Geo372
Vertiefung GIScience

Introducing (spatial) databases

Herbstsemester

Ross Purves

Last week

- We looked at the use of GIS in **decision making**
- Explored a number of ways of combing data to make decisions in **multi-criteria evaluation** (MCE) which formalised much of what we have done in the past
- Real world decisions require that we **embed GIS in a process** where we consult a wide range of stakeholders in the decision making process
- **Spatial Decision Support** Systems aim to provide a framework for such a process – key is that the GIS expert does not work alone

Learning objectives

- You can **define** the terms **database** and **DBMS**
- You can **describe** the **basic elements** of a **database table** and **operations** that are carried out on it
- You know the **strengths** and **weaknesses** of **DBMS** and you can give **examples** of their use
- You **understand** why **indexing** improves performance, and can give an example of the application of an index to a simple dataset
- You can **define** what properties a **spatial database** must have, and can **describe spatial indexing techniques**

Outline

- We are going to look at how **large volumes** of data are typically **stored** and **manipulated** in computers using databases
- First we'll look at the basic element of a database – the **database table**
- Then we will explore some **typical database applications** and their **requirements**
- We will explore how we can **index data**, to make access to it faster
- Finally, we will look at what makes **spatial databases** different, and how spatial data can be indexed

Health warning

- This lecture is intended as an **introduction** to **database ideas**
- Many (most) of you have not encountered databases before formally – this lecture aims to help you understand some **basic ideas** about **databases** and how we can use them with **geographic data**
- This lecture serves as a "teaser" for material in **Geo874/875** (Introduction to databases/ Spatial databases) as Masters courses
- Finally, if you are also studying informatics the **aspatial material** should be familiar

What is a database?

- A **database** is simply a **collection of data**, which are generally related to a particular theme, and organised in some **sensible structure**
- Databases are very **commonly** used in all applications of computers as soon as we have data which we wish to **store** and **use many times**…
- Generally, we access databases through special software called a **database management system** (DBMS)
- We'll start by looking at some **definitions** and then how we **store data** in a database…

Some definitions

- **Data** – data are the raw stuff that we store in a computer -> they need not be human readable
- **Database** – A database stores data, typically about a particular subject, in a structured way
- **Database management system** – A DBMS is a **software tool** for the efficient and effective storage, management and retrieval of data stored in databases

Data Base Management Systems

- There are different types of DBMS
  - **Relational-DBMS** are most common – they link tables which are two dimensional
  - **Object-DBMS** aimed to solve some weaknesses of RDBMS, most especially encapsulating rich properties and behaviours of objects (e.g. spatial data)
  - **Object-relational-DBMS** a mix between the previous two (essentially some capability to deal with objects is bolted onto the RDBMS)
- All of these models are based around **well structured** and carefully defined **schemas** (formal definitions)
- However, another family of database approaches exists – so-called **NoSQL** which are gaining popularity
- We're going to consider the properties of DBMS in a general way

Storing structured data

Primary key

FID	LENGTH	OBJECTID	OBJECTORIG	OBJECTVAL	BRIDGETYPE	TUNNELTYPE
0	304.10075	2767797	LK25	NS_Bahn1		
1	88.25705	2767805	LK25	NS_Bahn1		
2	106.87428	2767818	LK25	NS_Bahn1		
3	1356.1511	2767826	LK25	NS_Bahn2		
4	97.65828	2767830	LK25	NS_Bahn2		Tunnel
5	35.78934	2767831	LK25	NS_Bahn2	Bruecke	
6	249.2615	2767832	LK25	NS_Bahn2		
7	251.4928	2767833	LK25	NS_Bahn2		
8	491.41748	2767834	LK25	NS_Bahn2		
9	137.25998	2767835	LK25	NS_Bahn2		
10	164.96806	2767838	LK25	NS_Bahn2		
11	226.429	2767840	LK25	NS_Bahn2	Bruecke	
12	156.58021	2767841	LK25	NS_Bahn2		
13	35.17745	2767842	LK25	NS_Bahn2		
14	374.42415	2767843	LK25	NS_Bahn2	Bruecke	
15	101.97348	2767844	LK25	NS_Bahn2		
16	89.61592	2767846	LK25	NS_Bahn2		
17	90.05697	2767847	LK25	NS_Bahn2		
18	902.72458	2767849	LK25	NS_Bahn2		
19	445.24305	2767850	LK25	SS_Bahn1		
20	45.93035	2767860	LK25	I_Geleis		
21	316.53979	2767861	LK25	I_Geleis		
22	74.2783	2767862	LK25	I_Geleis		
23	130.83548	2767863	LK25	I_Geleis		
24	561.60188	2767875	LK25	I_Geleis		
25	315.67545	2767880	LK25	I_Geleis		
26	118.97345	2767881	LK25	I_Geleis		
27	241.46238	2767882	LK25	I_Geleis		

Record

Field

Typically a **RDBMS** manages individual **tables**, containing **records** which each hold items of data. The table itself is built from **fields**, at least one of which should have a defined **primary key** – a value which is guaranteed to be unique for every record.

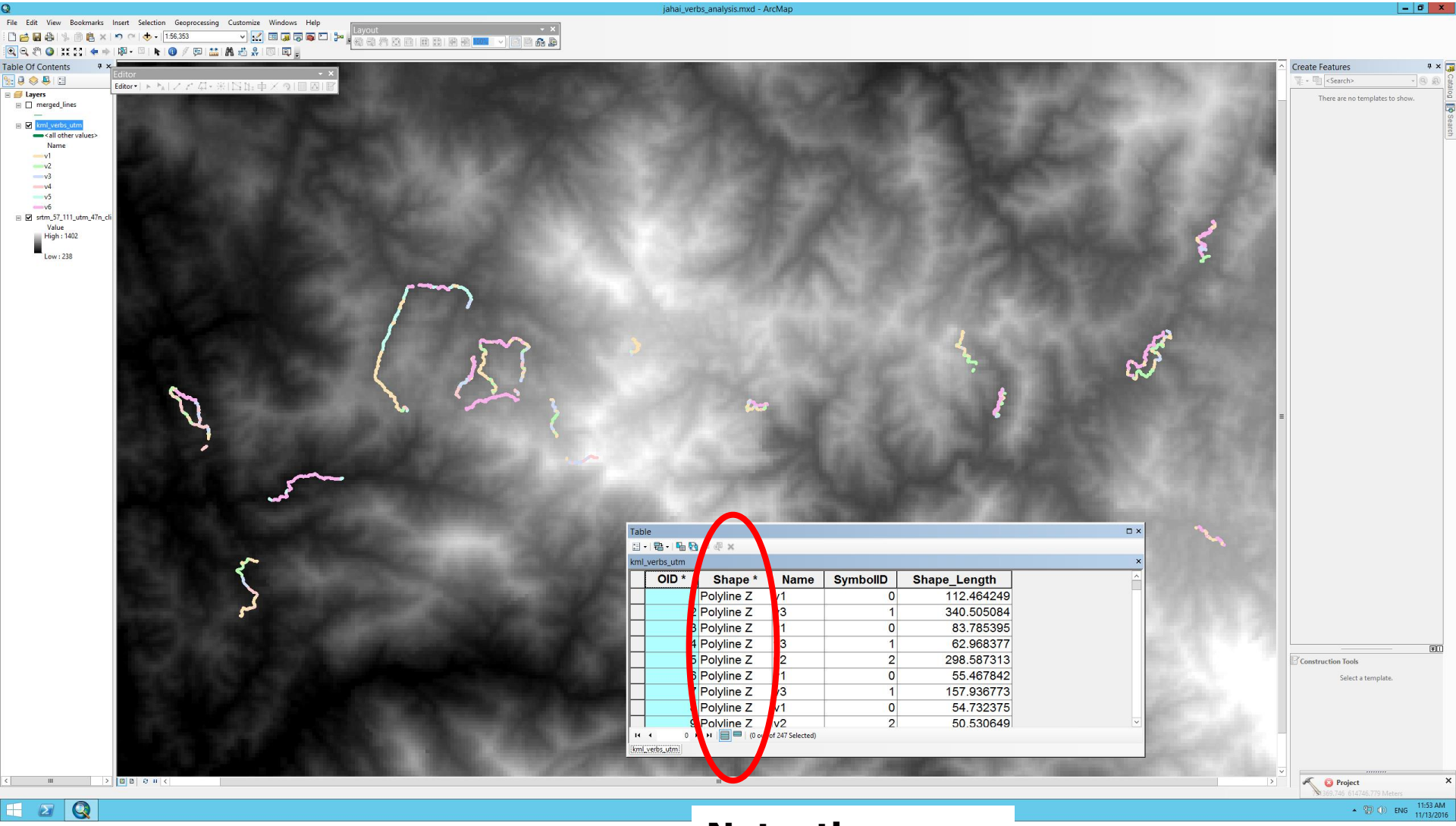
More on storing structured data

- The **fields** have defined data types (e.g. integer, float, string)
- It is also possible to identify **allowed values** in a field (e.g. only integers between 0-10; PLZ must be a 4 digit integer – 8055)
- The choice of **primary key** is very important – for instance all of your **Modulpruefungen** results are stored in a database
  - What **properties** must the primary key have?
  - What do you think should be the **primary key**?
  - What **should not** be the primary key?

Properties of database tables

- These properties were **formally defined** by Codd of IBM in 1970 (cited in Longley et al., pp. 200)
  - Only **one value** is stored in a cell at the intersection of a row and column
  - All values in a **column** are about the same **"subject"**
  - Every **row is unique** (though individual entries in a record need not be unique)
  - There is **no significance** to the **order** of the rows or columns

Why must every row be unique?



Note, these are objects

Joining tables

- Joining is a way of linking different tables which share some data…
- You have all **joined tables** in practicals
- You used an appropriate **key** – a field found in both tables and joined the two tables together
- There are 3 sorts of relationships which can occur when we have an appropriate key
  - **One to one:** Every ***ship*** has ONE ***captain*** (and every ***captain*** has ONE ***ship***)
  - **One to many (or many to one):** A ***Kanton*** has MANY ***Gemeinden*** (but each ***Gemeinde*** has only ONE ***Kanton***)
  - **Many to many:** A ***road*** passes through MANY ***cantons*** and a ***canton*** has MANY ***roads***

Basic database tasks

- Having created a database (and by definition a schema, the three most basic database tasks are
  - **Data entry**
  - **Data retrieval**
  - **Data deletion**
- The most common day to day task is data retrieval - **individual records** which match some **query** are returned
- Records either **match** or **don't match** a query… (e.g.: your attribute selections in ArcGIS)

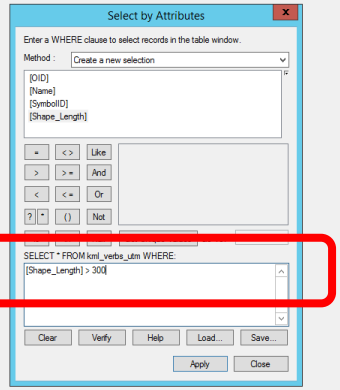
Querying a database

- Typically, we **query** a database with a **database query language**
- The most common example is **SQL** (Structured or Standard Query Language)
- With SQL we can **query** and **manipulate** database tables
  - E.g.: SELECT * FROM eis WHERE "LENGTH" > 300
  - This query says find all the records in the table eis whose field LENGTH have a value greater than 300
  - The result is the set of records for which this is true
  - We can combine queries with logical operators (e.g. AND/ OR/ XOR)
  - Queries must be valid

Table

kml_verbs_utm

OID *	Shape *	Name	SymbolID	Shape_Length
1	Polyline Z	v1	0	112.464249
2	Polyline Z	v3	1	340.505084
3	Polyline Z	v1	0	83.785395
4	Polyline Z	v3	1	62.968377
5	Polyline Z	v2	2	298.587313
6	Polyline Z	v1	0	55.467842
7	Polyline Z	v3	1	157.936773
8	Polyline Z	v1	0	54.732375
9	Polyline Z	v2	2	50.530649
10	Polyline Z	v1	0	54.308924
11	Polyline Z	v3	1	175.356974
12	Polyline Z	v1	0	229.138188
13	Polyline Z	v3	1	60.337968
14	Polyline Z	v1	0	104.423673
15	Polyline Z	v3	1	49.599391
16	Polyline Z	v1	0	119.71671
17	Polyline Z	v2	2	441.180343
18	Polyline Z	v1	0	106.402742
19	Polyline Z	v2	2	66.700558
20	Polyline Z	v1	0	99.683821
21	Polyline Z	v3	1	238.787435
22	Polyline Z	v1	0	252.743232
23	Polyline Z	v2	2	690.995194
24	Polyline Z	v1	0	106.878846
25	Polyline Z	v4	3	13.021452
26	Polyline Z	v2	2	27.645339
27	Polyline Z	v6	4	151.034687
28	Polyline Z	v1	0	163.640498
29	Polyline Z	v6	4	409.001877
30	Polyline Z	v1	0	18.709112
31	Polyline Z	v6	4	475.715773
32	Polyline Z	v1	0	124.964175
33	Polyline Z	v6	4	1543.979221
34	Polyline Z	v3	1	126.09966
35	Polyline Z	v2	2	43.864065
36	Polyline Z	v1	0	213.746508
37	Polyline Z	v2	2	166.353431
38	Polyline Z	v1	0	130.719591
39	Polyline Z	v3	1	74.30571
40	Polyline Z	v1	0	54.076111
41	Polyline Z	v2	2	89.648142
42	Polyline Z	v3	1	76.91482
43	Polyline Z	v2	2	100.060841
44	Polyline Z	v6	4	1323.408767
45	Polyline Z	v1	0	67.911546
46	Polyline Z	v6	4	384.178846



0 (0 out of 247 Selected)

kml_verbs_utm

Project

Table

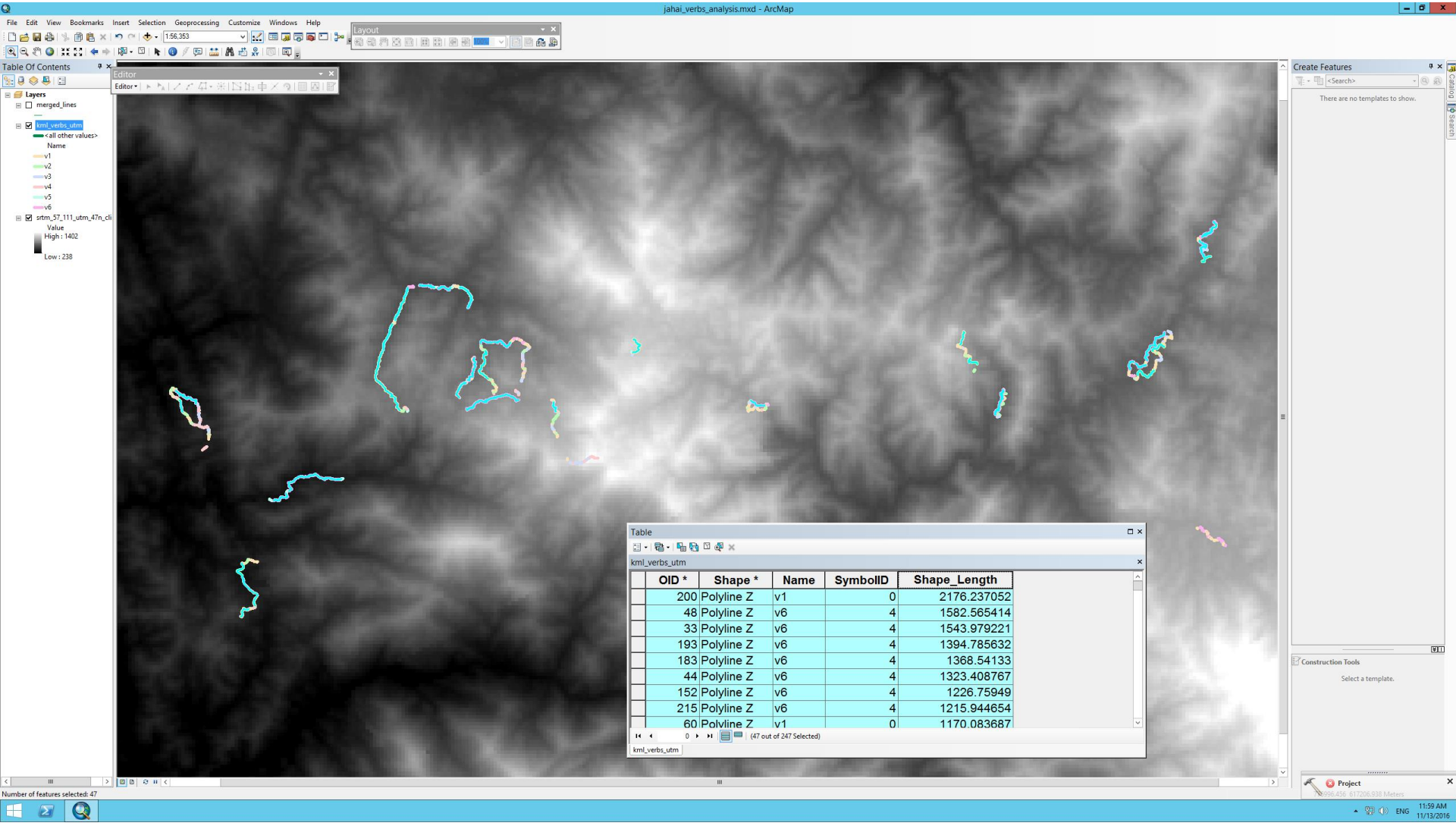
kml_verbs_utm

OID *	Shape *	Name	SymbolID	Shape_Length
1	Polyline Z	v1	0	112.464249
2	Polyline Z	v3	1	340.505084
3	Polyline Z	v1	0	83.785395
4	Polyline Z	v3	1	62.968377
5	Polyline Z	v2	2	298.587313
6	Polyline Z	v1	0	55.467842
7	Polyline Z	v3	1	157.936773
8	Polyline Z	v1	0	54.732375
9	Polyline Z	v2	2	50.530649
10	Polyline Z	v1	0	54.308924
11	Polyline Z	v3	1	175.356974
12	Polyline Z	v1	0	229.138188
13	Polyline Z	v3	1	60.337968
14	Polyline Z	v1	0	104.423673
15	Polyline Z	v3	1	49.599391
16	Polyline Z	v1	0	119.71671
17	Polyline Z	v2	2	441.180343
18	Polyline Z	v1	0	106.402742
19	Polyline Z	v2	2	66.700558
20	Polyline Z	v1	0	99.683821
21	Polyline Z	v3	1	238.787435
22	Polyline Z	v1	0	252.743232
23	Polyline Z	v2	2	690.995194
24	Polyline Z	v1	0	106.878846
25	Polyline Z	v4	3	13.021452
26	Polyline Z	v2	2	27.645339
27	Polyline Z	v6	4	151.034687
28	Polyline Z	v1	0	163.640498
29	Polyline Z	v6	4	409.001877
30	Polyline Z	v1	0	18.709112
31	Polyline Z	v6	4	475.715773
32	Polyline Z	v1	0	124.964175
33	Polyline Z	v6	4	1543.979221
34	Polyline Z	v3	1	126.09966
35	Polyline Z	v2	2	43.864065
36	Polyline Z	v1	0	213.746508
37	Polyline Z	v2	2	166.353431
38	Polyline Z	v1	0	130.719591
39	Polyline Z	v3	1	74.30571
40	Polyline Z	v1	0	54.076111
41	Polyline Z	v2	2	89.648142
42	Polyline Z	v3	1	76.91482
43	Polyline Z	v2	2	100.060841
44	Polyline Z	v6	4	1323.408767
45	Polyline Z	v1	0	67.911546
46	Polyline Z	v6	4	384.178846

(47 out of 247 Selected)

kml_verbs_utm

Project


jahai_verbs_analysis.mxd - ArcMap
File Edit View Bookmarks Insert Selection Geoprocessing Customize Windows Help
Table Of Contents
Layers
merged_lines
kml_verbs_utm
<all other values>
Name
v1
v2
v3
v4
v5
v6
srtm_57_111_utm_47n_cli
Value
High : 1402
Low : 238
Editor
Create Features
There are no templates to show.
Construction Tools
Select a template.
Table
kml_verbs_utm
OID * Shape * Name SymbolID Shape_Length
200 Polyline Z v1 0 2176.237052
48 Polyline Z v6 4 1582.565414
33 Polyline Z v6 4 1543.979221
193 Polyline Z v6 4 1394.785632
183 Polyline Z v6 4 1368.54133
44 Polyline Z v6 4 1323.408767
152 Polyline Z v6 4 1226.75949
215 Polyline Z v6 4 1215.944654
60 Polyline Z v1 0 1170.083687
(47 out of 247 Selected)
Number of features selected: 47
Project
ENG
11:59 AM
11/13/2016

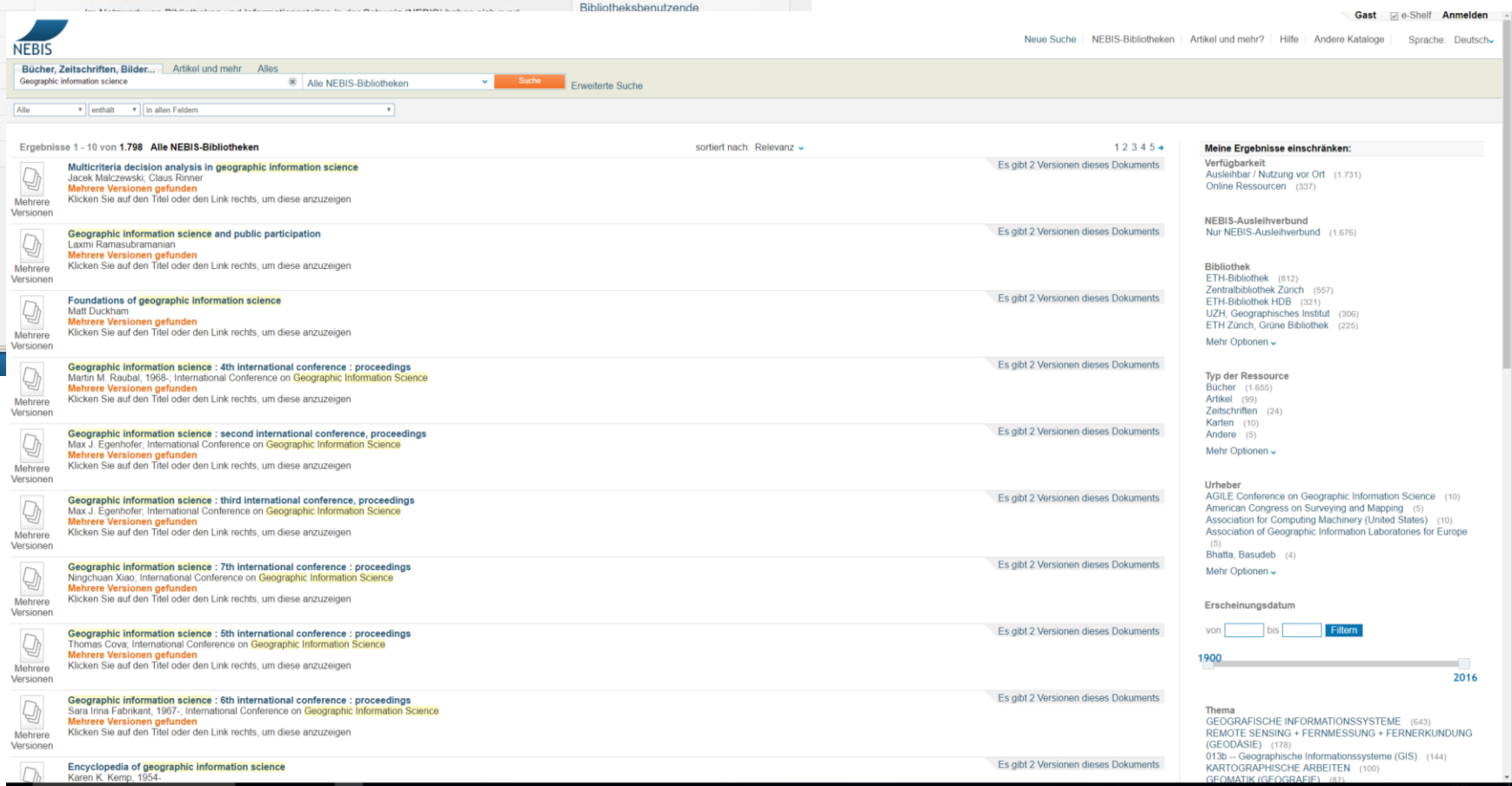
Using databases

- **Data collection** and **maintenance** are expensive
- If we spend a lot of money on something, it makes sense to think about **efficient**, **structured and safe ways** to store and use it
- Databases typically store sets of data related to a particular business or problem i.e.:
  - customer records for an online shop;
  - lists of plant types associated with forest parcels;
  - lists of books in the warehouse of an online shop.
- Databases and database management systems (DBMS) provide us with powerful ways of storing, retrieving and analysing data…

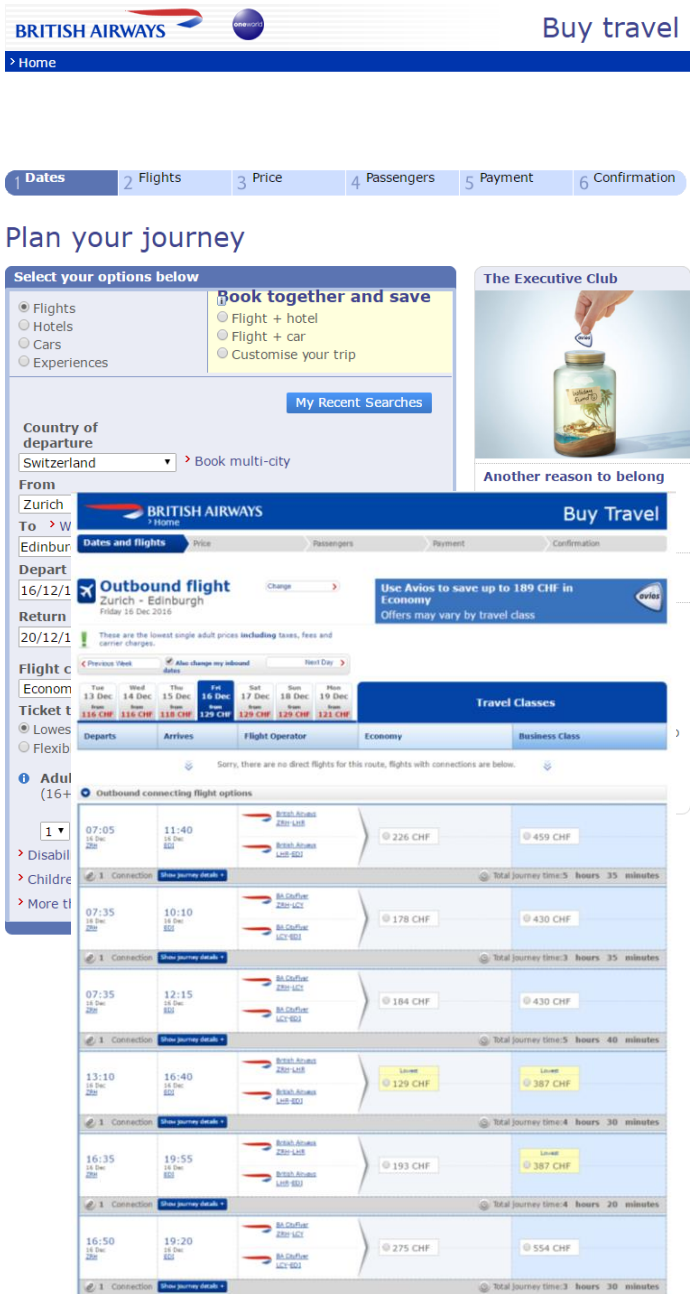
Allowing many users at the same time

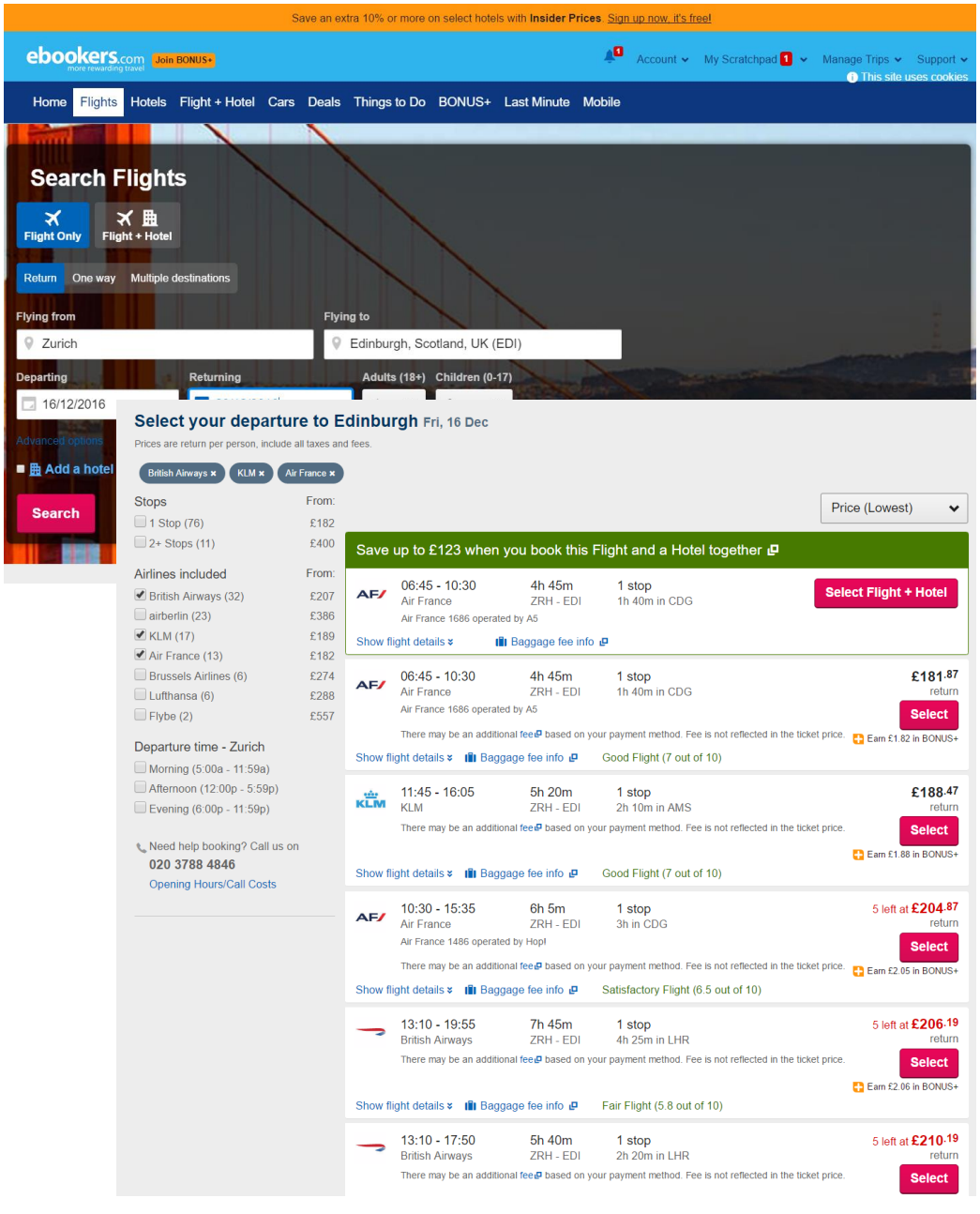
Many database applications must cope with **very large** numbers of simultaneous users

Retrieving similar data many times



Sharing data between many applications





Shared data

- Very common that **multiple applications** have **access** to the **same data**
- **Data structure constant**, but **view** of data differs (e.g. currency in flight bookings –implications?)
- Some **applications** may be linked to **different databases** (e.g. eBookers flight prices are not only those of British Airways…)
- Different database users have **different rights** – e.g. new flights are **entered** by British Airways and accessed by both booking systems
- What problems could occur when **multiple users** access a booking system (through one or more applications?)
- The **separation of applications** from **data** is a major advantage of the use of DBMS

Transactions

- A **transaction** is a group of changes made to a database
- **Integrity** is maintained by **only changing** the database when all the **stages are complete**
- If for some reason the changes **can't be completed**, the database is returned to its original state
- Integrity must also be maintained when **multiple users** access a database

Transaction examples

- When a **ticket is sold** the database record is **locked** until the transaction is complete (i.e. the sale is authorised by the customer's credit card) – during this period the same seat cannot be sold
- Typically such **transactions are fast**, and thus the locking is not evident to the user
- More **complex operations** (e.g. editing the points representing a polygon) may last a long time and locking the database my prevent other users from carrying out their tasks…

Allow access to large volumes of data

- Databases and DBMS allow storage and access to **very large volumes** of data
- Data storage is independent of users' CPU/ hard disk (central storage)
- Gathering of such large volumes of data also has important **security considerations**

Tages-Anzeiger

Front Zürich Schweiz International **Wirtschaft** Börse Sport Kultur Reisen Wissen Auto Blogs Panorama Mehr

Unternehmen Konjunktur Geld Karriere Recht & Konsum Rechtsfragen Vorbörse Weiterbildung World Economic Forum Bildstrecken

Was Sie zum Skelett der Panama Papers wissen müssen

Die Datenbank beinhaltet Infos zu allen Offshorefirmen der Kanzlei Mossack Fonseca. Fünf Fragen und Antworten.

Die Datenbank des ICIJ gibt nur einen kleinen Teil des Lecks preis, das die Panama Papers ausmacht: Fensterputzer im Bankenviertel von Panama-Stadt. Foto: Arnulfo Franco (AP)

Welche Daten hat das International Consortium of

Artikel zum Thema

Journalisten veröffentlichen Datenbank zu Panama Papers

Other properties of DBMS

- Collecting data together in one place **reduces redundancy** and **maintains consistency**
- Data **sharing** is made easier
- Database design **formalises data collection** and **storage** at an **organisational level**
- **Organisational knowledge** about data is represented in the database design
- Databases make ensuring **data integrity** and **persistence** (not losing data) **easier** – data are protected and should be secure (but see Panama papers example)

Advantages and disadvantages of DBMS

- Collecting data in one place reduces redundancy
- Reduced redundancy should reduce costs
- Applications are data independent
- Data sharing easier
- Security and standards (e.g. metadata entry) can be enforced
- DBMS are well suited to large user numbers
- Different users can be assigned differing rights (view some, view all, edit some of the database)
- Initial costs of migrating to DBMS are high
- System can be inflexible – e.g. I can't find out which students are in which semester, because this information is not stored!
- For short projects, with single users, add unnecessary complexity (reason we use file geodatabases and not geodatabases)
- Database performance can be poor, especially with complex data types and indexing must be well thought out

List after Longley et al., p. 195)

Question…

- You have a **telephone book**, **alphabetically** ordered, with **4 names per page** and **50 pages**
- What is the **largest number of records** you need to look at to find any name?
- What is the **least number of records** you need to look at?

Indexing and retrieving data

- Imagine we want to **search** a large database for "**Ross Purves**"
- A typical example would be looking in a **telephone book**
- How do you **search** a telephone book?
- **Indexing** is concerned with methods to make searching (and thus data retrieval) **more efficient**
- The **worst case** is always that for **n records**, we must **examine n records** to find what we are looking for…

Indexes

- Typical **real world database applications** have very large numbers (**often millions**) of records
- If we want to **search databases quickly** we must **build indexes** – remember from earlier, there is no significance to the order of rows in a database
- **Indexing reduces** the number of records that must be searched when we make a query
- We **index fields** on which searches are **commonly made** and which have **many unique values** (i.e. there is little point indexing a field with values such as male/ female)
- Indexes require us to **store more data** (the index) in additional fields and make adding new data slower (we have to rebuild the index too)

B-tree index

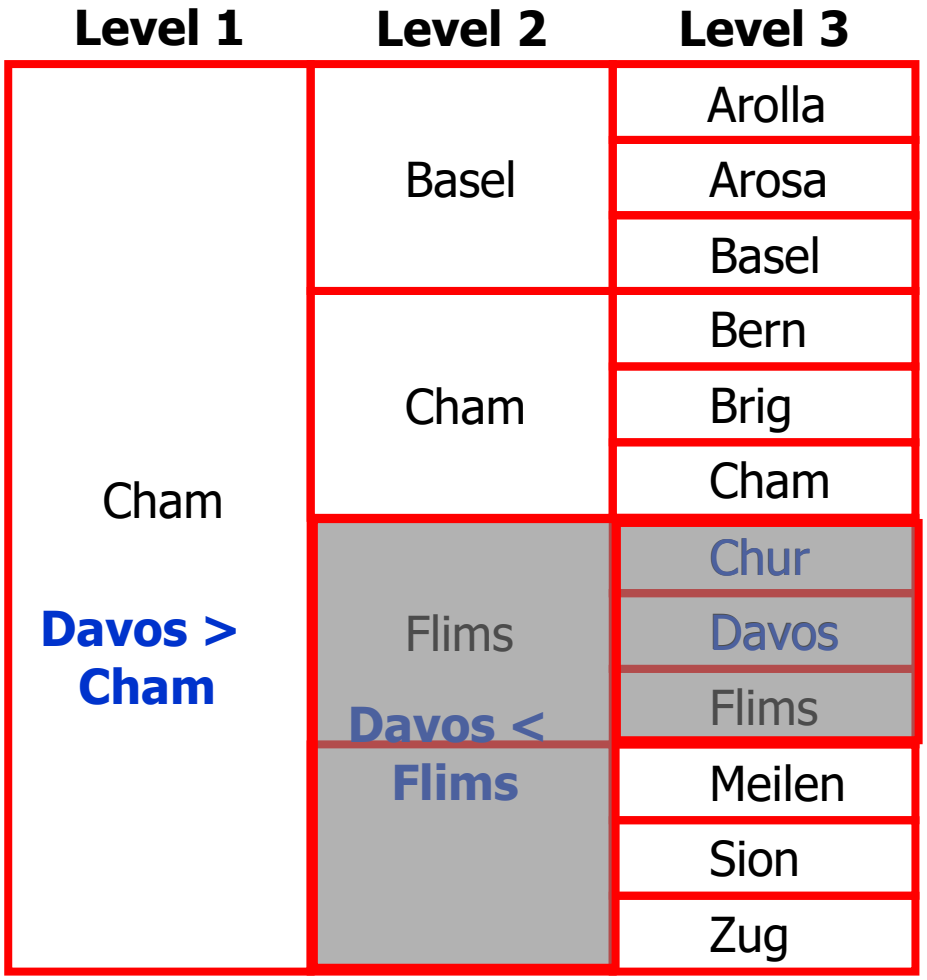
- A very commonly used index in commercial databases is the **B-tree index** (Balanced-tree)
- In the B-tree we **order the data** and store a given number of values at each level
- We can **traverse the tree quickly** by using pointers from each level
- B-trees are called balanced trees because there are always the **same number of leaves** at the bottom of the tree (contrast later with quadtrees)

B-tree example

Original data

Arolla
Arosa
Basel
Bern
Brig
Cham
Chur
Davos
Flims
Meilen
Sion
Zug

Indexed data



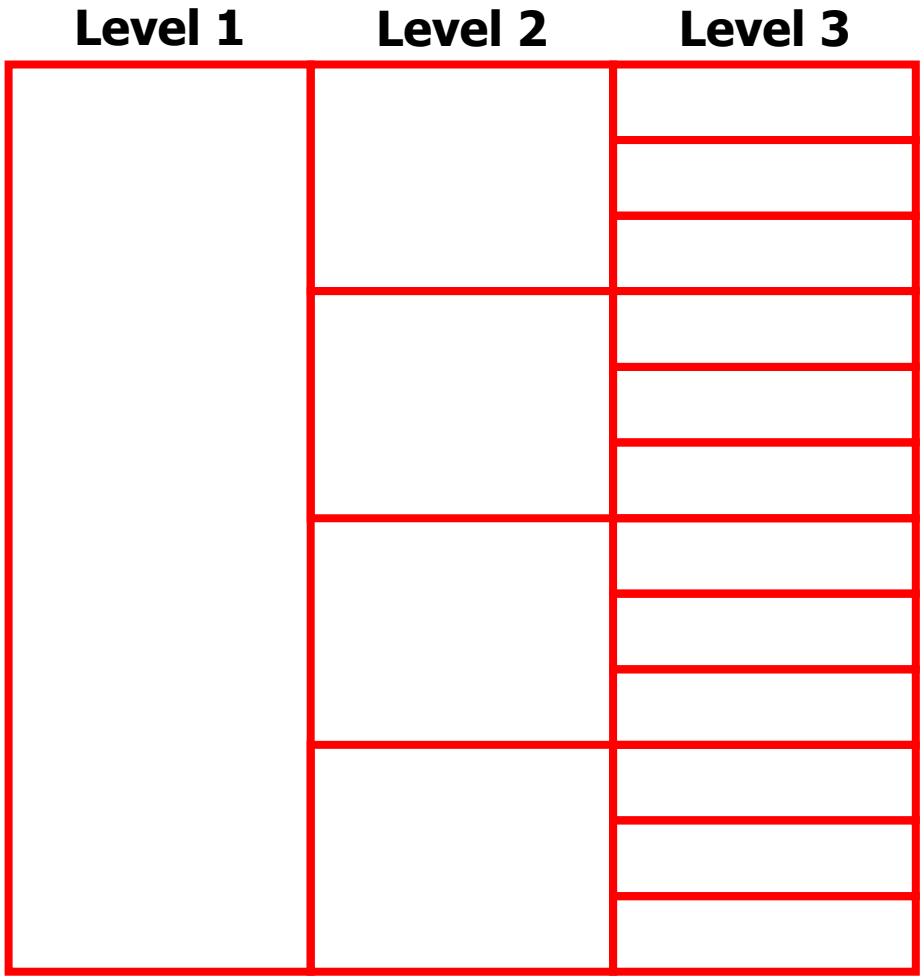
- At each level we compare with our query and decide which child to go to
- At the final level we search all records in the node
- Example query: **Davos**

Exercise – fill in a B-tree index for these data

Original data

21
14
32
17
100
3
72
23
10
10
70
-20

Indexed data



You need to make one extra step here…

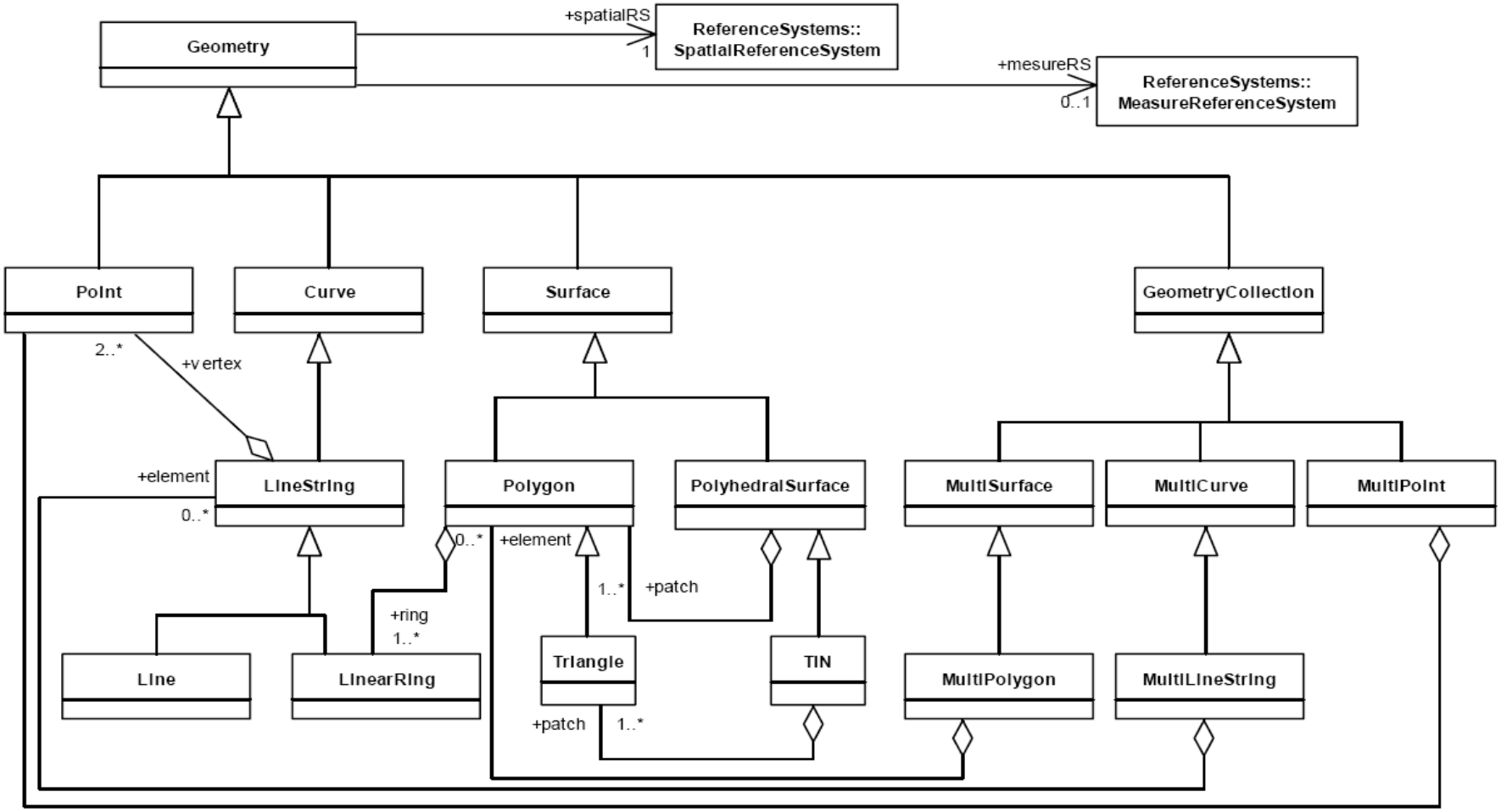
Spatial databases

- Up to now we have considered **standard**, **aspatial**, representations of databases
- **Güting** (1994) offered the following basic definition of a spatial database:
  1. A **spatial database system** is a **database system**
  2. It offers **spatial data types** in its data model and a **query language**
  3. It supports spatial data types in its implementation, providing at least **spatial indexing** and **efficient algorithms for spatial join**

Spatial databases

- Güting's definition matches well with what is **commonly implemented**
- For example, the **Open Geospatial Consortium** specify a **Simple Feature Specification**, which details what properties a spatial database should have
- Many **commercial** and **open-source databases implement** all or some of this standard – for example systems may implement some operations only for the **minimum-bounding rectangle** of polygons (much faster and simpler)

OGC geometry types



From: http://portal.opengeospatial.org/files/?artifact_id=25355

Example spatial operations supported by OGC Simple Feature Specification

Selection of methods for testing spatial relations

Equals (anotherGeometry: Geometry): Integer — Returns 1 (TRUE) if *this* geometric object is "spatially equal" to anotherGeometry.

Disjoint (anotherGeometry: Geometry): Integer — Returns 1 (TRUE) if *this* geometric object is "spatially disjoint" from anotherGeometry.

Touches (anotherGeometry: Geometry): Integer — Returns 1 (TRUE) if *this* geometric object "spatially touches" anotherGeometry.

Contains (anotherGeometry: Geometry): Integer — Returns 1 (TRUE) if *this* geometric object "spatially contains" anotherGeometry.

Create a matrix showing which relationships are possible for points, lines and polygons

Selection of methods for spatial analysis

Distance (anotherGeometry: Geometry):Double — Returns the shortest distance between any two Points in the two geometric objects as calculated in the spatial reference system of *this* geometric object. Because the geometries are closed, it is possible to find a point on each geometric object involved, such that the distance between these 2 points is the returned distance between their geometric objects.

Buffer (distance: Double): Geometry — Returns a geometric object that represents all Points whose distance from *this* geometric object is less than or equal to distance. Calculations are in the spatial reference system of *this* geometric object. Because of the limitations of linear interpolation, there will often be some relatively small error in this distance, but it should be near the resolution of the coordinates used.

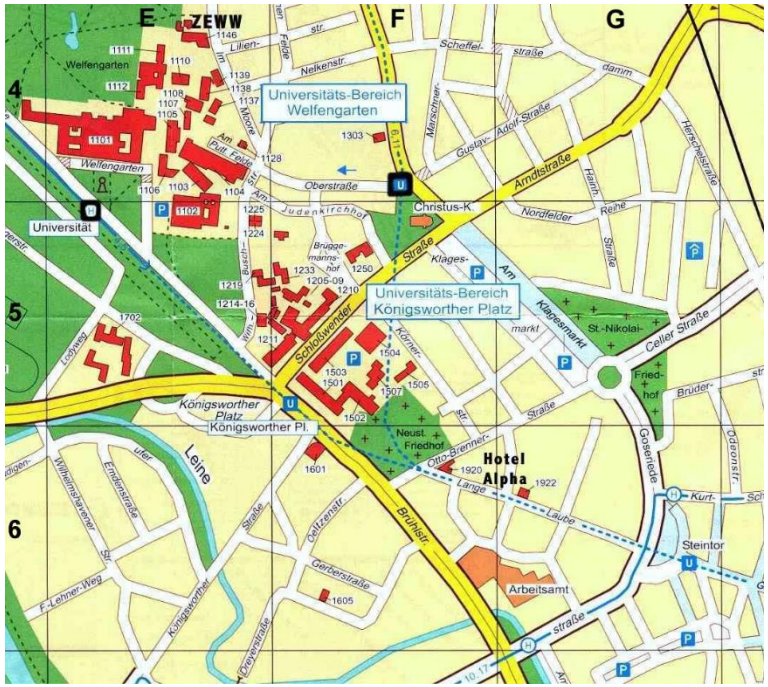
These are all operations you have seen before – the key thing here is that a spatial database must have data types and a query language which allow us to make such queries

Indexing spatial data

- A key problem with spatial data in databases is **indexing**
- For example, consider the query – "find all towns within 20km of Basel"
- The result set will include towns in France, Switzerland and Germany, so the **database** is likely to be **European**
- Comparing the **distance** of every town in Europe from Basel would take a very long time
- One solution would be to **calculate distances** and **index** those with a **B-tree** as above
- However, in a **spatial database**, we can **index** the **geometry** to make answering such queries more efficient
- We will look at **two common ways** of indexing spatial data

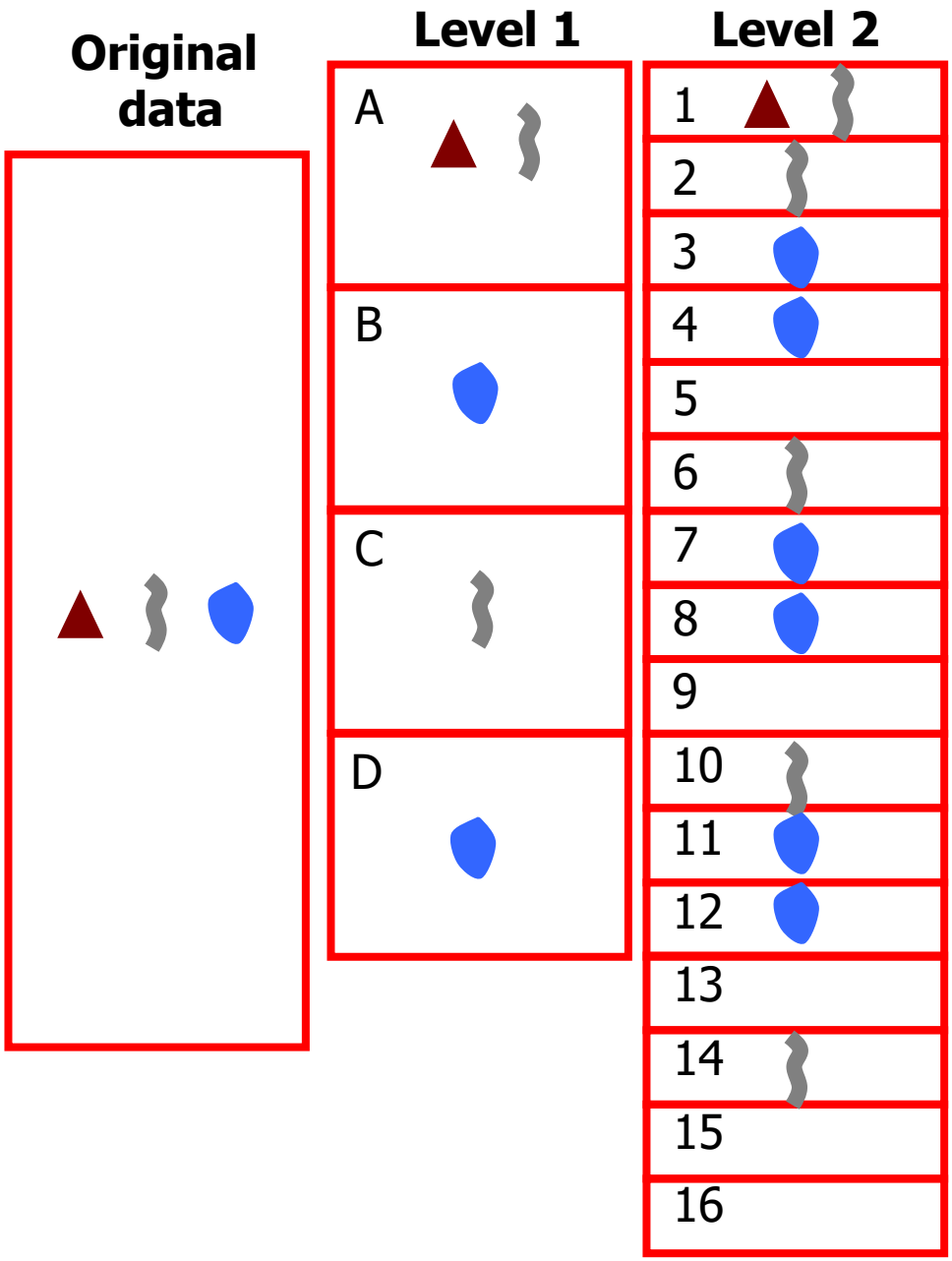
Grid indexes

- The simplest spatial index is known as the **grid index**
- Grid indexes are a **regular mesh** applied to geographic objects
- They are commonly used to **index street plans**
- The **grid size** should be chosen with respect to the objects stored
- According to Longley et al., **three grid levels** are sufficient for good performance

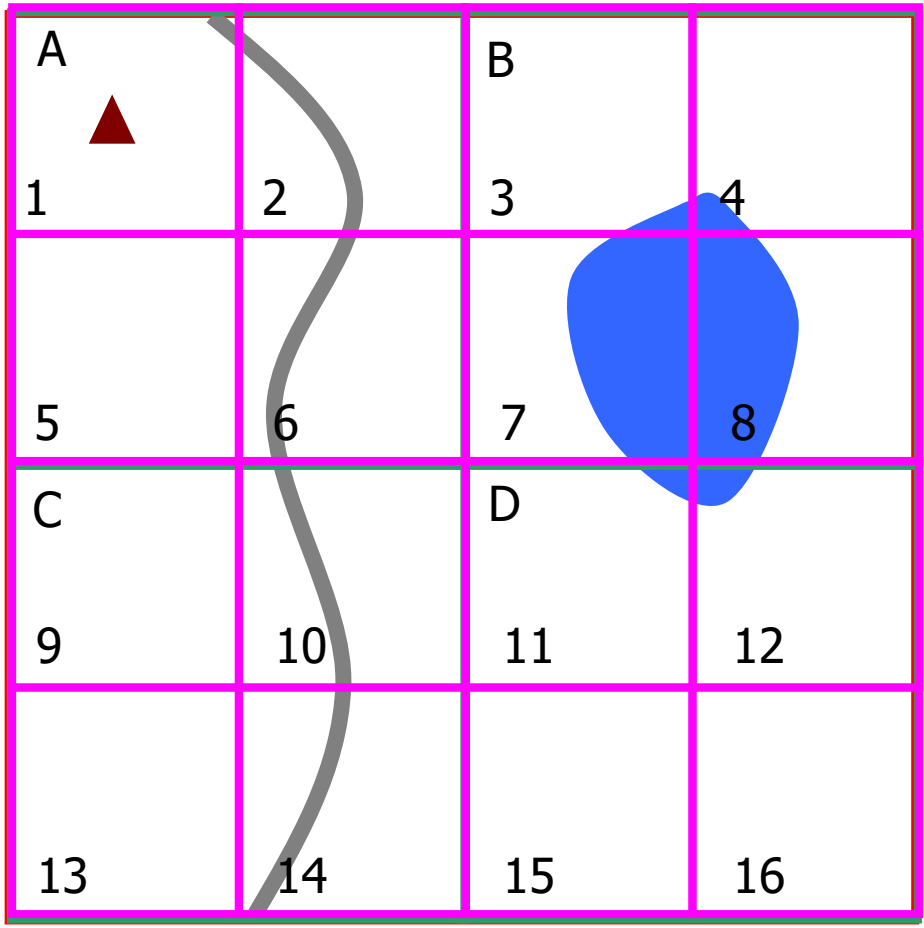


From: http://www.unics.uni-hannover.de/zeww/MapZEWW.jpg

Spatial indexes



Grid index example

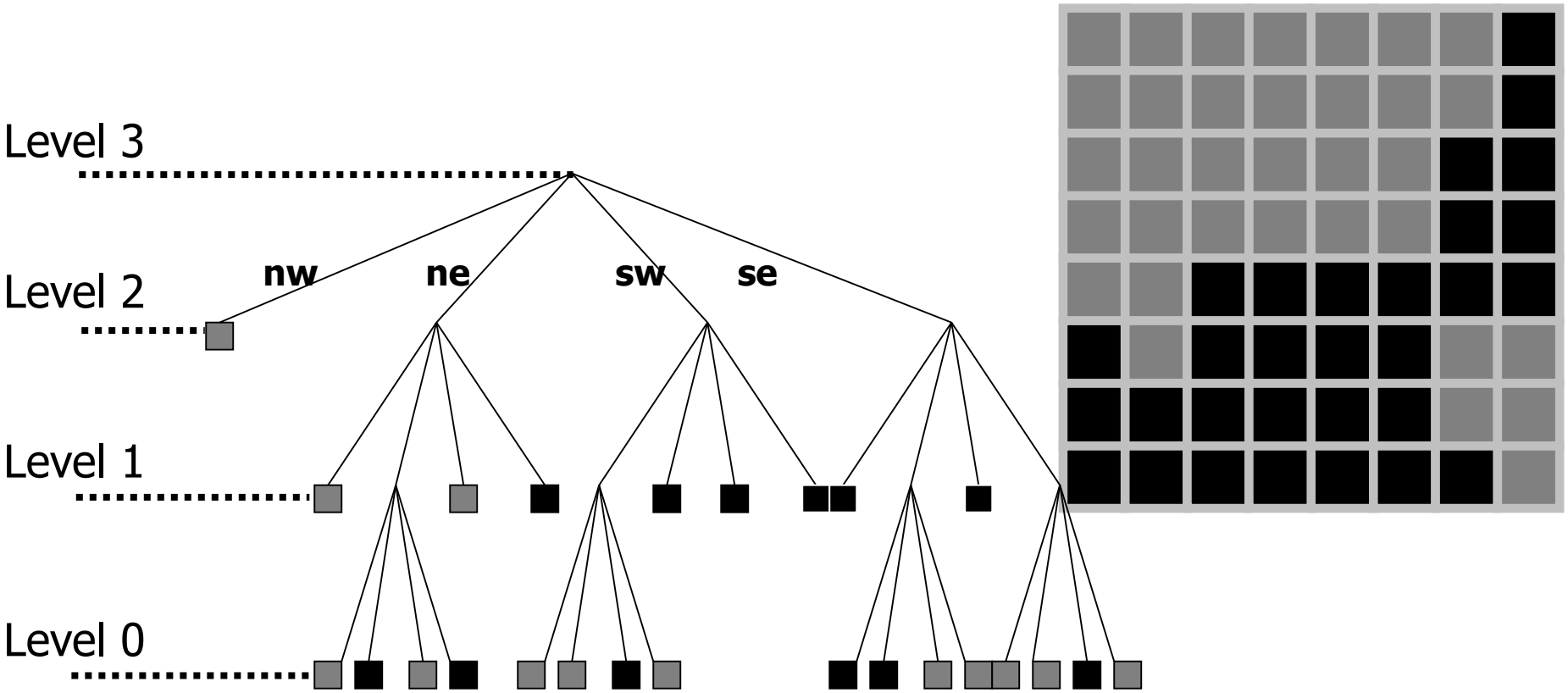


Introducing quadtrees

"a class of hierarchical data structures whose common property is that they are based on the principle of recursive decomposition of space" (Samet, 1989)

- We are interested here in quadtrees which allow us to represent regions of space where a value is constant - **region quadtrees**
- Region quadtrees are based on the **successive division** of a bounded area into **equal-sized quadrants**

An example quadtree

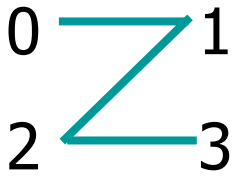


Some notes on our quadtree

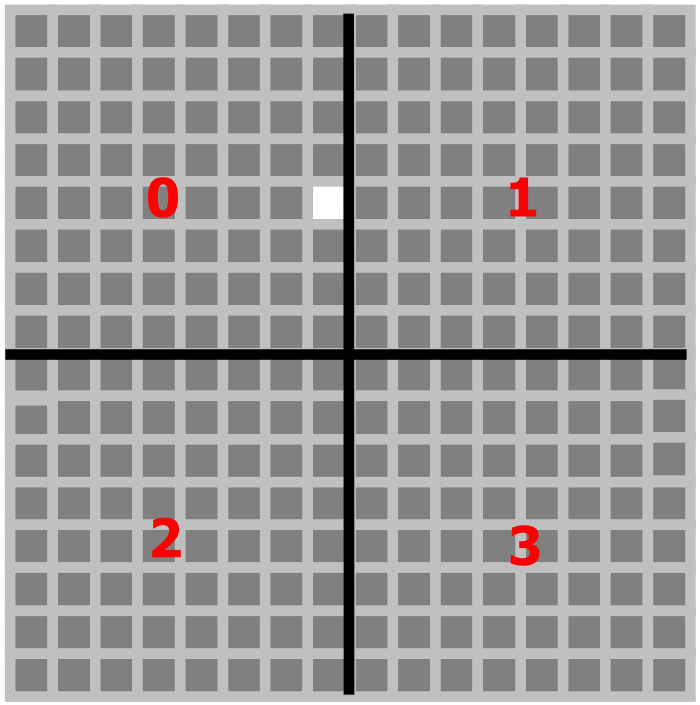
- The **division** of space in a region quadtree is **regular** -> other types of quadtree exist
- The quadtree only divides space further if it is necessary (c.f. grid index)
- The **stopping criteria** is that a quadrant is completely filled by an object
- For the raster case, a quadtree requires at most **n+1** levels, where, for a square raster, nRows = nCols = 2^n

Example - Creating a quadtree and its index

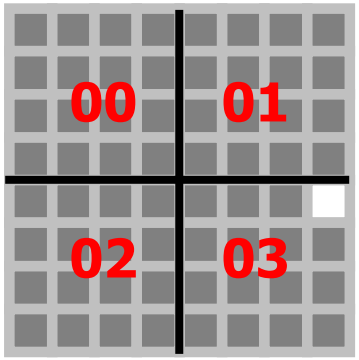
- The raster is progressively subdivided into equal sized regions, indexed using **Morton ordering**

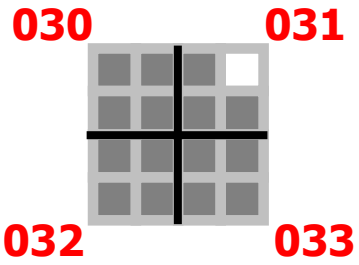


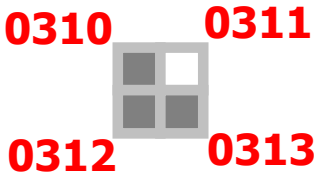
- If a region is homogeneous it need not be further subdivided



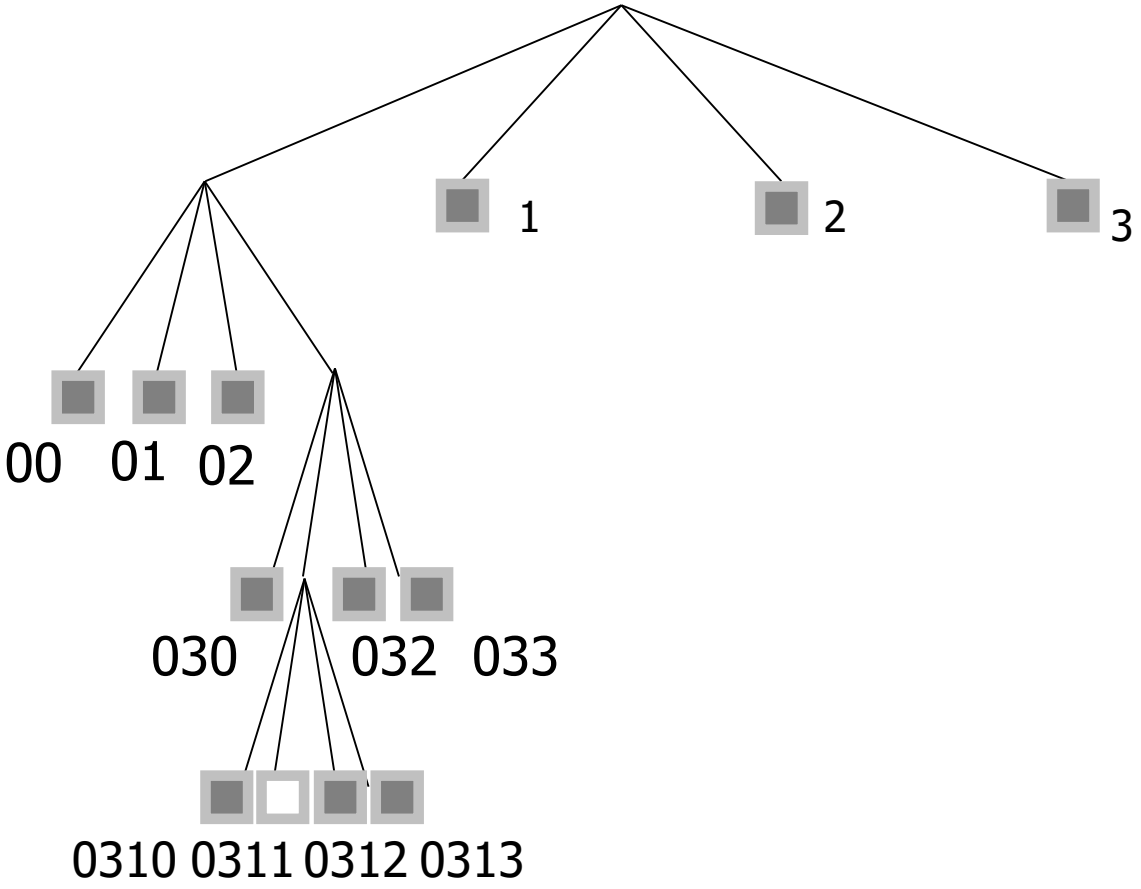
16 x 16 grid with one different cell at row 4, column 7







The final quadtree and index



Index

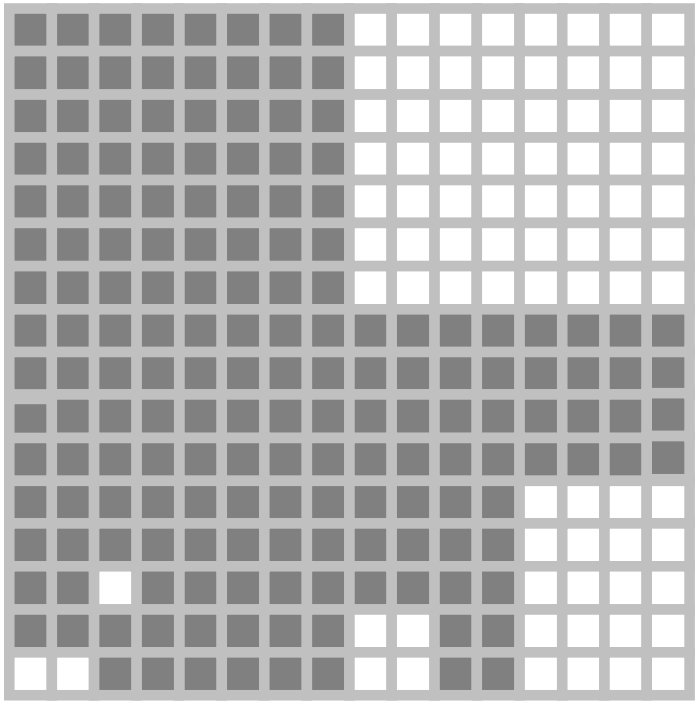
00[00]: Grey
01[00]: Grey
02[00]: Grey
030[0]: Grey
0310: Grey
0311: White
0312: Grey
0313: Grey
032[0]: Grey
033[0]: Grey
1[000]: Grey
2[000]: Grey
3[000]:Grey

The raster had dimensions $2^4 x 2^4$ and thus the quadtree has 5 levels

The final quadtree can be stored in a 1D index such as a B-tree allowing rapid access to individual values

Exercise

- Subdivide the following raster as in the previous example to create a quadtree
- How many values are stored in the final index?



Spatial databases - summary

- Spatial databases must represent **spatial database types**, allow **querying** and provide an **indexing mechanism**
- The **OGC specifications** are a good place to start investigating what a spatial database should provide
- Most spatial database implementations **simplify** elements of the problem
- Many **spatial indexing strategies** exist – **using any** will significantly improve performance

Summary

- I introduced some basic database concepts – if you found this interesting then think about the Masters courses
- In practical GIS applications, an understanding of the databases is nowadays essential (file-based approaches such as shape files are **not scalable**)
- DBMSs are the key tools for working with databases – we'll talk a bit more about their practical use next week when we look at **GIS and the internet**

References

- Burrough et al. 2015. Principles of Geographic Information Systems. Chapter 3, 3.3: Database structures: data organization in the computer
- Longley et al. 2015. Geographic Information Systems and Science. Chapter 9: Creating and maintaining geographic databases
- Güting. R.H. 1994. An introduction to Spatial Database Systems. VLDB Journal,3, 357-399.
- Samet, H. (1990a) The Design and Analysis of Spatial Data Structures. Reading, MA: Addison-Wesley
- OGC Simple Feature Specification (http://portal.opengeospatial.org/files/?artifact_id=18241)
- GITTA lesson "Einführung in Datenbanksysteme" (http://gitta.info/IntroToDBS/de/html/index.html)