Last week

- We needed to extend the relational model to handle space
  - Geometry attributes (Geometry objects, reference systems)
  - Spatial relationships (strong relationship, nothing, topologically)
  - Spatial methods for (efficient) spatial queries (supported by spatial indexes)
- We modeled spatial data with object-relational DBMS
  - We looked at different possibilites to model spatial entities
- We discussed standardisation
  - This showed us many aspects of geometry types implemented in spatial databases
- We mentioned Simple Features
  - Base standard that defines all the geometric 'things' we can use in a spatial database

Geo875 | FS24
University of Zürich

2. Lecture Spatial Databases

Spatial Data Types (in depth)

Rolf Meile

Eidg. Forschungsanstalt für Wald, Schnee und Landschaft (WSL)

Swiss Federal Institute for Forest, Snow and Landscape Research

Zhiyong Zhou

Dept. of Geography, University of Zürich

Learning Objectives

- ✓ Gain deeper understanding of how spatial data types are implemented and used accross database technologies
- ✓ We get an idea how basic spatial entity types of an ER can be modeled and then be transfered to a spatial database
- ✓ We try to get a deeper insight to the opensource system consisting of PostgreSQL with PostGIS
- ✓ Understand that creating and storing geometries is not trivial: we need constructors, spatial reference systems, and we need validity checks

Overview

1. **Spatial extensions to relational databases**
    1. PostGIS in conjunction with PostgreSQL
    2. Esri ArcSDE + databases
    3. Oracle Spatial an extension to Oracle database
2. From ER to spatial tables – a case study
3. PostGIS and PostgreSQL particulars
4. Geometry creation and SRID

Spatial extensions to relational databases

- Extensions handle all aspects of geometries and others
- Spatial databases therefore support:
  - Spatial attributes – spatial data types (vectors, topology, rasters)
  - Spatial methods, functions
  - Spatial indexes
- Extensions (different products) are similar in a users perspective; but be aware of details;
  - PostGIS: ST_Contains(geomA, geomB) vs. ST_ContainsProperly(geomA, geomB)
- Only standards guarantee interoperability between systems; see last lectures industry-standards OGC/ISO SQL/MM aka 'Open GIS'; example: ST_Contains(geomA, geomB);
- A myriad of geo functions alongside of standards

Spatial extensions to relational databases

Ways of installation are different

a) Software installed through the client ArcCatalog (ESRI)
b) Just an option when installing the DBMS (Oracle)
c) Package installation in addition to DBMS (PostGIS); apt / yum;
   Activating is needed for each database you create within your DBMS installation

```
CREATE DATABASE mypersonaldb;   -- our course db was named geodb

CREATE EXTENSION postgis;

SELECT postgis_full_version();   -- check installation, detailed version information
```

Handling geometries using PostGIS

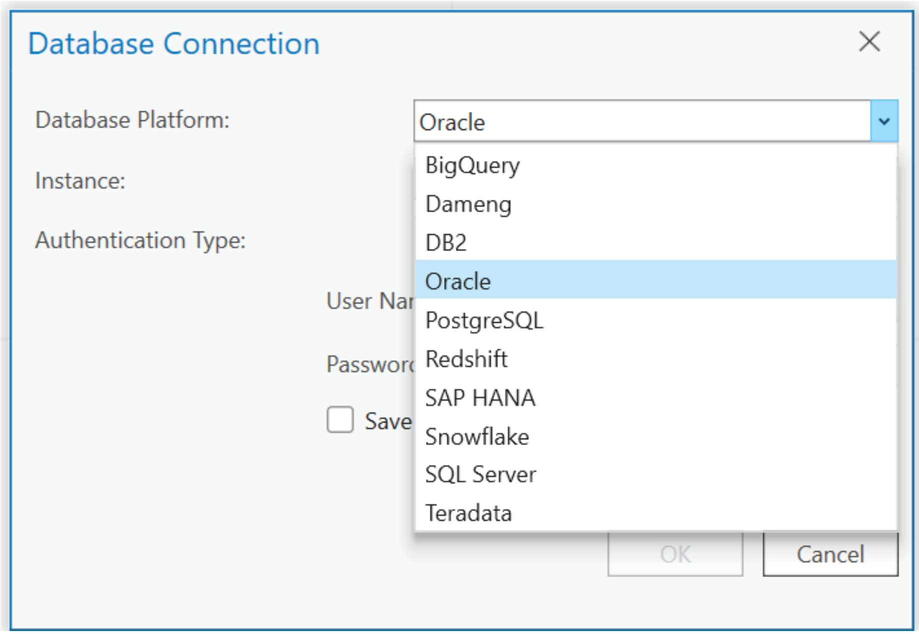
- PostGIS under the umbrella of the OSGeo foundation
  - Extension to PostgreSQL database;
  - Expressed in the public schema; thus available to all users;
  - Extends pure PostgreSQL database with spatial data types, indexes, functions and methods;
  - Optimized for integration with QGIS and other open source clients and servers (MapServer); works well with commercial products, too;
  - Offers functionality - server side - through incorporated open-source libraries
    - Proj4: Provides projection support
    - Geometry Engine Open Source (GEOS): Advanced geometry-processing support
    - Geospatial Data Abstraction Library (GDAL): Provides many advanced raster processing features
    - Computational Geometry Algorithms Library (CGAL/SFCGAL): Enables advanced 3D analysis
  - Create your own stored (geo-)functions
  - **It's all free and open-source**
- More details in the chapter after next

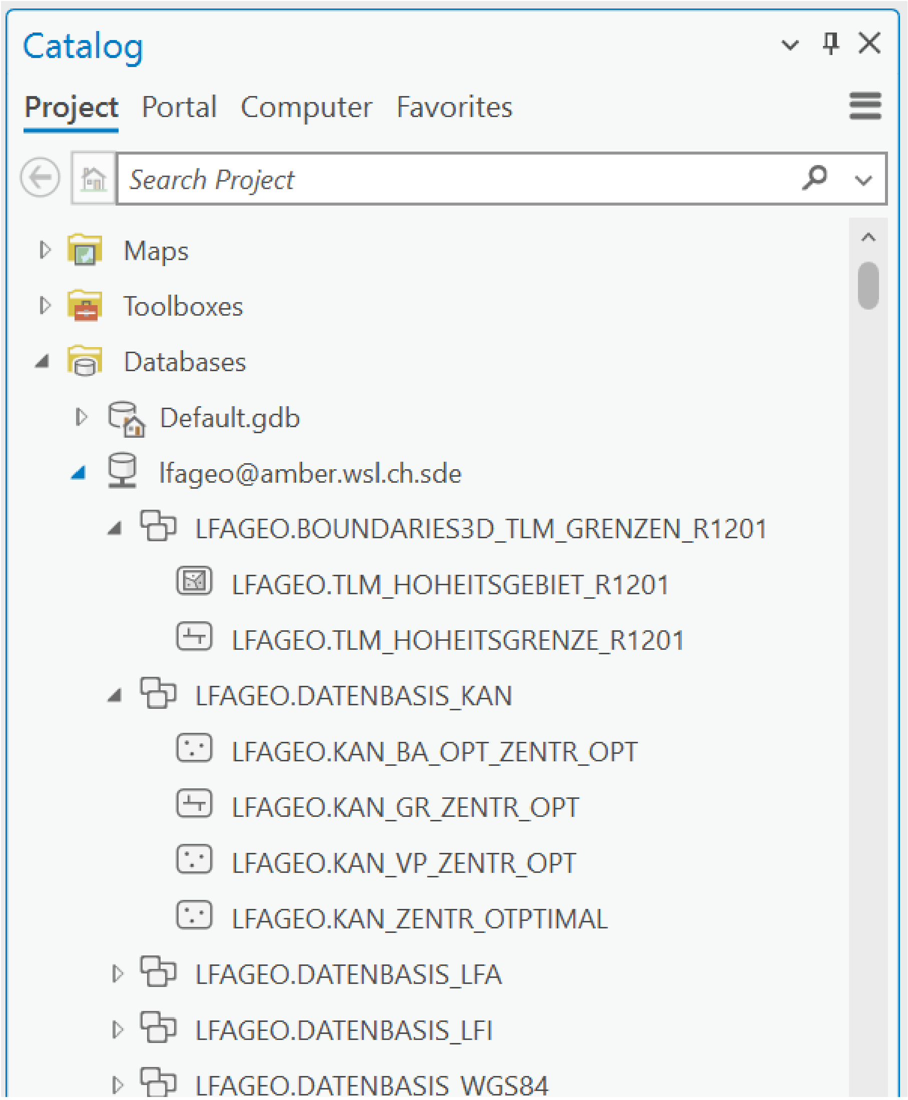
Handling geometries using ArcSDE

- ESRI ArcSDE (part of ArcGISServer)
  - Proprietary plugin for Oracle, DB2, Informix, and PostgreSQL;
  - Expressed as the DB user **SDE** (schema SDE);
  - Extends vanilla databases with spatial data types, indexes, functions and methods;
  - Manages/stores the spatial data of other users
  - Optimized for integration with ESRI GIS products (Desktop ArcGIS, Web, Catalog,…);
  - **It's not free but often used in large organizations**
- <u>Feature class</u>: the logical/physical equivalent of a spatial table: homogeneous attributes and one single type of geometry;
- <u>Feature dataset</u>: holds a number of related feature classes with the same spatial extent, coordinate system, resolution, for storing topologies, networks, TINs... – e.g., optical fiber network with nodes and edges

Looking at ArcSDE stored data in an Oracle DB

- ArcGISPro
  - Looks pretty much the same as in a file-geodatabase
  - Same wordings: feature class, feature datasets, etc.



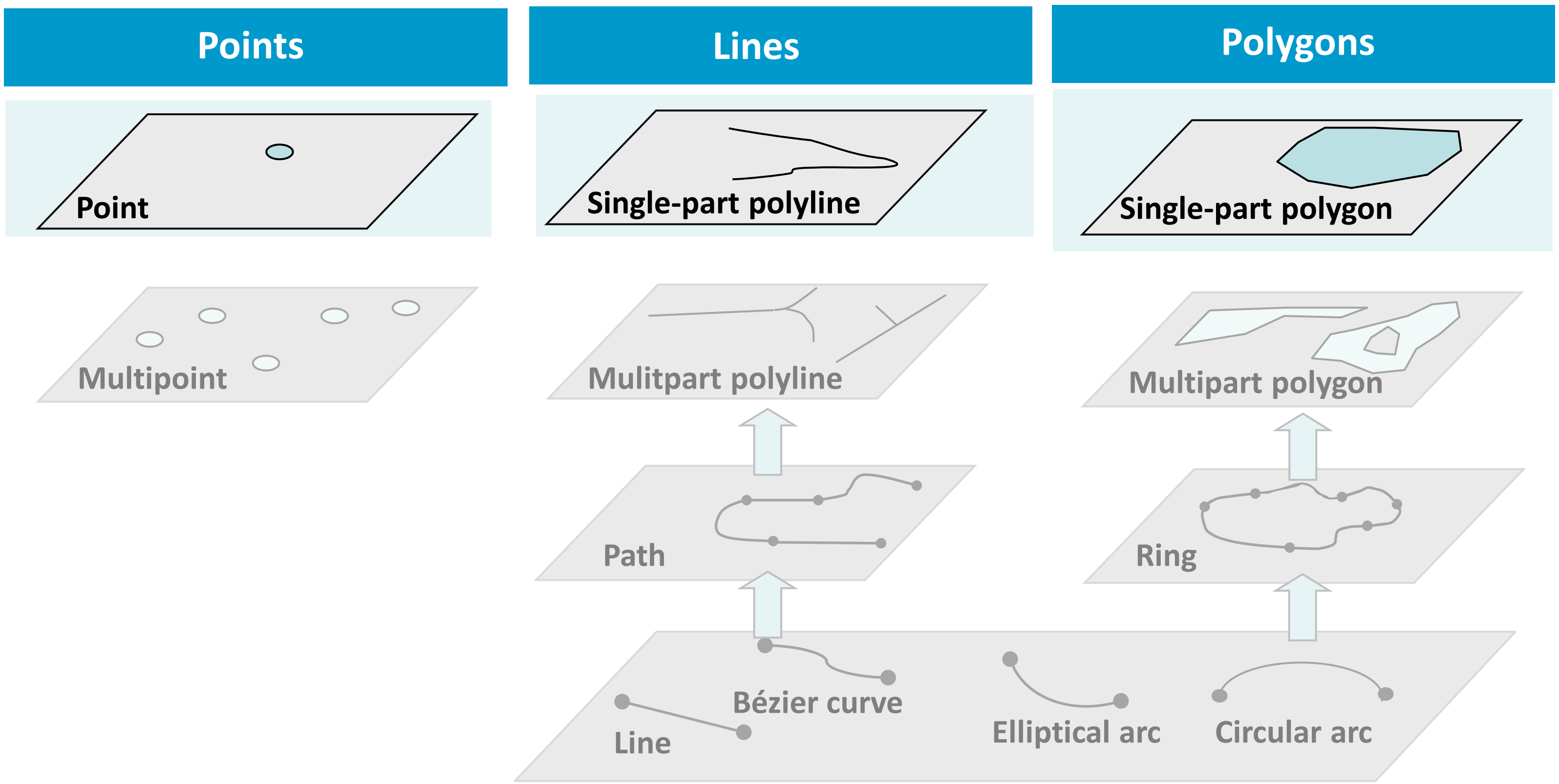


ArcSDE Geometry storage integration

Three storage data types supported by ArcSDE

1. As **SDE.ST_Geometry** (ADT)
   - Close to the SF standard; recommended/implented by ESRI;
   - Can be used in all databases (good support by ESRI products)
   - Geometries retrievable through SQL, incl. spatial queries
2. As **MDSYS.SDO_Geometry** (ADT)
   - Close to SF standard; Uses Oracle’s data type although ESRI integration;
   - Can visualize data using ESRI products;
   - Can use Oracle products for modeling and management;
   - Requires Oracle and ESRI licenses
3. As **BLOB**:
   - Old but stable, performant, legacy; No querying using SQL;
   - Complicated integration in the background
   - Geometries only retrievable using ESRI products (vendor lock-in)

ArcSDE Geometry types



Special feature: Storing Bézier curves and elliptical arcs
Additional geometry types: TINs, Raster
Software topology implemented

Handling geometries using Oracle Spatial

- Oracle Spatial
  - MDSYS.SDO_Geometry is the vector data type
  - A wrapper is used for all SDO_Geometries and methods (MDSYS.ST_Geometry), introducing compatibility with the standards
  - Good support by commercial products (GeoMedia, ArcMap) but also by QGIS and others
  - **It's not free but used in industry systems**
- Topology, Rasters, Metrics supported, too
- Implementation is very 'close' to the DB (as PostGIS is)
- Used to be one of the first spatial databases

Overview

1. Spatial extensions to relational databases
    1. PostGIS in conjunction with PostgreSQL
    2. Esri ArcSDE + databases
    3. Oracle Spatial an extension to Oracle database
2. **From ER to spatial tables – a case study**
3. PostGIS and PostgreSQL particulars
4. Geometry creation and SRID

ER model to tables

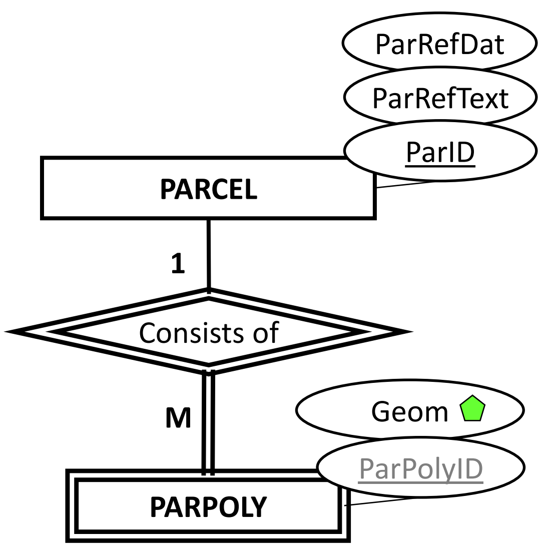
How do I get my (spatial) tables?

Simple use case with one to many relationship with weak entity type:

- „Each parcel can consist of one or more parcel polygons.“
- „Each parcel polygon must belong to exactly one parcel.“

Remember: modeling multi (-polygons, …)

- This example thinks the singlepart way; **separating semantics from geometries**;

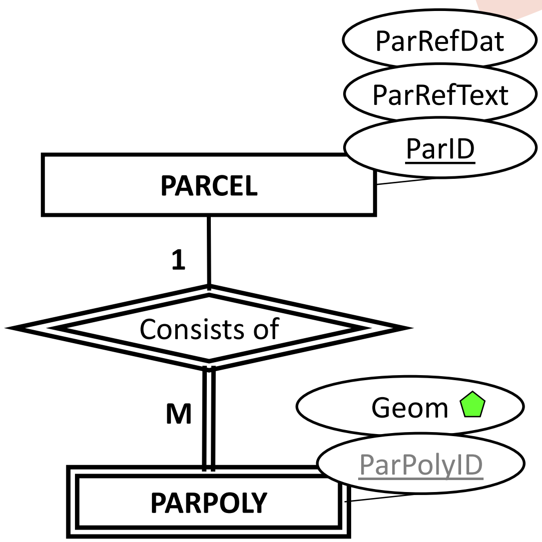


Exercise 1

Provide the relations for the ER model of the previous slide (on paper/screen). Rules of geo874 apply...

Overview of the ER to DB process

1. ER-Model with spatial entity type



2. Derive relations
(adhoc exercise)

3. Physical DB model

parcel	parcel_pk
parid (PK)	**integer**
parreftext	varchar(250)
parrefdat	timestamp

parpoly_parcel_fk

parpoly	parpoly_pk
parpolyid(PK)	**integer**
parid(PK)	**integer**
geom	**geometry(polygon,2056)**

4. Define tables parcel and parpoly with SQL

5. Import external spatial data to a temporary spatial table 'as is'

6. Transfer data from your temporary spatial table to parpoly with SQL (with transformations!)

7. Digitise/insert (spatial) data with QGIS or SQL

x. How about data of parcel table?

Exercise 2

If we had modeled the parcel as a spatial entity type with geometry subtype having multiple polygons. How would our ER -> relations -> database structures look like?

Short discussion

Overview

1. Spatial extensions to relational databases
   1. PostGIS in conjunction with PostgreSQL
   2. Esri ArcSDE + databases
   3. Oracle Spatial an extension to Oracle database
2. From ER to spatial tables – a case study
3. ▶ **PostGIS and PostgreSQL particulars**
4. Geometry creation and SRID

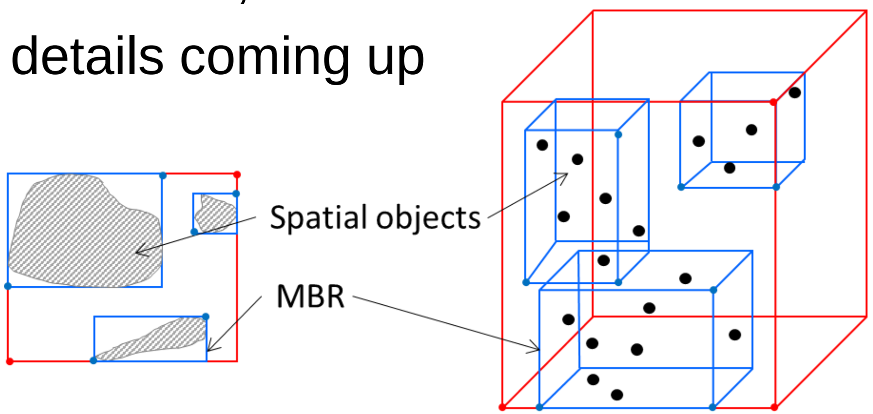
PostGIS Geometry storage

- Geometry type (planar)
  - Cartesian math
  - **Subtypes** for more constraining getting more data integrity: POINT, LINESTRING, LINESTRINGZ, LINESTRINGM, MULTIPOLYGON, GEOMETRYCOLLECTION, POLYHEDRALSURFACEZ, TINZ, ... not all listed
- Geography type (spherical)
  - Lines and polygons are drawn on the earth's curved surface
  - for lat/lon usage
  - only WGS84 as spatial reference system available
  - not all PostGIS functions work on geography
- Raster type (multiband cells)
  - Space as grid with rectangular cells
- Topology
  - World as a network of connected nodes, edges and faces with common borders

PostGIS beyond storage

- Spatial indices
  - no spatial table without a spatial index; you have to do it!
  - main implementation is fast and named GiST (a R-Tree derivate); B-tree also available;
  - make sure you create or have a spatial index if you own a table with a spatial attribute:

    ```
    CREATE INDEX idx_restaurants_geom ON restaurants
    USING gist(geom);
    ```
  - but: sometimes they're created automatically; e.g. while importing with QGIS, but not always, please check;
  - full lecture on all the whys and details coming up



(a) 2D (b) 3D

https://gistbok.ucgis.org/bok-topics/spatial-indexing

PostGIS beyond storage cont.

- Transformation and reprojection
  - remember: built in libraries
  - Example: Reproject a spatial table (**`mytable`**) with a subtyped column named **`geom`** of type **`geometry(MultiLineString,21781)`**

-> the column gets a (new) subtype with a new SRID, respectively
-> all records inside get reprojected
-> watch out: GUI tools sometimes only show main types; see FAQ for a solution

```
-- this converts it all (table physically changed/stored)
ALTER TABLE user50.mytable
ALTER COLUMN geom TYPE geometry(MultiLineString,2056)
USING ST_Transform(geom,2056);

-- on the fly reprojection in a query (nothing stored!)
SELECT ST_Transform(geom,2056) AS geom_trsfrmd
 FROM user50.my_unprojected_21781_table;
```

Limitations

- Geodetic support
- Handling curves limited and not complete
- Topology ?
- pgRouting ?
- Z-value handling
- 3D space living features like polyhedral surfaces and tins

But: it gets incredibly better with each release, e.g.

`ST_FrechetDistance (geometry1, geometry2, float densifyFrac)` V2.4

`ST_ChaikinSmoothing(geometry, nIterations, preserveEndPoints)` V2.5

`ST_GeneratePoints(geometry, nPoints)` V3.0

- version 3.5 is now current

`ST_ReducePrecision(geometry, precision)` V3.1

Overview

1. Spatial extensions to relational databases
   1. PostGIS in conjunction with PostgreSQL
   2. Esri ArcSDE + databases
   3. Oracle Spatial an extension to Oracle database
2. From ER to spatial tables – a case study
3. PostGIS and PostgreSQL particulars
4. **Geometry creation and SRID**

How to generate geometries?

- How does a new spatial table look like?
  - we have a spatial attribute of at least type geometry
  - better: subtyped with srid to constrain the geometries we will later insert into that table

- What is in the spatial table? Probably nothing yet…
  a) We could import geometries and use functions that convert these 'external' geometries to our geometry type (work done by external apps: QGIS, GDAL, other tools)
  b) We could transfer, transform or cast geometries already in the DB (staging/temporary) to our destination spatial table; (work done by SQL in the database)
  c) We can digitize geometries having the spatial database table (layer) editable in QGIS
  d) We can construct our geometries 'by text' with SQL, but how?

Geometry construction

- SQL: with a constructor you can 'make a thing of a desired type' from scratch; in the end using simple base types;
- Recall similiar things:
  - entity type **vs.** instantiated entity
  - object-oriented programming: type **vs.** object
- Some samples without geometry:
  - construct an object of simple base type like number, string: `345, 4` or `'hey you'`. What about object `Rolf` ? Not as simple...
  - construct an object of type lecturer; **create_lecturer(**`'Rolf'`**)** results in object `Rolf` ; need a function that hiddenly does the job for me!
  - construct an object of type timestamp **fn(**parameters**)**

```
make_timestamp(year int, month int, day int, hour int, min int, sec double precision)
```

    Example:

```
SELECT make_timestamp(2024, 11, 15, 9, 17, 43);
```

Geometry construction cont.

- Construct an object geometry type **fn**(representation_geometry)
- Different possibilities to get (aka serialize) a geometry:
  - fn1(text representation)
  - fn2(binary representation)
  - fn3(existing geometry to be morphed, split, ...)
- Examples

```
1. SELECT ST_GeomFromText('POINT(-100 28 1)',4326);
   SELECT ST_GeomFromText('POLYGON((10 28 ,9 29 ,7 30
   ,10 28 ))');
2. SELECT
   ST_GeomFromWKB(E'\\001\\001\\000\\000\\000\\321
   \\256B\\312O\\304Q\\300\\347\\030\\220\\275\\336%E
   @',4326);  -- E' safely escaping \ backslash
3. SELECT ST_Centroid(ST_GeomFromText( 'POLYGON((10
   28 ,9 29 ,7 30 ,10 28 ))'));
```

Geometry construction cont.

- More possibilities to serialize (to get a geometry) available
  - **`ST_GeomFromGML()`**
    `-- Geography Markup Language`
  - **`ST_GeomFromGeoJSON()`**
    `-- Geography Javascript Object Notation`
  - **`ST_GeomFromKML()`**
    `-- Keyhole Markup Format`
- Deserialize: from geometry type to a different format (type)
  - gml, geojson, kml...
  - **`ST_AsGML()`**, **`ST_AsGeoJson()`**, **`ST_AsKML()`** .... and of course **`ST_AsText`** or **`ST_AsEWKT()`**

Why serialize?

- We need to fit geometries into the typed (defined) column of the spatial table by using 'simpler' types

Why deserialize?

- Query spatial information out of the database in a desired/readable format

Example initialisations - real world

- Polygon based on (well-known) text and srid
```
SELECT ST_GeomFromText('POLYGON ((2682180 1235646,
2682799 1234338, 2686473 1235082, 2684750 1237992,
2682780 1240320, 2680160 1236665, 2682180
1235646))',2056);
```

- Line
```
SELECT ST_GeomFromText('LINESTRING (2682580 1235699,
2682799 1234338)',2056);
```

- Point
```
SELECT ST_GeomFromText('POINT (2682580
1235699)',2056);
```

How do we get and store values of the vertices?

- Using SQL: Type the numbers/text of the WKT-representation; tedious..., but easy for points
- GUIs do the same with a lot of 'hidden' steps
  - Mouse/Screen coordinate system
  - Browser: HTML & Javascript Libraries
  - Browser: Screen to real-world coordinates calculations; geometry info
  - Browser: Sending to web server (application running there)
  - Web server: Application opens connection to database server
  - Web server: Transforms the geo information from browser to SQL(!)
  - Web server: Stores the data in a spatial table
- QGIS for digitalising directly on spatial database tables

Webapplication **geoadmin.ch**

Exercise 3

a) Provide the code to construct a Polygon of type **geometry** representing a triangle with the coordinates 4,5 and 7,8 and 10,4 in an undefined coordinate system (on paper/screen).

b) Provide the code to construct a Line with the coordinates 4,5 and 7,5 8,9 3,8 also in an undefined coordinate system (on paper/screen).

SRID: Spatial Reference ID

Geometry constructors ask for SRID – what is it exactly?

- SRID: Spatial Reference System ID. Values should match values in a table available in the public schema (comes with installation).
  - Switzerland's new 'CH1903+_LV95' has SRID = 2056
  - Switzerland's old 'CH1903_LV03' has SRID = 21781
  - Query the information:

```
SELECT sr.* FROM public.spatial_ref_sys sr WHERE
sr.srid=2056;
```

- These values are standardised as EPSG codes.
- See www.epsg.org
- https://spatialreference.org/
- Own values could be added

Exercise 4

Note the differences in the values for this polygon definition. Discuss with your neighbour, go online and find out:

- Why do we here have so many brackets in `SELECT ST_GeomFromText('POLYGON((8 47,10 30,10 10,30 5,45 20,8 47),(30 20,20 15,20 25,30 20))', 4326)`?
- What does `4326` stand for? Find its textual definition.
- What are the values in e.g., `8 47` representing?
- Is this a real-world example?

Summary

- We have discussed different products that can store spatial data in databases
- Geometries require the specification of geometry types; for constructing we need vertices in text form and spatial reference systems
- We need functions to create geometries/geometry objects
- We took a more detailed look at PostGIS and layed out different spatial data types that can be stored with this data extension