Spatial Databases I

Advanced Topics in Databases

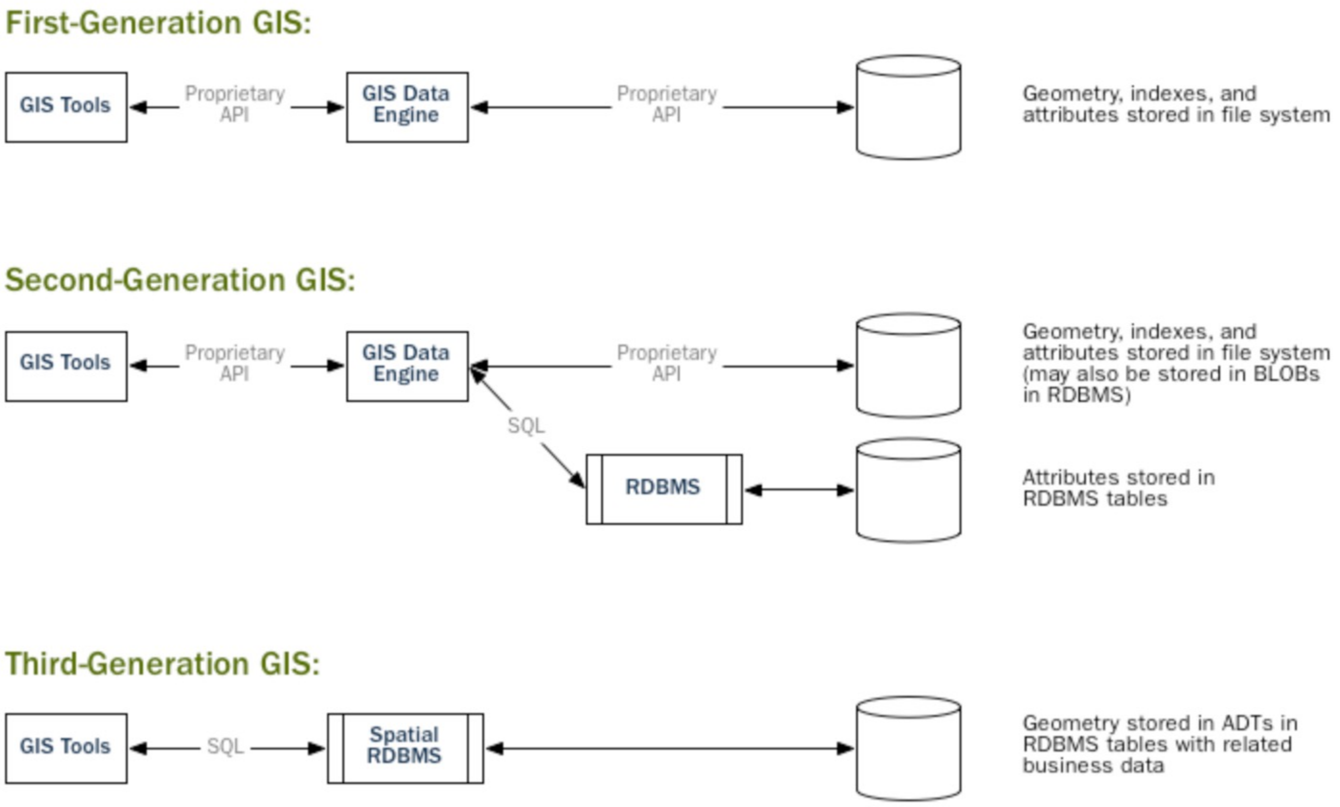
What is a Spatial Database?

- **Spatial databases store and manipulate spatial objects like any other object in the database.**
- There are three aspects that associate *spatial* data with a database – data types, indexes, and functions.
  1. **Spatial data types** refer to shapes such as point, line, and polygon;
  2. Multi-dimensional **spatial indexing** is used for efficient processing of spatial operations;
  3. **Spatial functions**, posed in SQL, are for querying of spatial properties and relationships.
- Combined, spatial data types, indexes, and functions provide a flexible structure for optimized performance and analysis.

In the Beginning

- In legacy first-generation GIS implementations, all spatial data is stored in flat files and special GIS software is required to interpret and manipulate the data. These first-generation management systems are designed to meet the needs of users where all required data is within the user's organizational domain. They are proprietary, self-contained systems specifically built for handling spatial data.
- Second-generation spatial systems store some data in relational databases (usually the "attribute" or non-spatial parts) but still lack the flexibility afforded with direct integration.
- **True spatial databases were born when people started to treat spatial features as first class database objects.**
- Spatial databases fully integrate spatial data with a relational database. The system orientation changes from GIS-centric to database-centric.

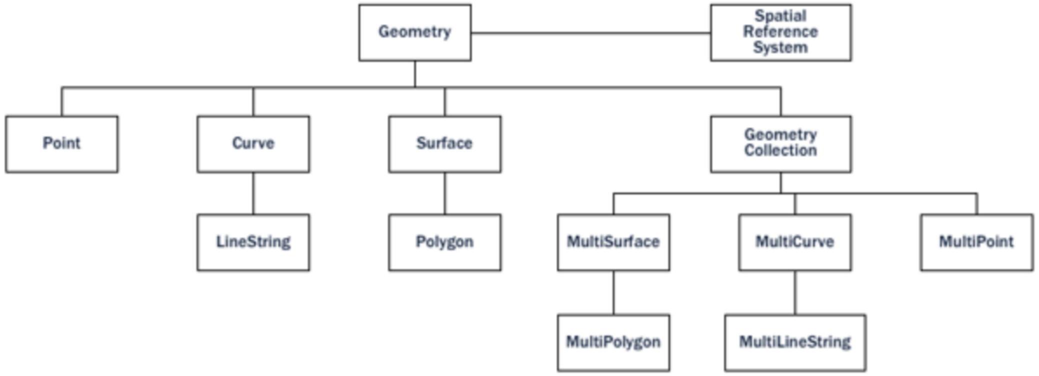
Evolution of GIS Architectures


First-Generation GIS:
GIS Tools
Proprietary API
GIS Data Engine
Proprietary API
Geometry, indexes, and attributes stored in file system
Second-Generation GIS:
GIS Tools
Proprietary API
GIS Data Engine
Proprietary API
Geometry, indexes, and attributes stored in file system (may also be stored in BLOBs in RDBMS)
SQL
RDBMS
Attributes stored in RDBMS tables
Third-Generation GIS:
GIS Tools
SQL
Spatial RDBMS
Geometry stored in ADTs in RDBMS tables with related business data

Spatial Data Types

- An ordinary database has strings, numbers, and dates. A spatial database adds additional (spatial) types for representing **geographic features**.
- These spatial data types abstract and encapsulate spatial structures such as boundary and dimension. In many respects, spatial data types can be understood simply as shapes.
- Spatial data types are organized in a type hierarchy. Each sub-type inherits the structure (attributes) and the behavior (methods or functions) of its super-type.

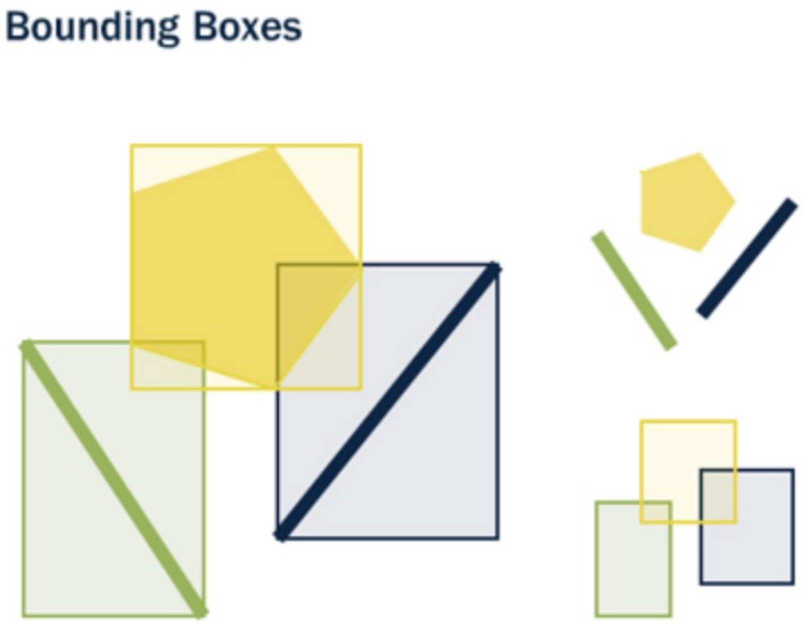
Geometry Hierarchy



Spatial Indexes and Bounding Boxes

- An ordinary database provides indexes to allow for fast and random access to subsets of data. Indexing for standard types (numbers, strings, dates) is usually done with B-tree indexes.
- A B-tree partitions the data using the natural sort order to put the data into a hierarchical tree. The natural sort order of numbers, strings, and dates is simple to determine – every value is less than, greater than or equal to every other value.
- But because polygons can overlap, can be contained in one another, and are arrayed in a two-dimensional (or more) space, a B-tree cannot be used to efficiently index them.
- Real spatial databases provide a “spatial index” that instead answers the question “which objects are within this particular bounding box?”.
- A bounding box is the smallest rectangle – parallel to the coordinate axes – capable of containing a given feature.

Spatial Indexes and Bounding Boxes



Bounding Boxes

Spatial Indexes and Bounding Boxes

- Bounding boxes are used because answering the question “is A inside B?” is very computationally intensive for polygons but very fast in the case of rectangles. Even the most complex polygons and linestrings can be represented by a simple bounding box.
- Indexes have to perform quickly in order to be useful. So instead of providing exact results, as B-trees do, spatial indexes provide approximate results. The question “what lines are inside this polygon?” will be instead interpreted by a spatial index as “what lines have bounding boxes that are contained inside this polygon’s bounding box?”
- The actual spatial indexes implemented by various databases vary widely. The most common implementations are the R-Tree and Quadtree (used in PostGIS), but there are also grid-based indexes and GeoHash indexes implemented in other spatial databases.

Spatial Functions

- For manipulating data during a query, an ordinary database provides functions such as concatenating strings, performing hash operations on strings, doing mathematics on numbers, and extracting information from dates.
- A spatial database provides a complete set of functions for analyzing geometric components, determining spatial relationships, and manipulating geometries.
- These spatial functions serve as the building block for any spatial project.

Spatial Functions

- The majority of all spatial functions can be grouped into one of the following five categories:
  - **Conversion**: Functions that convert between geometries and external data formats.
  - **Management**: Functions that manage information about spatial tables and PostGIS administration.
  - **Retrieval**: Functions that retrieve properties and measurements of a Geometry.
  - **Comparison**: Functions that compare two geometries with respect to their spatial relation.
  - **Generation**: Functions that generate new geometries from others.

PostGIS

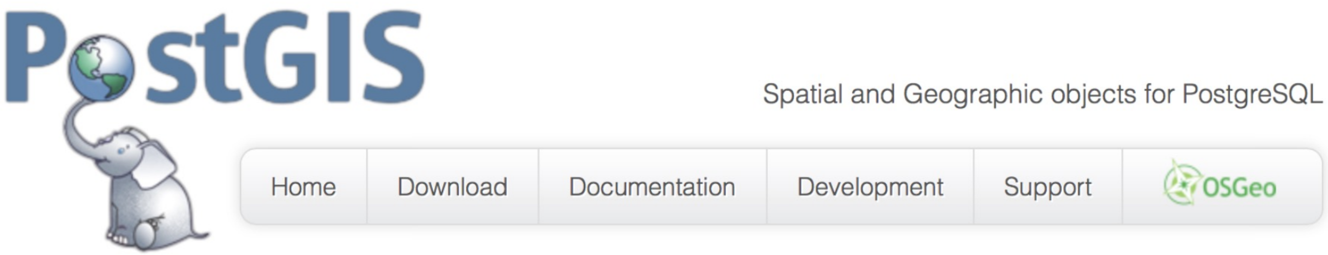
- PostGIS turns the PostgreSQL Database Management System into a spatial database by adding support for the three features:

  - spatial types,
  - spatial indexes,
  - and spatial functions.

- Because it is built on PostgreSQL, PostGIS automatically inherits important “enterprise” features as well as open standards for implementation.

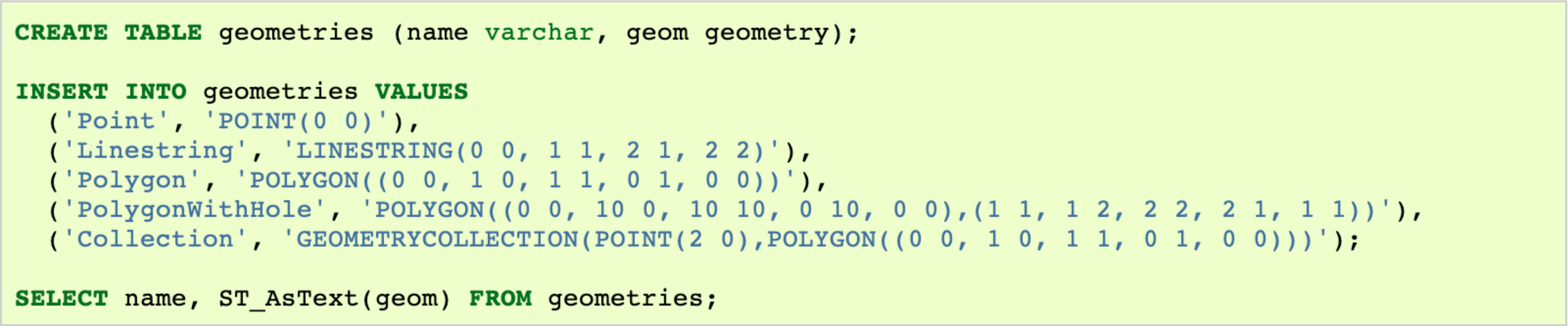
PostGIS

- Installation instructions on: www.postgis.net



Spatial and Geographic objects for PostgreSQL

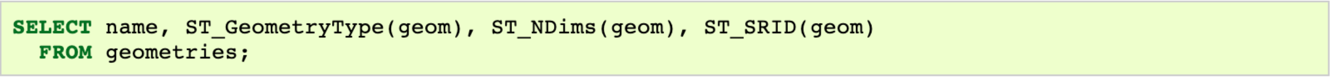
Creating simple spatial data



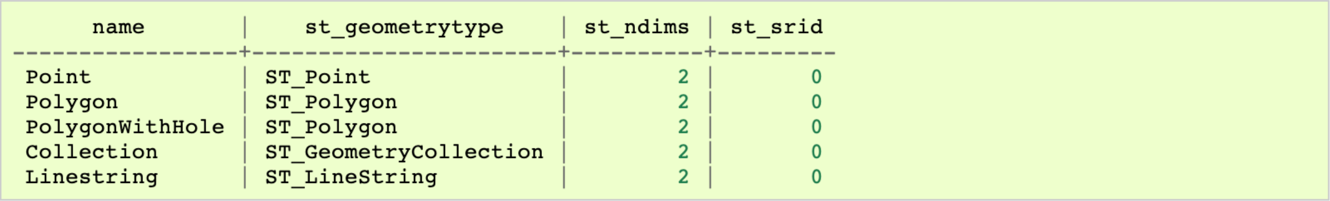
```sql
CREATE TABLE geometries (name varchar, geom geometry);

INSERT INTO geometries VALUES
  ('Point', 'POINT(0 0)'),
  ('Linestring', 'LINESTRING(0 0, 1 1, 2 1, 2 2)'),
  ('Polygon', 'POLYGON((0 0, 1 0, 1 1, 0 1, 0 0))'),
  ('PolygonWithHole', 'POLYGON((0 0, 10 0, 10 10, 0 10, 0 0),(1 1, 1 2, 2 2, 2 1, 1 1))'),
  ('Collection', 'GEOMETRYCOLLECTION(POINT(2 0),POLYGON((0 0, 1 0, 1 1, 0 1, 0 0)))');

SELECT name, ST_AsText(geom) FROM geometries;
```

Representing spatial objects

- Our example table contains a mixture of different geometry types.
- We can collect general information about each object using functions that read the geometry metadata.
  - ST_GeometryType(geometry) returns the type of the geometry
  - ST_NDims(geometry) returns the number of dimensions of the geometry
  - ST_SRID(geometry) returns the spatial reference identifier number of the geometry



```
SELECT name, ST_GeometryType(geom), ST_NDims(geom), ST_SRID(geom)
  FROM geometries;
```

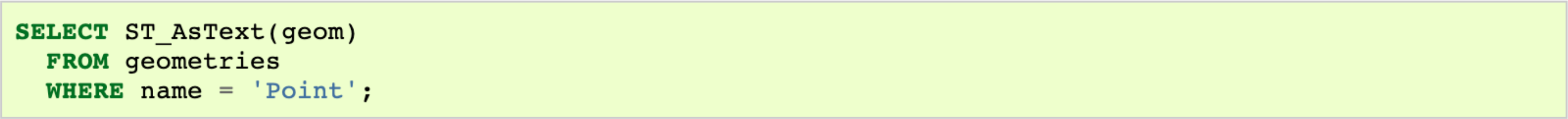


```
      name       |    st_geometrytype    | st_ndims | st_srid
-----------------+-----------------------+----------+---------
 Point           | ST_Point              |        2 |       0
 Polygon         | ST_Polygon            |        2 |       0
 PolygonWithHole | ST_Polygon            |        2 |       0
 Collection      | ST_GeometryCollection |        2 |       0
 Linestring      | ST_LineString         |        2 |       0
```

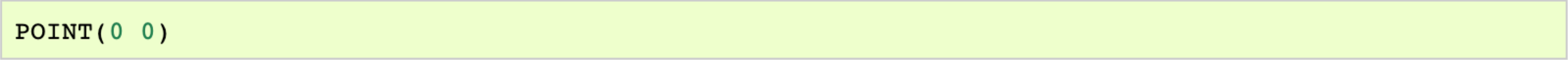
Points

Point
Multipoint with 4 parts

- A spatial point represents a single location on the Earth.
- This point is represented by a single coordinate (including either 2-, 3- or 4-dimensions).
- Points are used to represent objects when the exact details, such as shape and size, are not important at the target scale.
- For example, cities on a map of the world can be described as points, while a map of a single state might represent cities as polygons.



```
SELECT ST_AsText(geom)
  FROM geometries
  WHERE name = 'Point';
```

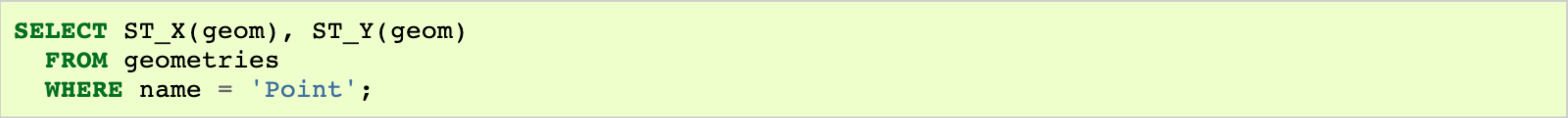


```
POINT(0 0)
```

Points

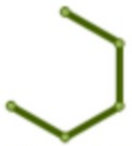
Point
Multipoint with 4 parts

- Some of the specific spatial functions for working with points are:
  - ST_X(geometry) returns the X ordinate
  - ST_Y(geometry) returns the Y ordinate
- So, we can read the ordinates from a point like this:

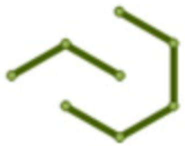


```
SELECT ST_X(geom), ST_Y(geom)
  FROM geometries
  WHERE name = 'Point';
```

Linestrings

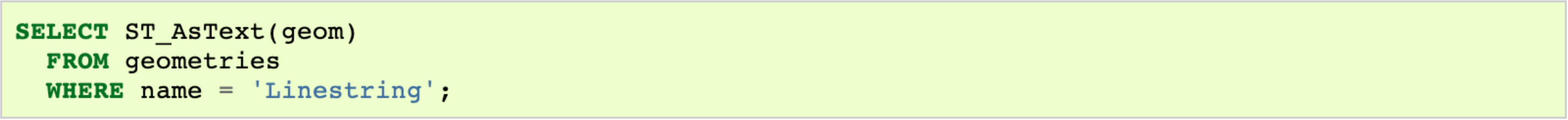


Simple non-closed linestring

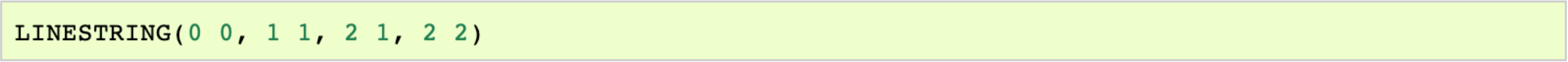


Simple multilinestring defined by 4 endpoints of 2 elements

- A linestring is a path between locations. It takes the form of an ordered series of two or more points.
- Roads and rivers are typically represented as linestrings.
- A linestring is said to be closed if it starts and ends on the same point.
- It is said to be simple if it does not cross or touch itself (except at its endpoints if it is closed).
- A linestring can be both closed and simple.
- The following SQL query will return the geometry associated with one linestring (in the ST_AsText column).

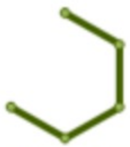


```
SELECT ST_AsText(geom)
  FROM geometries
  WHERE name = 'Linestring';
```

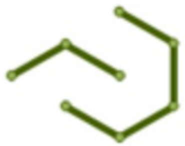


```
LINESTRING(0 0, 1 1, 2 1, 2 2)
```

Linestrings



Simple non-closed linestring



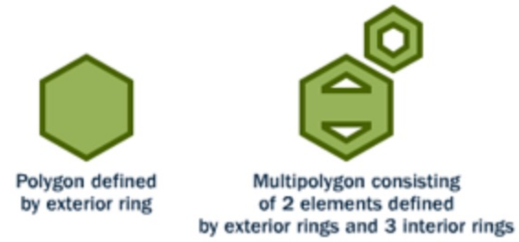
Simple multilinestring defined by 4 endpoints of 2 elements

- Some of the specific spatial functions for working with linestrings are:
  - ST_Length(geometry) returns the length of the linestring
  - ST_StartPoint(geometry) returns the first coordinate as a point
  - ST_EndPoint(geometry) returns the last coordinate as a point
  - ST_NPoints(geometry) returns the number of coordinates in the linestring
- So, the length of our linestring is:

```
SELECT ST_Length(geom)
  FROM geometries
  WHERE name = 'Linestring';
```

```
3.41421356237309
```

Polygons



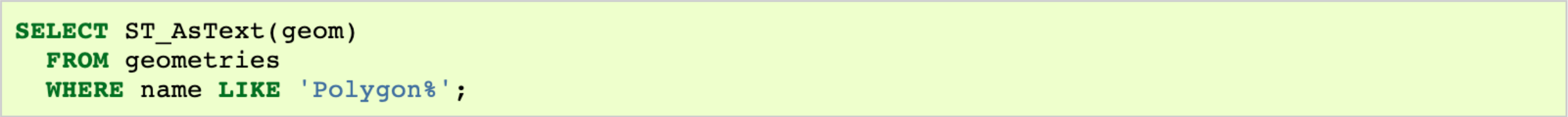
- A polygon is a representation of an area.
- The outer boundary of the polygon is represented by a ring.
- This ring is a linestring that is both closed and simple as defined previously.
- Holes within the polygon are also represented by rings.
- Polygons are used to represent objects whose size and shape are important.
- City limits, parks, building footprints or bodies of water are all commonly represented as polygons when the scale is sufficiently high to see their area.
- Roads and rivers can sometimes be represented as polygons.

Polygons

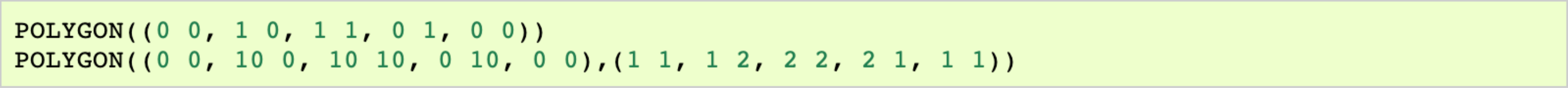
Polygon defined by exterior ring

Multipolygon consisting of 2 elements defined by exterior rings and 3 interior rings

- The following SQL query will return the geometry associated with one polygon (in the ST_AsText column):

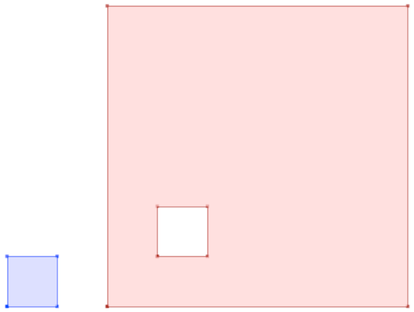


```sql
SELECT ST_AsText(geom)
  FROM geometries
  WHERE name LIKE 'Polygon%';
```

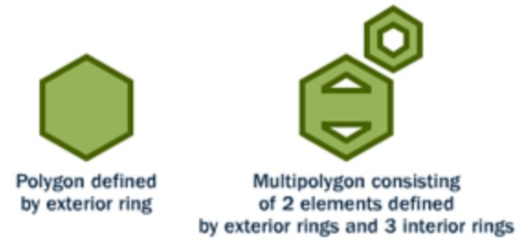


```
POLYGON((0 0, 1 0, 1 1, 0 1, 0 0))
POLYGON((0 0, 10 0, 10 10, 0 10, 0 0),(1 1, 1 2, 2 2, 2 1, 1 1))
```

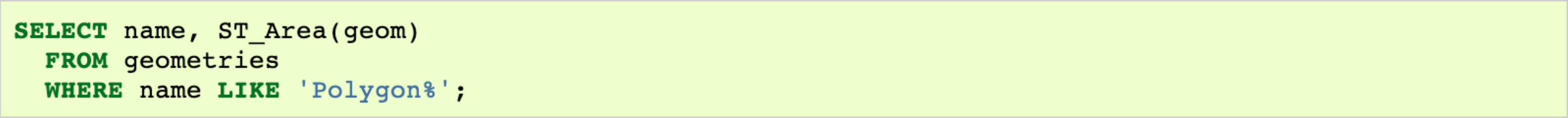
- The first polygon has only one ring.
- The second one has an interior "hole".



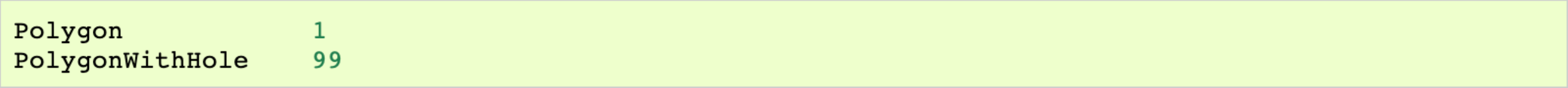
Polygons



- Some of the specific spatial functions for working with polygons are:
  - ST_Area(geometry) returns the area of the polygons
  - ST_NRings(geometry) returns the number of rings (usually 1, more if there are holes)
  - ST_ExteriorRing(geometry) returns the outer ring as a linestring
  - ST_InteriorRingN(geometry,n) returns a specified interior ring as a linestring
  - ST_Perimeter(geometry) returns the length of all the rings
- We can calculate the area of our polygons using the area function:



```
SELECT name, ST_Area(geom)
  FROM geometries
  WHERE name LIKE 'Polygon%';
```

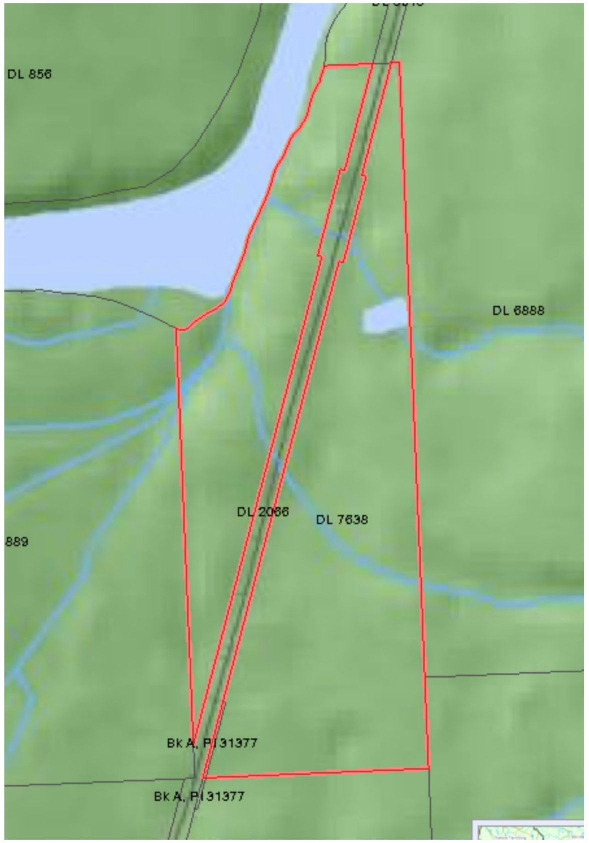


```
Polygon                1
PolygonWithHole        99
```

- Note that the polygon with a hole has an area that is the area of the outer shell (a 10x10 square) minus the area of the hole (a 1x1 square).

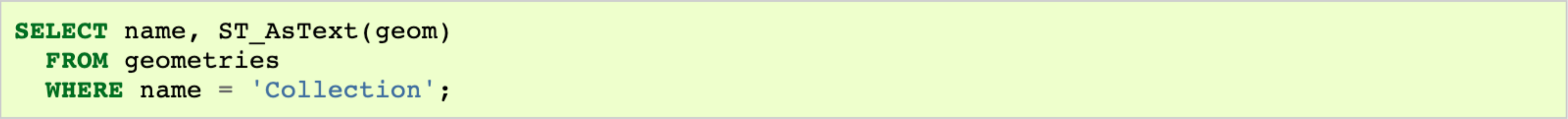
Collections

- There are four collection types, which group multiple simple geometries into sets:
  - **MultiPoint**, a collection of points
  - **MultiLineString**, a collection of linestrings
  - **MultiPolygon**, a collection of polygons
  - **GeometryCollection**, a heterogeneous collection of any geometry (including other collections)
- Collections are another concept that shows up in GIS software more than in generic graphics software.
- They are useful for directly modeling real world objects as spatial objects.
- For example, how to model a lot that is split by a right-of-way? As a MultiPolygon, with a part on either side of the right-of-way.

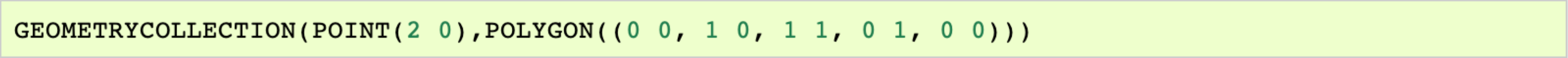


Collections

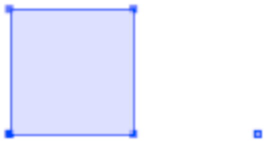
- Our example collection contains a polygon and a point:



```sql
SELECT name, ST_AsText(geom)
  FROM geometries
  WHERE name = 'Collection';
```



```
GEOMETRYCOLLECTION(POINT(2 0),POLYGON((0 0, 1 0, 1 1, 0 1, 0 0)))
```



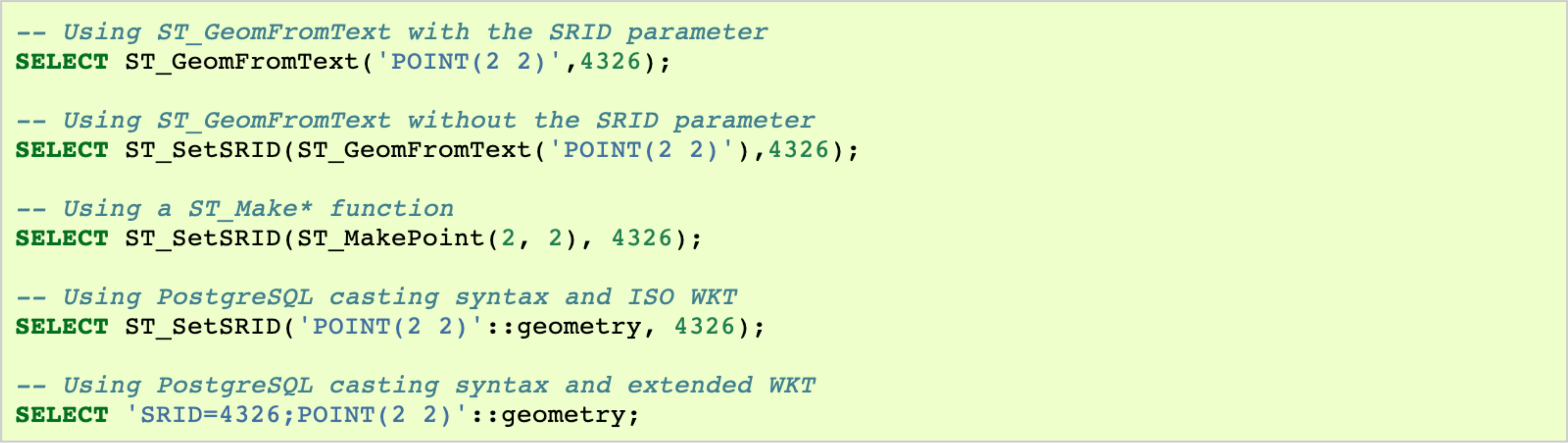
- Some of the specific spatial functions for working with collections are:
  - ST_NumGeometries(geometry) returns the number of parts in the collection
  - ST_GeometryN(geometry,n) returns the specified part
  - ST_Area(geometry) returns the total area of all polygonal parts
  - ST_Length(geometry) returns the total length of all linear parts

Geometry Input and Output

- Within the database, geometries are stored on disk in a format only used by the PostGIS program.
- In order for external programs to insert and retrieve useful geometries, they need to be converted into a format that other applications can understand.
- PostGIS supports emitting and consuming geometries in a large number of formats:
  - Well-known text (WKT)
    - ST_GeomFromText(text, srid) returns geometry
    - ST_AsText(geometry) returns text
    - ST_AsEWKT(geometry) returns text
  - Well-known binary (WKB)
    - ST_GeomFromWKB(bytea) returns geometry
    - ST_AsBinary(geometry) returns bytea
    - ST_AsEWKB(geometry) returns bytea
  - Geographic Mark-up Language (GML)
    - ST_GeomFromGML(text) returns geometry
    - ST_AsGML(geometry) returns text
  - Keyhole Mark-up Language (KML)
    - ST_GeomFromKML(text) returns geometry
    - ST_AsKML(geometry) returns text
  - GeoJSON
    - ST_AsGeoJSON(geometry) returns text
  - Scalable Vector Graphics (SVG)
    - ST_AsSVG(geometry) returns text

Geometry Input and Output

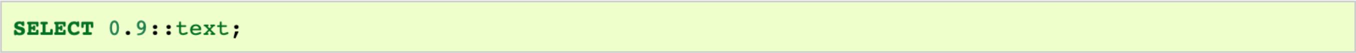
- In addition to the ST_GeometryFromText function, there are many other ways to create geometries from well-known text or similar formatted inputs:



```
-- Using ST_GeomFromText with the SRID parameter
SELECT ST_GeomFromText('POINT(2 2)',4326);

-- Using ST_GeomFromText without the SRID parameter
SELECT ST_SetSRID(ST_GeomFromText('POINT(2 2)'),4326);

-- Using a ST_Make* function
SELECT ST_SetSRID(ST_MakePoint(2, 2), 4326);

-- Using PostgreSQL casting syntax and ISO WKT
SELECT ST_SetSRID('POINT(2 2)'::geometry, 4326);

-- Using PostgreSQL casting syntax and extended WKT
SELECT 'SRID=4326;POINT(2 2)'::geometry;
```

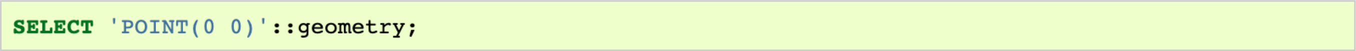
Casting from Text

- The WKT strings we've see so far have been of type 'text' and we have been converting them to type 'geometry' using PostGIS functions like ST_GeomFromText().
- PostgreSQL includes a short form syntax that allows data to be converted from one type to another, the casting syntax, oldata::newtype. So for example, this SQL converts a double into a text string.



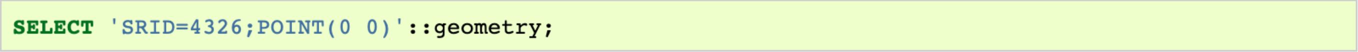
```
SELECT 0.9::text;
```

- Less trivially, this SQL converts a WKT string into a geometry:



```
SELECT 'POINT(0 0)'::geometry;
```

- One thing to note about using casting to create geometries: unless you specify the SRID, you will get a geometry with an unknown SRID. You can specify the SRID using the "extended" well-known text form, which includes an SRID block at the front:



```
SELECT 'SRID=4326;POINT(0 0)'::geometry;
```

Function List

- **ST_Area**: Returns the area of the surface if it is a polygon or multi-polygon. For “geometry” type area is in SRID units. For “geography” area is in square meters.
- **ST_AsText**: Returns the Well-Known Text (WKT) representation of the geometry/geography without SRID metadata.
- **ST_AsBinary**: Returns the Well-Known Binary (WKB) representation of the geometry/geography without SRID meta data.
- **ST_EndPoint**: Returns the last point of a LINESTRING geometry as a POINT.
- **ST_AsEWKB**: Returns the Well-Known Binary (WKB) representation of the geometry with SRID meta data.
- **ST_AsEWKT**: Returns the Well-Known Text (WKT) representation of the geometry with SRID meta data.
- **ST_AsGeoJSON**: Returns the geometry as a GeoJSON element.
- **ST_AsGML**: Returns the geometry as a GML version 2 or 3 element.
- **ST_AsKML**: Returns the geometry as a KML element. Several variants. Default version=2, default precision=15.
- **ST_AsSVG**: Returns a Geometry in SVG path data given a geometry or geography object.

Function List

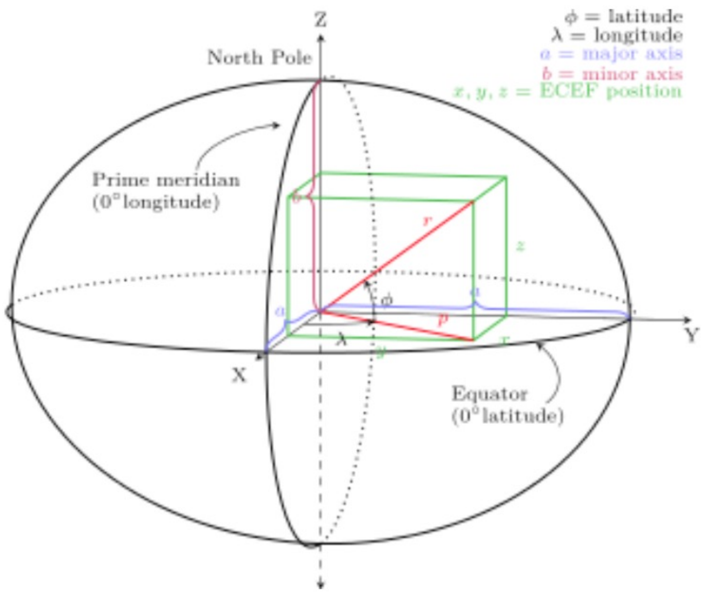
- **ST_ExteriorRing**: Returns a line string representing the exterior ring of the POLYGON geometry. Return NULL if the geometry is not a polygon. Will not work with MULTIPOLYGON
- **ST_GeometryN**: Returns the 1-based Nth geometry if the geometry is a GEOMETRYCOLLECTION, MULTIPOINT, MULTILINESTRING, MULTICURVE or MULTIPOLYGON. Otherwise, return NULL.
- **ST_GeomFromGML**: Takes as input GML representation of geometry and outputs a PostGIS geometry object.
- **ST_GeomFromKML**: Takes as input KML representation of geometry and outputs a PostGIS geometry object
- **ST_GeomFromText**: Returns a specified ST_Geometry value from Well-Known Text representation (WKT).
- **ST_GeomFromWKB**: Creates a geometry instance from a Well-Known Binary geometry representation (WKB) and optional SRID.
- **ST_GeometryType**: Returns the geometry type of the ST_Geometry value.
- **ST_InteriorRingN**: Returns the Nth interior linestring ring of the polygon geometry. Return NULL if the geometry is not a polygon or the given N is out of range.
- **ST_Length**: Returns the 2d length of the geometry if it is a linestring or multilinestring. geometry are in units of spatial reference and geography are in meters (default spheroid)

Function List

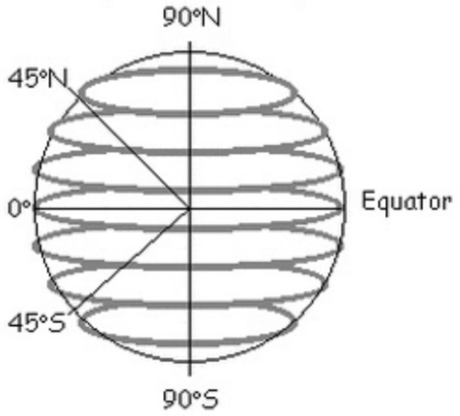
- **ST_NDims**: Returns coordinate dimension of the geometry as a small int. Values are: 2,3 or 4.
- **ST_NPoints**: Returns the number of points (vertexes) in a geometry.
- **ST_NRings**: If the geometry is a polygon or multi-polygon returns the number of rings.
- **ST_NumGeometries**: If geometry is a GEOMETRYCOLLECTION (or MULTI*) returns the number of geometries, otherwise return NULL.
- **ST_Perimeter**: Returns the length measurement of the boundary of an ST_Surface or ST_MultiSurface value. (Polygon, Multipolygon)
- **ST_SRID**: Returns the spatial reference identifier for the ST_Geometry as defined in spatial_ref_sys table.
- **ST_StartPoint**: Returns the first point of a LINESTRING geometry as a POINT.
- **ST_X**: Returns the X coordinate of the point, or NULL if not available. Input must be a point.
- **ST_Y**: Returns the Y coordinate of the point, or NULL if not available. Input must be a point.

Geographic coordinate system

Latitude, Longitude, Altitude

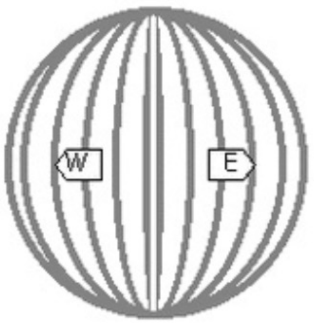


Latitude
(North/South)



Latitude varies from 0° at the equator to 90° North and South at the poles

Longitude
(West/East)

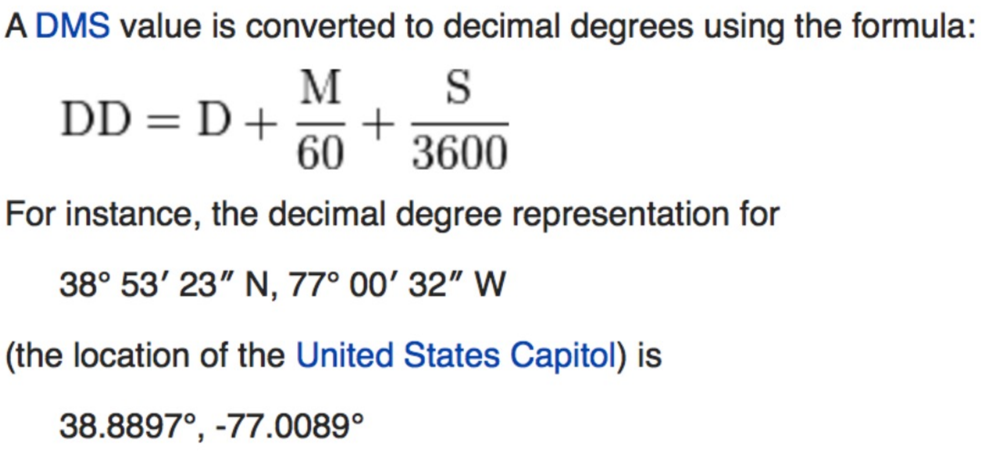


Longitude varies from 0° at Greenwich to 180° East and West

GPS in Decimal Degrees

- WGS 84 Spheroid

# name	longitude	latitude
Shanghai	121.47	31.23
Bombay	72.82	18.96
Karachi	67.01	24.86
Buenos Aires	-58.37	-34.61
Delhi	77.21	28.67
Istanbul	29	41.1
Manila	120.97	14.62
Sao Paulo	-46.63	-23.53



A DMS value is converted to decimal degrees using the formula:

$$\mathrm{DD} = \mathrm{D} + \frac{\mathrm{M}}{60} + \frac{\mathrm{S}}{3600}$$

For instance, the decimal degree representation for

38° 53′ 23″ N, 77° 00′ 32″ W

(the location of the United States Capitol) is

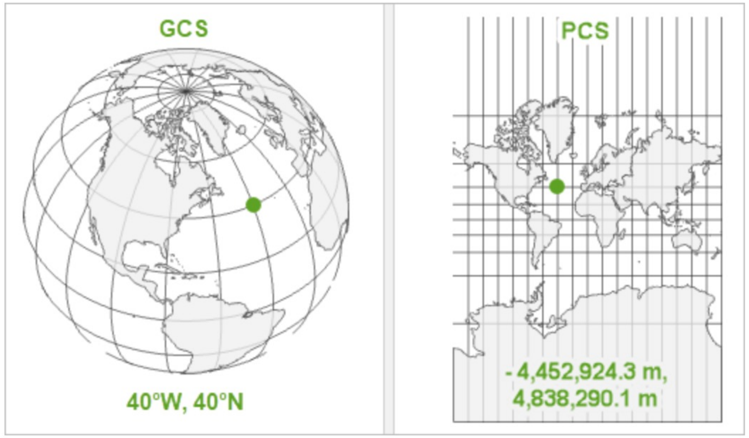
38.8897°, -77.0089°

Map Projection

- To establish the position of a geographic location on a map, a map projection is used to convert geodetic coordinates to plane coordinates on a map;
- It projects the datum ellipsoidal coordinates and height onto a flat surface of a map. The datum, along with a map projection applied to a grid of reference locations, establishes a grid system for plotting locations.
- Common map projections in current use include the Universal Transverse Mercator (UTM), the Military Grid Reference System (MGRS), the United States National Grid (USNG), the Global Area Reference System (GARS) and the World Geographic Reference System (GEOREF).
- Coordinates on a map are usually in terms northing N and easting E offsets relative to a specified origin.

Geographic coordinate system vs. projected coordinate system

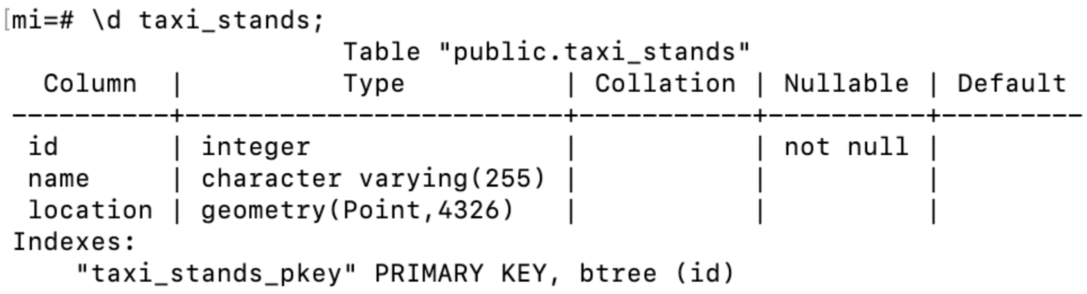
- What is the difference between a geographic coordinate system (GCS) and a projected coordinate system (PCS)?
  - A GCS defines where the data is located on the earth's surface.
  - A PCS tells the data how to draw on a flat surface, like on a paper map or a computer screen.
  - A GCS is round, and so records locations in angular units (usually degrees).
  - A PCS is flat, so it records locations in linear units (usually meters).



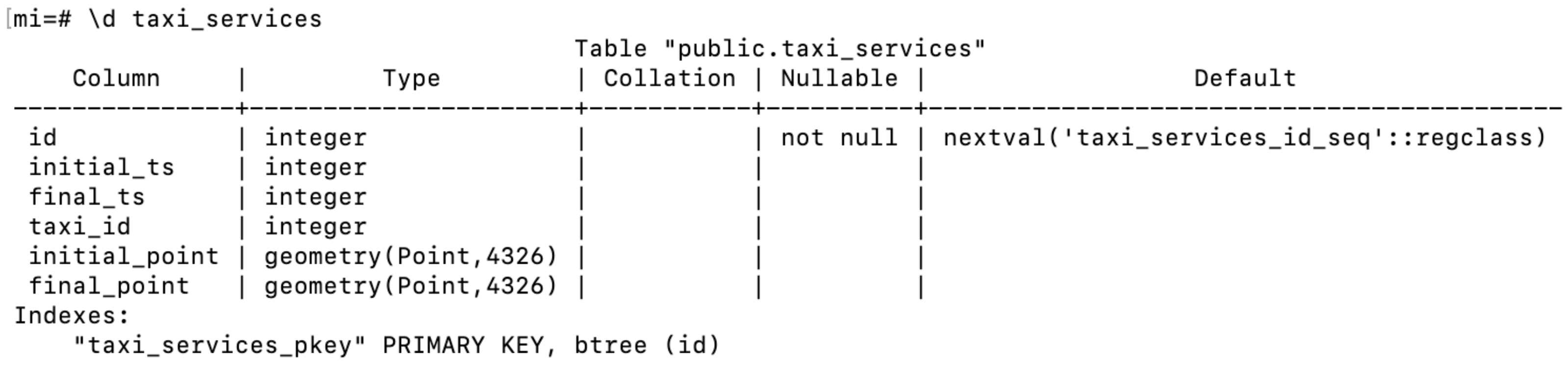
The Taxi Dataset

- A table taxi_stands with the location of taxi stands in Porto.
- A table taxi_services with start and end data about taxi_services in Porto.
- location, initial_point and final_point are in WGS84 coordinate system (SRID=4326).
- To add a projected coordinate system optimized to Portugal (SRID=3763) use:
  - alter table taxi_stands add proj_location geometry(Point,3763);
  - update taxi_stands set proj_location=st_transform(location::geometry,3763);

taxi_stands and taxi_services tables



```
[mi=# \d taxi_stands;
                     Table "public.taxi_stands"
  Column  |          Type          | Collation | Nullable | Default
----------+------------------------+-----------+----------+---------
 id       | integer                |           | not null |
 name     | character varying(255) |           |          |
 location | geometry(Point,4326)   |           |          |
Indexes:
    "taxi_stands_pkey" PRIMARY KEY, btree (id)
```



```
[mi=# \d taxi_services
                                      Table "public.taxi_services"
    Column     |         Type         | Collation | Nullable |                  Default
---------------+----------------------+-----------+----------+-------------------------------------------
 id            | integer              |           | not null | nextval('taxi_services_id_seq'::regclass)
 initial_ts    | integer              |           |          |
 final_ts      | integer              |           |          |
 taxi_id       | integer              |           |          |
 initial_point | geometry(Point,4326) |           |          |
 final_point   | geometry(Point,4326) |           |          |
Indexes:
    "taxi_services_pkey" PRIMARY KEY, btree (id)
```