# Subqueries

We examine the use of a complete **SELECT** statement embedded within another **SELECT** statement. The results of this inner **SELECT** statement (or **subselect**) are used in the outer statement to help determine the contents of the final result. A sub-select can be used in the **WHERE** and **HAVING** clauses of an outer SELECT statement, where it is called **a subquery or nested query**. **Subselects** may also appear in **INSERT**, **UPDATE**, and **DELETE** statements.

There are three types of subquery:

- A *scalar subquery* returns a single column and a single row, that is, a single value. In principle, a scalar subquery can be used whenever a single value is needed. Example 6.19 uses a scalar subquery.
- A *row subquery* returns multiple columns, but only a single row. A row subquery can be used whenever a row value constructor is needed, typically in predicates.
- A *table subquery* returns one or more columns and multiple rows. A table subquery can be used whenever a table is needed, for example, as an operand for the IN predicate.

## Using a subquery with equality

*List the staff who work in the branch at '163 Main St'.*

```
SELECT staffNo, fName, lName, position
FROM Staff
WHERE branchNo = (SELECT branchNo
                  FROM Branch
                  WHERE street = '163 Main St');
```

The inner SELECT statement (SELECT branchNo FROM Branch . . .) finds the branch number that corresponds to the branch with street name '163 Main St' (there will be only one such branch number, so this is an example of a scalar subquery). Having obtained this branch number, the outer SELECT statement then retrieves the details of all staff who work at this branch. In other words, the inner SELECT returns a result table containing a single value 'B003', corresponding to the branch at '163 Main St', and the outer SELECT becomes:

```
SELECT staffNo, fName, lName, position
FROM Staff
WHERE branchNo = 'B003';
```

**Result:**

| staffNo | fName | lName | position |
|---|---|---|---|
| SG37 | Ann | Beech | Assistant |
| SG14 | David | Ford | Supervisor |
| SG5 | Susan | Brand | Manager |

We can think of the subquery as producing a temporary table with results that can be accessed and used by the outer statement. A subquery can be used immediately following a relational operator (=, <, >, <=, > =,< >) in a WHERE clause, or a HAVING clause. The subquery itself is always enclosed in parentheses.

## <u>Using a subquery with an aggregate function</u>

*List all staff whose salary is greater than the average salary, and show by how much their salary is greater than the average.*

```
SELECT staffNo, fName, lName, position,
       salary – (SELECT AVG(salary) FROM Staff) AS salDiff
FROM Staff
WHERE salary > (SELECT AVG(salary) FROM Staff);
```

First, note that we cannot write 'WHERE salary > AVG(salary)', because aggregate functions cannot be used in the WHERE clause. Instead, we use a subquery to find the average salary, and then use the outer SELECT statement to find those staff with a salary greater than this average. In other words, the subquery returns the average salary as £17,000. Note also the use of the scalar subquery in the SELECT list to determine the difference from the average salary. The outer query is reduced then to:

```
SELECT staffNo, fName, lName, position, salary – 17000 AS salDiff
FROM Staff
WHERE salary > 17000;
```

---

**Result:**

| staffNo | fName | lName | position | salDiff |
|---|---|---|---|---|
| SL21 | John | White | Manager | 13000.00 |
| SG14 | David | Ford | Supervisor | 1000.00 |
| SG5 | Susan | Brand | Manager | 7000.00 |

The following rules apply to subqueries:

(1) The ORDER BY clause may not be used in a subquery (although it may be used in the outermost SELECT statement).

(2) The subquery SELECT list must consist of a single column name or expression, except for subqueries that use the keyword EXISTS

(3) By default, column names in a subquery refer to the table name in the FROM clause of the subquery. It is possible to refer to a table in a FROM clause of an outer query by qualifying the column name (see following).

(4) When a subquery is one of the two operands involved in a comparison, the subquery must appear on the right-hand side of the comparison. For example, it would be incorrect to express the previous example as:

```
SELECT staffNo, fName, lName, position, salary
FROM Staff
WHERE (SELECT AVG(salary) FROM Staff) < salary;
```

because the subquery appears on the left-hand side of the comparison with salary.

**Nested subqueries: use of IN**

*List the properties that are handled by staff who work in the branch at '163 Main St'.*

```
SELECT propertyNo, street, city, postcode, type, rooms, rent
FROM PropertyForRent
WHERE staffNo IN (SELECT staffNo
                  FROM Staff
                  WHERE branchNo = (SELECT branchNo
                                    FROM Branch
                                    WHERE street = '163 Main St'));
```

Working from the innermost query outwards, the first query selects the number of the branch at **'163 Main St'**. The second query then selects those staff who work at this branch number. In this case, there may be more than one such row found, and so we cannot use the equality condition **(=)** in the outermost query. Instead, we use the **IN** keyword. The outermost query then retrieves the details of the properties that are managed by each member of staff identified in the middle query.

**Result:**

| propertyNo | street | city | postcode | type | rooms | rent |
|---|---|---|---|---|---|---|
| PG16 | 5 Novar Dr | Glasgow | G12 9AX | Flat | 4 | 450 |
| PG36 | 2 Manor Rd | Glasgow | G32 4QX | Flat | 3 | 375 |
| PG21 | 18 Dale Rd | Glasgow | G12 | House | 5 | 600 |

## ANY and ALL

The keywords ANY and ALL may be used with subqueries that produce a single column of numbers. If the subquery is preceded by the keyword ALL, the condition will be true only if it is satisfied by all values produced by the subquery. If the subquery is preceded by the keyword ANY, the condition will be true if it is satisfied by any (one or more) values produced by the subquery. If the subquery is

empty, the ALL condition returns true, the ANY condition returns false. The ISO standard also allows the qualifier SOME to be used in place of ANY.

## Use of ANY/SOME

*Find all staff whose salary is larger than the salary of at least one member of staff at branch B003.*

```
SELECT staffNo, fName, lName, position, salary
FROM Staff
WHERE salary > SOME (SELECT salary
                     FROM Staff
                     WHERE branchNo = 'B003');
```

---

Although this query can be expressed using a **subquery** that finds the minimum salary of the staff at branch B003 and then an outer query that finds all staff whose salary is greater than this number (see Example 6.20), an alternative approach uses the **SOME/ ANY** keyword. The inner query produces the set {12000, 18000, 24000} and the outer query selects those staff whose salaries are greater than any of the values in this set (that is, greater than the minimum value, 12000). This alternative method may seem more natural than finding the minimum salary in a subquery.

**Result:**

| staffNo | fName | lName | position | salary |
|---|---|---|---|---|
| SL21 | John | White | Manager | 30000.00 |
| SG14 | David | Ford | Supervisor | 18000.00 |
| SG5 | Susan | Brand | Manager | 24000.00 |

## <u>Use of ALL</u>

*Find all staff whose salary is larger than the salary of every member of staff at branch B003.*

```
SELECT staffNo, fName, lName, position, salary
FROM Staff
WHERE salary > ALL (SELECT salary
                    FROM Staff
                    WHERE branchNo = 'B003');
```

---

This example is very similar to the previous example. Again, we could use a **subquery** to find the maximum salary of staff at branch **B003** and then use an outer query to find all staff whose salary is greater than this number. However, in this example we use the **ALL** keyword.

**Result:**

| staffNo | fName | lName | position | salary |
|---|---|---|---|---|
| SL21 | John | White | Manager | 30000.00 |

## Multi-table Queries

All the examples we have considered so far have a major limitation: the columns that are to appear in the result table must all come from a single table. In many cases, this is insufficient to answer common queries that users will have. To combine columns from several tables into a result table, we need to use a **join** operation. The SQL join operation combines information from two tables by forming pairs of related rows from the two tables. The row pairs that make up the joined table are those where the matching columns in each of the two tables have the same value.

If we need to obtain information from more than one table, the choice is between using a subquery and using a join. If the final result table is to contain columns from different tables, then we must use a join. To perform a join, we simply include more than one table name in the FROM clause, using a comma as a separator, and typically including a WHERE clause to specify the join column(s). It is also possible to use an **alias** for a table named in the FROM clause. In this case, the alias is separated from the table name with a space. An alias can be used to qualify a column name whenever there is ambiguity regarding the source of the column name. It can also be used as a shorthand notation for the table name. If an alias is provided, it can be used anywhere in place of the table name.

## Sorting a join

*For each branch office, list the staff numbers and names of staff who manage properties and the properties that they manage.*

```
SELECT s.branchNo, s.staffNo, fName, lName, propertyNo
FROM Staff s, PropertyForRent p
WHERE s.staffNo = p.staffNo
ORDER BY s.branchNo, s.staffNo, propertyNo;
```

In this example, we need to join the Staff and **PropertyForRent** tables, based on the primary key/foreign key attribute (**staffNo**). To make the results more readable, we have ordered the output using the branch number as the major sort key and the staff number and property number as the minor keys.

**Result:**

| branchNo | staffNo | fName | lName | propertyNo |
|---|---|---|---|---|
| B003 | SG14 | David | Ford | PG16 |
| B003 | SG37 | Ann | Beech | PG21 |
| B003 | SG37 | Ann | Beech | PG36 |
| B005 | SL41 | Julie | Lee | PL94 |
| B007 | SA9 | Mary | Howe | PA14 |

## Three-table join

*For each branch, list the staff numbers and names of staff who manage properties, including the city in which the branch is located and the properties that the staff manage.*

```
SELECT b.branchNo, b.city, s.staffNo, fName, lName, propertyNo
FROM Branch b, Staff s, PropertyForRent p
WHERE b.branchNo = s.branchNo AND s.staffNo = p.staffNo
ORDER BY b.branchNo, s.staffNo, propertyNo;
```

The result table requires columns from three tables: Branch (**branchNo** and **city**), Staff (**staffNo, fName and lName**), and **PropertyForRent** (**propertyNo**), so a join must be used. The Branch and Staff details are joined using the condition (**b.branchNo = s.branchNo**) to link each branch to the staff who work there. The Staff and PropertyForRent details are joined using the condition (**s.staffNo = p.staffNo**) to link staff to the properties they manage.

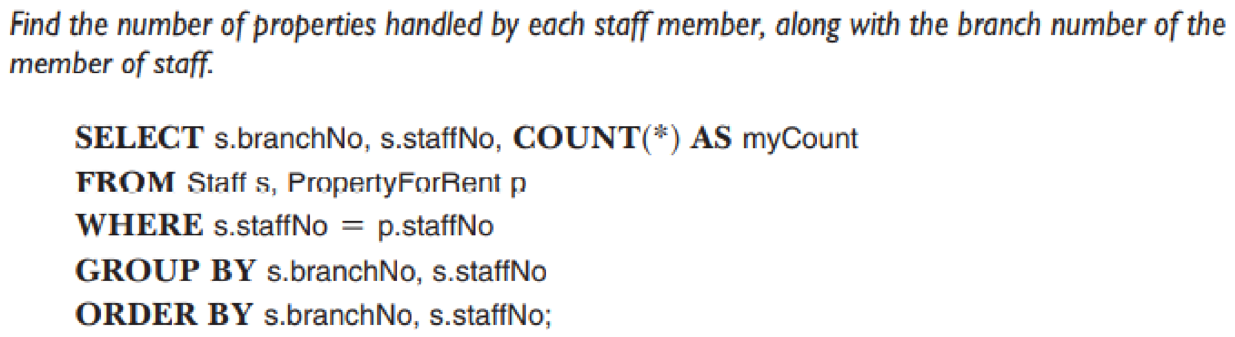
**Result:**

| branchNo | city | staffNo | fName | lName | propertyNo |
|---|---|---|---|---|---|
| B003 | Glasgow | SG14 | David | Ford | PG16 |
| B003 | Glasgow | SG37 | Ann | Beech | PG21 |
| B003 | Glasgow | SG37 | Ann | Beech | PG36 |
| B005 | London | SL41 | Julie | Lee | PL94 |
| B007 | Aberdeen | SA9 | Mary | Howe | PA14 |

Note again that the SQL standard provides alternative formulations for the FROM and WHERE clauses, for example:

```
FROM (Branch b JOIN Staff s USING branchNo) AS bs
                JOIN PropertyForRent p USING staffNo
```

**Multiple grouping columns**

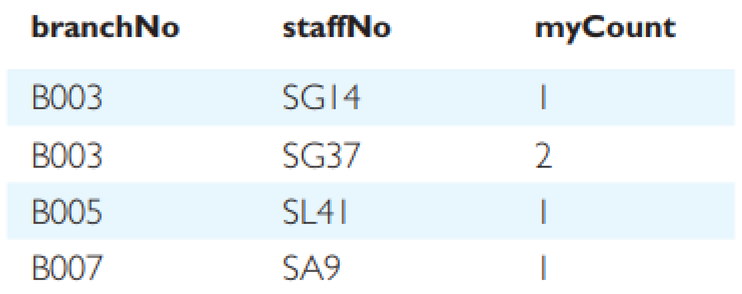


*Find the number of properties handled by each staff member, along with the branch number of the member of staff.*

```
SELECT s.branchNo, s.staffNo, COUNT(*) AS myCount
FROM Staff s, PropertyForRent p
WHERE s.staffNo = p.staffNo
GROUP BY s.branchNo, s.staffNo
ORDER BY s.branchNo, s.staffNo;
```

To list the required numbers, we first need to find out which staff actually manage properties. This can be found by joining the Staff and **PropertyForRent** tables on the **staffNo** column, using the **FROM/WHERE** clauses. Next, we need to form groups consisting of the branch number and staff number, using the **GROUP BY** clause. Finally, we sort the output using the **ORDER BY** clause.

**Result:**



| branchNo | staffNo | myCount |
|---|---|---|
| B003 | SG14 | 1 |
| B003 | SG37 | 2 |
| B005 | SL41 | 1 |
| B007 | SA9 | 1 |

**END**