JOINS

Keys

Primary Keys:

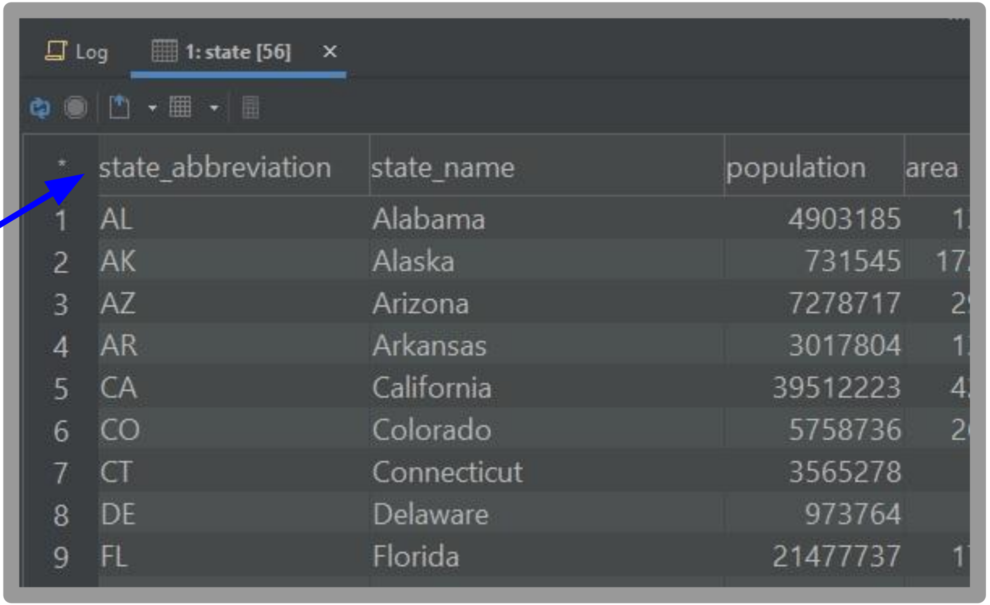
- Uniquely identify records in a table.
- Leveraged to allow us to define relationships between tables.

Keys

Natural Primary Keys:

Use a piece of table data that is unique for each record.

state_abbreviation can be used as a natural primary key.



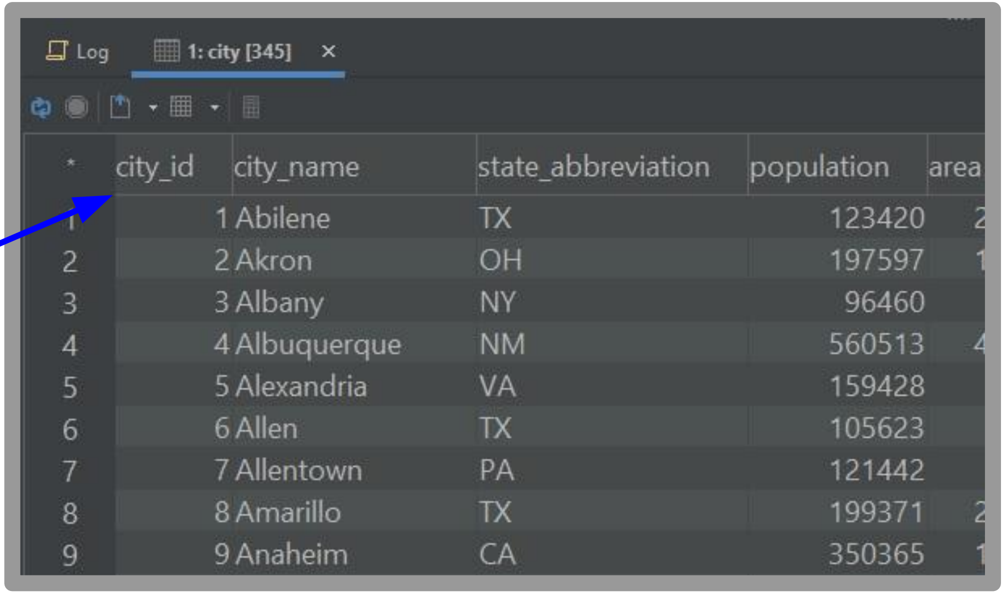
*	state_abbreviation	state_name	population	area
1	AL	Alabama	4903185	1
2	AK	Alaska	731545	17
3	AZ	Arizona	7278717	2
4	AR	Arkansas	3017804	1
5	CA	California	39512223	4
6	CO	Colorado	5758736	2
7	CT	Connecticut	3565278	
8	DE	Delaware	973764	
9	FL	Florida	21477737	1

Keys

Surrogate Primary Keys:

Use a generated unique identifier when the data does not contain a natural one.

The number used in the `city_id` field is auto-generated and is used as a key since the data does not have a good natural key,

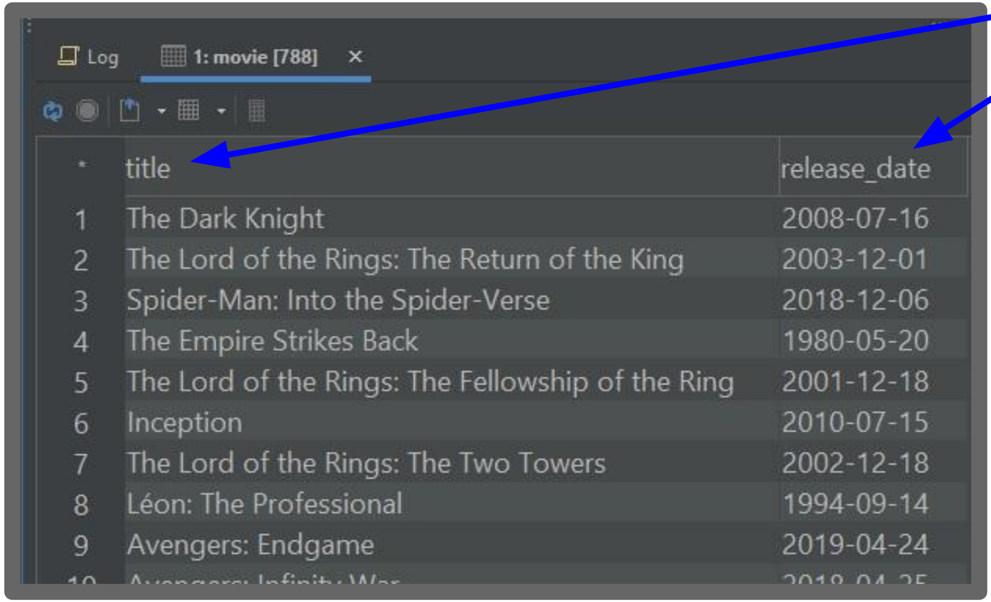


*	city_id	city_name	state_abbreviation	population	area
1	1	Abilene	TX	123420	2
2	2	Akron	OH	197597	1
3	3	Albany	NY	96460	
4	4	Albuquerque	NM	560513	4
5	5	Alexandria	VA	159428	
6	6	Allen	TX	105623	
7	7	Allentown	PA	121442	
8	8	Amarillo	TX	199371	2
9	9	Anaheim	CA	350365	1

Keys

Composite Primary Keys:

A primary key made up of multiple fields.



	title	release_date
1	The Dark Knight	2008-07-16
2	The Lord of the Rings: The Return of the King	2003-12-01
3	Spider-Man: Into the Spider-Verse	2018-12-06
4	The Empire Strikes Back	1980-05-20
5	The Lord of the Rings: The Fellowship of the Ring	2001-12-18
6	Inception	2010-07-15
7	The Lord of the Rings: The Two Towers	2002-12-18
8	Léon: The Professional	1994-09-14
9	Avengers: Endgame	2019-04-24
10	Avengers: Infinity War	2018-04-25

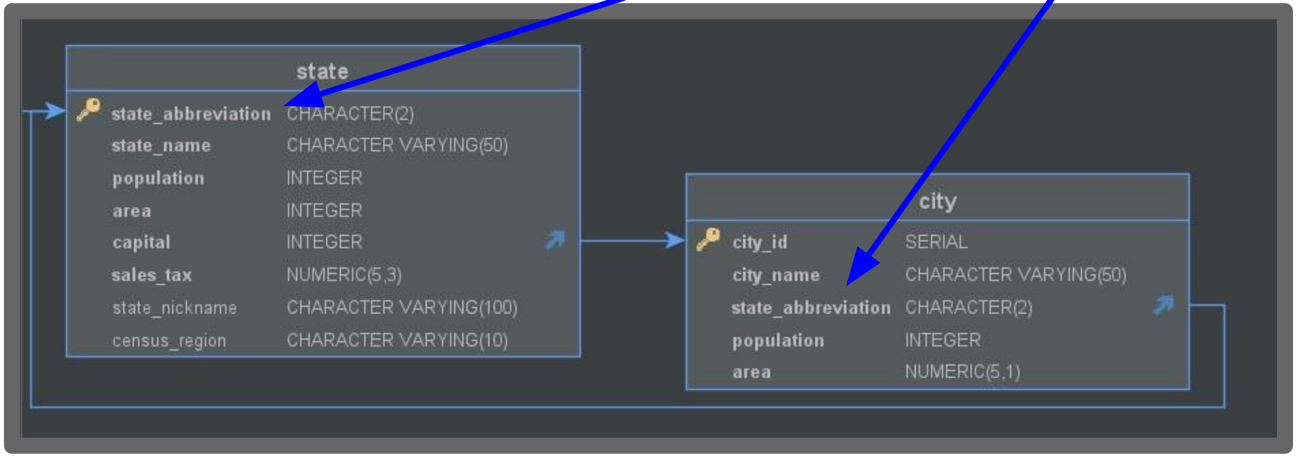
`title` + `release_date` would be unique and can be used as a key.

Keys

Foreign Key:

A field that references a primary key in another table.

(Allows enforcement of data integrity)



state

🔑 state_abbreviation	CHARACTER(2)
state_name	CHARACTER VARYING(50)
population	INTEGER
area	INTEGER
capital	INTEGER
sales_tax	NUMERIC(5,3)
state_nickname	CHARACTER VARYING(100)
census_region	CHARACTER VARYING(10)

city

🔑 city_id	SERIAL
city_name	CHARACTER VARYING(50)
state_abbreviation	CHARACTER(2)
population	INTEGER
area	NUMERIC(5,1)

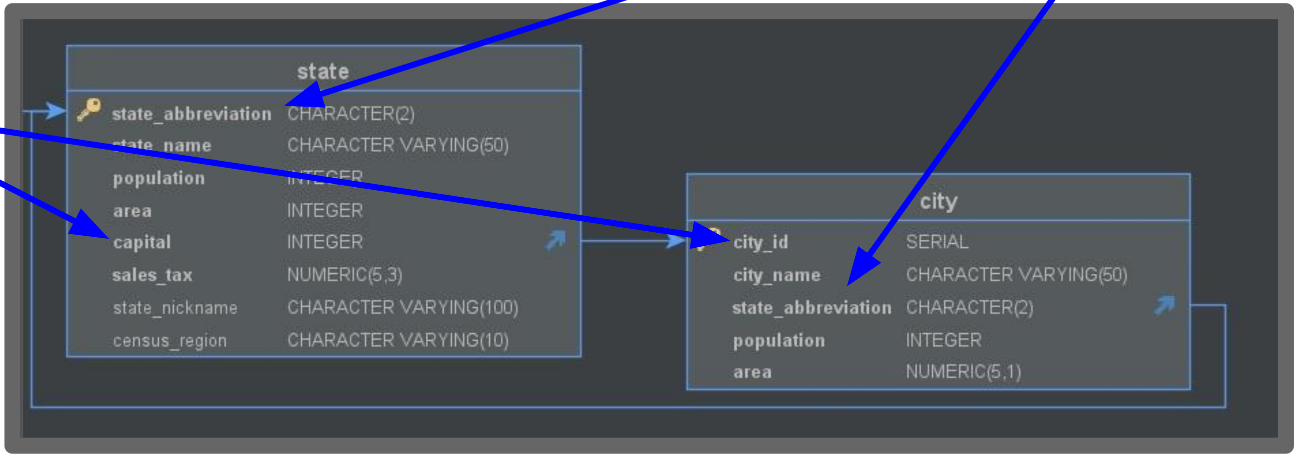
The **`state_abbreviation`** field in the **`city`** table references the **`state_abbreviation`** field in the **`state`** table.

Keys

Foreign Key:

A field that references a primary key in another table.

(Allows enforcement of data integrity)



state

state_abbreviation	CHARACTER(2)
state_name	CHARACTER VARYING(50)
population	INTEGER
area	INTEGER
capital	INTEGER
sales_tax	NUMERIC(5,3)
state_nickname	CHARACTER VARYING(100)
census_region	CHARACTER VARYING(10)

city

city_id	SERIAL
city_name	CHARACTER VARYING(50)
state_abbreviation	CHARACTER(2)
population	INTEGER
area	NUMERIC(5,1)

The **`capital`** field in the **`state`** table references the **`city_id`** field of the **`city`** table.

The **`state_abbreviation`** field in the **`city`** table references the **`state_abbreviation`** field in the **`state`** table.

Cardinality

One-To-One (1:1)

One row in table A relates to one row in table B.

Example:

Each record in Person table has one corresponding record in SSN (Social Security Number) table.

Cardinality

One-To-Many (1:N OR 1:M)

One row in table A may relate to multiple rows in table B.

Example:

Each record in Address table may related to multiple records in Person table.

Cardinality

Many-To-Many (M:N or N:M)

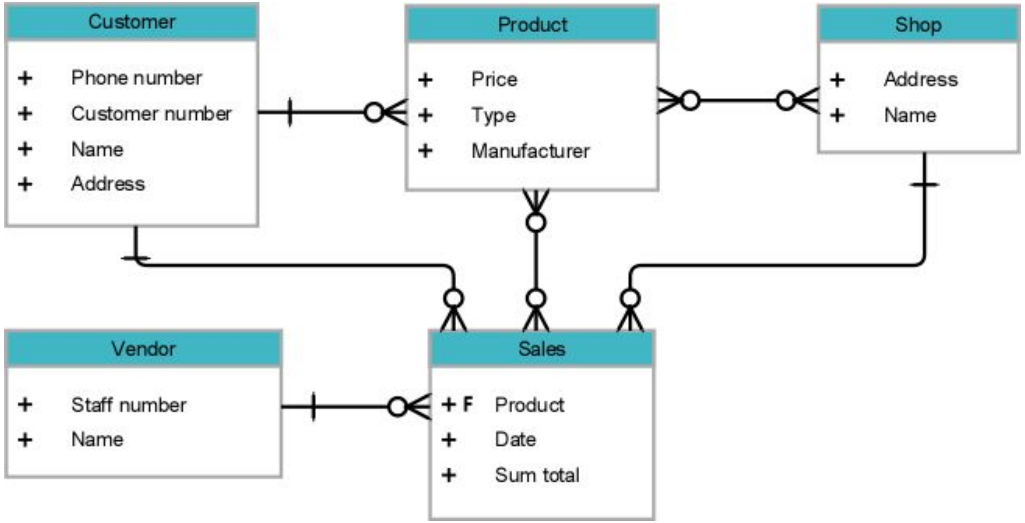
Many rows in table A may relate to many rows in table B.

Example:

Each record in Film table may relate to multiple records in Actor table and each record in Actor table may relate to multiple records in Film table.

Implemented via join tables (stay tuned...)

Entity Relationship Diagram (ERD)



Customer

- + Phone number
- + Customer number
- + Name
- + Address

Product

- + Price
- + Type
- + Manufacturer

Shop

- + Address
- + Name

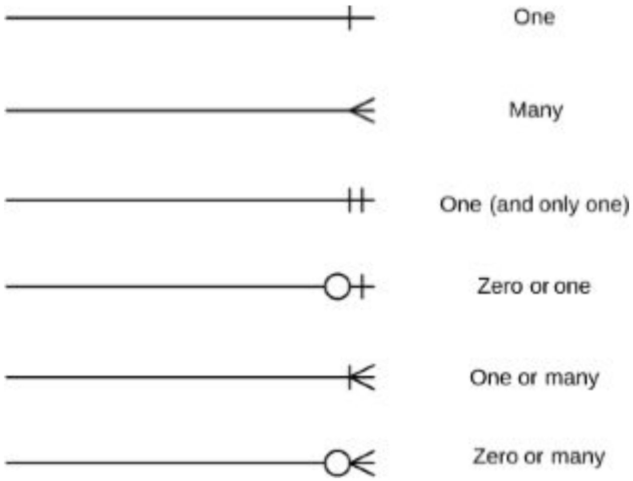
Vendor

- + Staff number
- + Name

Sales

- + F Product
- + Date
- + Sum total

ERD Cardinality



One

Many

One (and only one)

Zero or one

One or many

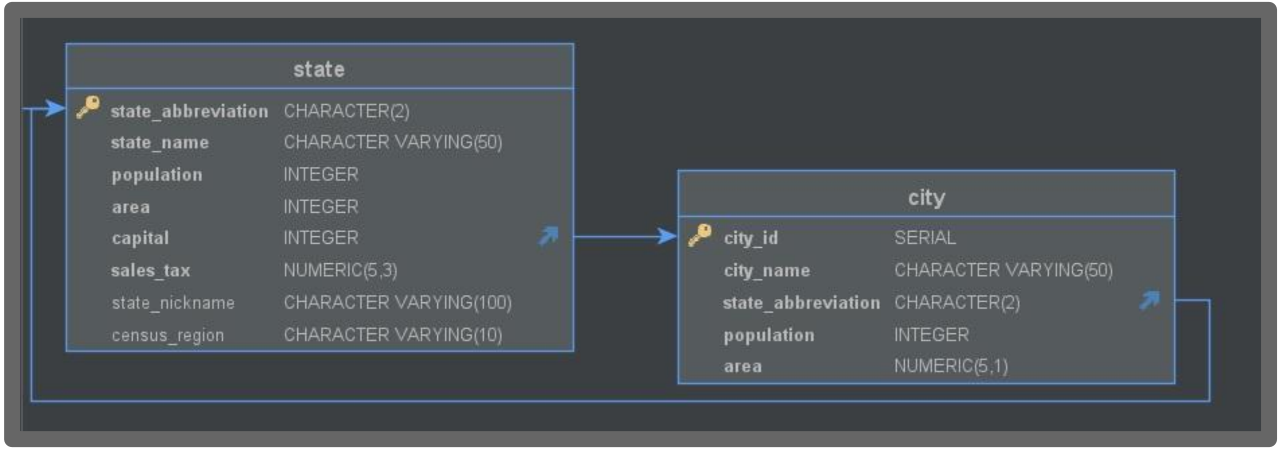
Zero or many

JOINS

Joins allow us to relate data between tables to query data in whatever ways makes sense.

We could relate data from the country and city tables to provide one set of data.

To get a city's country we would join the **`city`** table's **`countrycode`** field to the **`country`** tables's **`code`** field.



state	
state_abbreviation	CHARACTER(2)
state_name	CHARACTER VARYING(50)
population	INTEGER
area	INTEGER
capital	INTEGER
sales_tax	NUMERIC(5,3)
state_nickname	CHARACTER VARYING(100)
census_region	CHARACTER VARYING(10)

city	
city_id	SERIAL
city_name	CHARACTER VARYING(50)
state_abbreviation	CHARACTER(2)
population	INTEGER
area	NUMERIC(5,1)

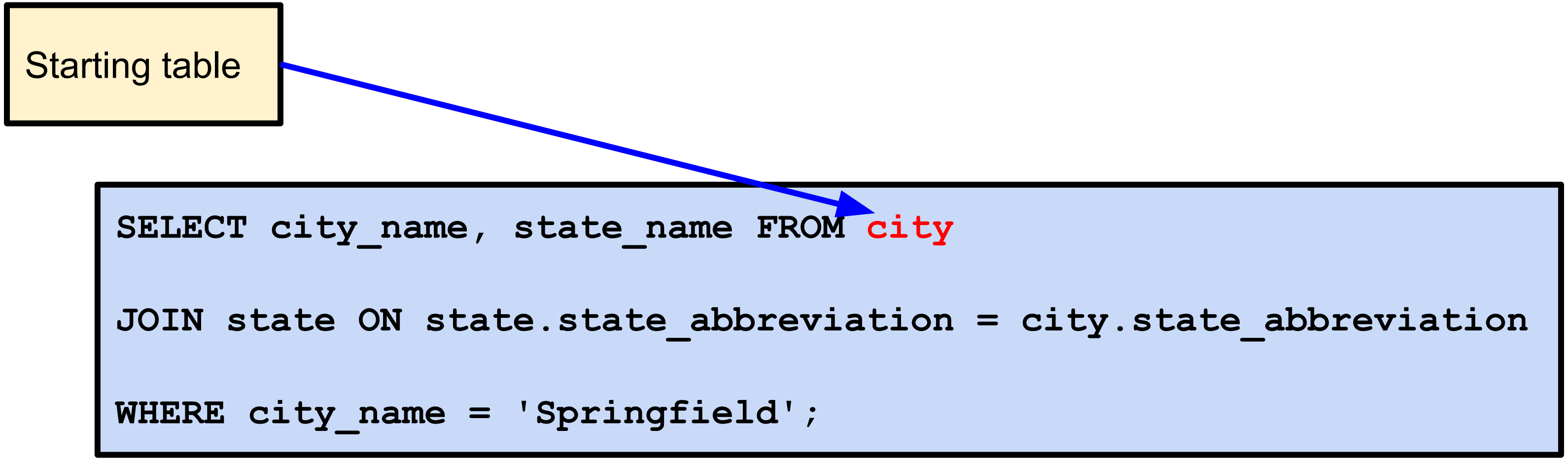
JOINS

ANATOMY OF A JOIN STATEMENT

```
SELECT city_name, state_name FROM city

JOIN state ON state.state_abbreviation = city.state_abbreviation

WHERE city_name = 'Springfield';
```

JOINS

ANATOMY OF A JOIN STATEMENT

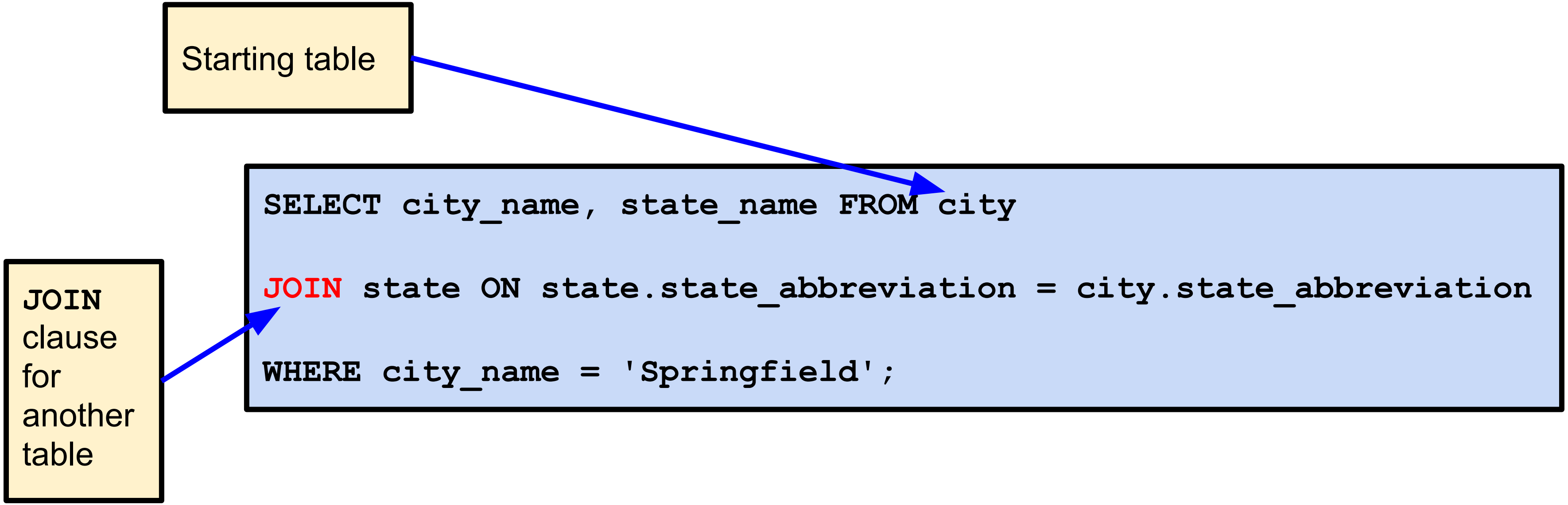


Starting table

```
SELECT city_name, state_name FROM city

JOIN state ON state.state_abbreviation = city.state_abbreviation

WHERE city_name = 'Springfield';
```

JOINS

ANATOMY OF A JOIN STATEMENT



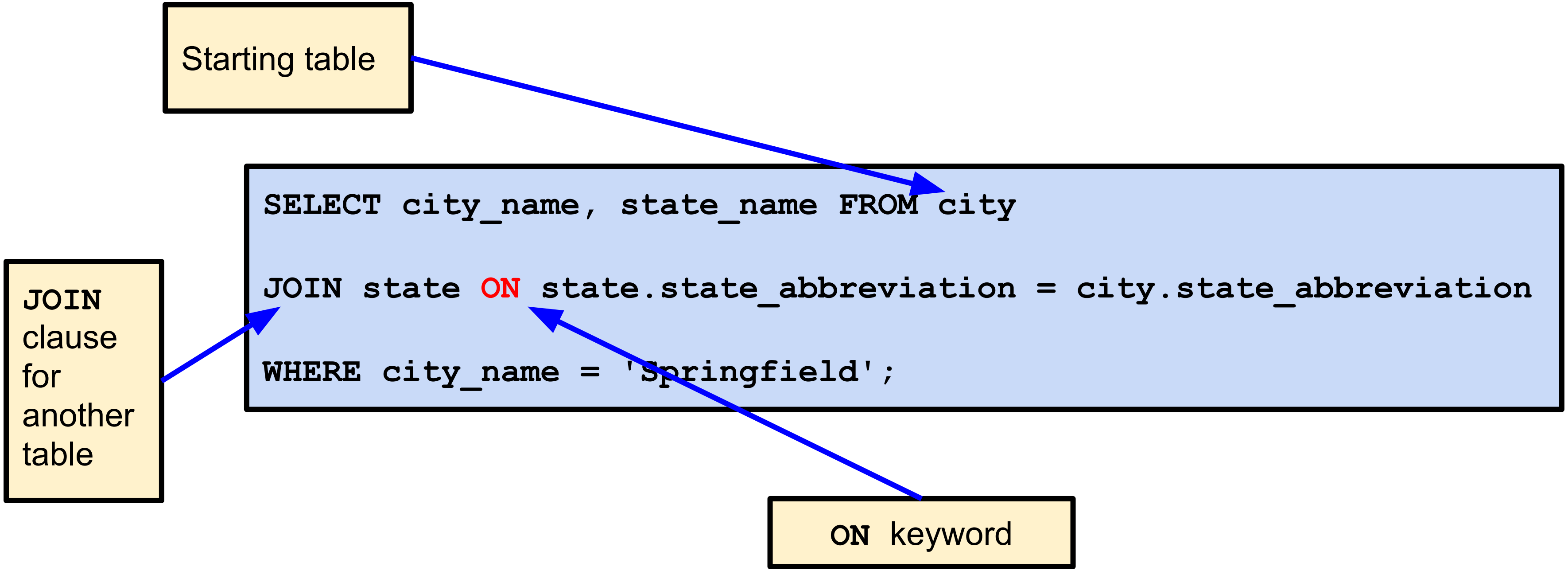
Starting table

```
SELECT city_name, state_name FROM city

JOIN state ON state.state_abbreviation = city.state_abbreviation

WHERE city_name = 'Springfield';
```

JOIN clause for another table

JOINS

ANATOMY OF A JOIN STATEMENT



```
SELECT city_name, state_name FROM city

JOIN state ON state.state_abbreviation = city.state_abbreviation

WHERE city_name = 'Springfield';
```

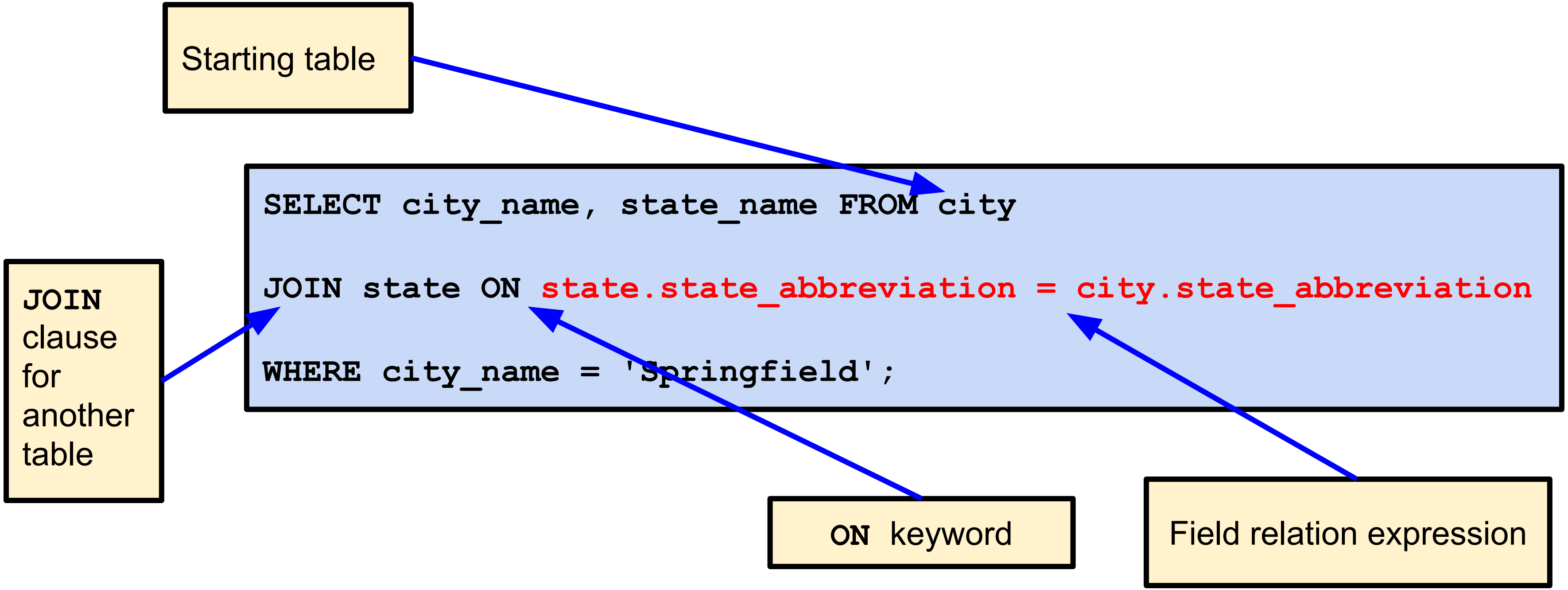
Starting table

JOIN clause for another table

ON keyword

JOINS

ANATOMY OF A JOIN STATEMENT



Starting table

```
SELECT city_name, state_name FROM city

JOIN state ON state.state_abbreviation = city.state_abbreviation

WHERE city_name = 'Springfield';
```

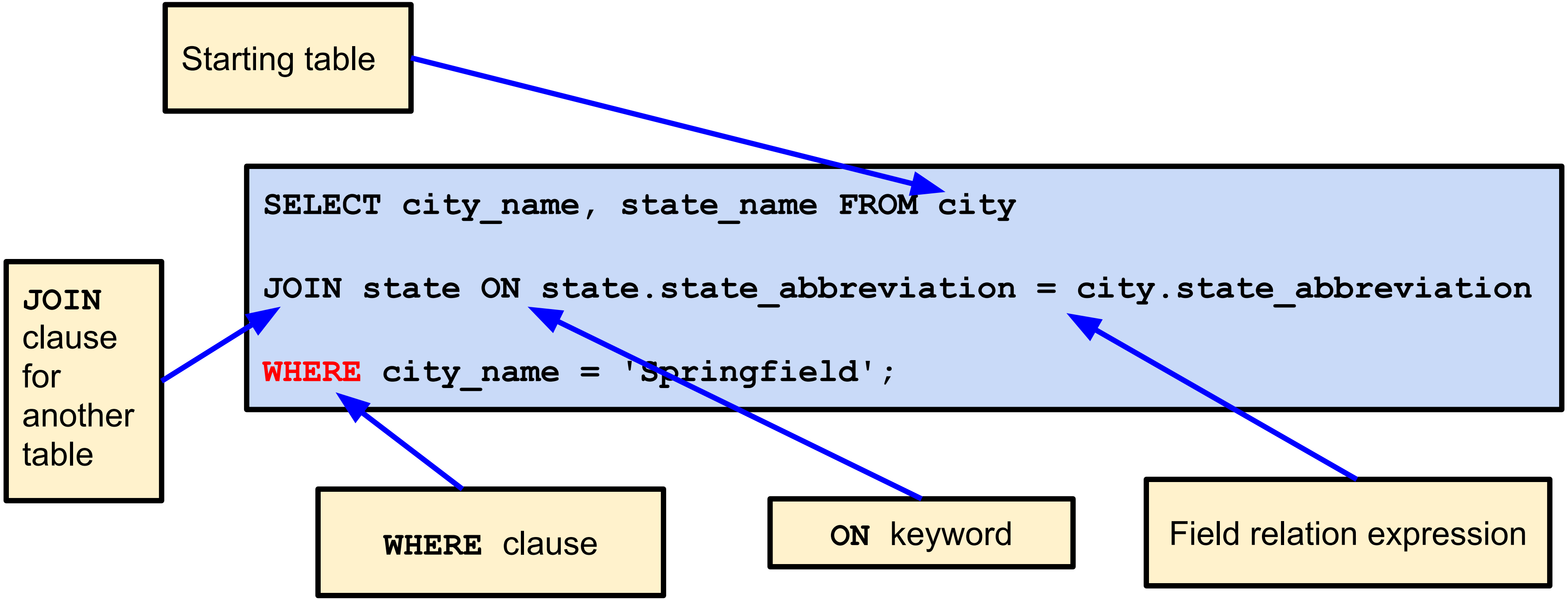
JOIN clause for another table

ON keyword

Field relation expression

JOINS

ANATOMY OF A JOIN STATEMENT



```
SELECT city_name, state_name FROM city

JOIN state ON state.state_abbreviation = city.state_abbreviation

WHERE city_name = 'Springfield';
```

Starting table

JOIN clause for another table

WHERE clause

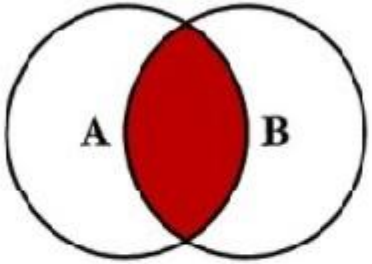
ON keyword

Field relation expression

JOINS

INNER JOINS

Inner joins allow us to query data that is the intersection of two tables.



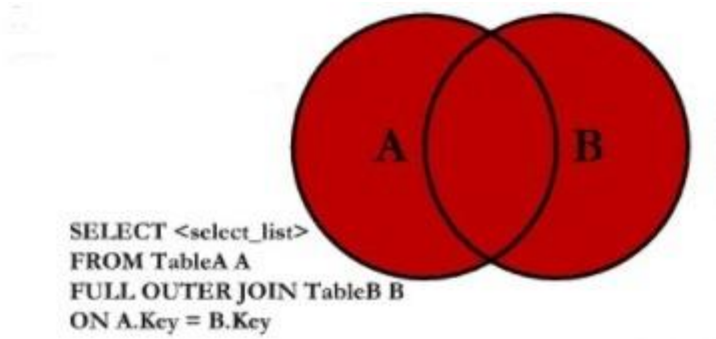
```
SELECT <select_list>
FROM TableA A
INNER JOIN TableB B
ON A.Key = B.Key
```

JOINS

OUTER JOINS

When performing an Inner Join, rows from either table that are unmatched in the other table are not returned. In an outer join, unmatched rows in one or both tables can be returned. There are a few types of outer joins.

A Full Outer Join returns the data from both tables, including unmatched data.

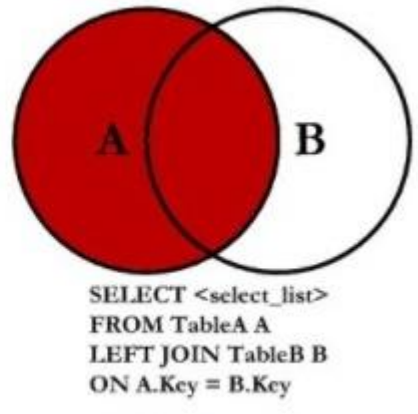


```
SELECT <select_list>
FROM TableA A
FULL OUTER JOIN TableB B
ON A.Key = B.Key
```

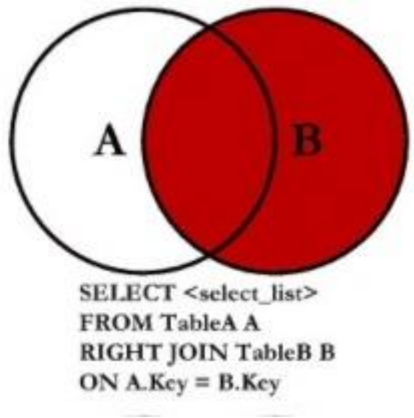
JOINS

LEFT AND RIGHT JOINS

Left and Right Outer Joins allow us to include unmatched data from either the “Left” or “Right” table data. Left and Right refer to the table’s position in the from/join statement. Left and Right Outer Joins are usually referred to as Left and Right Joins.

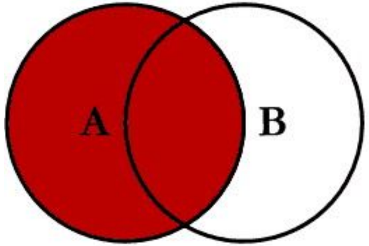


```
SELECT <select_list>
FROM TableA A
LEFT JOIN TableB B
ON A.Key = B.Key
```

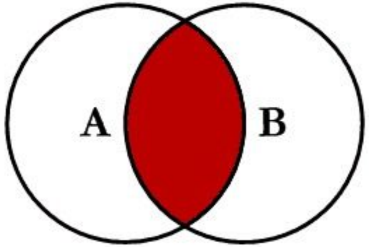


```
SELECT <select_list>
FROM TableA A
RIGHT JOIN TableB B
ON A.Key = B.Key
```

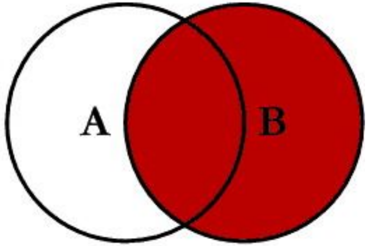
SQL JOINS



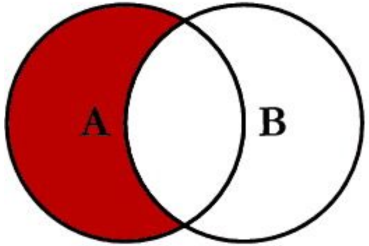
```
SELECT <select_list>
FROM TableA A
LEFT JOIN TableB B
ON A.Key = B.Key
```



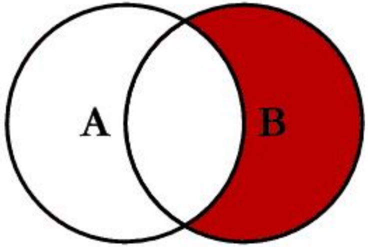
```
SELECT <select_list>
FROM TableA A
INNER JOIN TableB B
ON A.Key = B.Key
```



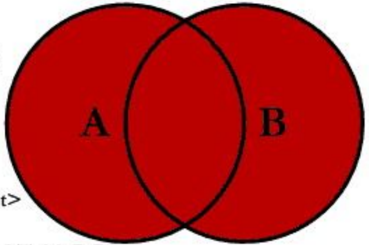
```
SELECT <select_list>
FROM TableA A
RIGHT JOIN TableB B
ON A.Key = B.Key
```



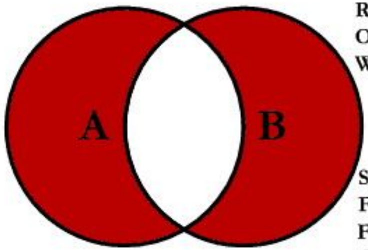
```
SELECT <select_list>
FROM TableA A
LEFT JOIN TableB B
ON A.Key = B.Key
WHERE B.Key IS NULL
```



```
SELECT <select_list>
FROM TableA A
RIGHT JOIN TableB B
ON A.Key = B.Key
WHERE A.Key IS NULL
```



```
SELECT <select_list>
FROM TableA A
FULL OUTER JOIN TableB B
ON A.Key = B.Key
```



```
SELECT <select_list>
FROM TableA A
FULL OUTER JOIN TableB B
ON A.Key = B.Key
WHERE A.Key IS NULL
OR B.Key IS NULL
```

Setting Up the joinsdb Database (Optional)

Set Up Database JoinsDB in PgAdmin

In PgAdmin:

- **Create a new database called `joinsdb`**
- **Use the `JoinsLesson-JoinsDB.sql` script in the `JoinsDB` folder of the `lecture` folder to set up the schema and data**
- **The `JoinsLesson-JoinsExamples.sql` script in the `JoinsDB` folder of the `lecture` folder contains the examples we will be walking through**

JOINS

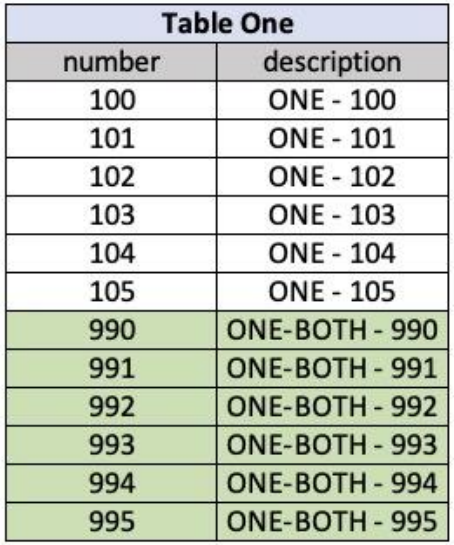


Table One	
number	description
100	ONE - 100
101	ONE - 101
102	ONE - 102
103	ONE - 103
104	ONE - 104
105	ONE - 105
990	ONE-BOTH - 990
991	ONE-BOTH - 991
992	ONE-BOTH - 992
993	ONE-BOTH - 993
994	ONE-BOTH - 994
995	ONE-BOTH - 995

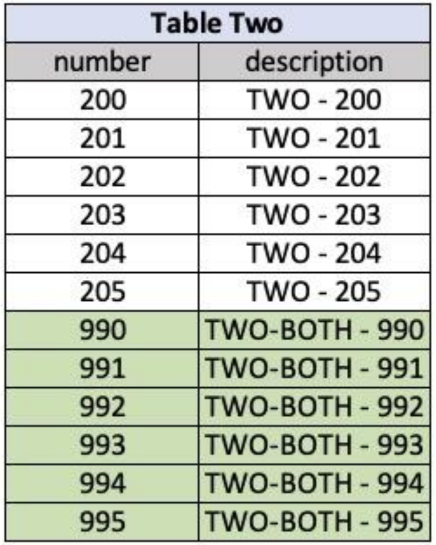
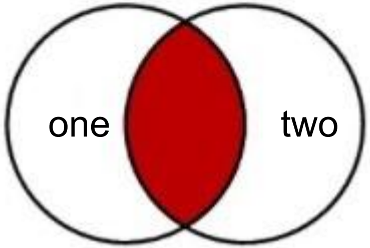


Table Two	
number	description
200	TWO - 200
201	TWO - 201
202	TWO - 202
203	TWO - 203
204	TWO - 204
205	TWO - 205
990	TWO-BOTH - 990
991	TWO-BOTH - 991
992	TWO-BOTH - 992
993	TWO-BOTH - 993
994	TWO-BOTH - 994
995	TWO-BOTH - 995

JOINS

Inner Join (Default)

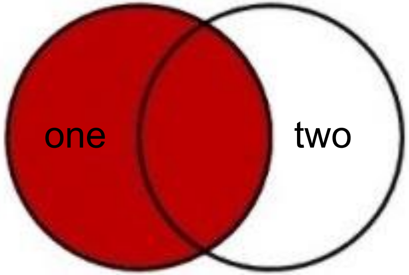
```
SELECT one.number AS one_number, one.description AS one_description,
two.number AS two_number, two.description AS two_description

FROM one

JOIN two ON one.number = two.number;
```



JOINS

Left Join

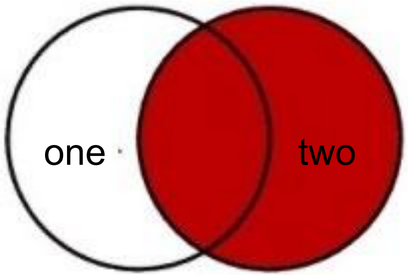
```
SELECT one.number AS one_number, one.description AS one_description,
two.number AS two_number, two.description AS two_description

FROM one

LEFT JOIN two ON one.number = two.number;
```



JOINS

Right Join

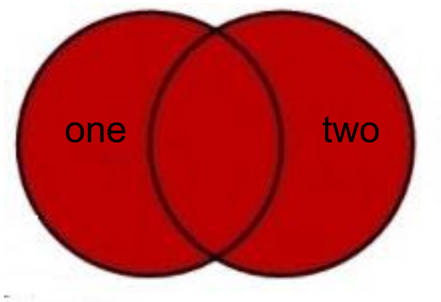
```
SELECT one.number AS one_number, one.description AS one_description,
two.number AS two_number, two.description AS two_description

FROM one

RIGHT JOIN two ON one.number = two.number;
```



JOINS

FULL OUTER JOIN

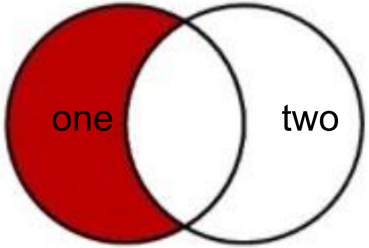
```
SELECT one.number AS one_number, one.description AS one_description,
two.number AS two_number, two.description AS two_description

FROM one

FULL OUTER JOIN two ON one.number = two.number;
```



JOINS

Useful Variation: Left Table Values Only

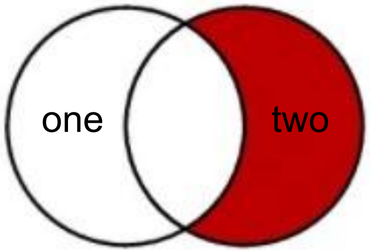
```
SELECT one.number AS one_number, one.description AS one_description,
two.number AS two_number, two.description AS two_description

FROM one

LEFT JOIN two ON one.number = two.number

WHERE two.number IS NULL;
```



JOINS

Useful Variation: Right Table Values Only

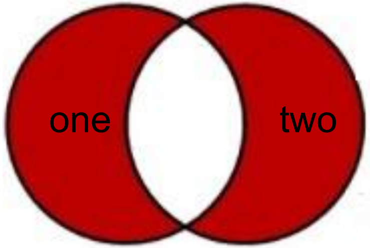
```
SELECT one.number AS one_number, one.description AS one_description,
two.number AS two_number, two.description AS two_description

FROM one

RIGHT JOIN two ON one.number = two.number

WHERE one.number IS NULL
```



JOINS

Useful Variation: Left or Right Table Values But Not Both

```
SELECT one.number AS one_number, one.description AS one_description,
two.number AS two_number, two.description AS two_description

FROM one

FULL OUTER JOIN two ON one.number = two.number

WHERE one.number IS NULL OR two.number IS NULL
```



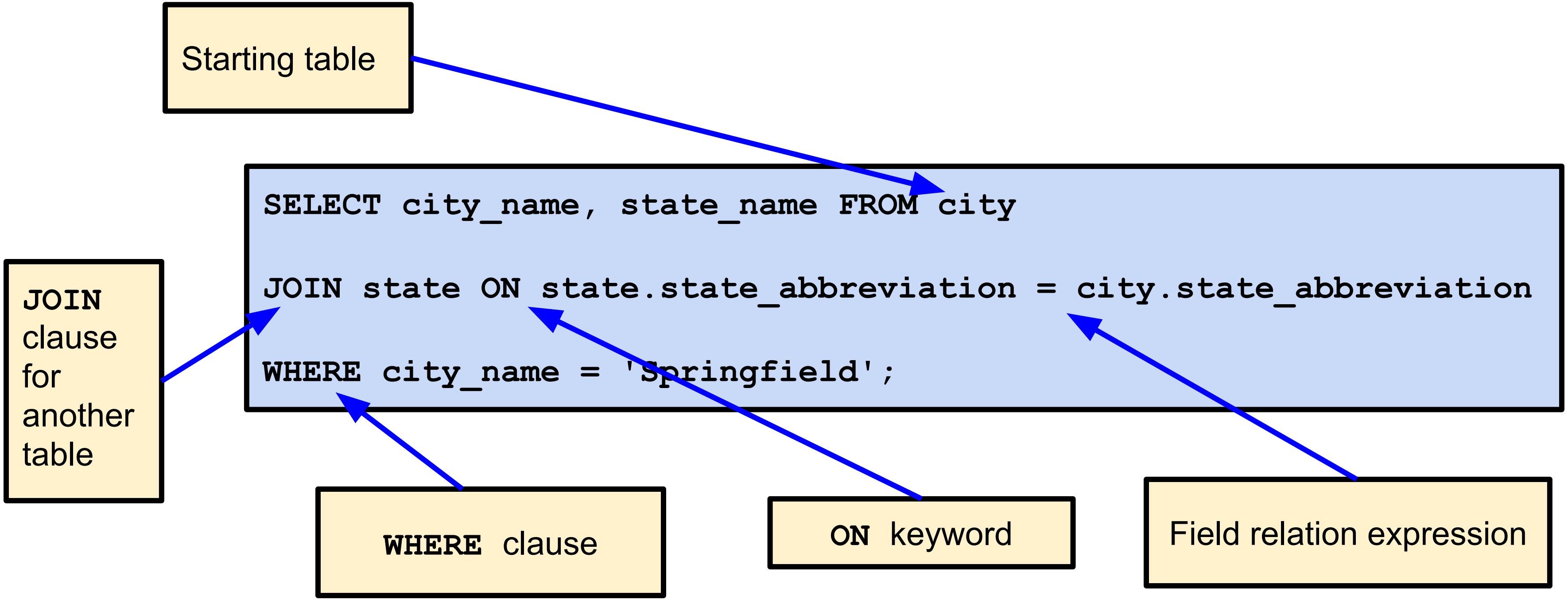
Let's Join Some Tables!!!!!!!!!!!

Joins using MovieDB Database

Open **MovieDB_ERD.png**

JOINS

ANATOMY OF A JOIN STATEMENT



```
SELECT city_name, state_name FROM city

JOIN state ON state.state_abbreviation = city.state_abbreviation

WHERE city_name = 'Springfield';
```

Starting table

JOIN clause for another table

WHERE clause

ON keyword

Field relation expression

UNIONS

A SQL Union:

- Combines the results of two or more queries into a single result set.
- The number of columns involved as well as the data types for those columns in each query **MUST BE THE SAME**.
- Duplicate rows are removed

Example:

Faculty and student contact info is stored in separate tables but we want a combined list of faculty and students in the campus directory.

UNIONS

Sample SQL Union

```
SELECT population FROM city

UNION

SELECT population FROM state
```