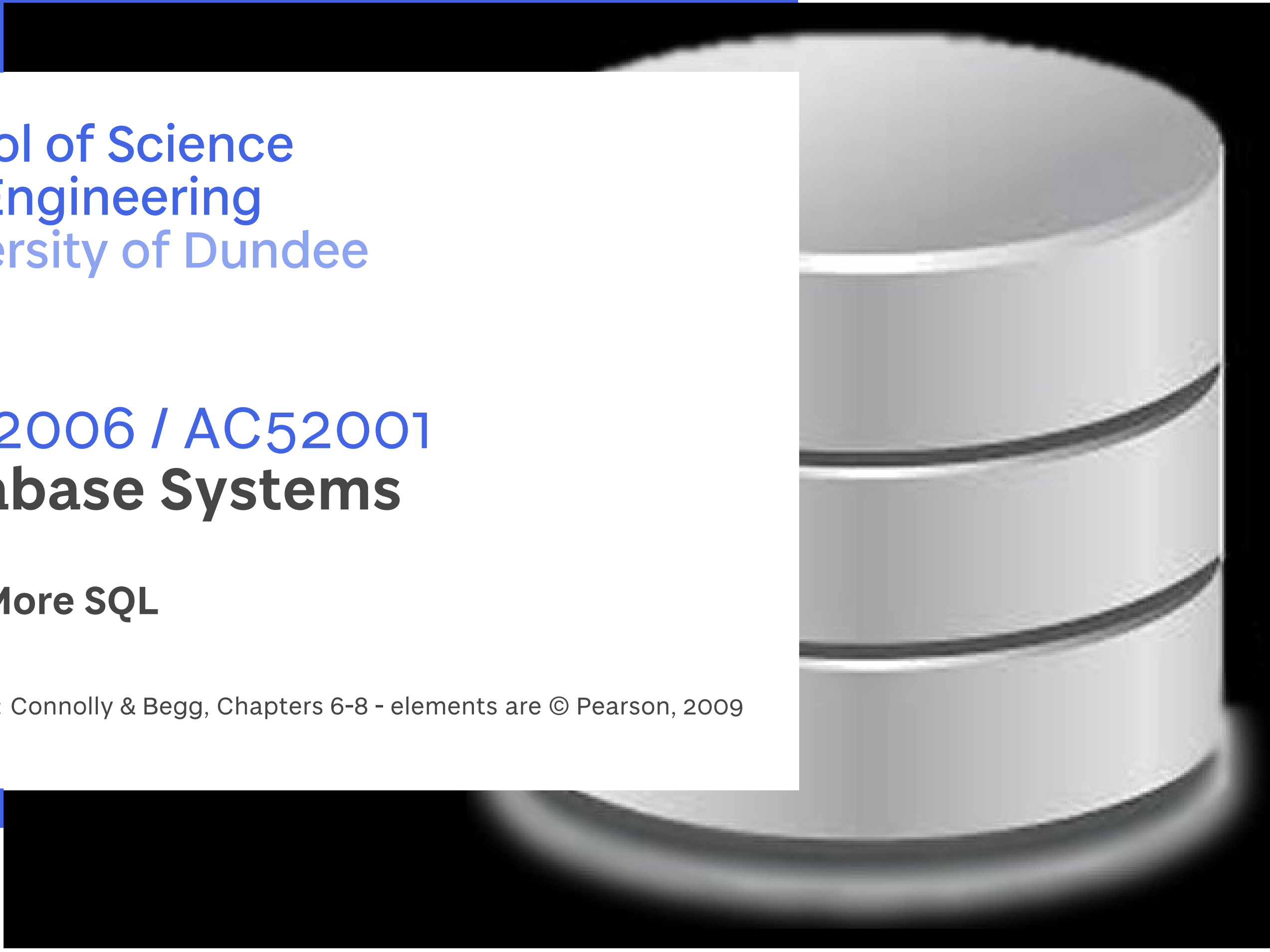


School of Science
and Engineering
University of Dundee

AC32006 / AC52001
Database Systems

Even More SQL

Reference: Connolly & Begg, Chapters 6-8 - elements are © Pearson, 2009

In this video …

→queries across **multiple** tables

→types of joins

→a few more SQL commands and functions

Relationships

Remember that relationships i.e. links between tables, are not actually *stored* - they are *virtual* within a relational database!

However, the database tables will have been designed to *facilitate linking* (using appropriate primary and foreign keys) … but the actual links between the tables are only made dynamically when the database is *manipulated* e.g. using a query

Multi-Table Queries

- →We can use subqueries if the result columns come from a single table (in each part)
- →If result columns come from *more than one table*, we must use a join
- →There are several ways to perform a join:
  - → simplest is to include *more than one table* in the **`FROM`** clause using a comma as a separator and typically include a **`WHERE`** clause to specify the joining columns
  - → SQL performs the join *itself* as required

Multi-Table Queries

- We can use an alias for a table named in the `FROM` clause:
  - using short alias names saves us some typing when formulating an SQL query
  - alias is separated from table name using a space
  - alias can be used to qualify column names when there is ambiguity

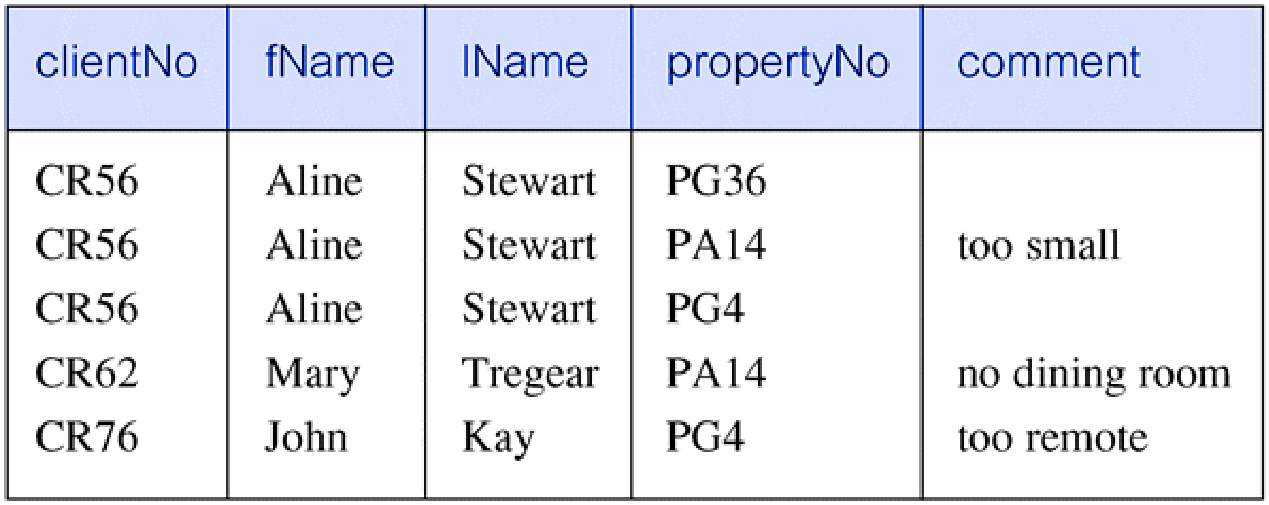
Example - Simple Join

→ List names of all clients who have viewed a property, along with any comment supplied:

```
SELECT c.clientNo, fName, lName,
           propertyNo, comment
     FROM Client c, Viewing v
     WHERE c.clientNo = v.clientNo;
```

Example - Simple Join

→ Only those rows from both tables which have *identical values in the clientNo columns* (`c.clientNo = v.clientNo`) are included in the result:



clientNo	fName	lName	propertyNo	comment
CR56	Aline	Stewart	PG36	
CR56	Aline	Stewart	PA14	too small
CR56	Aline	Stewart	PG4	
CR62	Mary	Tregear	PA14	no dining room
CR76	John	Kay	PG4	too remote

Alternative JOIN Constructs

→SQL provides *alternative ways* to explicitly specify joins:

```
FROM Client c JOIN Viewing v ON
    c.clientNo = v.clientNo
FROM Client JOIN Viewing USING clientNo
FROM Client NATURAL JOIN Viewing
```

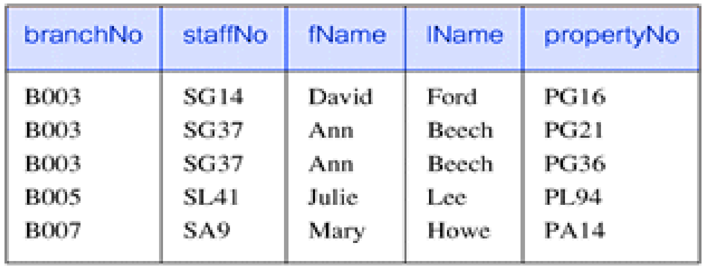
→In each, **FROM** replaces the original **FROM** and **WHERE**; however, note that the first produces a table with *two identical* clientNo columns

Natural Join joins on column names which are *the same* across both tables

Example - Sorting a join

→ For each branch, list numbers and names of staff who manage properties, and the properties that they manage:

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



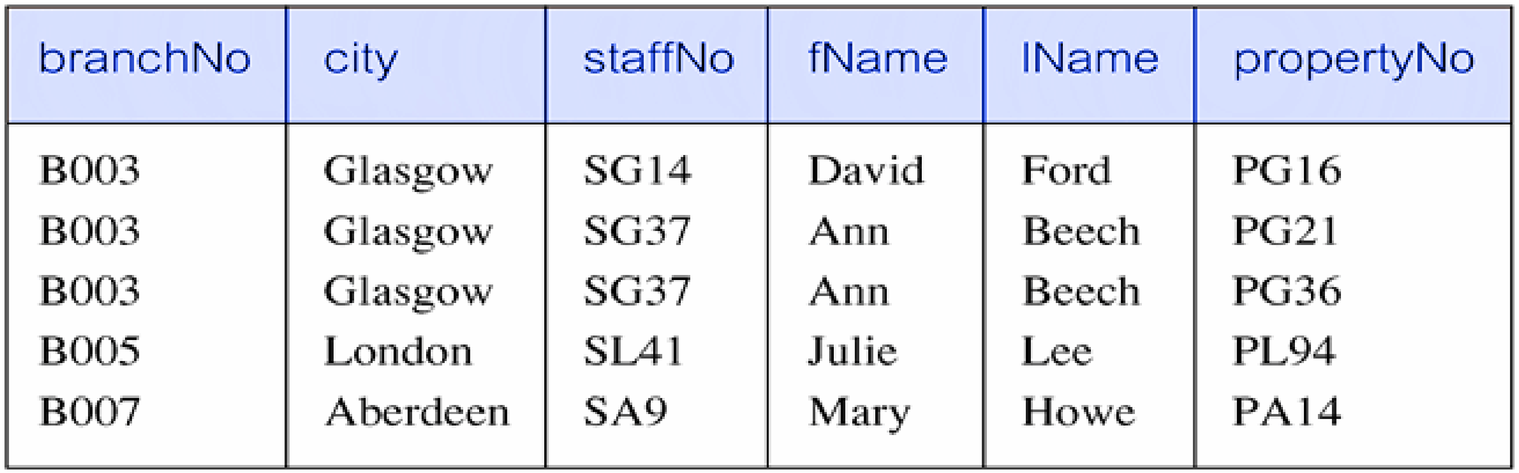
branchNo	staffNo	fName	lName	propertyNo
B003	SG14	David	Ford	PG16
B003	SG37	Ann	Beech	PG21
B003	SG37	Ann	Beech	PG36
B005	SL41	Julie	Lee	PL94
B007	SA9	Mary	Howe	PA14

Example - Three-Table Join

→ For each branch, list staff who manage properties, including the city where branch is located and the properties that they manage:

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

Example - Three Table Join



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

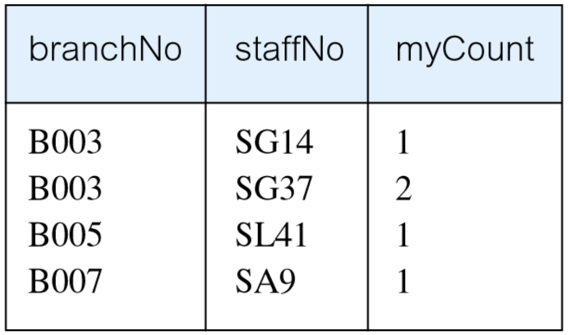
→ Alternative formulation for **FROM** and **WHERE**:

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

Example - Multiple Grouping Columns

→Find the total number of properties handled by each staff member:

```
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;
```



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

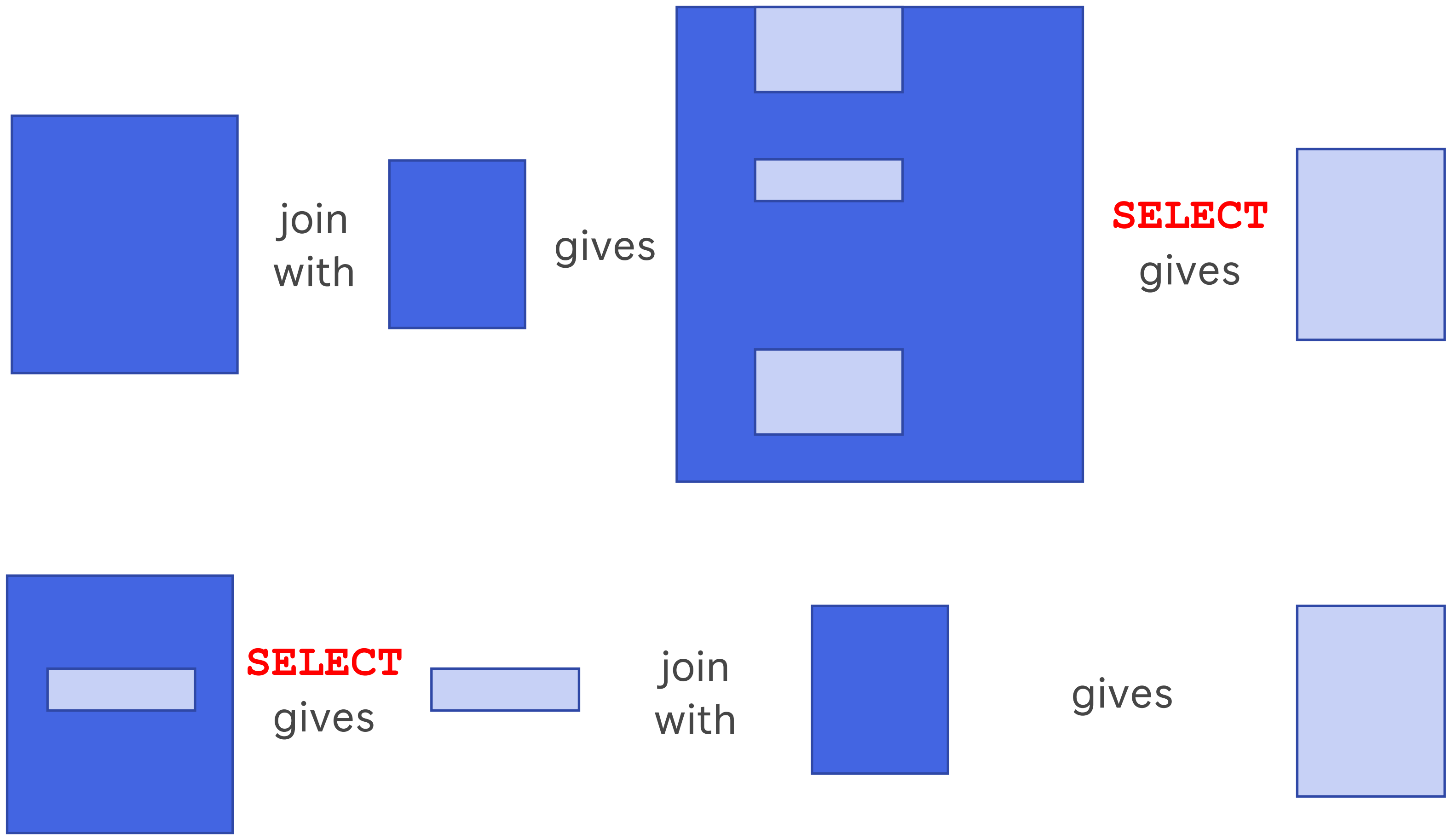
Multi-Table Queries - caveat

Although unseen by the user, any operation which joins tables creates a *temporary table* - although temporary, this table still requires memory/disk space and processing power!

The temporary table may be *much larger* than the tables that it is generated from, so we may need to be *careful* about how we create joins.

If query performance is bad, we may need to reformulate a query e.g. such that **`SELECT`**s are done *before* joins in order to minimize the size of the temporary join table.

Multi-Table Queries - beware



Computing a Join (manually)

Procedure for generating results of a join is:

1. Form the *Cartesian product* of the tables named the in **`FROM`** clause
2. If there is a **`WHERE`** clause, apply the search condition to each row of the product table, retaining those rows which satisfy the condition
3. For each remaining row, determine the value of each item in the **`SELECT`** list to produce a single row in the result table

Computing a Join (manually)

4. If `DISTINCT` has been specified, eliminate any duplicate rows from the result table
5. If there is an `ORDER BY` clause, sort result table as required

→SQL provides a special format of `SELECT` for Cartesian product:

```
SELECT [DISTINCT | ALL] {* |
    columnList}
    FROM Table1 CROSS JOIN Table2
```

Example - Performing a Join

People

Name	Job
Fred Smith	Manager
Jim Spriggs	Supervisor
Tom Wapcaplet	Trainee

Pay

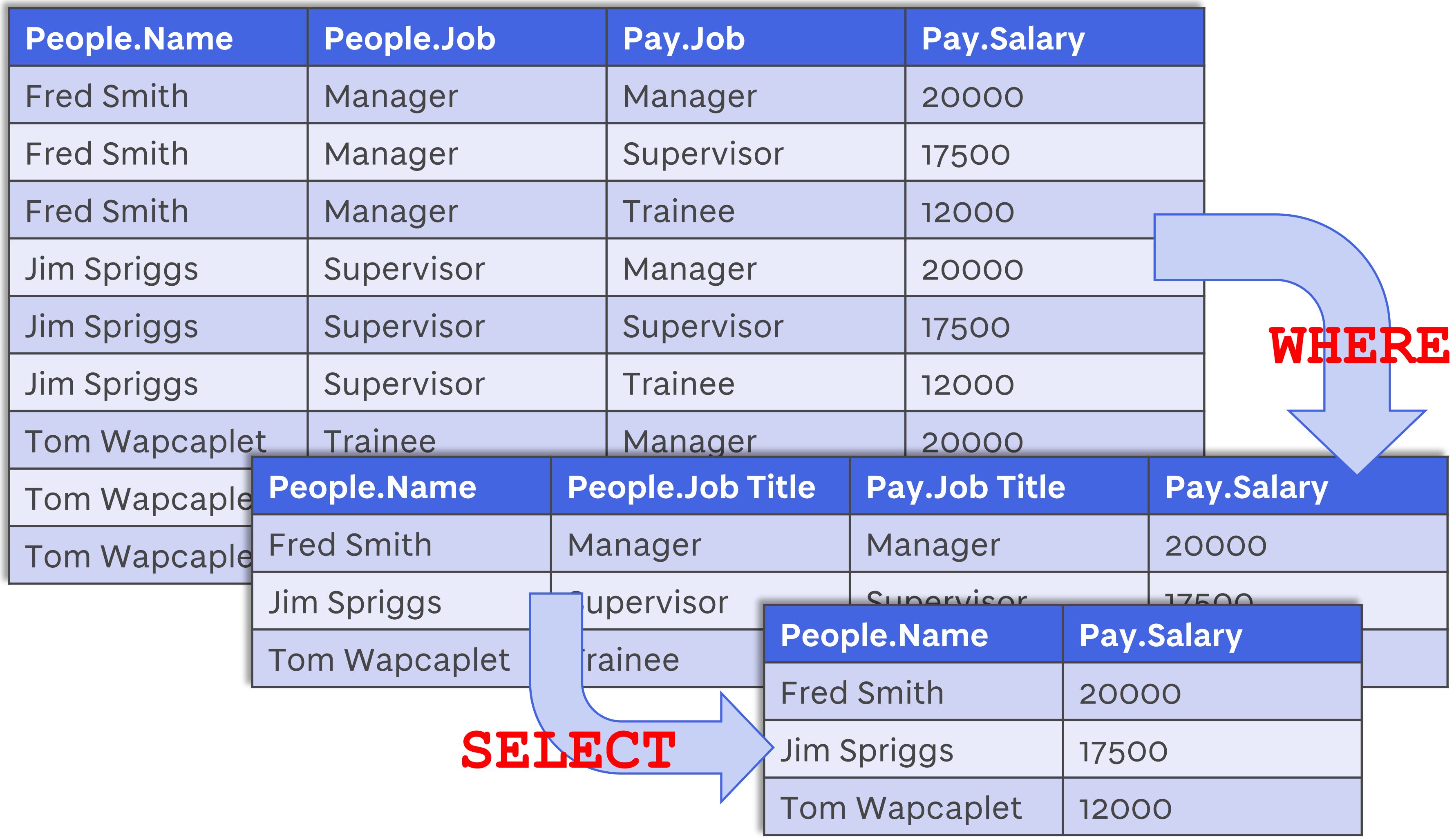
Job	Salary
Manager	20000
Supervisor	17500
Trainee	12000

→List each person together with their salary:

```
SELECT p.people, w.salary
     FROM People p, Pay w
     WHERE p.Job = w.job;
```

an Inner Join

Example - Performing a Join

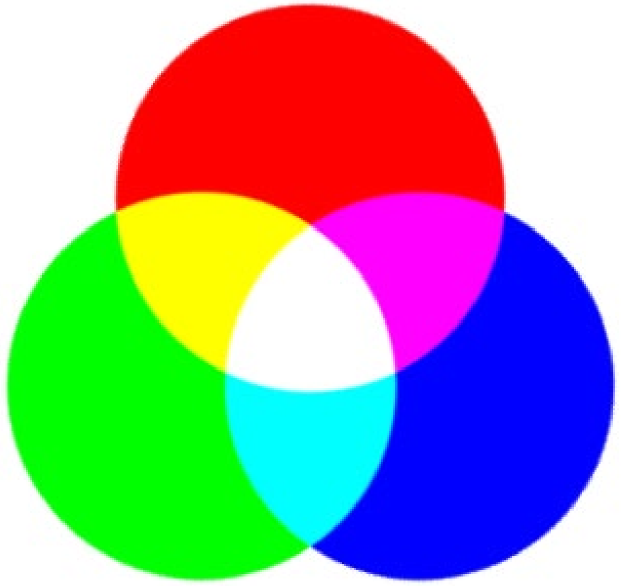


People.Name	People.Job	Pay.Job	Pay.Salary
Fred Smith	Manager	Manager	20000
Fred Smith	Manager	Supervisor	17500
Fred Smith	Manager	Trainee	12000
Jim Spriggs	Supervisor	Manager	20000
Jim Spriggs	Supervisor	Supervisor	17500
Jim Spriggs	Supervisor	Trainee	12000
Tom Wapcaplet	Trainee	Manager	20000
Tom Wapcaple			
Tom Wapcaple			

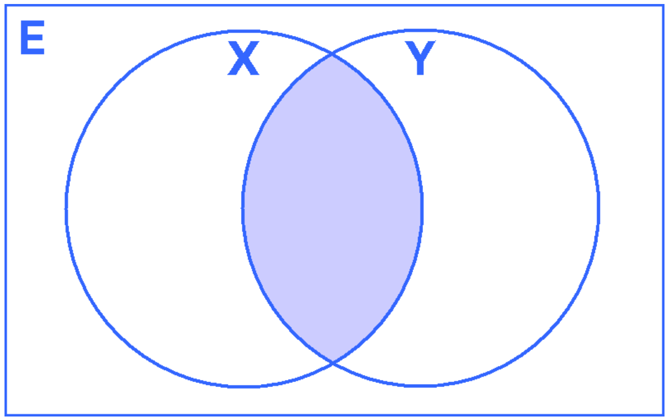
People.Name	People.Job Title	Pay.Job Title	Pay.Salary
Fred Smith	Manager	Manager	20000
Jim Spriggs	upervisor	Supervisor	17500
Tom Wapcaplet	rainee		

People.Name	Pay.Salary
Fred Smith	20000
Jim Spriggs	17500
Tom Wapcaplet	12000

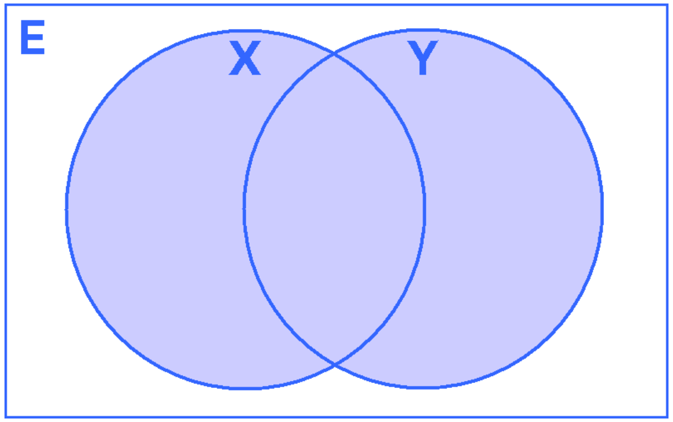
More types of JOIN



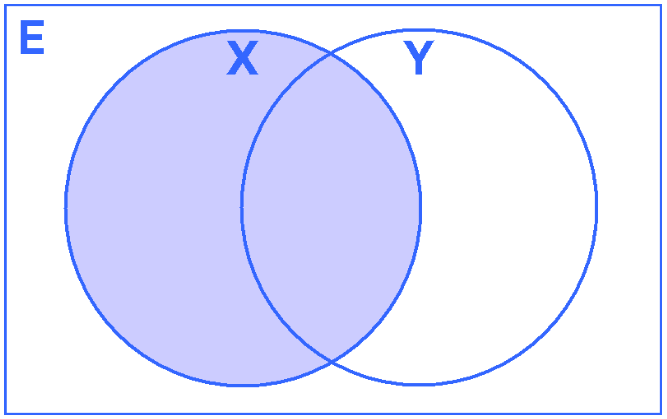
JOIN types



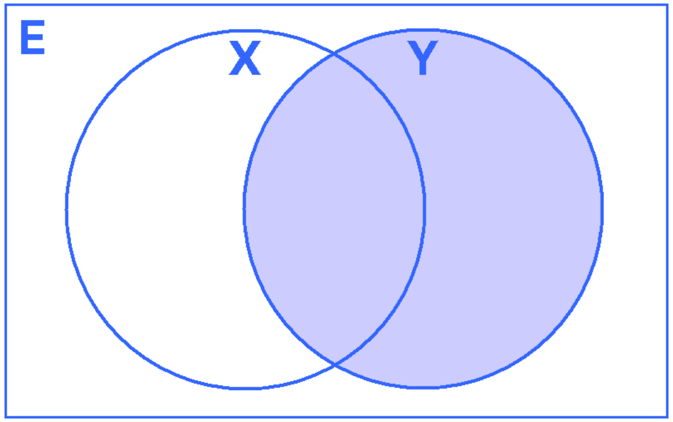
(INNER) JOIN



CROSS JOIN (Cartesian join)



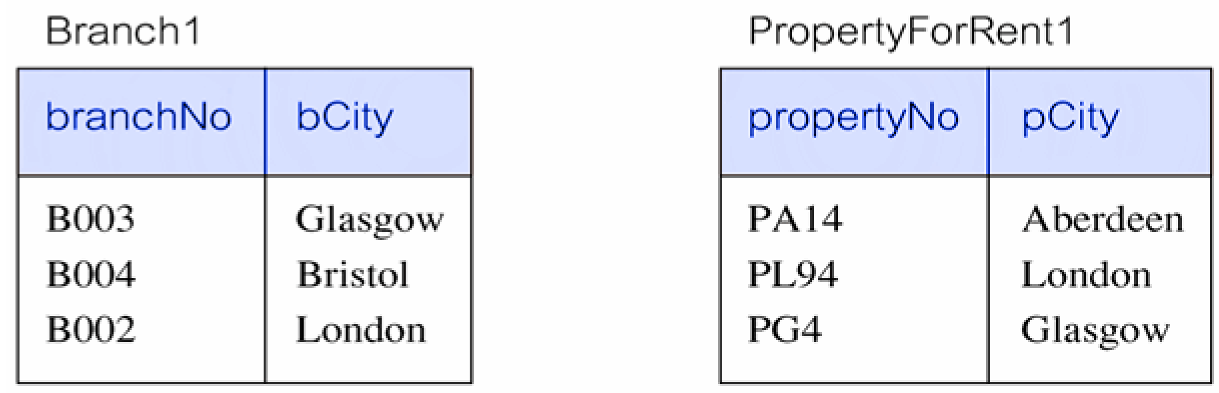
LEFT OUTER JOIN



RIGHT OUTER JOIN

More Joins

- With an inner join, if one row of a joined table is unmatched, that row is *omitted* from the result table
- To include unmatched rows in result table, use an outer join
- Consider:



Branch1

branchNo	bCity
B003	Glasgow
B004	Bristol
B002	London

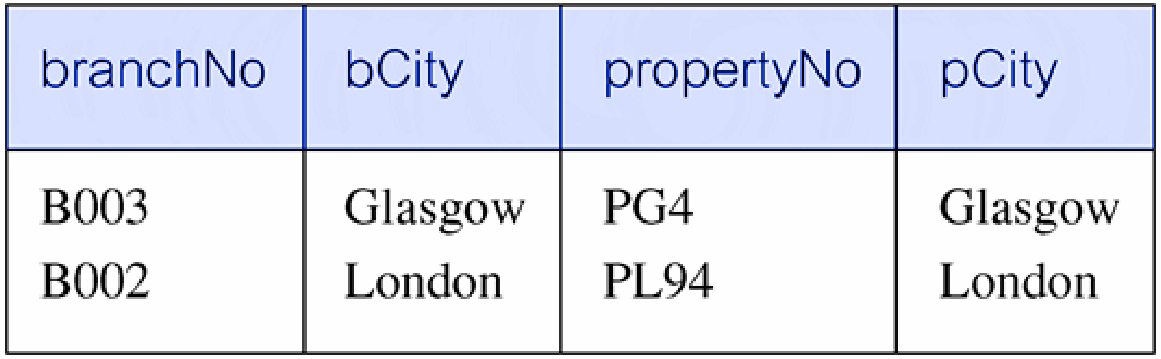
PropertyForRent1

propertyNo	pCity
PA14	Aberdeen
PL94	London
PG4	Glasgow

Example - Inner Join

The (inner) join of these two tables:

```
SELECT b.*, p.*
    FROM Branch1 b, PropertyForRent1 p
    WHERE b.bCity = p.pCity;
```



branchNo	bCity	propertyNo	pCity
B003	Glasgow	PG4	Glasgow
B002	London	PL94	London

Example - Inner Join

- → The result table has two rows where the cities are the same
- → … but there are no rows corresponding to branches in Bristol and Aberdeen (as these values are *unmatched*)

Example - Left Outer Join

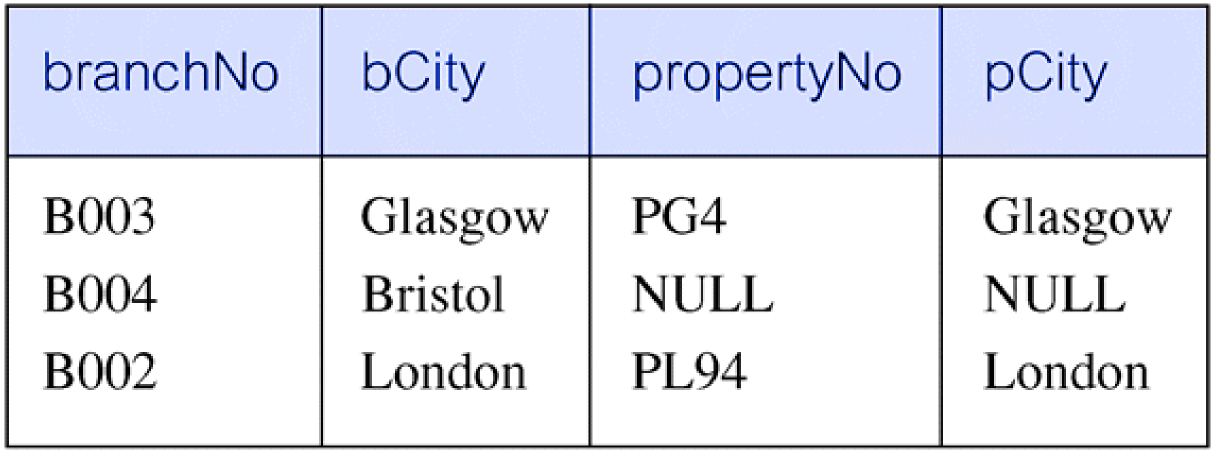
→ List branches and properties that are in same city along with any unmatched branches:

```
SELECT b.*, p.*
    FROM Branch1 b LEFT JOIN
        PropertyForRent1 p ON
        b.bCity = p.pCity;
```

Example - Left Outer Join

→ Left outer join will include all rows from first (left) table *even where unmatched* with rows from second (right) table

→ Columns from second table are filled with NULLs



branchNo	bCity	propertyNo	pCity
B003	Glasgow	PG4	Glasgow
B004	Bristol	NULL	NULL
B002	London	PL94	London

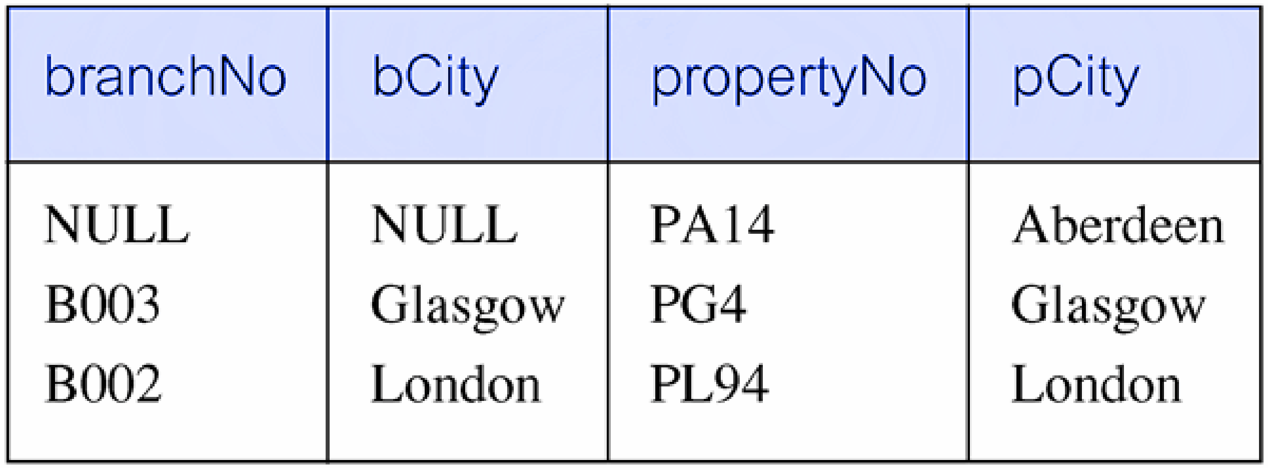
Example - Right Outer Join

→List branches and properties in same city and any unmatched properties:

```
SELECT b.*, p.*
    FROM Branch1 b RIGHT JOIN
        PropertyForRent1 p ON
        b.bCity = p.pCity;
```

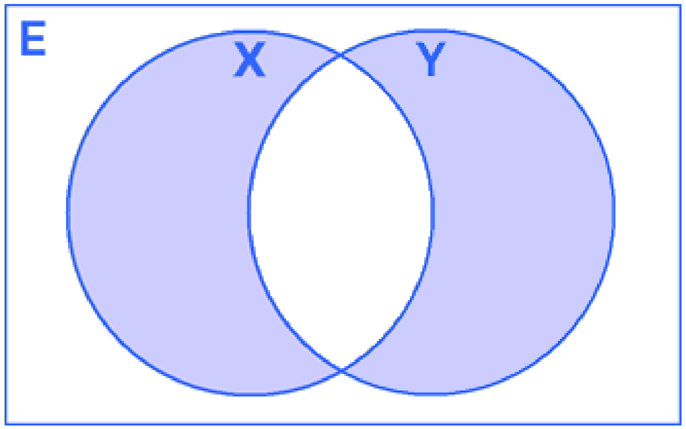
Example - Right Outer Join

- → Right outer join will include all rows from second (right) table *even where unmatched* with rows from first (left) table
- → Columns from first table are filled with NULLs



branchNo	bCity	propertyNo	pCity
NULL	NULL	PA14	Aberdeen
B003	Glasgow	PG4	Glasgow
B002	London	PL94	London

Example - Full (Outer) Join



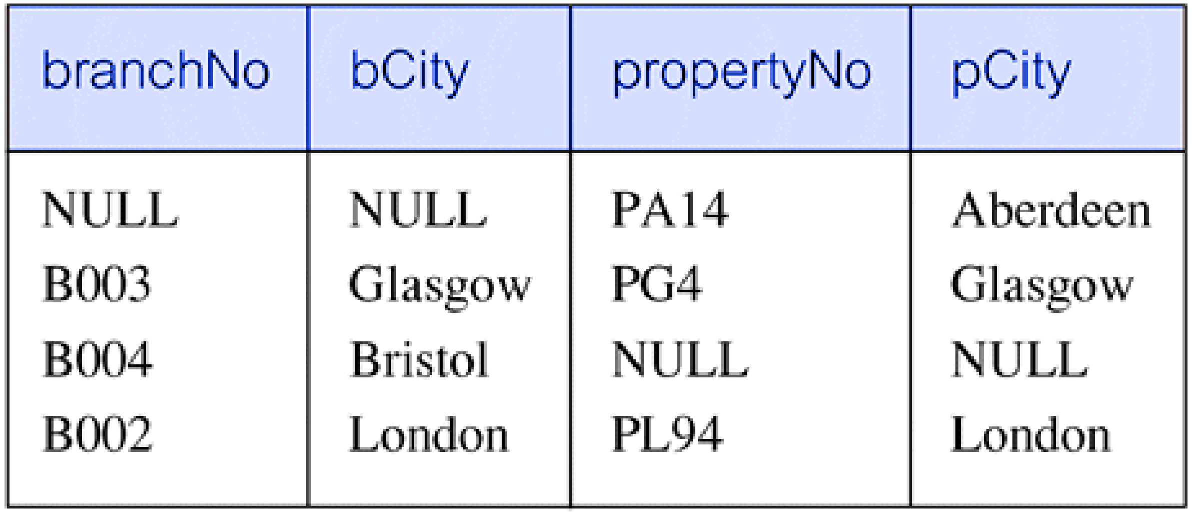
→List branches and properties in same city and any unmatched branches or properties:

Keyword is optional

```
SELECT b.*, p.*
     FROM Branch1 b FULL OUTER JOIN
          PropertyForRent1 p ON
          b.bCity = p.pCity;
```

Example - Full (Outer) Join

- Full outer join includes all rows including those that are unmatched in *either* direction
- Unmatched columns are filled with NULLs
- *Not the same* as a Cartesian cross join



branchNo	bCity	propertyNo	pCity
NULL	NULL	PA14	Aberdeen
B003	Glasgow	PG4	Glasgow
B004	Bristol	NULL	NULL
B002	London	PL94	London

EXISTS and NOT EXISTS

- → **`EXISTS`** and **`NOT EXISTS`** are for use *only with subqueries*
- → Produce a simple true/false result
- → True *if and only if* there exists *at least one row* in result table returned by the subquery
- → False if subquery returns an *empty result table*
- → **`NOT EXISTS`** is the negated form of **`EXISTS`**

EXISTS and NOT EXISTS

- As (**`NOT`**) **`EXISTS`** checks only for *existence* or *non-existence* of rows in subquery result table, the subquery can contain any number of columns
- It is common for subqueries following (**`NOT`**) **`EXISTS`** to be of the form:

```
    (SELECT * ...)
```

Example - Query using EXISTS

→Find all staff who work in a London branch:

```
SELECT staffNo, fName, lName, position
    FROM Staff s WHERE EXISTS
        (SELECT *
            FROM Branch b
            WHERE s.branchNo = b.branchNo
            AND city = 'London');
```

staffNo	fName	lName	position
SL21	John	White	Manager
SL41	Julie	Lee	Assistant

Example - Query using EXISTS

→ Note, search condition `s.branchNo = b.branchNo` is necessary to consider correct branch record for *each* member of staff

→ If omitted, we would get *all* staff records listed out because subquery:

```
SELECT *
    FROM Branch WHERE city='London'
```

would *always* be true, so query would be:

```
SELECT staffNo, fName, lName, position
    FROM Staff WHERE true;
```

Summary

We have seen:

→ multi-table queries using simple joins

→ queries with more sophisticated joins

→ more SQL querying functions

University of Dundee

dundee.ac.uk