The person Table

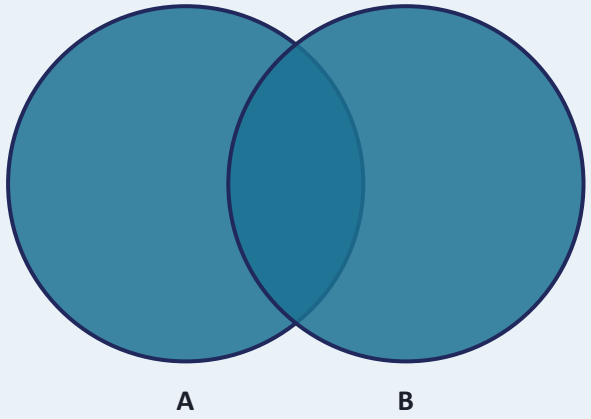
A small table we'll use throughout this deck

name	gender	occupation
Sarah Johnson	Female	Nurse
Michael Chen	Male	Nurse
David Martinez	Male	Doctor
Emily Davis	Female	Doctor
James Wilson	Male	Nurse
Olivia Brown	Female	Engineer
Robert Taylor	Male	Engineer
Sophia Anderson	Female	Nurse
Daniel Thomas	Male	Nurse
Ava Garcia	Female	Engineer

Three Set Operations

Combine two sets to produce another set

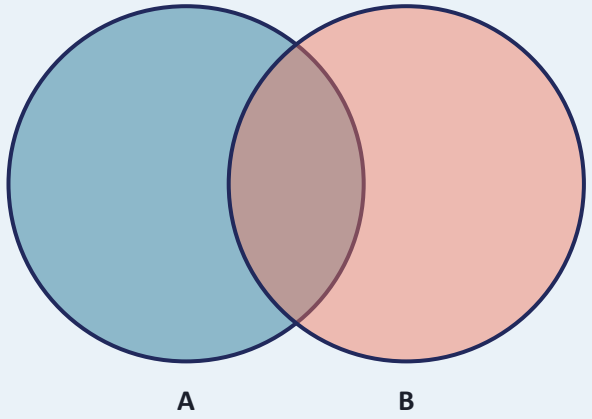
UNION



Everything in A, or B, or both. Duplicates removed.

$\{ 1, 2 \} \cup \{2, 3 \} = \{ 1, 2, 3 \}$

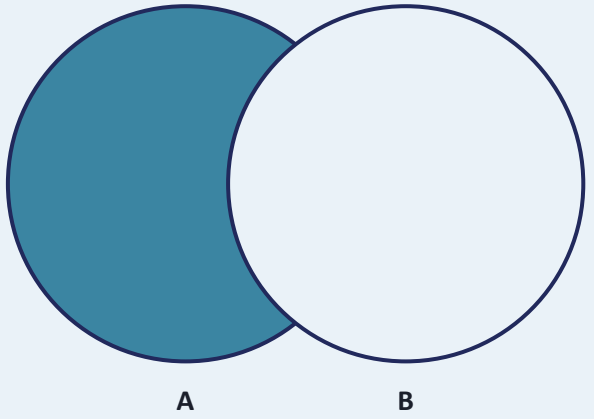
INTERSECT



Only the items that appear in both A and B.

$\{ 1, 2 \} \cap \{2, 3 \} = \{ 2 \}$

EXCEPT



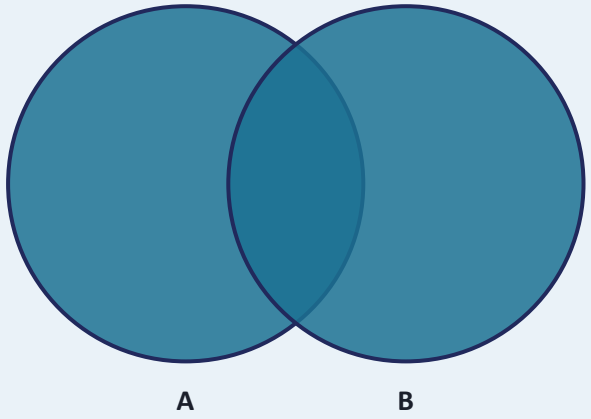
Items in A that do NOT also appear in B.

$\{ 1, 2 \} - \{2, 3 \} = \{ 1 \}$

SQL Set Operations

Combine the results of two queries that return the same columns

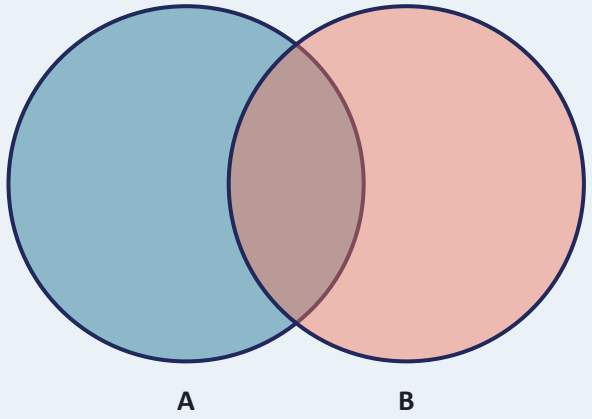
UNION



Everything in A, or B, or both. Duplicates removed.

```
SELECT * FROM A
UNION
SELECT * FROM B
```

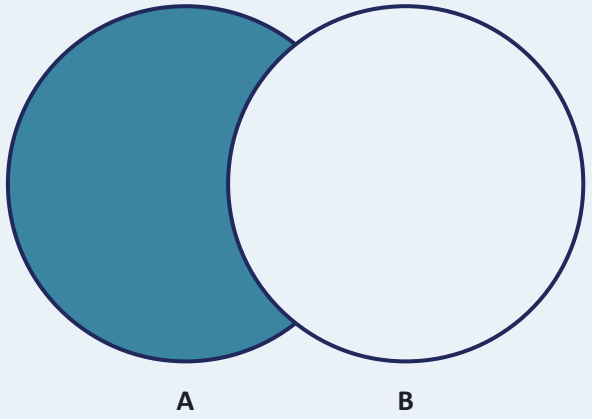
INTERSECT



Only the rows that appear in both A and B.

```
SELECT * FROM A
INTERSECT
SELECT * FROM B
```

EXCEPT

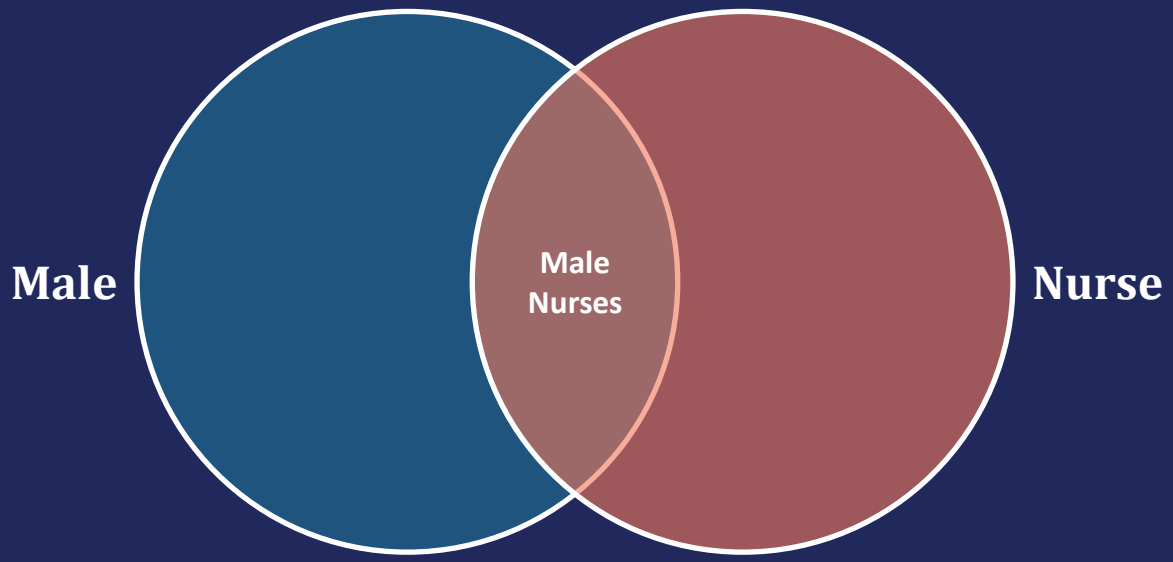


Rows in A that do NOT also appear in B.

```
SELECT * FROM A
EXCEPT
SELECT * FROM B
```

Who are the male nurses?

A person qualifies only if they belong to BOTH groups — that's a set intersection.



In set notation

Male ∩ Nurse

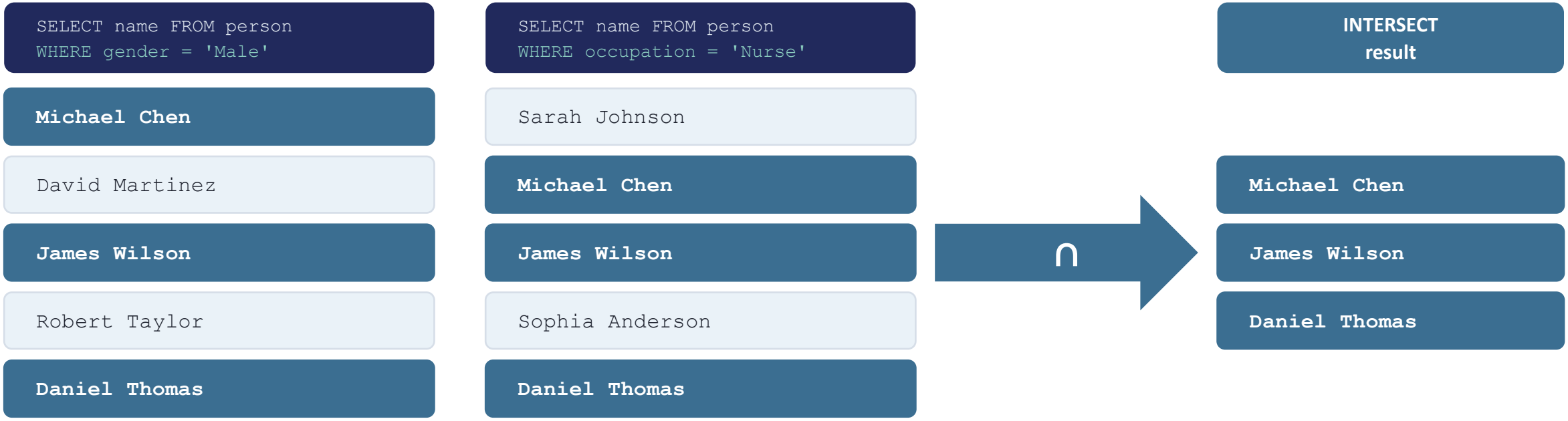
∩ = *intersection* = *“in both”*

Answer set: people who are gender = Male AND occupation = Nurse

Answering It With SQL

INTERSECT keeps only the rows that come back from both SELECT statements

```
SELECT name FROM person WHERE gender = 'Male' INTERSECT SELECT name FROM person WHERE occupation = 'Nurse';
```



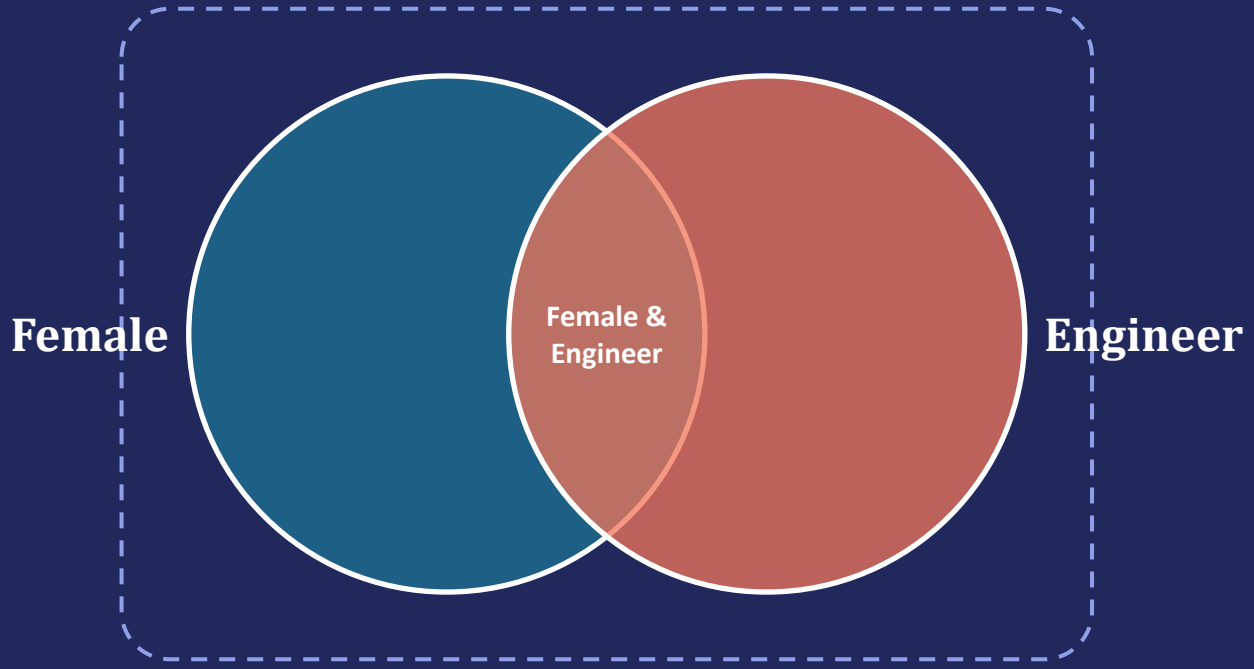
Highlighted names appear in both lists — only they survive into the INTERSECT result.

Same answer, one query using AND:

```
SELECT name FROM person WHERE gender = 'Male' AND occupation = 'Nurse';
```

Give a list of people who are either females or engineers

A person qualifies if they belong to EITHER group (or both) — that's a set union.



In set notation

Female ∪ Engineer

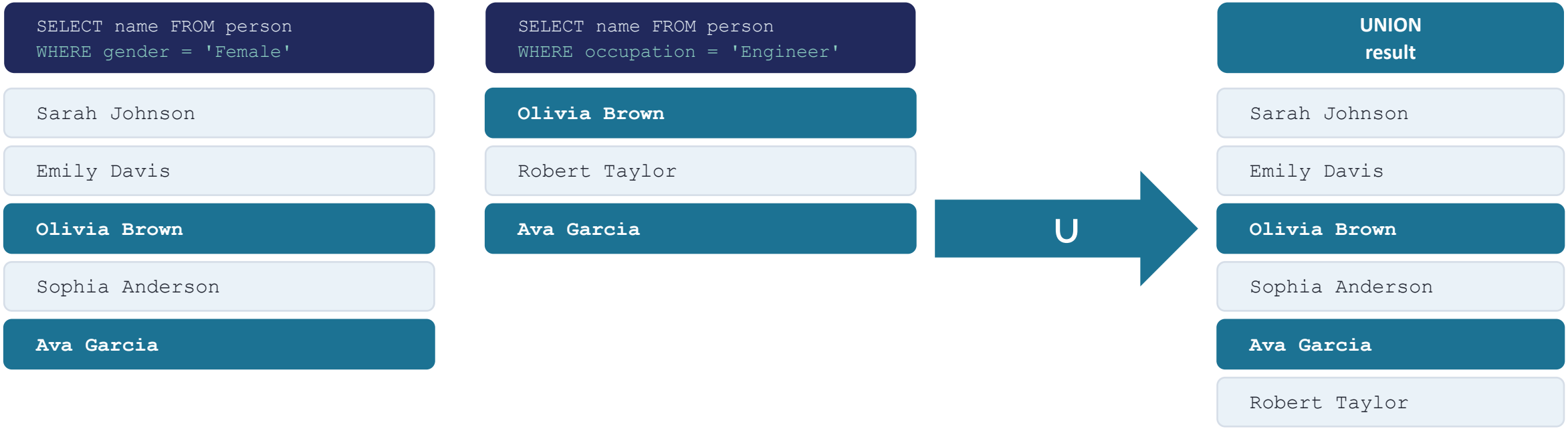
∪ = union = "either, or both"

Answer set: people who are gender = Female OR occupation = Engineer

Answering It With SQL

UNION combines rows from both SELECT statements and drops duplicates

```
SELECT name FROM person WHERE gender = 'Female' UNION SELECT name FROM person WHERE occupation = 'Engineer';
```



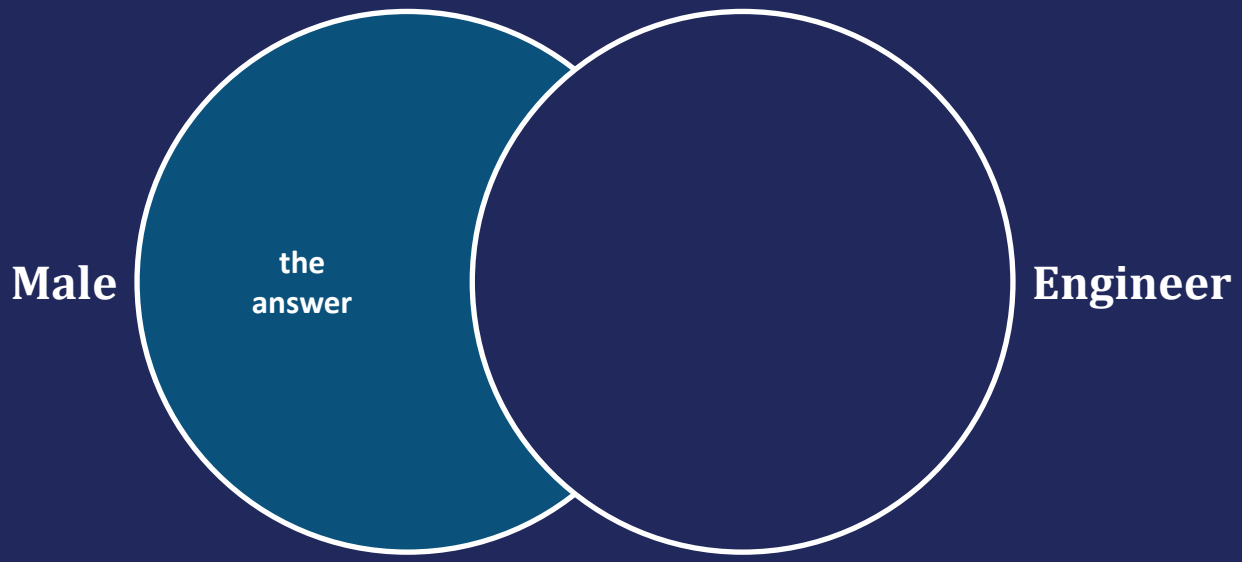
Highlighted names appear in both lists — they're kept only once in the UNION result.

Same answer, one query using OR:

```
SELECT name FROM person WHERE gender = 'Female' OR occupation = 'Engineer';
```

What men are not engineers?

A person qualifies if they're in group A but NOT in group B — that's a set difference.



In set notation

Male – Engineer

– = difference = "in A, not in B"

Answer set: people who are gender = Male AND NOT occupation = Engineer

Answering It With SQL

EXCEPT keeps rows from the first SELECT that don't also appear in the second

```
SELECT name FROM person WHERE gender = 'Male' EXCEPT SELECT name FROM person WHERE occupation = 'Engineer';
```

SELECT name FROM person WHERE gender = 'Male'	SELECT name FROM person WHERE occupation = 'Engineer'		EXCEPT result
Michael Chen	Olivia Brown		**Michael Chen**
David Martinez	~~Robert Taylor~~		**David Martinez**
James Wilson	Ava Garcia	−	**James Wilson**
~~Robert Taylor~~ *removed*			**Daniel Thomas**
Daniel Thomas			

Robert Taylor appears in both lists, so EXCEPT removes him — everyone else in the Male list survives.

Same answer, one query using <>:

```
SELECT name FROM person WHERE gender = 'Male' AND occupation <> 'Engineer';
```

The person Table, Revisited

Some people hold more than one occupation — they simply appear in more than one row

name	gender	occupation
Sarah Johnson	Female	Nurse
Michael Chen	Male	Nurse
David Martinez	Male	Doctor
Emily Davis	Female	Doctor
James Wilson	Male	Nurse
James Wilson	Male	Engineer
Olivia Brown	Female	Engineer
Robert Taylor	Male	Engineer
Sophia Anderson	Female	Nurse
Daniel Thomas	Male	Nurse
Ava Garcia	Female	Engineer
Ava Garcia	Female	Doctor

The Negation Trap

"What men are not engineers?" — <> produces wrong results now

```
SELECT name FROM person WHERE gender = 'Male' AND occupation <> 'Engineer';
```

The Male records, checked one row at a time:

name	gender	occupation	occupation <> 'Engineer'
Michael Chen	Male	Nurse	✓
David Martinez	Male	Doctor	✓
James Wilson	Male	Nurse	✓
James Wilson	Male	**Engineer**	—
Robert Taylor	Male	**Engineer**	—
Daniel Thomas	Male	Nurse	✓

↑ James Wilson has one row without a checkmark (he's an engineer) — but his ***other*** row still checks out, so he wrongly appears in the results.

What it actually returns:

Michael Chen	✓ *correct*
David Martinez	✓ *correct*
James Wilson	✗ *he's an engineer!*
Daniel Thomas	✓ *correct*

Splitting the Table: Person + Employed

Person

PersonID	Gender	Name
1	Male	Michael Chen
2	Male	David Martinez
3	Male	**James Wilson**
4	Female	Olivia Brown
5	Male	Robert Taylor

Employed

PersonID	Occupation	Salary
1	Nurse	$71,000
2	Doctor	$145,000
3	Nurse	$69,000
3	Engineer	$98,000
4	Engineer	$95,000
5	Engineer	$101,000

The IN Operator

Match a column against a list of values — a shortcut for chained ORs

The query:

```
SELECT Name FROM Person WHERE PersonID IN (3, 5);
```

IN is true if PersonID matches ***any*** value in the list.

The same thing, spelled out with OR:

```
SELECT Name FROM Person WHERE PersonID = 3 OR PersonID = 5;
```

Both give the exact same result — IN is just easier to read and write as the list grows.

Result:

Name
James Wilson
Robert Taylor

View Tables ›

IN With a Nested Query

The list of IDs doesn't have to be typed by hand — a subquery can compute it

```
SELECT Name FROM Person WHERE gender = 'Male'
AND PersonID IN (SELECT PersonID FROM Employed WHERE Occupation = 'Engineer');
```

Step 1 — the subquery finds every Engineer's PersonID: → `(3, 4, 5)`

Substituting the result back into the original query:

```
SELECT Name FROM Person WHERE gender = 'Male' AND PersonID IN (3, 4, 5);
```

Step 2 — check every Male person against that list:

PersonID	Name	In Engineer list?
1	~~Michael Chen~~	—
2	~~David Martinez~~	—
3	**James Wilson**	✓
5	**Robert Taylor**	✓

Note: The nested SELECT must return exactly one column.

NOT IN Solves It Correctly

"What men are not engineers?" — the subquery accounts for every occupation row

```
SELECT Name FROM Person WHERE gender = 'Male'
AND PersonID NOT IN (SELECT PersonID FROM Employed WHERE Occupation = 'Engineer');
```

Step 1 — same subquery as before, the Engineer PersonIDs: → `(3, 4, 5)`

Step 2 — check every Male person against that list, with NOT IN:

PersonID	Name	NOT IN Engineer list?
1	**Michael Chen**	✓
2	**David Martinez**	✓
3	~~James Wilson~~	✗ engineer
5	~~Robert Taylor~~	✗ engineer

Introducing EXISTS

Check whether a matching row exists, instead of comparing to a list

Find the names of all male engineers:

```
SELECT Name FROM Person WHERE gender = 'Male'
AND EXISTS (SELECT * FROM Employed
    WHERE Employed.PersonID = Person.PersonID
    AND Employed.Occupation = 'Engineer');
```

EXISTS returns TRUE if the subquery finds at least one matching row.

The **highlighted line** ties the subquery back to the outer row —evaluated fresh for every candidate row in Person

Result:

James Wilson
Robert Taylor

How EXISTS Works

For every candidate row, the subquery runs fresh and asks: does anything match?

Case: a matching row exists

PersonID 3
James Wilson

```
EXISTS (SELECT * FROM Employed
WHERE PersonID = 3 AND Occupation = 'Engineer')
```

✓ found (3, Engineer, $98,000)
EXISTS = TRUE → row is kept

Case: no matching row

PersonID 1
Michael Chen

```
EXISTS (SELECT * FROM Employed
WHERE PersonID = 1 AND Occupation = 'Engineer')
```

— no matching row found
EXISTS = FALSE → row is excluded

The loop: SQL repeats this check for every row Person offers — Michael Chen, David Martinez, James Wilson, Robert Taylor — keeping only the rows where the subquery finds a match.

The EXISTS Loop, Row 1 of 4

PersonID 1 — Michael Chen: does Employed contain a matching Engineer row?

```
EXISTS (SELECT * FROM Employed WHERE
  Employed.PersonID = Person.PersonID AND Occupation = 'Engineer');
```

Male Person Records

PersonID	Name
1	**Michael Chen**
2	David Martinez
3	James Wilson
5	Robert Taylor

Employed — PersonID = 1

PersonID	Occupation	Salary
1	Nurse	$71,000

No Engineer row found for PersonID 1 → EXISTS = FALSE → row is excluded

The EXISTS Loop, Row 2 of 4

PersonID 2 — David Martinez: does Employed contain a matching Engineer row?

```
EXISTS (SELECT * FROM Employed WHERE
  Employed.PersonID = Person.PersonID AND Occupation = 'Engineer');
```

Male Person Records

PersonID	Name
1	Michael Chen
2	**David Martinez**
3	James Wilson
5	Robert Taylor

Employed — PersonID = 2

PersonID	Occupation	Salary
2	Doctor	$145,000

No Engineer row found for PersonID 2 → EXISTS = FALSE → row is excluded

The EXISTS Loop, Row 3 of 4

PersonID 3 — James Wilson: does Employed contain a matching Engineer row?

```
EXISTS (SELECT * FROM Employed WHERE
  Employed.PersonID = Person.PersonID AND Occupation = 'Engineer');
```

Male Person Records

PersonID	Name
1	Michael Chen
2	David Martinez
3	**James Wilson**
5	Robert Taylor

Employed — PersonID = 3

PersonID	Occupation	Salary
3	Nurse	$69,000
3	**Engineer** ✓	$98,000

Engineer row found for PersonID 3 ($98,000) → EXISTS = TRUE → row is kept

The EXISTS Loop, Row 4 of 4

PersonID 5 — Robert Taylor: does Employed contain a matching Engineer row?

```
EXISTS (SELECT * FROM Employed WHERE
  Employed.PersonID = Person.PersonID AND Occupation = 'Engineer');
```

Male Person Records

PersonID	Name
1	Michael Chen
2	David Martinez
3	James Wilson
5	**Robert Taylor**

Employed — PersonID = 5

PersonID	Occupation	Salary
5	**Engineer** ✓	$101,000

Engineer row found for PersonID 5 ($101,000) → EXISTS = TRUE → row is kept

EXISTS Without Correlation? A Trap

Drop the join condition, and EXISTS stops asking about any particular person

```
SELECT Name FROM Person WHERE gender = 'Male'
AND EXISTS (SELECT * FROM Employed WHERE Occupation = 'Engineer');
                          ✗  no condition connecting Employed to Person.PersonID
```

EXISTS is TRUE for ***every*** Person row — not just the engineers.

Result — every Male row comes back, whether they're an engineer or not:

Michael Chen	David Martinez	James Wilson	Robert Taylor
✗ not an engineer!	✗ not an engineer!	is an engineer	is an engineer

NOT EXISTS

"What men are not engineers?" — flip EXISTS to NOT EXISTS

```
SELECT Name FROM Person WHERE gender = 'Male'
AND NOT EXISTS (SELECT * FROM Employed
    WHERE Employed.PersonID = Person.PersonID
    AND Employed.Occupation = 'Engineer');
```

Checking every Male person, correlated row by row:

PersonID	Name	NOT EXISTS engineer row?
1	**Michael Chen**	✓
2	**David Martinez**	✓
3	~~James Wilson~~	✗ exists
5	~~Robert Taylor~~	✗ exists

Result:

Michael Chen

David Martinez

Exactly matches the NOT IN result from before.

IN/NOT IN and EXISTS/NOT EXISTS are two different mechanisms for the same relational idea — both correctly account for ***every*** occupation row a person has, which is exactly what tripped up the original <> attempt.

Three Ways to Answer: What Men Are Not Engineers?

EXCEPT, NOT IN, and NOT EXISTS all reach the same answer

1. EXCEPT

```
SELECT PersonID
FROM Person
WHERE gender = 'Male'
EXCEPT
SELECT PersonID
FROM Employed
WHERE Occupation= 'Engineer'
```

2. NOT IN

```
SELECT PersonID
FROM Person
WHERE gender = 'Male'
AND PersonID NOT IN
  (SELECT PersonID
   FROM Employed
   WHERE Occupation = 'Engineer');
```

3. NOT EXISTS

```
SELECT PersonID
FROM Person p
WHERE gender = 'Male'
AND NOT EXISTS
  (SELECT * FROM Employed e
   WHERE e.PersonID = p.PersonID AND
     e.Occupation = 'Engineer');
```

A Difference Problem with Two Columns

"List the PersonID and Name of men who are not engineers"

1. EXCEPT

```
SELECT PersonID, Name
FROM Person
WHERE gender = 'Male'
EXCEPT
SELECT PersonID, Name FROM Person
WHERE PersonID IN
   (SELECT PersonID FROM Employed
    WHERE Occupation = 'Engineer');
```

2. NOT IN

```
SELECT PersonID, Name
FROM Person
WHERE gender = 'Male'
AND PersonID NOT IN
  (SELECT PersonID FROM Employed
   WHERE Occupation = 'Engineer');
```

3. NOT EXISTS

```
SELECT PersonID, Name
FROM Person p
WHERE gender = 'Male'
AND NOT EXISTS
  (SELECT * FROM Employed e
   WHERE e.PersonID = p.PersonID AND
   e.Occupation = 'Engineer');
```

Notice: EXCEPT needs its second SELECT to match column-for-column (PersonID, Name) with the first SELECT

Where Do the Conditions Go?

In A − B, solved with NOT IN or NOT EXISTS — 2-column layout

Conditions that define set A *(who's even a candidate)* belong in the **outer** query. Conditions that define set B *(what disqualifies someone)* belong in the **inner** subquery.

Set A condition (outer query): gender = 'Male'

Set B condition (inner query): Occupation = 'Engineer'

Using NOT IN:

```
SELECT Name FROM Person
WHERE gender = 'Male'
AND PersonID NOT IN
  (SELECT PersonID FROM Employed
   WHERE Occupation = 'Engineer');
```

Using NOT EXISTS:

```
SELECT Name FROM Person
WHERE gender = 'Male'
AND NOT EXISTS
  (SELECT * FROM Employed
   WHERE Employed.PersonID = Person.PersonID
   AND Occupation = 'Engineer');
```

Same split every time: outer query = “who's in the running” (A), inner subquery = “what rules them out” (B). The join line (PersonID = PersonID) is neither — it's just the bridge.