SQL:
INTERSECT
vs.
INNER JOIN

Consider these two tables of employees

```sql
SELECT *
FROM #Employees1;
```

	first_name	last_name
1	Guy	Gilbert
2	Kevin	Brown
3	Roberto	Tamburello
4	Rob	Walters
5	Rob	Walters
6	Thierry	D'Hers
7	David	Bradley
8	David	Bradley
9	JoLynn	*NULL*
10	Ruth	Ellerbrock

```sql
SELECT *
FROM #Employees2;
```

	forename	surname
1	Thierry	D'Hers
2	David	Bradley
3	David	Bradley
4	JoLynn	*NULL*
5	Ruth	Ellerbrock
6	Gail	Erickson
7	Barry	Johnson
8	Jossef	Goldberg
9	Terri	Duffy
10	Sidney	Higa

We want to find which employees are in both tables

```sql
SELECT *
FROM #Employees1;
```

	first_name	last_name
1	Guy	Gilbert
2	Kevin	Brown
3	Roberto	Tamburello
4	Rob	Walters
5	Rob	Walters
6	Thierry	D'Hers
7	David	Bradley
8	David	Bradley
9	JoLynn	*NULL*
10	Ruth	Ellerbrock

```sql
SELECT *
FROM #Employees2;
```

	forename	surname
1	Thierry	D'Hers
2	David	Bradley
3	David	Bradley
4	JoLynn	*NULL*
5	Ruth	Ellerbrock
6	Gail	Erickson
7	Barry	Johnson
8	Jossef	Goldberg
9	Terri	Duffy
10	Sidney	Higa

INTERSECT finds the distinct intersection of two sets

We use the INTERSECT operator between two queries. The queries are evaluated and the result-sets are compared

```sql
SELECT first_name, last_name
FROM #Employees1
INTERSECT
SELECT forename, surname
FROM #Employees2;
```

Column order and column count are important, but the column names don't need to be the same, they must only return *compatible data types*

The column names from the query left of (above) the INTERSECT operator are used in the result set

	first_name	last_name
1	David	Bradley
2	JoLynn	*NULL*
3	Ruth	Ellerbrock
4	Thierry	D'Hers

Duplicates are not returned

NULL comparisons are respected

INNER JOIN will also return rows present in both sets

We can return as many columns as we need, not just those being compared

```
SELECT *
FROM #Employees1 e1
    INNER JOIN #Employees2 e2
        ON  e1.first_name = e2.forename
        AND e1.last_name = e2.surname;
```

Column name differences are accounted for in the JOIN predicates

If we SELECT *, columns from both sides of the join are returned

NULL comparisons are not respected – they are not returned

	first_name	last_name	forename	surname
1	Thierry	D'Hers	Thierry	D'Hers
2	David	Bradley	David	Bradley
3	David	Bradley	David	Bradley
4	David	Bradley	David	Bradley
5	David	Bradley	David	Bradley
6	Ruth	Ellerbrock	Ruth	Ellerbrock

Each match is returned – so in case of duplicates, each row in one table is duplicated by the number of matches in the other

We can rectify the duplicates with explicit changes to the SELECT clause

```sql
SELECT DISTINCT e1.first_name, e1.last_name
FROM #Employees1 e1
    INNER JOIN #Employees2 e2
        ON  e1.first_name = e2.forename
        AND e1.last_name = e2.surname;
```

List columns explicitly and add the DISTINCT keyword

	first_name	last_name
1	David	Bradley
2	Ruth	Ellerbrock
3	Thierry	D'Hers

NULL comparisons are still not respected – they are not returned

To ensure NULL values are returned, we can modify the JOIN predicate

```sql
SELECT DISTINCT e1.first_name, e1.last_name
FROM #Employees1 e1
    INNER JOIN #Employees2 e2
        ON  e1.first_name = e2.forename
        AND COALESCE(e1.last_name,'') = COALESCE(e2.surname,'');
```

Use COALESCE to evaluate to the first non-null parameter.

In this case, if the column is NULL, use an empty string instead.

	first_name	last_name
1	David	Bradley
2	JoLynn	*NULL*
3	Ruth	Ellerbrock
4	Thierry	D'Hers

Because an empty string is a value (whereas NULL is not), the comparison is valid in a join predicate, so the row with the NULL last_name is returned

INTERSECT vs INNER JOIN

	INTERSECT	INNER JOIN
Returns rows which are in both tables	✓	✓
Respects NULL values by default	✓	✗
Column order is flexible	✗	✓
Column count is flexible	✗	✓
Removes duplicates by default	✓	✗