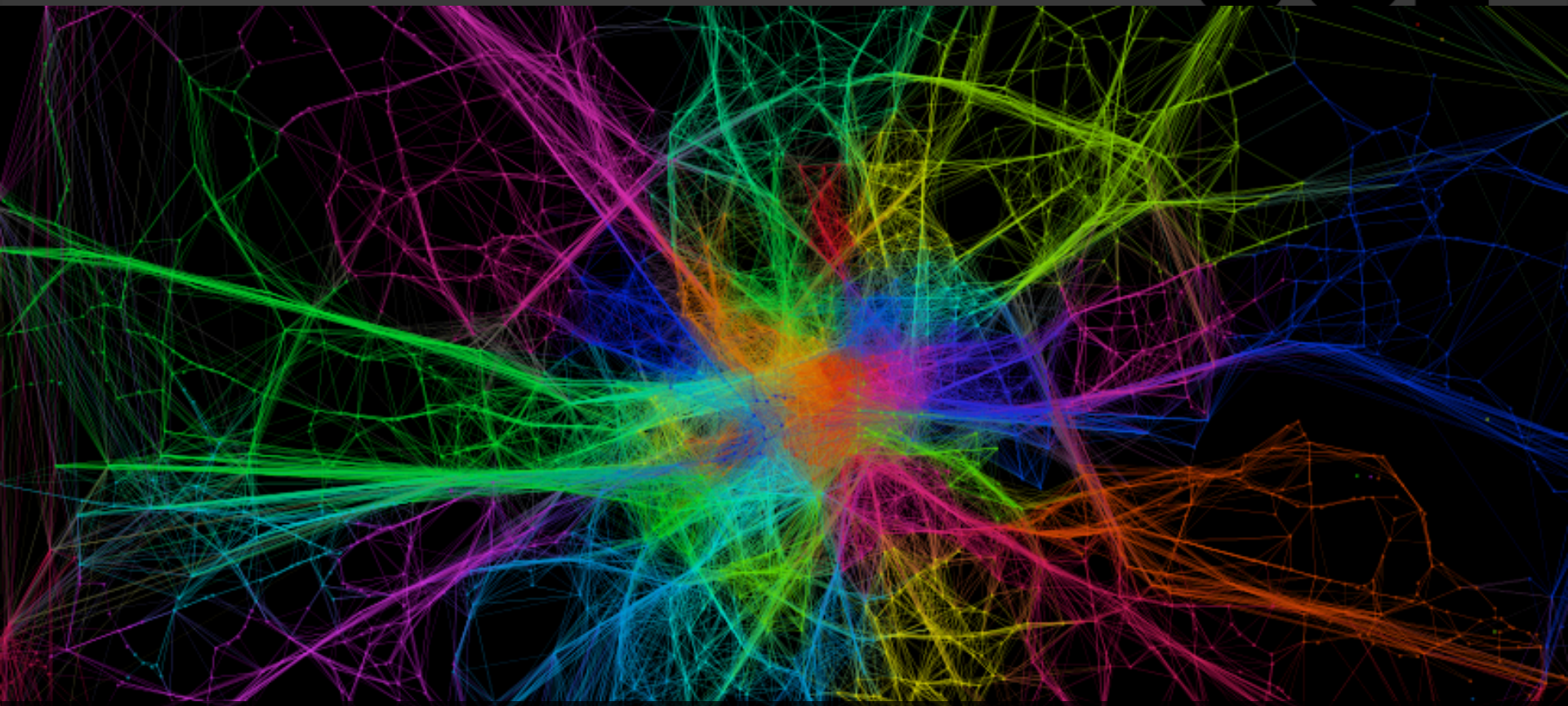
UCL



Advanced SQL

CASA0006: Data Science for Spatial Systems
CASA0009: Spatial Data Capture and Analysis

Steven Gray

Recap

What we already know

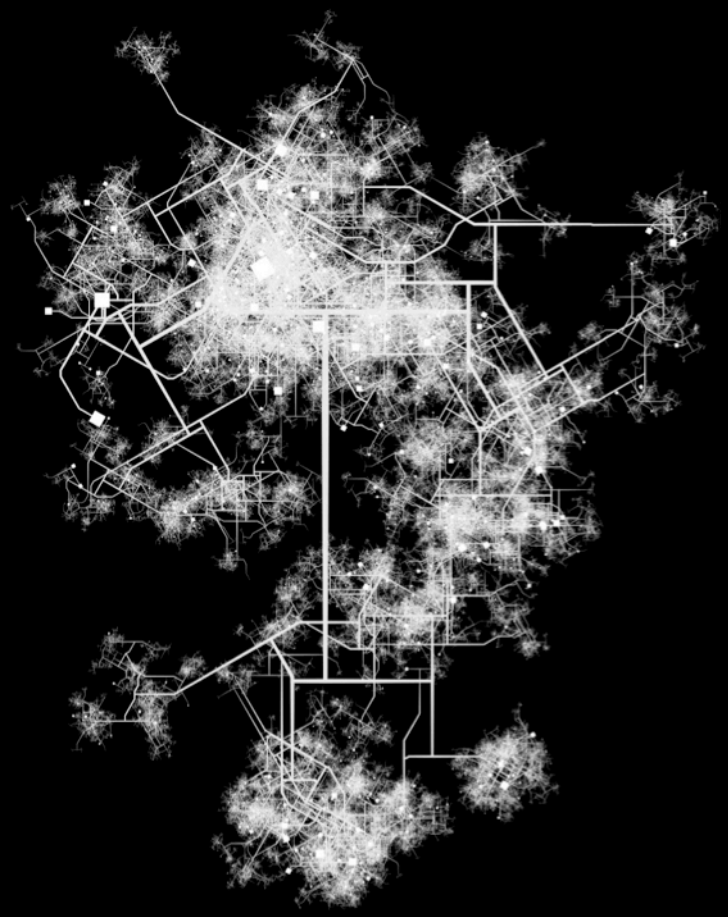
- SELECTing data
- Grouping data
- Ordering data
- Amending table contents

```
SELECT * FROM cities
WHERE continent = 'EU'
ORDER BY pop;
```

```
SELECT country_cd, AVG(pop) AS mean FROM cities
GROUP BY country_cd
HAVING mean > 100000;
```

```
INSERT INTO cities (id,name,lat,lon)
VALUES (99999,'Grantham','Test',52.918, 0.638);
```

Outline



1. What We Already Know
2. Working with Multiple Tables
   a. Join Syntax
   b. Querying Joined Tables
3. Temporary Tables
   a. Purpose
   b. Process
4. Space and Time
   a. Time Formats and Placeholders
   b. Spatial Data
5. Types of SQL
   a. DBMS Engines
   b. Syntax Differences

UCL

Working with Multiple Tables

Join Syntax
Querying Joined Tables

SELECT … WHERE

Across Multiple Tables

To query across tables you must **join them on shared values**, there are a number of ways to do this

A join on a WHERE clause returns all **records with matching keys** from both tables

```
SELECT * FROM cities, countries
WHERE cities.country_cd = countries.country_cd;
```

```
SELECT cities.name, cities.pop/country.pop
FROM cities, countries
WHERE cities.country_cd = countries.country_cd
AND cities.pop > 5000000;
```

```
SELECT a.name, a.pop/b.pop AS prop_pop
FROM cities a, countries b
WHERE a.country_cd = b.country_cd;
```

Table aliases

JOINS

Multiple Tables

Alternatively, **JOIN** syntax controls which combinations of records are returned

```
SELECT * FROM cities a
INNER JOIN countries b
ON a.country_cd = b.country_cd;
```

First table

Joined table

INNER not actually required, but makes syntax clearer

Same as WHERE clause, but clearer when lots of tables are involved

```
SELECT * FROM cities a
LEFT JOIN countries b
ON a.country_cd = b.country_cd;
```

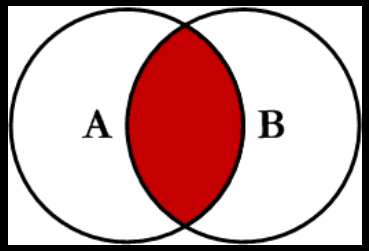
Returns **ALL** from first table, PLUS joined records from second table

```
SELECT * FROM cities a
RIGHT JOIN countries b
ON a.country_cd = b.country_cd;
```

Return **ALL** rows from second table, PLUS joined records from first table

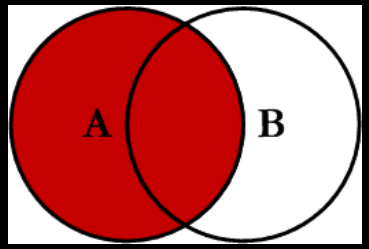
JOINS

For Reference: Types of Join



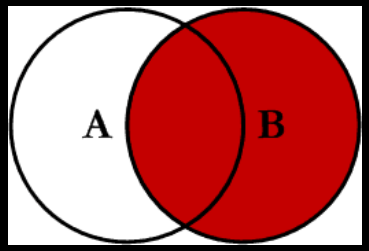
Inner JOIN

```
SELECT <select_list>
FROM Table_A A
INNER JOIN Table_B B
ON A.Key = B.Key
```



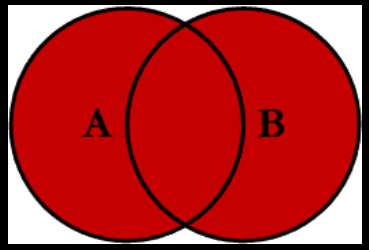
Left JOIN

```
SELECT <select_list>
FROM Table_A A
LEFT JOIN Table_B B
ON A.Key = B.Key
```



Right JOIN

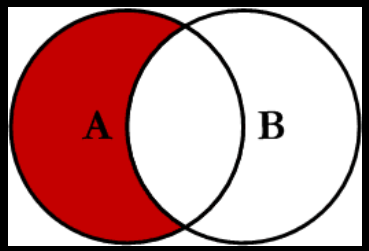
```
SELECT <select_list>
FROM Table_A A
RIGHT JOIN Table_B B
ON A.Key = B.Key
```



Outer JOIN

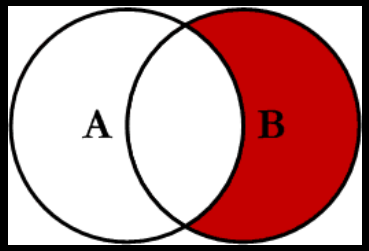
Not available in MySQL

```
SELECT <select_list>
FROM Table_A A
FULL OUTER JOIN Table_B B
ON A.Key = B.Key
```



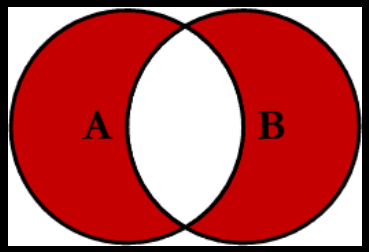
Left excluding JOIN

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



Right excluding JOIN

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



Outer excluding JOIN

Not available in MySQL

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

http://www.codeproject.com/Articles/33052/Visual-Representation-of-SQL-Joins

Temporary Tables

Purpose
Process

Temporary Tables

Creation from queries

Temporary tables provide a repository for query results, enabling quick joins between queried and existing datasets

```
CREATE TEMPORARY TABLE euro_cities
AS
(SELECT name, country_cd FROM cities
WHERE continent = 'EU');
```

Uses a query to specify table definitions

```
SELECT * FROM euro_cities
INNER JOIN countries
ON euro_cities.country_cd = countries.country_cd;
```

Enables join on the predefined subset, rather than full table join followed by SELECT

WARNING Temporary tables disappear at the end of a client session!

Space and Time

Time Formats and Placeholders
Spatial Data

Dates and Times

Formats

Use of date and time data requires a range of **special data types** and **formats** that you should be aware of

Data Type	Format
DATE	'0000-00-00'
TIME	'00:00:00'
DATETIME	'0000-00-00 00:00:00'
TIMESTAMP	'0000-00-00 00:00:00'

Date Time

TIMESTAMP captures timezone DATETIME does not

When stored within one of these types, the data can be extracted in different date and time formats

Dates and Times

Querying

Querying date and time data is pretty easy, but requires a careful formatting

```
SELECT * FROM table WHERE date >= '2010-11-17';

SELECT * FROM table WHERE date < '2010-11-17 18:45:01';

SELECT * FROM table WHERE date BETWEEN '2010-11-17
18:45:01' AND '2010-11-18 12:30';
```

NOTICE THE FORMAT

`Year-Month-Date Hour:Minute:Second`

Dates and Times

Returning

Requesting date or time data requires a more careful specification, using formatting functions and placeholders

```
SELECT DATE_FORMAT(date, '%Y/%m/%d') FROM table;
```

- `DATE_FORMAT` — SQL function to format the contents of a TIMESTAMP column
- `date` — Column name
- `'%Y/%m/%d'` — Desired format

```
2015/01/30
```

```
SELECT DATE_FORMAT(date, '%d-%m-%Y %H:%i') FROM table;
```

```
30-01-2015 10:50
```

```
SELECT DATE_FORMAT(date, '%H:%i.%s') FROM table;
```

```
10:50.22
```

Date Formats

For Reference: Date and Time Placeholders

Specifier	Description
%a	Abbreviated weekday name (Sun..Sat)
%b	Abbreviated month name (Jan..Dec)
%c	Month, numeric (0..12)
%D	Day of the month with English suffix
%d	Day of the month, numeric (00..31)
%e	Day of the month, numeric (0..31)
%f	Microseconds (000000..999999)
%H	Hour (00..23)
%h	Hour (01..12)
%I	Hour (01..12)
%i	Minutes, numeric (00..59)
%j	Day of year (001..366)
%k	Hour (0..23)
%l	Hour (1..12)
%M	Month name (January..December)
%m	Month, numeric (00..12)

Specifier	Description
%p	AM or PM
%r	Time, 12-hour (hh:mm:ss followed by AM or PM)
%S	Seconds (00..59)
%s	Seconds (00..59)
%T	Time, 24-hour (hh:mm:ss)
%u	Week (01..53), where Sunday is the first day of the week
%v	Week (01..53), where Monday is the first day of the week
%W	Weekday name (Sunday..Saturday)
%w	Day of the week (0=Sunday..6=Saturday)
%Y	Year, numeric, four digits
%y	Year, numeric (two digits)
%%	A literal “%” character
%x	x, for any “x” not listed above

Spatial Data

Formats and Functions

Spatial data types are simple formats for storing spatial data, enabling use of MySQL's spatial analysis methods

Data Types

```
GEOMETRY
POINT
LINESTRING
POLYGON
```

Functions

```
ST_Contains(g1, g2)
ST_Crosses(g1, g2)
ST_Distance(g1, g2)
ST_Intersects(g1, g2)
ST_Overlaps(g1, g2)
ST_Touches(g1, g2)
ST_Within(g1, g2)
```

Require GEOMETRY data types as inputs

Function names and availability changes with MySQL versions

e.g.

```
SELECT ST_Distance(pt1, pt2);

SELECT * FROM coords
WHERE ST_Distance(pt1, pt2) < 0.05;
```

UCL

Types of SQL

DBMS Engines
Syntax Differences

SQL Engines

DBMS Alternatives

Open Source SQL

- MySQL (#2)
- PostgreSQL (#4)
- SQLite (#9)
- MariaDB

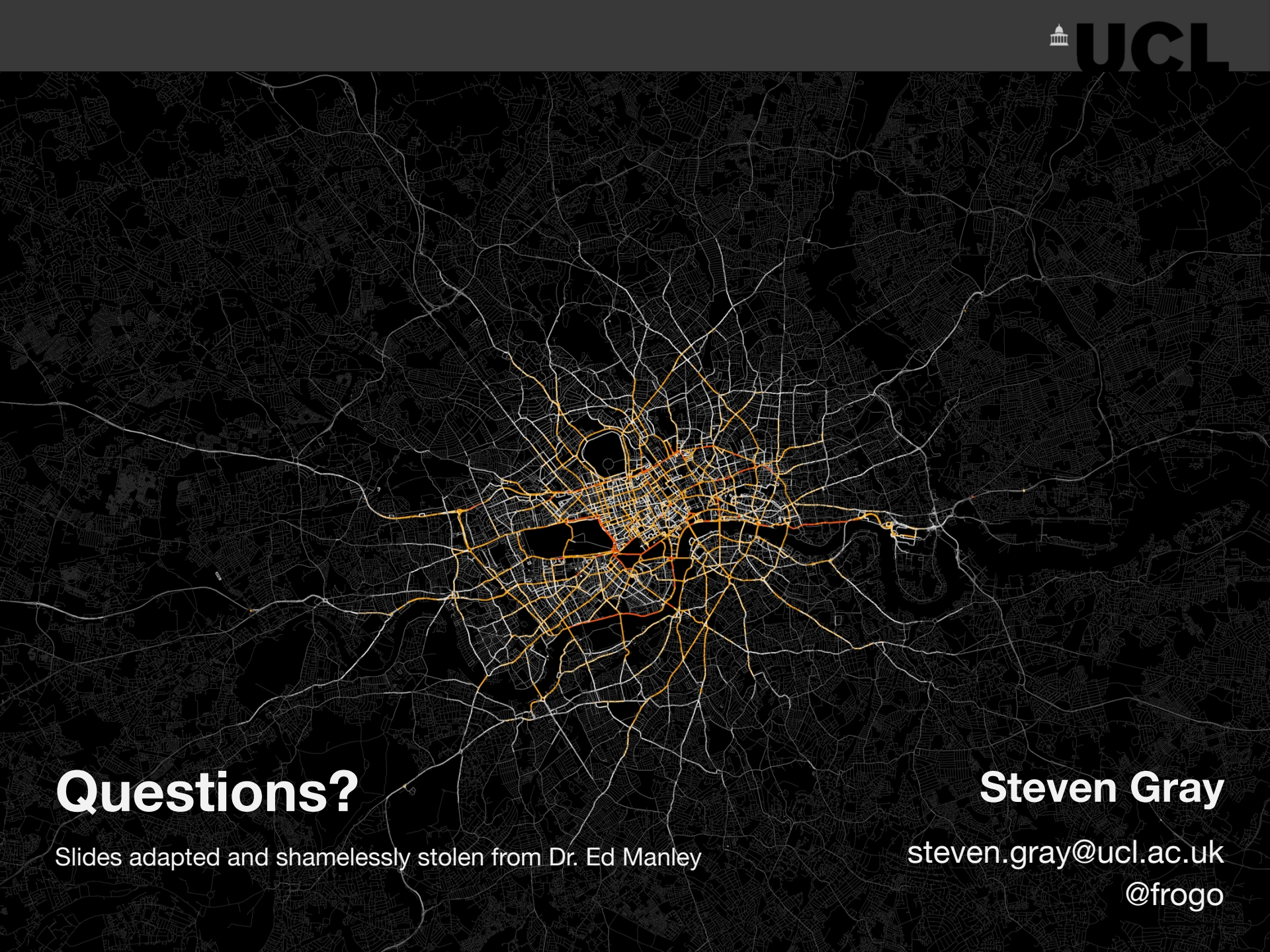
Proprietary SQL

- Oracle (#1)
- MS SQL Server (#3)
- IBM DB2 (#6)
- MS Access (#7)
- Teradata (#12)

NoSQL

- Graph-based (e.g. Neo4j (#23))
- Document-based (e.g. MongoDB (#5))
- Distributed structure (e.g. Hadoop)

SQL Syntax Differences
http://troels.arvin.dk/db/rdbms/



Questions?

Slides adapted and shamelessly stolen from Dr. Ed Manley

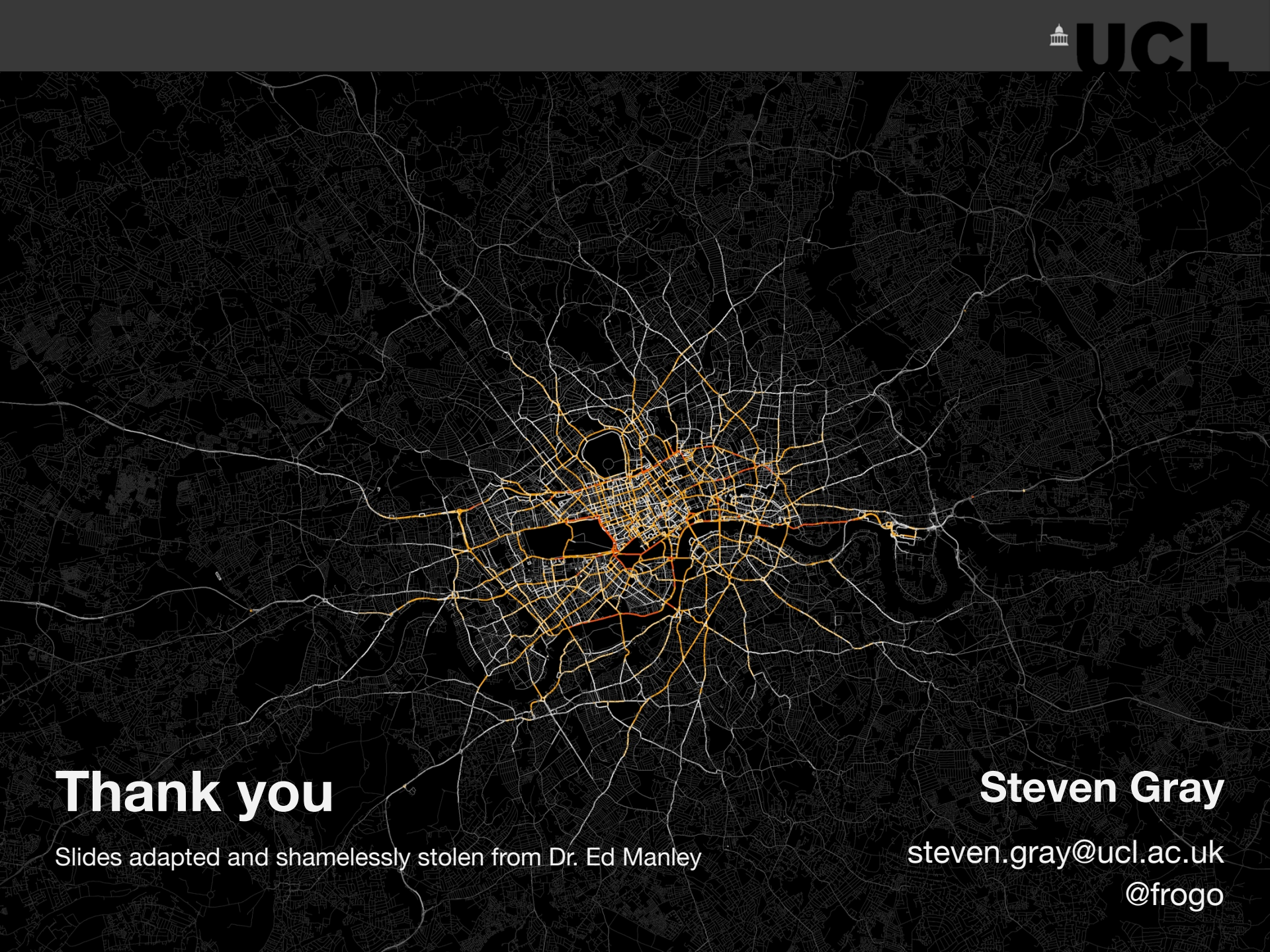
Steven Gray

steven.gray@ucl.ac.uk

@frogo

Workshop
Advanced SQL

- The workshop will focus on extending your SQL skills
- You'll learn how to query joined tables, use temporary tables, use indexes, and work with spatial and temporal data
- You'll continue using MySQL Workbench, connecting to the database you setup last week
- **Download the Worksheet from Moodle**
- **There is also another quiz on Moodle to complete**
- Don't forget Slack **AnswerBot** is there to help, otherwise come see us at the **Drop-In** on Friday



Thank you

Slides adapted and shamelessly stolen from Dr. Ed Manley

Steven Gray

steven.gray@ucl.ac.uk

@frogo