Library Carpentry: Working with Data

11-12th May 2023

Emma Hobbs

What we will cover and objectives:

Day 3: Regular Expressions (Regex)

- What is regex?
- Why use regex?
- Identify potential use cases for regular expressions
- Recognize common regex metacharacters
- How to use regex in searches?

Day 4: SQL

- What is a relational database?
- Understand the difference between tables and databases
- Explain the purpose a database schema
- Create simple SQL queries to return rows an columns from a table in a database
- Create new columns and values in a database using SQL
- Use joins to perform queries across multiple tables in a database

Library Carpentry: Day 3 Introduction to Working with Data (Regular Expressions)

11th May 2023

Emma Hobbs

What we'll cover

- **What** is regex?
- **Why** use regex?
- **How** to use regex?
  - Identify potential use cases for regular expressions
  - Recognize common regex metacharacters
  - How to use regex in searches?

10:00-10:15 Introduction

10:15-11:00
Theory:
- What are regular expressions, and why use them?
- Matching letters, cases and numbers[]

Exercises: (breakout rooms) Using square brackets

11:00-11:15 BREAK

11:15-12:00
Recap the exercises
Theory:
- Wildcards and escaping: ., \., \d, \w and \s
- Finding substring: ^, $, \b

Exercises: (breakout rooms) Using special characters

12:00-13:00 LUNCH

13:00-14:00
Recap exercises
Theory: Wilcards and repeats: *, +, ?, {value}, |
Exercises: Remaining exercises from lesson 1: (breakout rooms)

14:00-14:15 BREAK

14:15-15:00
Recap exercises
MCQs: together (answer in the etherpad below)
Longer / real-word exercises (breakout rooms)
- Exercise: finding email addresses:
- Exercise: finding phone numbers:Recap and Wrap up

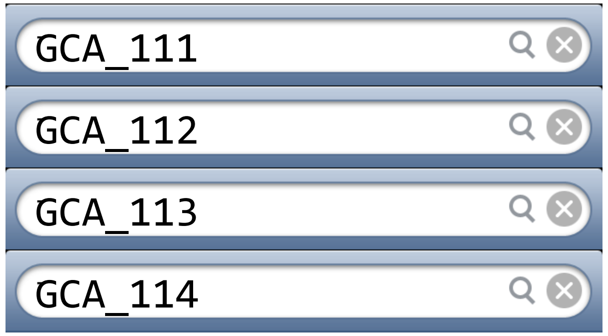
~~What are~~ **Why** regular expressions?

Table: employees

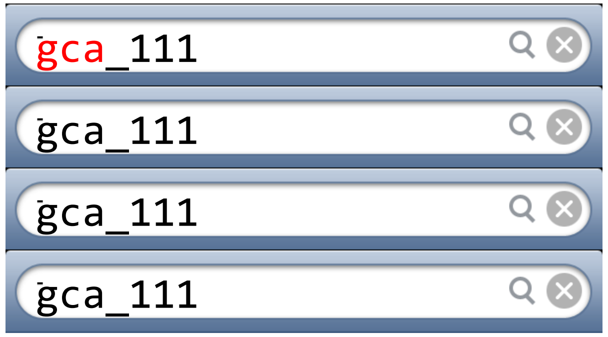
employee_id	**name**	**age**	**role**	**Country**
GCA_125	William Afton	53	Manager	US
gca_522	Henry Emily	52	Senior engineer	UK
Tkm_888	Mike Schmidt	35	Software engineer	UK
pom_124	Emily Grey	22	Intern	Spain

Common pattern (e.g. GCA_###) ← Non-standardised

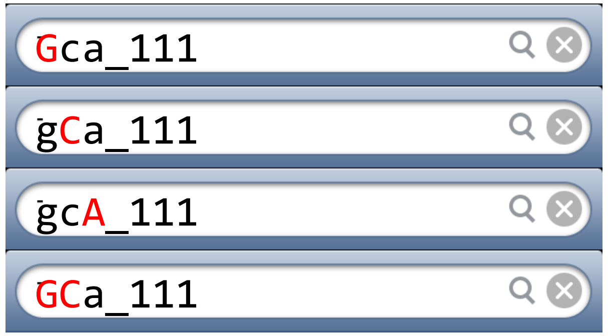
All combinations of digits:


GCA_111
GCA_112
GCA_113
GCA_114

& Repeat with lower case 'gca':


gca_111
gca_111
gca_111
gca_111

& All combinations of upper/lower case:


Gca_111
gCa_111
gcA_111
GCa_111

Search for data that matches a **pattern** of character, not direct matches

What are regular expressions? (Regex)

“Regular expressions are a concept and an implementation used in many different programming environments for sophisticated **pattern matching**” – The Carpentries

“Regex is able to capture a **pattern** in a string” – Towards Datascience

“Regex is a sequence of characters that define a search **pattern**” – Geeks for Geeks

- Method, approach, concept *– not a tool or package*
- String: a sequence of characters *– “a string”, “also a string 123”*
- Widely implemented
- Match:
  - Types of characters (upper case, digits, spaces, etc
  - Match patterns
  - Capture parts (substring) of an original string

How and why do regular expressions work?

- Use a combination of literal characters and metacharacters

Literal characters

One meaning

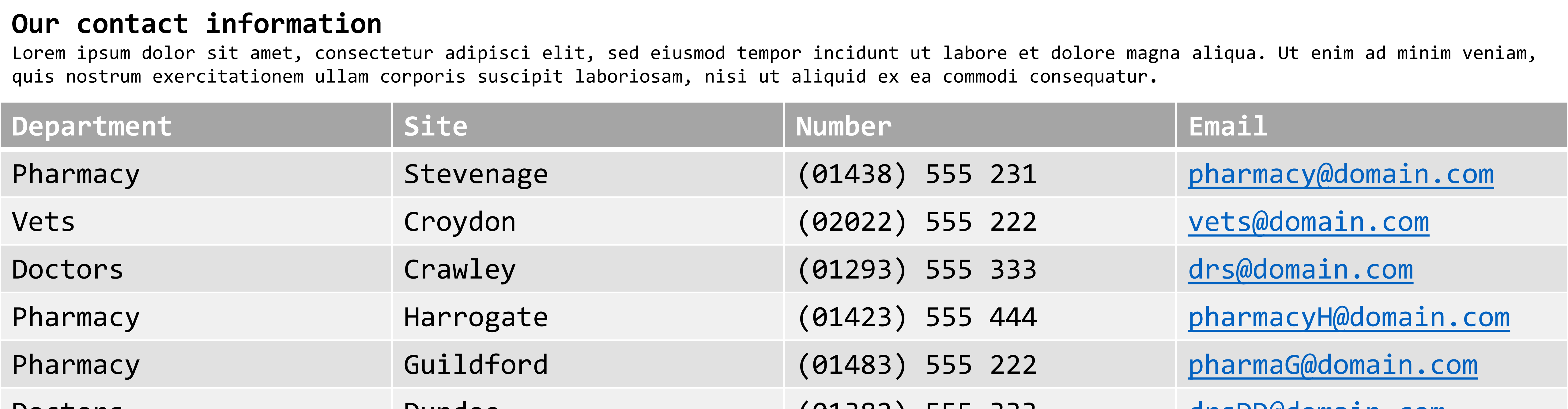
‘a’, ‘b’

Metacharacters

American Standard Code for Information Interchange (ASCII)

*Special meaning (\, *, ^ etc.)*

Often have more than one literal meaning



Our contact information

Lorem ipsum dolor sit amet, consectetur adipisci elit, sed eiusmod tempor incidunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrum exercitationem ullam corporis suscipit laboriosam, nisi ut aliquid ex ea commodi consequatur.

Department	Site	Number	Email
Pharmacy	Stevenage	(01438) 555 231	pharmacy@domain.com
Vets	Croydon	(02022) 555 222	vets@domain.com
Doctors	Crawley	(01293) 555 333	drs@domain.com
Pharmacy	Harrogate	(01423) 555 444	pharmacyH@domain.com
Pharmacy	Guildford	(01483) 555 222	pharmaG@domain.com
Doctors	Dundee	(01382) 555 333	drsDD@domain.com

Warning: Regex Syntax and interoperability

Metacharacters: [square brackets]

- Square brackets define a list *or* range of characters to be found

- **List** of potential characters to be found

```
[ABC] – will match ‘A’, ‘B’ or ‘C’
```

- **Range** of potential characters to be found

```
[A-Z] – will match any uppercase letter
```

- Case sensitive **range** of potential characters to be found

```
[A-Za-z] – will match any uppercase or lowercase letter
```

- Combine numbers and letters

```
[A-Za-z0-9] – will match any letter or number
```

Metacharacters: [square brackets]

Exercises:

1. Using square brackets: What will the regular expression Fr[ea]nc[eh] match?

2. Taking spaces into consideration:
   1. Type 'community' into the regex box (excluding the quotation marks). How many matches are there?
   2. Type 'community ' (community followed by a space) into the regex box (excluding the quotation marks). How many matches are there?
   3. Why are there a different number of matchets between 'community' and 'community '

3. Exploring effect of expressions matching different words: Change the expression to communi and you get 15 full matches of several words. Why?

4. Taking capitalisation into consideration: Type the expression '[Cc]ommuni'. You get 16 matches. Why?

Metacharacters: wildcard and escaping

- Full stop means **any** character

```
.
```

- Match a period/full stop by **escaping** the special character

```
\. – will match '.'
```

- Match a single digit – **escape** the d character

```
\d – will match any single digit (equivalent to [0-9])
```

- Match a letter digit – **escape** the w character

```
\w – will match any single letter (equivalent to [A-Za-z])
```

- Match a space, tab or new line – **escape** the s character

```
\s – will match any ' ', '    '(\t), and '\n'
```

Metacharacters: Regex finds substrings

- Match a string that starts with…

`^ what is written` **`after`** `^ will match the start of the string`

- Match at the end of a string

`$` `– what is written` **`before $`** `will match the end of a string`

- The pattern must match at a word boundary

`\b` `Putting this either side of a word stops the regular expression matching longer variants of words`

Metacharacters: Wildcards, escaping and substrings

Exercises:

1. Using special characters in regular expressions matches
2. Using dollar signs

3. Taking any character into consideration
4. Regex characters that indicate location

Metacharacters: wildcards and repeats

- matches the preceding element zero or more times

`* - ab*c = ac, abc, abbc, abbbc etc.`

- matches the preceding element one or more times

`+ - ab+c = abc, abbc, abbbc etc.`

- matches when the preceding character appears zero or one time

`?`

- matches the preceding character the number of times defined by VALUE

`{value} – {4} 4 repeats, {1,6} 1-6 repeats`

- means or

`|`

- renders an expression case-insensitive

`/I – equivalent to [A-Za-z]`

Metacharacters: Wildcards and repeats

Exercises from lesson 1:

First set of exercises in lesson 1:

1. [Oo]Rgani.e\w*
2. [Oo]rgani.e\w+$
3. ^[Oo]rgani.e\w?\b
4. ^[Oo]rgani.e\w?$
5. \b[Oo]rgani.e\w{2}\b
6. \b[Oo]rgani.e\b|\b[Oo]rgani.e\w{1}\b

Second set of exercises in lesson 1:

7. Introducing options
8. Case insensitivity
9. Word boundaries
10. Matching non-linguistic patterns
11. Matching digits
12. Matching dates
13. Matching multiple date formats
14. Matching publication formats

Library Carpentry: Day 4
Data Management with SQL

12th May 2023

Emma Hobbs

What we’ll cover

10:00-10:15 Introduction

10:15-11:00
Theory:
- What is SQL, and SQL vs SQLite
- What is a relational database

Practical
- Using DB Browser
- Start on: The SELECT and FROM statements

11:00-11:15 BREAK

11:15-12:00
Practical:
- Continue SELECT, FROM and WHERE statements
- Missing data

Exercises

12:00-13:00 LUNCH

13:00-14:00
Recap exercises
Practical:
- Creating new columns
- Aggregations
- Creating tables and views

14:00-14:15 BREAK

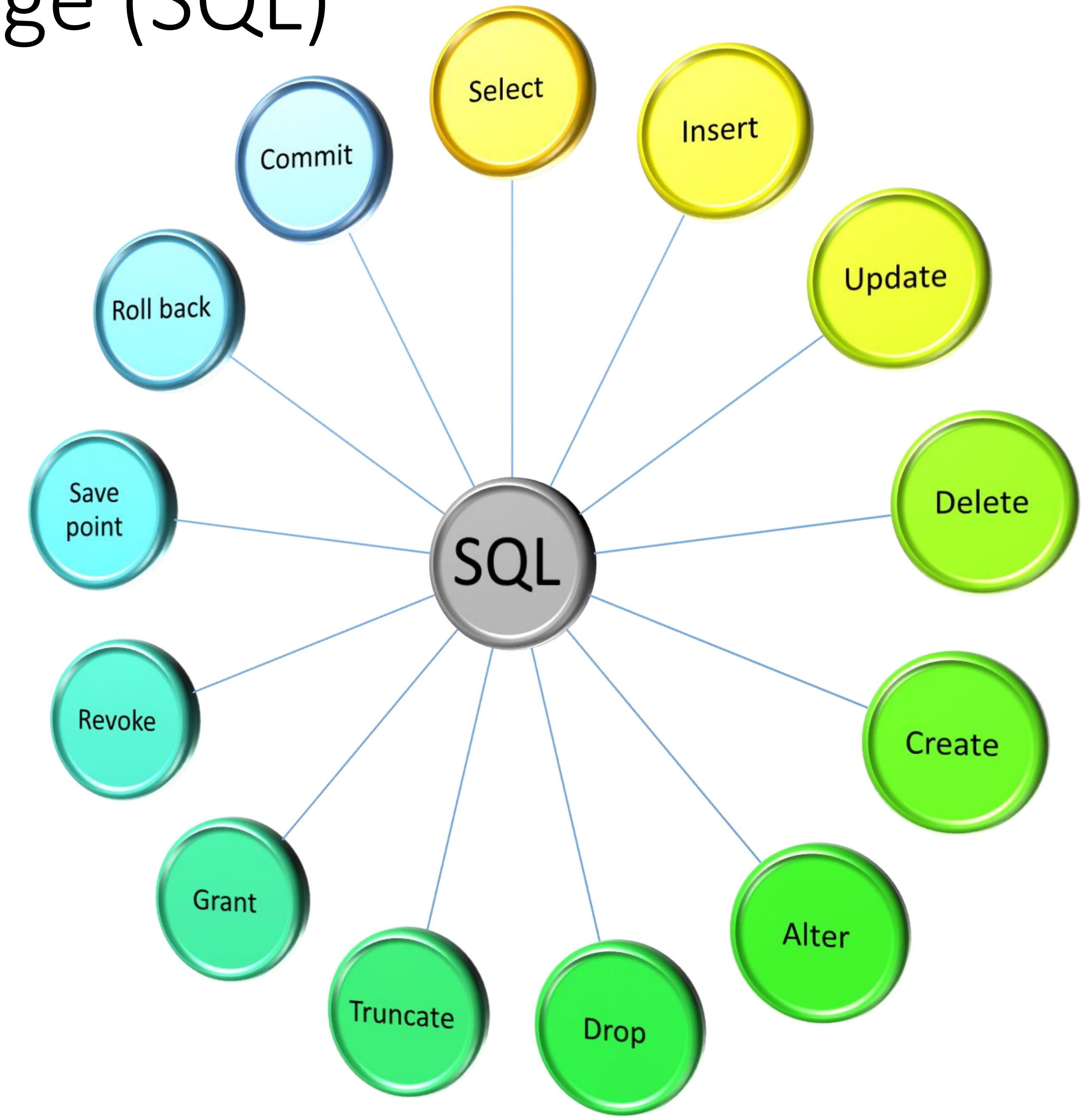
14:15-15:00
Theory and practical:
- Joins
- (if time) Using database tables in other environments
- (if time) The SQLite command line

Exercises

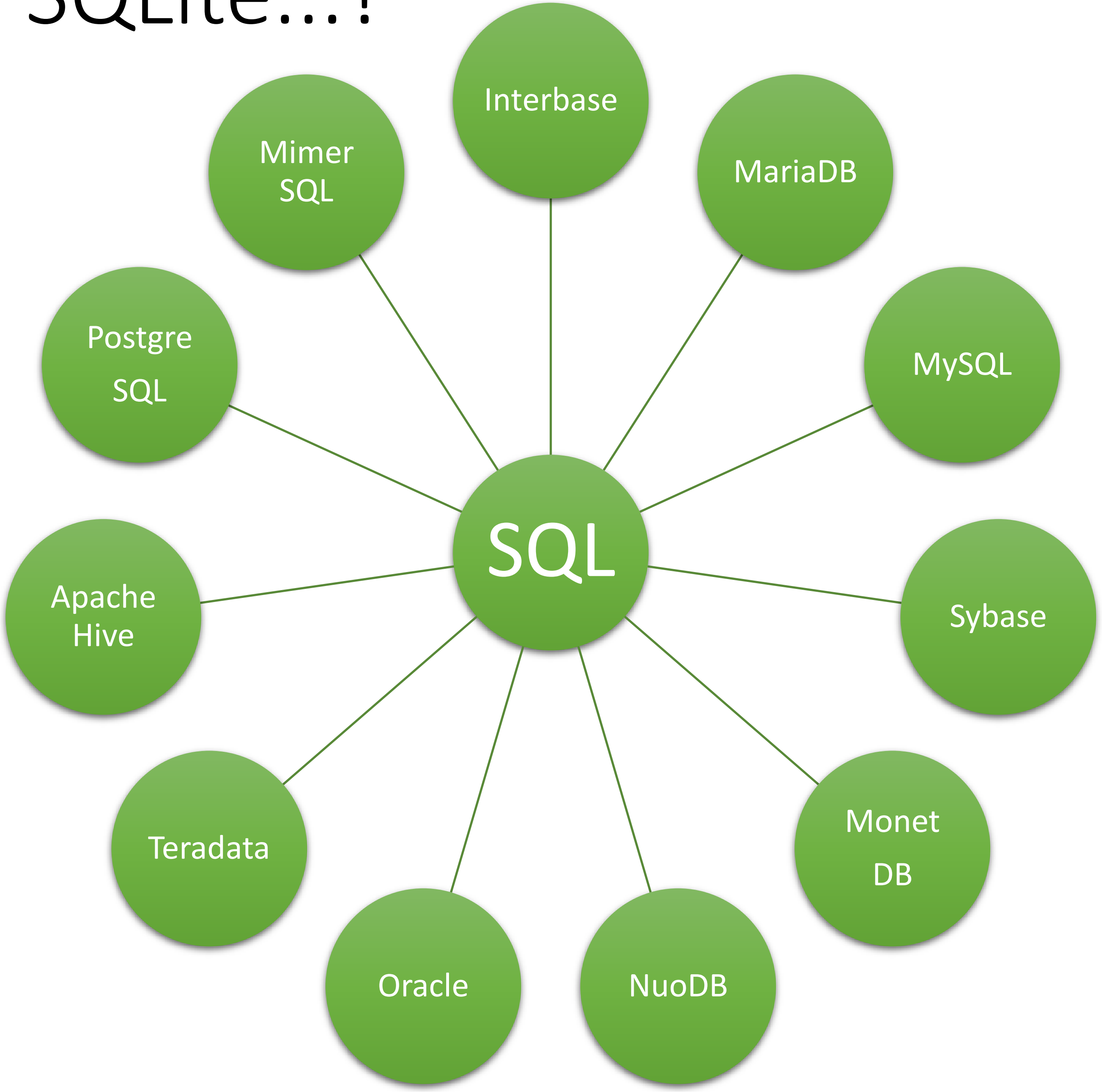
Recap and wrap up

Structured Query Language (SQL)

- Domain specific-language
- Consists of many *statements*



SQL, MySQL, SQLite...?



What is a database?

- What is a table?
- What is a database?
- What is the difference between a table and a database?

Table:

Has rows and columns

Row = observation

Column = variable

Variable

Name	Age	English	Maths
Jack	12	A	B
Anu	13	C	A
Zaynah	12	A	A

Observation

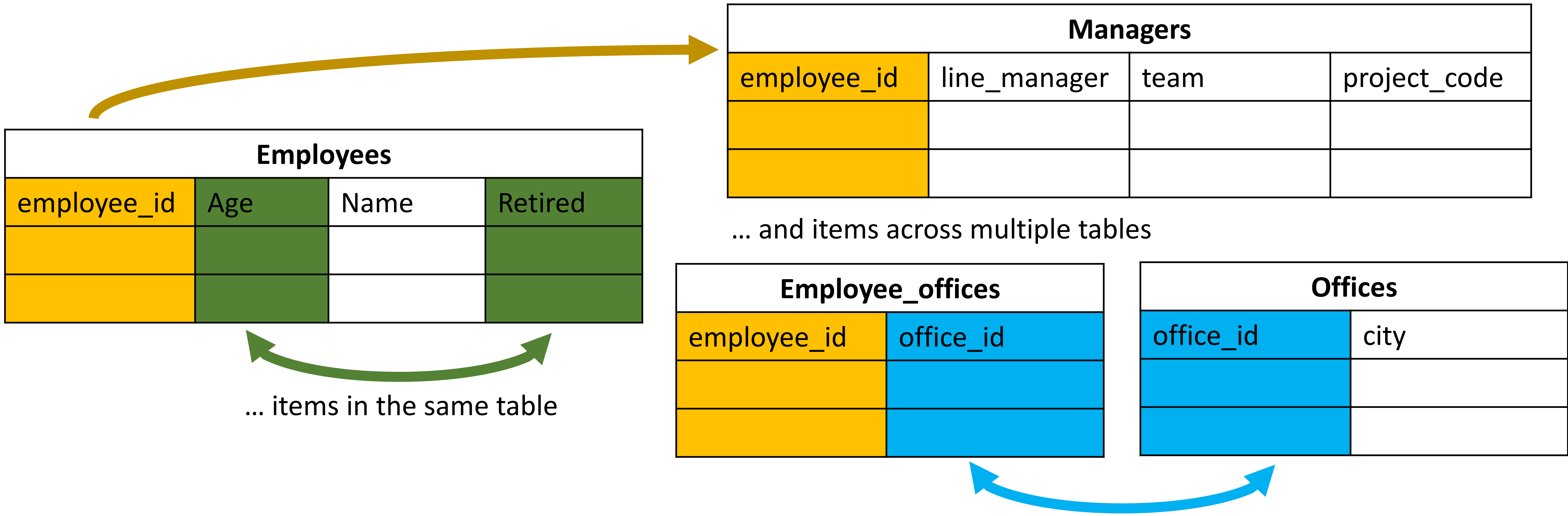
Database:

Structured set of data

Organised collection of data

What is a relational database?

- Collection of data that is organised into a set of tables
- Relationships can be defined between items



employee_id	Age	Name	Retired	Salary
1	25	Jack	N	32
2	26	Ahmed	N	32
3	25	Sam	N	32
4	25	Jamie	N	34

Employees

employee_id	Age	Name	Retired	salary_id
1	25	Jack	N	1
2	26	Ahmed	N	1
3	25	Sam	N	1
4	25	Jamie	N	2

Salaries

salary_id	salary
1	32
2	34

Data Types

Data type	Description
NULL	The value is a NULL value
INTEGER	The value is a signed integer, stored in 1, 2, 3, 4, 6, or 8 bytes depending on the magnitude of the value
REAL	The value is a floating point value, stored in 8-bytes
TEXT	The value is a text string
BLOB	The data is stored exactly as it was input, Used for binary data such as images.

Primary Keys & Foreign Keys

employees			
employee_id	Age	Name	Retired
1	25	Jack	0
2	26	Ahmed	0
3	25	Sam	0
4	25	Jamie	0

Primary Key (i.e. record ID number)

Cars			
registration	age	make	sold
KY05 KFH	18	Fait	1
EJ18 SKN	5	Hyundai	0
OH22 OOH	1	Kia	1
ZN72	1	Kia	0

Primary Keys & Foreign Keys

Employees

employee_id	Age	Name	Retired
Primary Key	25	Jack	0
2	26	Ahmed	0
3	25	Sam	0
4	25	Jamie	0

Offices

office_id	city
Primary Key	**Text**
1	London
2	Dundee

Primary Key (i.e. record ID number)

… is a Foreign Key in another table

Employees_Offices

employee_id	office_id
Foreign Key	**Foreign Key**
2	1
3	2
4	2

DB Browser

- Practical

The SELECT Statement

Key word	**Item**	***Use***
SELECT	Column names	*Columns we want to be returned from the database*
FROM	Table names	*Tables to retrieve the columns from*
WHERE	Conditions	*Filter returned data. i.e. only return values greater than 50*
GROUP BY	Column names	*Group data together by a common column name*
HAVING	Conditions	*Conditional when working with aggregate functions*
ORDER BY	Column names	*Order the returned data by the values in the given columns*
LIMIT	Integer	*Max number of rows to return*

Missing Data

- Practical

Joins: (*Recap Primary Keys & Foreign Keys*)

Employees

employee_id	Age	Name	Retired
Primary Key	25	Jack	0
2	26	Ahmed	0
3	25	Sam	0
4	25	Jamie	0

Primary Key (i.e. record ID number)

… is a Foreign Key in another table

Offices

office_id	City
Primary Key	**Text**
1	London
2	Dundee

Employee_Offices

employee_id	office_id
Foreign Key	**Foreign Key**
2	1
3	2
4	2

Joins

```
SELECT col_name_1, col_name_2
FROM table_2
JOIN table_2 ON table_1.primary_key = table_2.foreign_key
WHERE col_name_3 > condition
```

WHERE col_name_3 > condition

Table we want to join

Table to join on to

Table we want to join

Joins

```
SELECT col_name_1, col_name_2
FROM table_2
JOIN table_2 ON table_1.primary_key = table_2.foreign_key
```

Table we want to join

Table to join on to

Table we want to join

Joins

```
SELECT name
FROM Employees
```

Employees

employee_id	age	name	retired
Primary Key	*Integer*	*Name*	*Bool*
2	26	Ahmed	0
3	25	Sam	0
4	25	Jamie	0

Employee_Offices

employee_id	office_id
Foreign Key	***Foreign Key***
2	1
3	2
4	2

Offices

office_id	salary
Primary Key	***Text***
1	London
2	Dundee

Joins

```
SELECT name
FROM Employees
JOIN Employee_Offices ON
     Employees.employee_id = Employee_Offices.employee_id
```

Employees

employee_id	**age**	**name**	**retired**
Primary Key	*Integer*	*Name*	*Bool*
2	26	Ahmed	0
3	25	Sam	0
4	25	Jamie	0

Employee_Offices

employee_id	**office_id**
Foreign Key	***Foreign Key***
2	1
3	2
4	2

Offices

office_id	**City**
Primary Key	***Text***
1	London
2	Dundee

Joins

```
SELECT name
FROM Employees
JOIN Employee_Offices ON
     Employees.employee_id = Employee_Offices.employee_id
```

Employees				**Employee_Offices**	
employee_id	**age**	**name**	**retired**	**employee_id**	**office_id**
Primary Key	*Integer*	*Name*	*Bool*	***Foreign Key***	***Foreign Key***
2	26	Ahmed	0	2	1
3	25	Sam	0	3	2
4	25	Jamie	0	4	2

Offices	
office_id	**city**
Primary Key	***Text***
1	London
2	Dundee

Joins

```
SELECT name
FROM Employees
JOIN Employee_Offices ON
     Employees.employee_id = Employee_Offices.employee_id
JOIN Offices ON
     Employee_Offices.office_id = Offices.office_id
```

Employees				Employee_Offices	
employee_id	age	name	retired	employee_id	office_id
Primary Key	*Integer*	*Name*	*Bool*	***Foreign Key***	***Foreign Key***
2	26	Ahmed	0	2	1
3	25	Sam	0	3	2
4	25	Jamie	0	4	2

Offices	
office_id	city
Primary Key	***Text***
1	London
2	Dundee

Joins

```
SELECT name
FROM Employees
JOIN Employee_Offices ON
     Employees.employee_id = Employee_Offices.employee_id
JOIN Offices ON
     Employee_Offices.office_id = Offices.office_id
WHERE Offices.city = ‘London’
```

Employees				Employee_Offices		Offices	
employee_id	**age**	**name**	**retired**	**employee_id**	**office_id**	**office_id**	**city**
Primary Key	*Integer*	*Name*	*Bool*	***Foreign Key***	***Foreign Key***	***Primary Key***	***Text***
2	26	Ahmed	0	2	1	1	London
3	25	Sam	0	3	2	2	Dundee
4	25	Jamie	0	4	2	2	Dundee

Joins

```
SELECT name
FROM Employees
JOIN Employees_Offices ON
     Employees.employee_id = Employees_Offices.employee_id
JOIN Offices ON
     Employees_Offices.office_id = Offices.office_id
AND Offices.city = ‘London’
```

Employees				Employees_Offices		Offices	
employee_id	**age**	**name**	**retired**	**employee_id**	**office_id**	**office_id**	**city**
Primary Key	*Integer*	*Name*	*Bool*	***Foreign Key***	***Foreign Key***	***Primary Key***	***Text***
2	26	Ahmed	0	2	1	1	London

Types of Joins

```
SELECT name
FROM Employees
INNER JOIN Employee_Offices ON
    Employees.employee_id = Employee_Offices.employee_id
INNER JOIN Offices ON
    Employee_Offices.office_id = Offices.office_id
WHERE Offices.city = ‘London’
```

Employees				Employees_Offices		Offices	
employee_id	**age**	**name**	**retired**	**employee_id**	**office_id**	**office_id**	**city**
Primary Key	*Integer*	*Name*	*Bool*	***Foreign Key***	***Foreign Key***	***Primary Key***	***Text***
2	26	Ahmed	0	2	1	1	London
3	25	Sam	0	3	2	2	Dundee
4	25	Jamie	0	4	2		

Types of Joins

JOIN

Left table

Right table

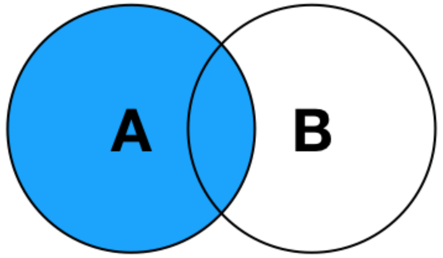
employees				Employees_Offices		Offices	
employee_id	**age**	**name**	**retired**	**employee_id**	**office_id**	**office_id**	**city**
Primary Key	*Integer*	*Name*	*Bool*	***Foreign Key***	***Foreign Key***	***Primary Key***	***Text***
2	26	Ahmed	0	2	1	1	London
3	25	Sam	0	3	2	2	Dundee
4	25	Jamie	0	4	2		

Left table

Right table

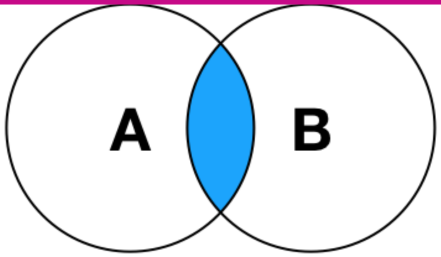
JOIN

Types of Joins

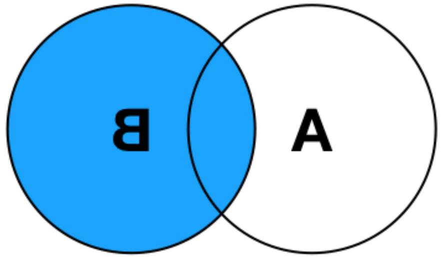


LEFT JOIN

(Left outer join)
Return rows from left and matched rows from right

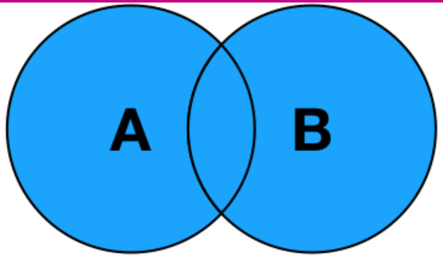


INNER JOIN
Return **matched** rows from both tables

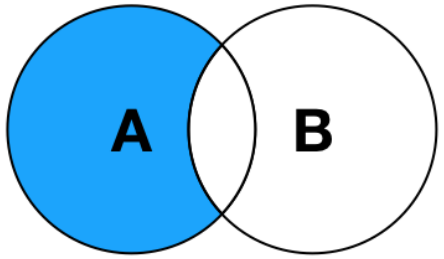


RIGHT JOIN

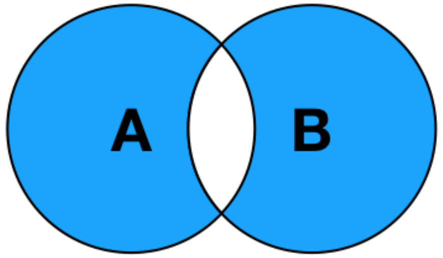
(Right outer join)
Return rows from right and matched rows from left



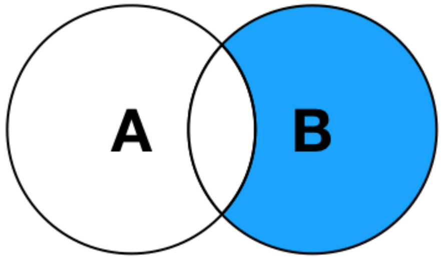
FULL OUTER JOIN
Return **all** rows from both tables



LEFT JOIN EXCLUDING INNER JOIN
Return rows from left that do not match rows in the right table

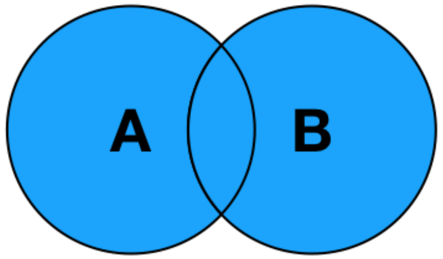


FULL OUTER JOIN EXCLUDING INNER JOIN

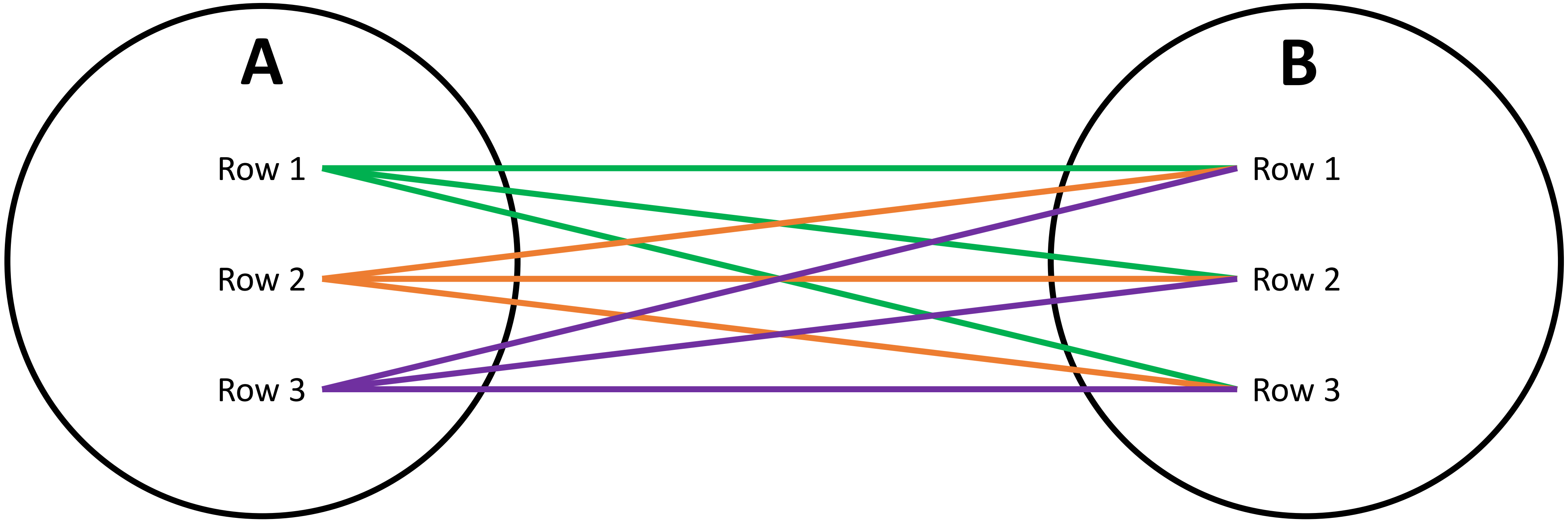


RIGHT JOIN EXCLUDING INNER JOIN
Return rows from right that do not match rows in the left table

Types of Joins



FULL OUTER JOIN
Return **all** rows from
both tables


A
Row 1
Row 2
Row 3
B
Row 1
Row 2
Row 3