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.

`code` can be used as a natural primary key.

Log | 1: COUNTRY [239]

*	code	name	continent	region
1	AFG	Afghanistan	Asia	Southern and Central Asia
2	NLD	Netherlands	Europe	Western Europe
3	ANT	Netherlands Antilles	North America	Caribbean
4	ALB	Albania	Europe	Southern Europe
5	DZA	Algeria	Africa	Northern Africa
6	ASM	American Samoa	Oceania	Polynesia
7	AND	Andorra	Europe	Southern Europe
8	AGO	Angola	Africa	Central Africa
9	AIA	Anguilla	North America	Caribbean
10	ATG	Antigua and Barbuda	North America	Caribbean

Keys

Surrogate Primary Keys:

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

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

Log | 1: CITY [1000]

*	id	name	countrycode	district	population
1	1	Kabul	AFG	Kabol	1780000
2	2	Qandahar	AFG	Qandahar	237500
3	3	Herat	AFG	Herat	186800
4	4	Mazar-e-Sharif	AFG	Balkh	127800
5	5	Amsterdam	NLD	Noord-Holland	731200
6	6	Rotterdam	NLD	Zuid-Holland	593321
7	7	Haag	NLD	Zuid-Holland	440900
8	8	Utrecht	NLD	Utrecht	234323
9	9	Eindhoven	NLD	Noord-Brabant	201843
10	10	Tilburg	NLD	Noord-Brabant	193238
11	11	Groningen	NLD	Groningen	172701

Keys

Composite Primary Keys:

A primary key made up of multiple fields.

Log | 1: rental [5]

*	rental_id	rental_date	inventory_id	customer_id
1	178	2005-05-26 04:21:46	1489	196
2	2861	2005-06-19 23:21:34	1489	116
3	6795	2005-07-12 16:41:00	1489	245
4	10390	2005-08-01 06:46:48	1489	502
5	14243	2006-02-14 15:16:03	1489	264

`inventory_id` + `rental_date` would be unique and can be used as a key.

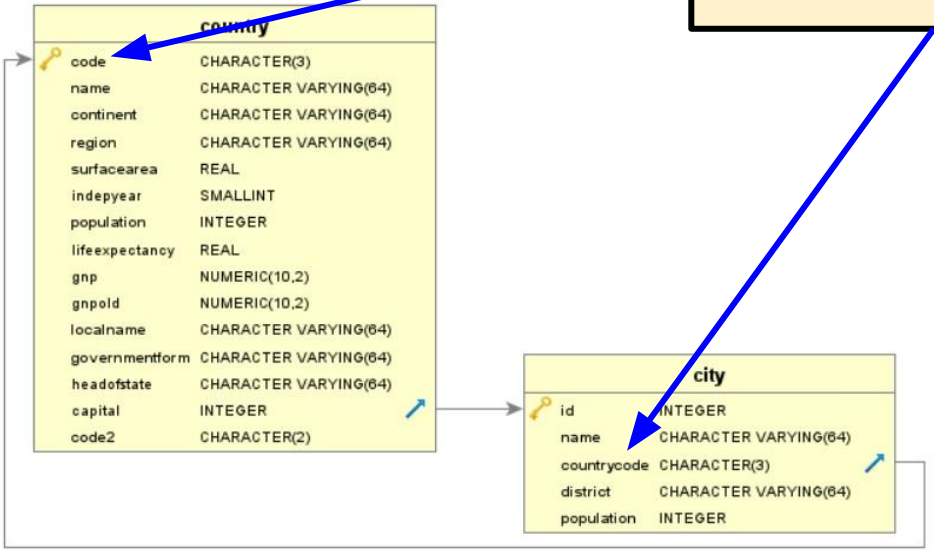
Keys

Foreign Key:

A field that references a primary key in another table.

(Allows enforcement of data integrity)

The **`countrycode`** field in the **`city`** table references the **`code`** field in the **`country`** table.



Keys

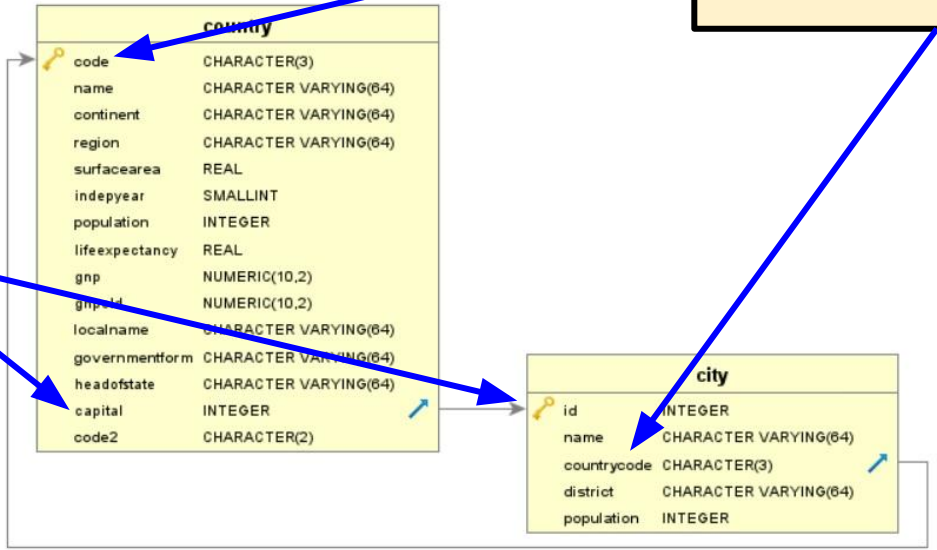
Foreign Key:

A field that references a primary key in another table.

(Allows enforcement of data integrity)

The **`countrycode`** field in the **`city`** table references the **`code`** field in the **`country`** table.

The **`capital`** field in the **`country`** table references the **`id`** field of the **`city`** 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)

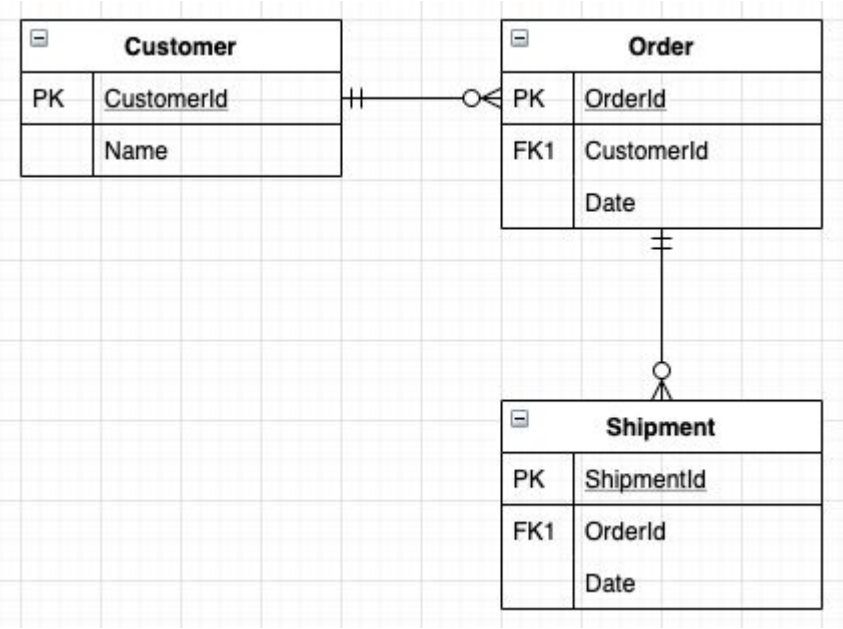
Many rows in table A may relate to many rows in table B.

Example:

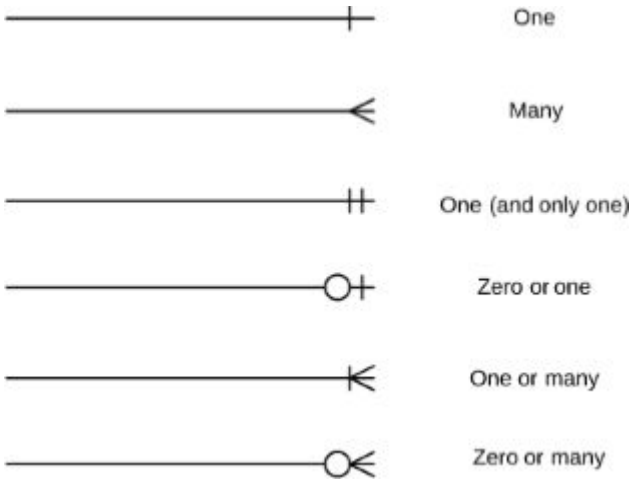
Each record in Film table may relate to multiple records in Actor table and each record in Actor table may relate to multiple records in Film table.

Implemented via join tables (stay tuned...)

Entity Relationship Diagram (ERD)



ERD Cardinality

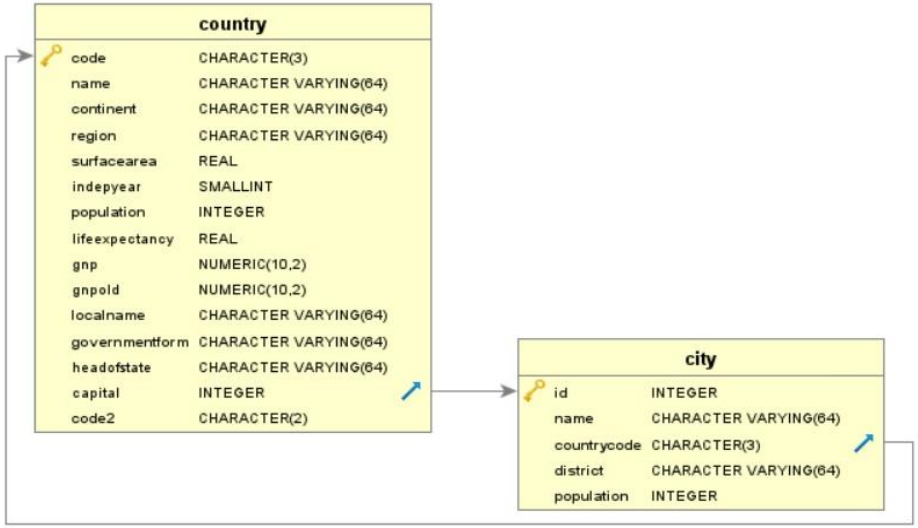


JOINS

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.



JOINS

ANATOMY OF A JOIN STATEMENT

```
SELECT * FROM payment

JOIN customer ON customer.customer_id = payment.customer_id

WHERE payment_id=1599
```

JOINS

ANATOMY OF A JOIN STATEMENT

Starting table

```
SELECT * FROM payment

JOIN customer ON customer.customer_id = payment.customer_id

WHERE payment_id=1599
```

JOINS

ANATOMY OF A JOIN STATEMENT

Starting table

```
SELECT * FROM payment

JOIN customer ON customer.customer_id = payment.customer_id

WHERE payment_id=1599
```

JOIN clause for another table

JOINS

ANATOMY OF A JOIN STATEMENT

Starting table

ON keyword

```
SELECT * FROM payment

JOIN customer ON customer.customer_id = payment.customer_id

WHERE payment_id=1599
```

JOIN clause for another table

JOINS

ANATOMY OF A JOIN STATEMENT

Starting table

ON keyword

```
SELECT * FROM payment

JOIN customer ON customer.customer_id = payment.customer_id

WHERE payment_id=1599
```

JOIN clause for another table

Field relation expression

JOINS

ANATOMY OF A JOIN STATEMENT

Starting table

ON keyword

```
SELECT * FROM payment

JOIN customer ON customer.customer_id = payment.customer_id

WHERE payment_id=1599
```

JOIN clause for another table

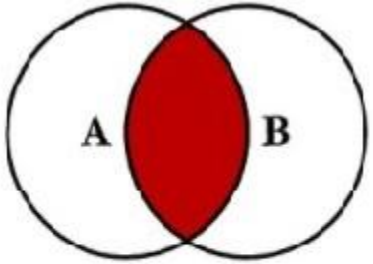
WHERE clause

Field relation expression

JOINS

INNER JOINS

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



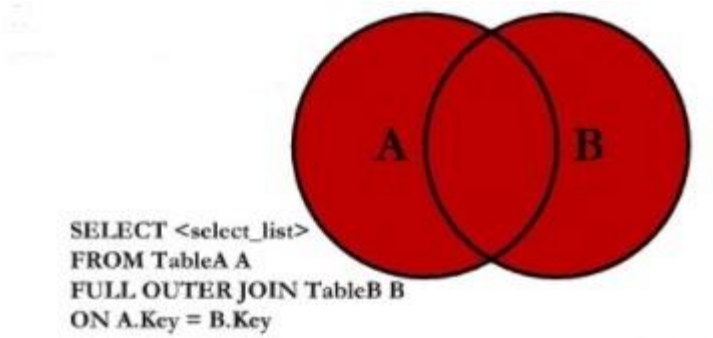

```
SELECT <select_list>
FROM TableA A
INNER JOIN TableB B
ON A.Key = B.Key
```

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.

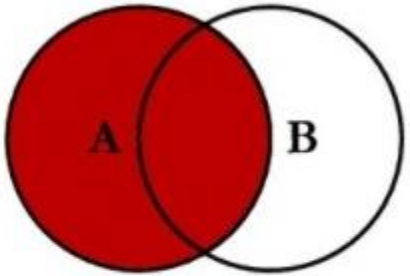
A Full Outer Join returns the data from both tables, including unmatched data.



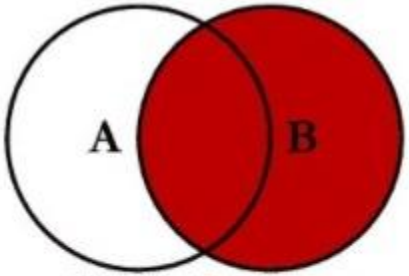
JOINS

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.

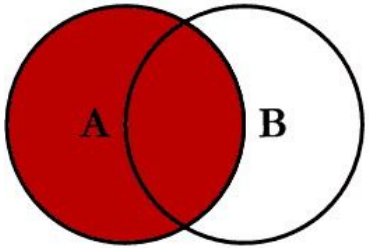



```
SELECT <select_list>
FROM TableA A
LEFT JOIN TableB B
ON A.Key = B.Key
```

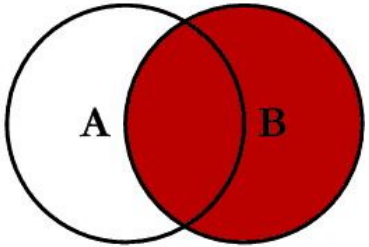



```
SELECT <select_list>
FROM TableA A
RIGHT JOIN TableB B
ON A.Key = B.Key
```

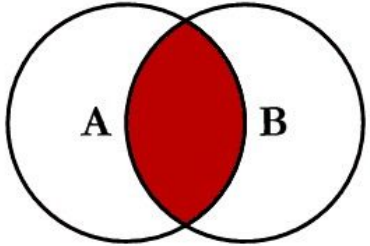
SQL JOINS



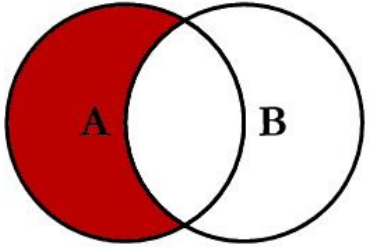

```
SELECT <select_list>
FROM TableA A
LEFT JOIN TableB B
ON A.Key = B.Key
```



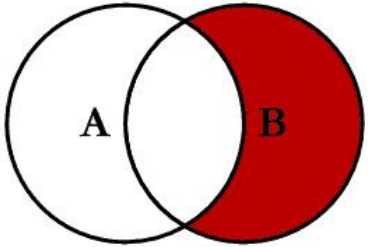

```
SELECT <select_list>
FROM TableA A
RIGHT JOIN TableB B
ON A.Key = B.Key
```



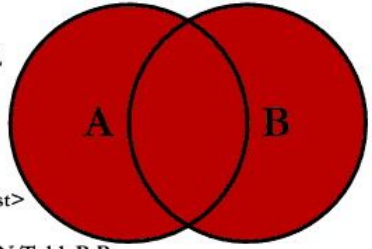

```
SELECT <select_list>
FROM TableA A
INNER JOIN TableB B
ON A.Key = B.Key
```



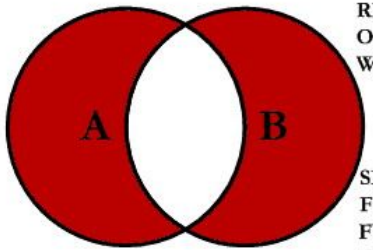

```
SELECT <select_list>
FROM TableA A
LEFT JOIN TableB B
ON A.Key = B.Key
WHERE B.Key IS NULL
```




```
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)

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

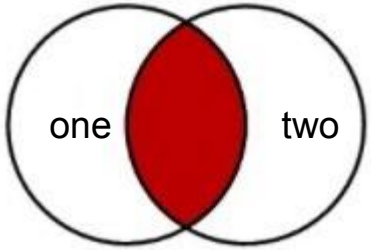
JOINS

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;
```



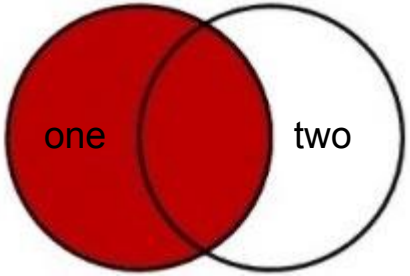
JOINS

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;
```



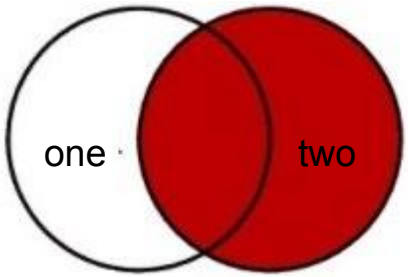
JOINS

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;
```



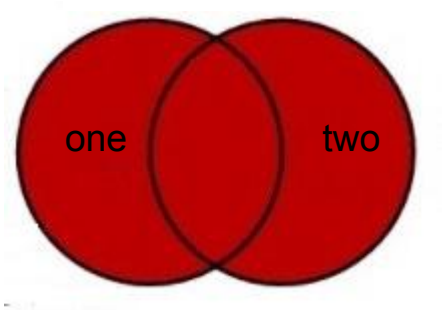
JOINS

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

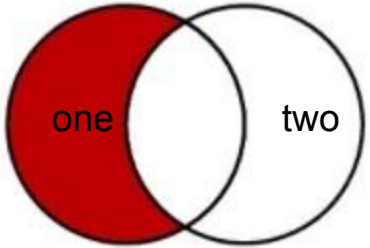
Useful Variation: Left Table Values Only

```
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

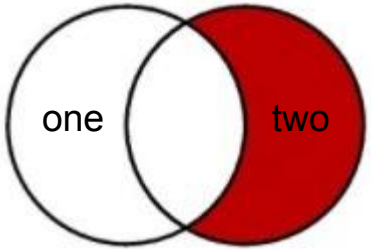
Useful Variation: Right Table Values Only

```
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

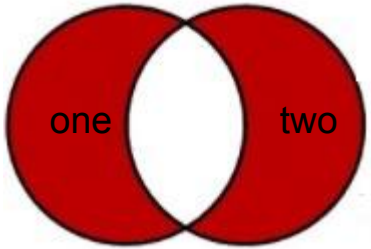
Useful Variation: Left or Right Table Values But Not Both

```
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
```



Setting Up the MovieDB Database

JOINS

Open **MovieDB_ERD.png**

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

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 first_name FROM actor

UNION

SELECT first_name FROM customer
```