Geospatial Databases

Lecture:
Introduction to PostgreSQL

Nikolas Mitrou, Professor ECE, NTUA
Anastasios Zafeiropoulos, Postdoc Researcher ECE, NTUA

PostgreSQL Data Types

NUMERIC TYPES

- INTEGER, 4 bytes, range (-32768, 32767)
  - SMALLINT, BIGINT
- REAL, 4 bytes, 6 decimal digits precision
- SERIAL, 4 bytes, autoincrementing integer (1, 214743647)
  - SMALLSERIAL, BIGSERIAL

CHARACTER TYPES

- VARCHAR(n), variable length with limit
- CHAR(n), fixed length
- TEXT, variable unlimited length

PostgreSQL Data Types

DATE/TIME TYPES

- DATE, 4 bytes, date (no time)
- TIME, 8/12 bytes time of day (with time zone)
- TIMESTAMP, 8 bytes, date and time

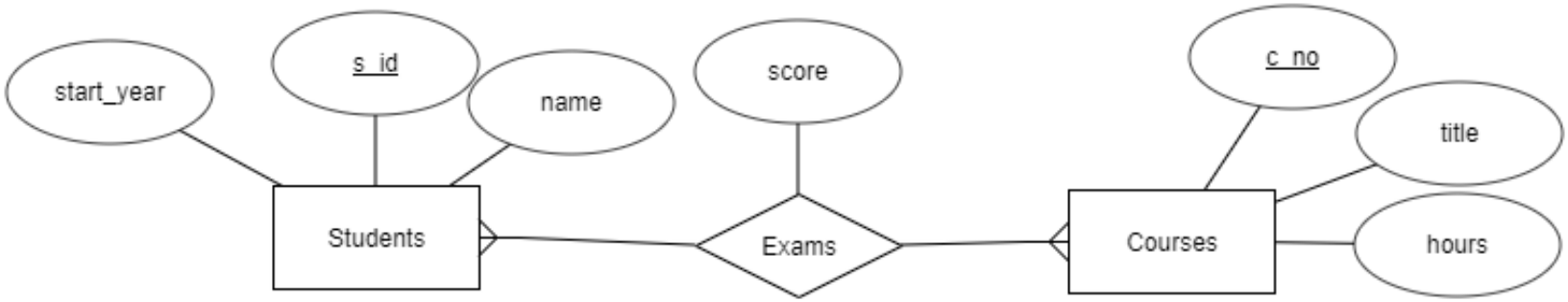
GEOMETRIC TYPES

- POINT, 16 bytes, point (x,y)
- LINE, 32 bytes, line {A,B,C}
- LINESEQ, 32 bytes, line segment ((x1,y1), (x2,y2))
- POLYGON, 40 + 16n bytes, [(x1,y1,…)]
- CIRCLE, 24 bytes, <(x,y), r> (center point and radius)

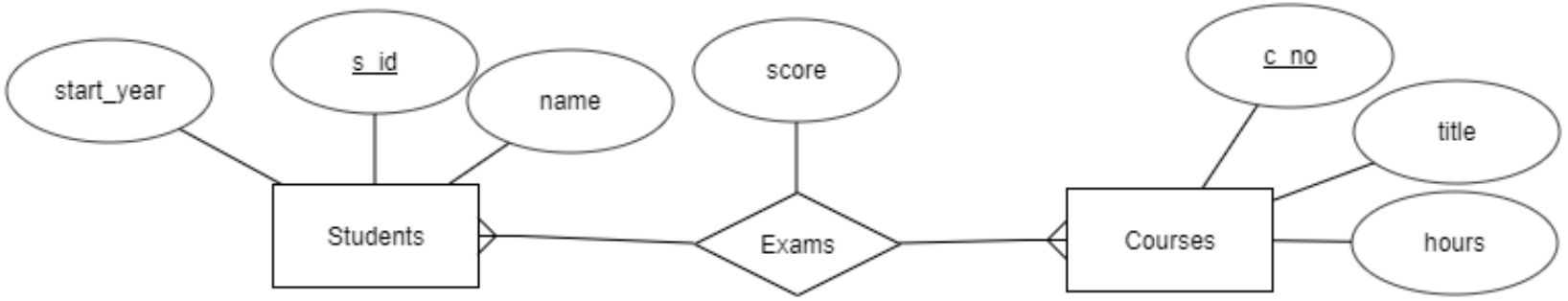
PostgreSQL – Basic Commands

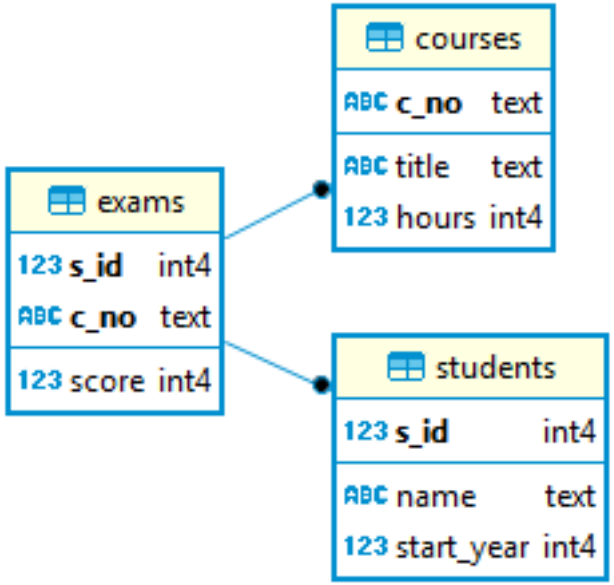
- PSQL commands (interactive terminal for Postgres)
  - \list or \l - list all databases
  - \l+ - list all databases with additional information
  - \d – list tables in a database
  - \d tablename - show table definition
  - \d+ tablename - show extended table information
  - \du – list all users
  - \q quit
  - \help
  - create database dbname; - create a database
  - drop database dbname; - delete a database
  - \c dbname; - connect to a database

PostgreSQL – Database Example



PostgreSQL – Database Example





PostgreSQL Basic Commands

- Create Table
  - CREATE TABLE table_name;
  - CREATE TABLE table_name (column_name TYPE column_constraint, table_constraint table_constraint) INHERITS existing_table_name;
- CONSTRAINTS
  - UNIQUE (column_list)
  - PRIMARY KEY (column_list)
  - CHECK (condition)
  - REFERENCES

```
CREATE TABLE courses (
  c_no text PRIMARY KEY,
  title text,
  hours integer);
```

```
CREATE TABLE students (
s_id integer PRIMARY KEY,
name text,
start_year integer);
```

PostgreSQL Basic Commands

- Insert Data
  - INSERT INTO table_name VALUES (values);
- Update Data
  - UPDATE table_name SET column1=value1, column2=value2 WHERE condition;
- Delete Data
  - DELETE FROM table_name WHERE condition;

```
INSERT INTO courses (c_no, title, hours)
VALUES ('CS301', 'Databases', 30),
('CS305', 'Networks', 60);
```

```
INSERT INTO students (s_id, name, start_year)
VALUES (1451, 'Anna', 2014),
(1432, 'Victor', 2014),
(1556, 'Nina', 2015);
```

```
UPDATE courses SET hours = hours*0.8 WHERE hours > 45;
```

PostgreSQL Basic Commands

```
CREATE TABLE exams(
s_id integer REFERENCES students(s_id) (ON DELETE
RESTRICT/CASCADE),
c_no text REFERENCES courses(c_no) (ON DELETE
RESTRICT/CASCADE),
score integer,
CONSTRAINT pk PRIMARY KEY(s_id, c_no));
```

```
INSERT INTO exams(s_id,
c_no, score)
VALUES (1451, 'CS301', 5),
(1556, 'CS301', 5),
(1451, 'CS305', 5),
(1432, 'CS305', 4);
```

```
UPDATE exams SET score  =
score*1.1 FROM courses
WHERE (courses.c_no =
exams.c_no AND
courses.hours > 10) ;
```

PostgreSQL Basic Commands

- Alter Table
  - ALTER TABLE table_name action;
  - ALTER TABLE table_name ADD COLUMN new_column_name TYPE;
  - ALTER TABLE table_name DROP COLUMN column_name;
  - ALTER TABLE table_name RENAME COLUMN column_name TO new_column_name;
  - ALTER TABLE table_name ALTER COLUMN column_name [SET NOT NULL| DROP NOT NULL]
  - ALTER TABLE table_name ADD CHECK expression;
- Delete Table
  - DROP TABLE table_name [CASCADE | RESTRICT]

ALTER TABLE students ADD COLUMN address text;

UPDATE students SET address = '……' WHERE name = '……';

ALTER TABLE students ALTER COLUMN name SET NOT NULL;

DROP TABLE students CASCADE;

ALTER TABLE exams ADD CONSTRAINT constraint s_id_fk FOREIGN KEY (s_id) REFERENCES students(s_id);

PostgreSQL Basic Commands

- Retrieve Data
  - SELECT * FROM table_name;
  - SELECT column1, column2 FROM table_name;
  - SELECT DISTINCT column1 FROM table_name;
  - SELECT column1 FROM table_name WHERE conditions;
  - Condition (=, <> (or !=), >, >=, <, <=, AND, OR)

```
SELECT * FROM courses;

SELECT title AS course_title, hours
FROM courses;

SELECT start_year FROM
students;

SELECT DISTINCT start_year
FROM students;

SELECT * FROM courses WHERE
hours > 45;
```

PostgreSQL Basic Commands

- Retrieve Data
  - SELECT column1, column_2 FROM table_name ORDER BY column1 ASC, column2 DESC;
  - SELECT * FROM table_name LIMIT n OFFSET m;
  - SELECT column1, aggregate_function(column2) FROM table_name GROUP BY column1;

SELECT * FROM exams ORDER BY score DESC;

SELECT * FROM exams ORDER BY score DESC LIMIT 2 OFFSET 1;

SELECT s_id, avg(score) FROM exams GROUP BY s_id;

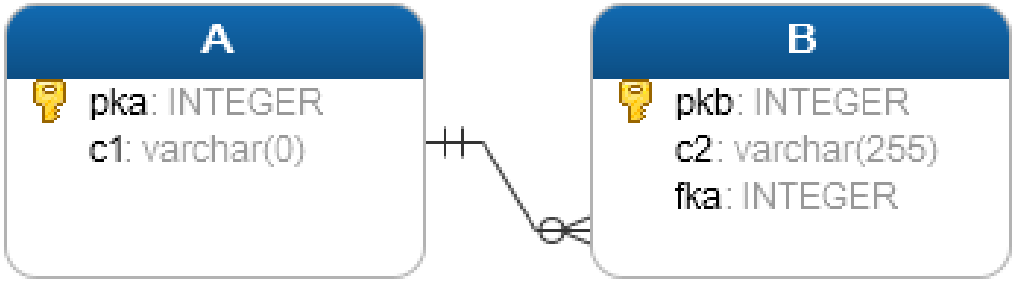
The GROUP BY clause divides the rows returned from the SELECT statement into groups. For each group, you can apply an aggregate function e.g., SUM to calculate the sum of items or COUNT to get the number of items in the groups.

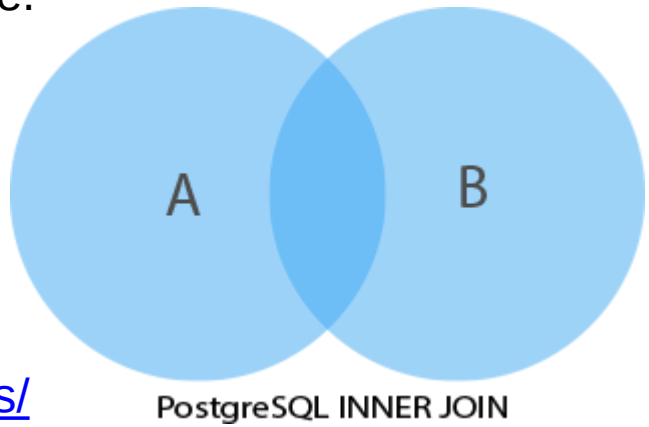
PostgreSQL Basic Commands

- INNER JOIN
  - SELECT A.pka, A.c1, B.pkb, B.c2 FROM A INNER JOIN B ON A .pka = B.fka;

SELECT students.s_id, name, start_year, score FROM students INNER JOIN exams ON students.s_id = exams.s_id;

If you want to select rows from the A table that have corresponding rows in the B table, you use the INNER JOIN clause.





PostgreSQL INNER JOIN

http://www.postgresqltutorial.com/postgresql-joins/

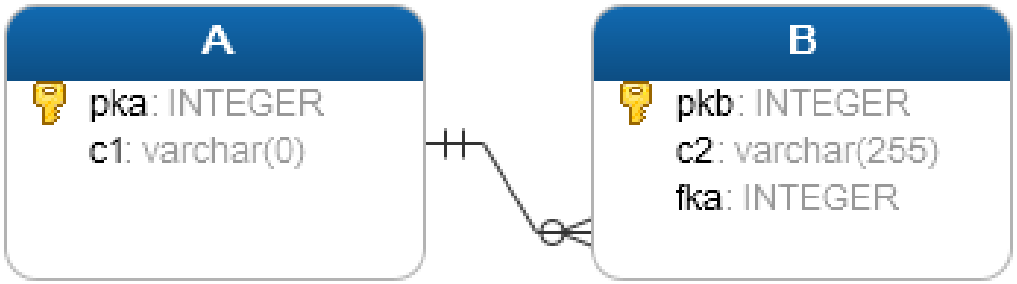
http://www.postgresqltutorial.com/postgresql-inner-join/

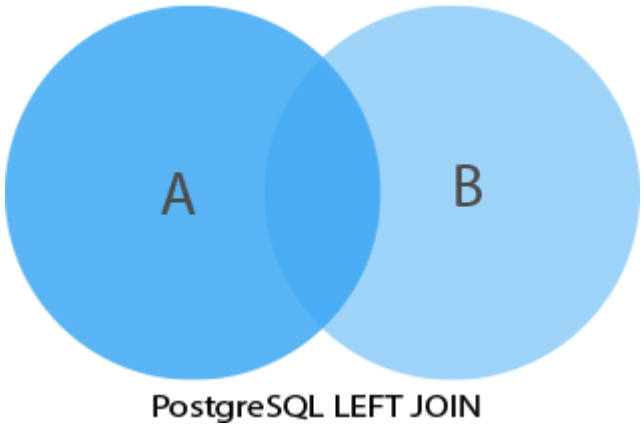
PostgreSQL Basic Commands

- LEFT JOIN
  - SELECT A.pka, A.c1, B.pkb, B.c2 FROM A LEFT JOIN B ON A .pka = B.fka;

```
SELECT students.name,
exams.score
FROM students
LEFT JOIN exams
ON students.s_id = exams.s_id
AND exams.c_no = 'CS305';
```

If you want to select rows from the A table which may or may not have corresponding rows in the B table, you use the LEFT JOIN clause.



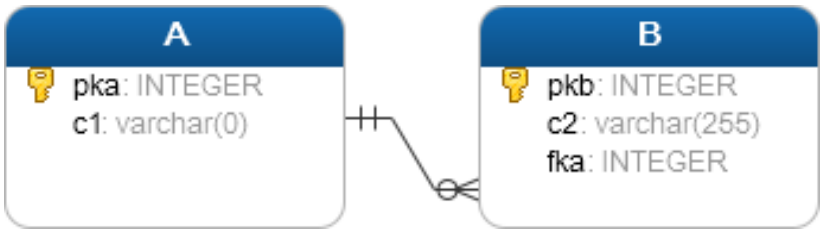


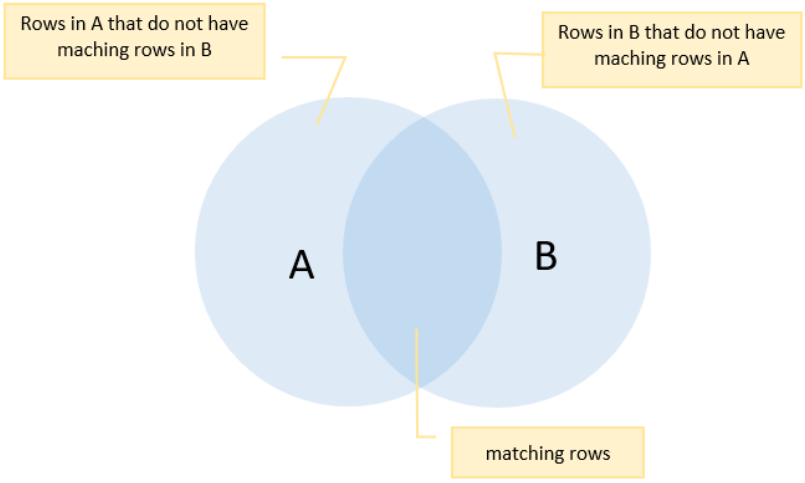
PostgreSQL Basic Commands

- FULL OUTER JOIN
  - SELECT * FROM A FULL [OUTER] JOIN B on A.id = B.id;

The full outer join combines the results of both left join and right join.

```
SELECT students.name,
exams.score
FROM students
FULL OUTER JOIN exams
ON students.s_id = exams.s_id
AND exams.c_no = 'CS305';
```





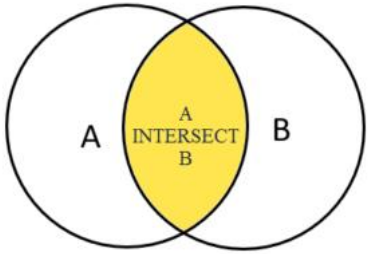
PostgreSQL Basic Commands

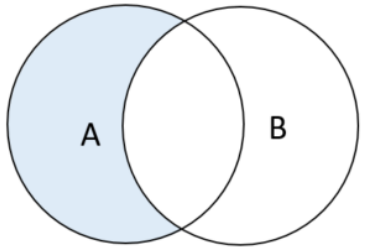
- **UNION**: combine result sets of two or more select statements
  - SELECT column1, column2 FROM table_name_1 UNION SELECT column1, column_2 FROM table_name_2;
  - Both queries must return the same number of columns.
  - The corresponding columns in the queries must have compatible data types.
  - ALL : does not remove duplicates
- **INTERSECT**: combine result sets of two or more select statements
  - SELECT column_list FROM A INTERSECT SELECT column_list FROM B;
- **EXCEPT**: returns distinct rows from the first (left) query that are not in the output of the second (right) query.
  - SELECT column_list FROM A WHERE condition_a EXCEPT SELECT column_list FROM B WHERE condition_b;
  - The number of columns and their orders must be the same in the two queries.
  - The data types of the respective columns must be compatible.

```
SELECT * FROM students UNION (ALL) SELECT * FROM students2;
```

```
SELECT name FROM students INTERSECT SELECT name FROM students2;
```

```
SELECT s_id FROM students EXCEPT SELECT s_id FROM exams WHERE exams.score < 5;
```





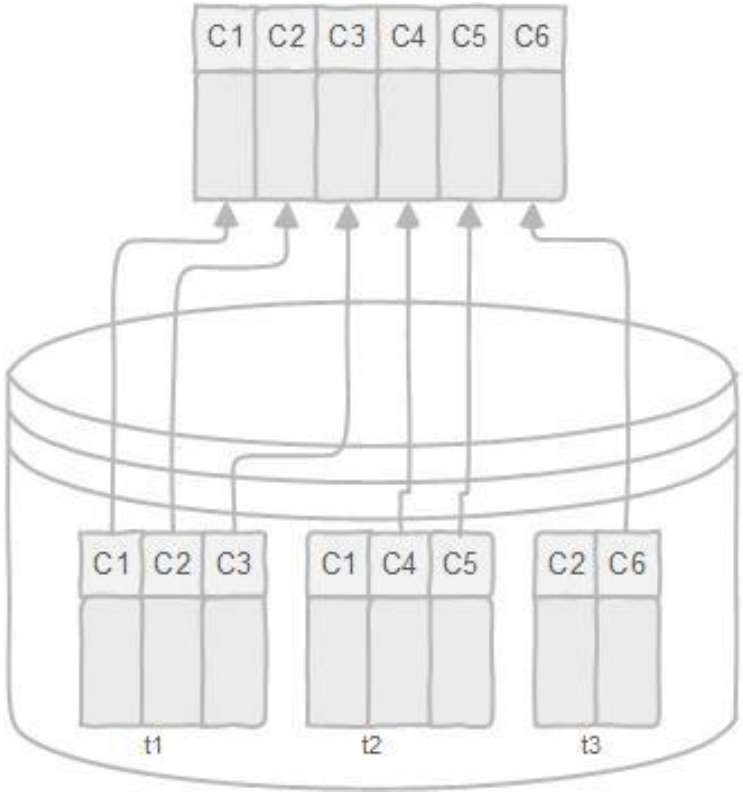
PostgreSQL Basic Commands

- SUBQUERY
  - Construct complex queries
  - PostgreSQL executes the query that contains a subquery in the following sequence:
    - First, executes the subquery.
    - Second, gets the result and passes it to the outer query.
    - Third, executes the outer query.

```
SELECT name, start_year
FROM students
WHERE s_id IN (SELECT s_id
FROM exams
WHERE c_no = 'CS305');
```

PostgreSQL Basic Commands

- VIEWS
  - CREATE VIEW view_name AS query;




```
CREATE VIEW results AS
SELECT s_id , c_no, name, title
FROM exams
JOIN students USING (s_id)
JOIN courses USING (c_no);
```

```
SELECT c1,c2,c3,c4,c5,c6
FROM t1
JOIN t2 USING (c1)
JOIN t3 USING (c2)
```

Data Input/Output

- By using the pgAdmin
  - Data Input/Output from csv or txt files
  - Database backup/restore

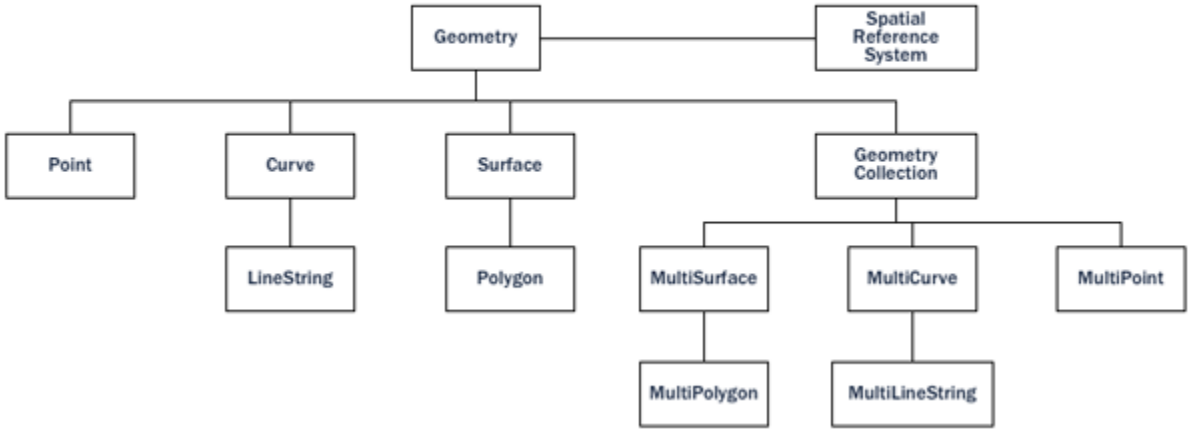
Extract SQL from a database via Pgadmin
Select Database --> Backup --> Provide filename
Select Format --> plain
Select Encoding --> UTF8
Backup

PostgreSQL and Postgis

- POSTGIS data: representation of geographical characteristics/features
- Spatial data types: point, line, polygon;
- PostGIS makes the PostgreSQL Database Management System able to manage spatial data and functions, supporting: spatial types, indexes, functions.

CREATE EXTENSION postgis;
//load the PostGIS spatial extension
SELECT postgis_full_version();
//confirm that PostGIS is installed by running a PostGIS function

Geometry Hierarchy



PostgreSQL and Postgis

```
CREATE TABLE cities ( id int4 primary key, name varchar(50),
geom geometry(POINT,4326) );
SELECT * from cities;
```

```
INSERT INTO cities (id, geom, name) VALUES
(1,ST_GeomFromText('POINT(-0.1257 51.508)',4326),'London,
England');
INSERT INTO cities (id, geom, name) VALUES
(2,ST_GeomFromText('POINT(-81.233 42.983)',4326),'London,
Ontario');
INSERT INTO cities (id, geom, name) VALUES
(3,ST_GeomFromText('POINT(27.91162491 -
33.01529)',4326),'East London,SA');
```

PostgreSQL and Postgis

```
SELECT * FROM cities;

SELECT id, ST_AsText(geom), ST_AsEwkt(geom), ST_X(geom),
ST_Y(geom) FROM cities;

SELECT p1.name, p2.name,
ST_DistanceSphere(p1.geom,p2.geom) FROM cities AS p1,
cities AS p2 WHERE p1.id > p2.id;
```

http://www.postgis.us/downloads/postgis21_cheatsheet.pdf