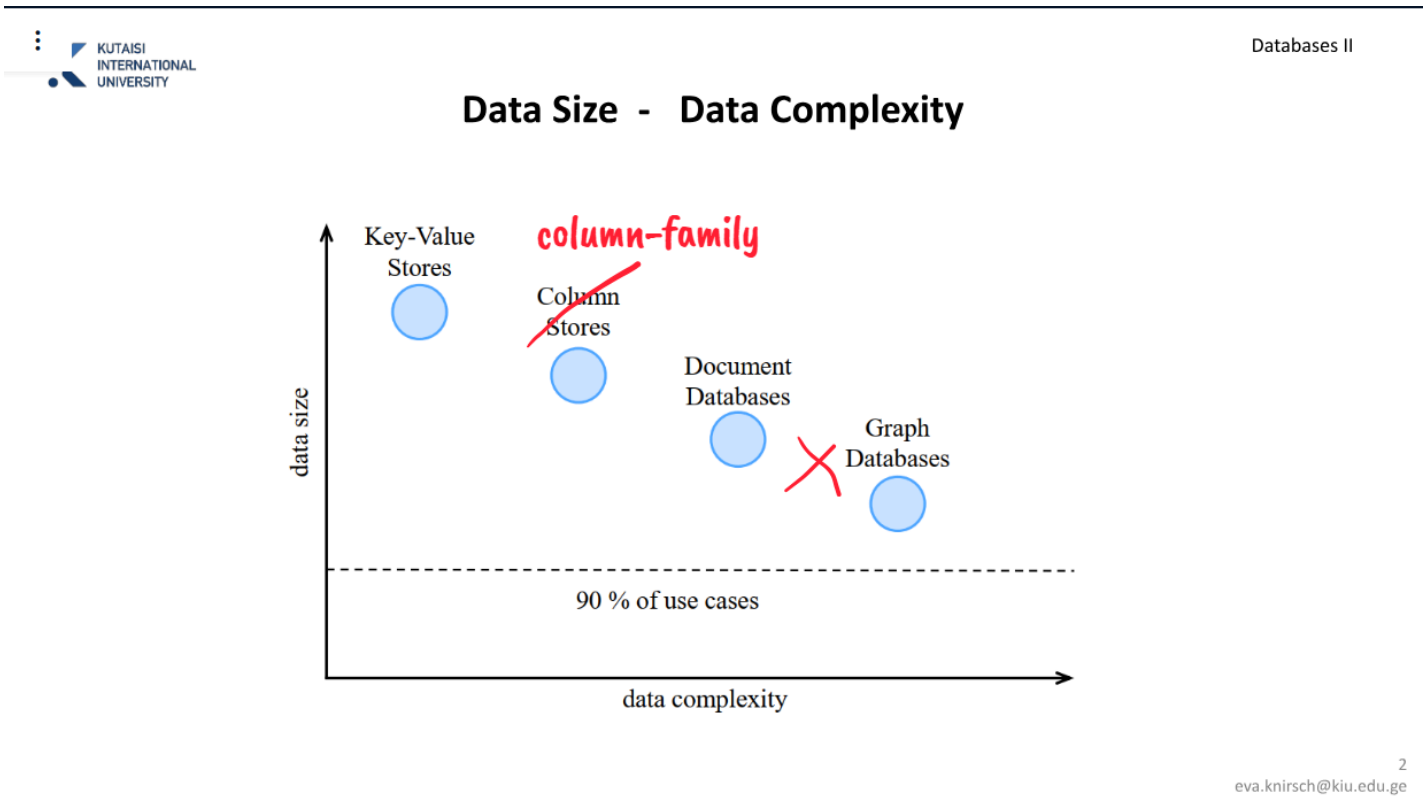
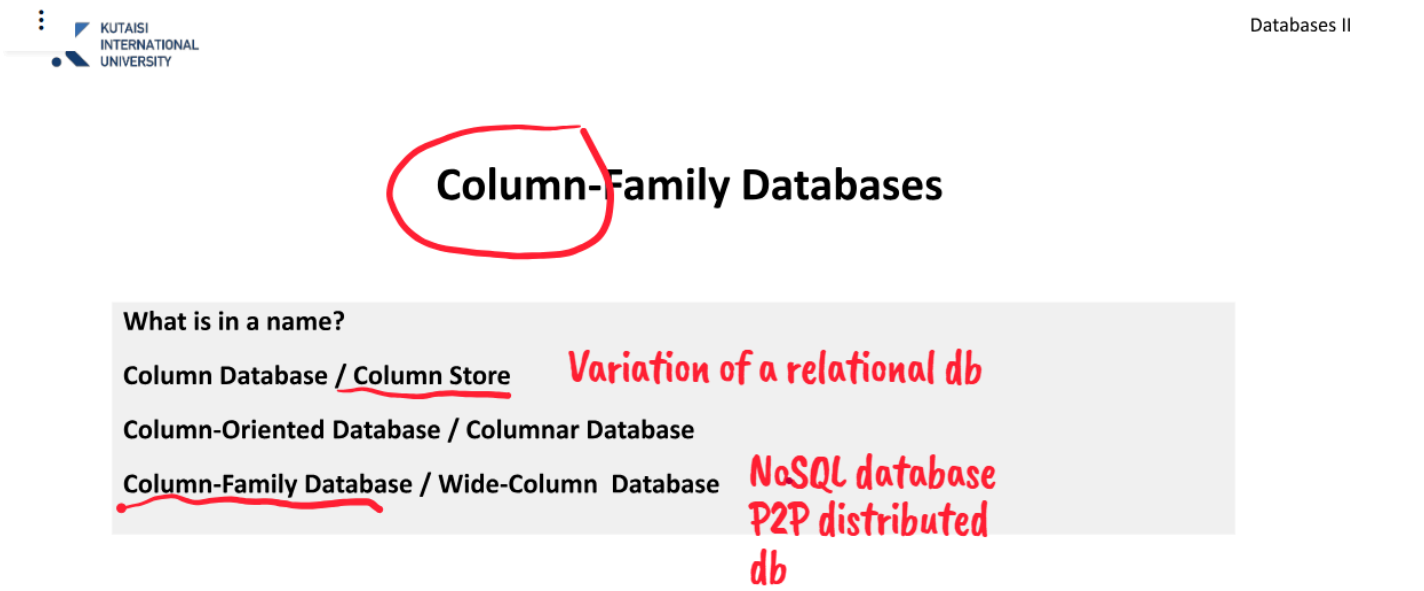
Lecture 14

Type: Lecture

Data Size - Data Complexity



relational databases on the right of the x-axis, on X



Row Format

- RDBMS are –logically - two-dimensional: columns (attributes) and rows (tuples).
- However, the storage of the data happens one-dimensionally – in a file (heap file or B-Tree).
  → how to map from two dimensions to one dimension for storage?

teacherID	name	postal	email	DoB	gender	education	remark
41	Krawinkel	10045	krawinkel@wir_wissen_alles.xx	1978-09-15	f	Master	
42	Kaiser	10045	kaiser@zirkus.xx	2001-11-30	f	Ph.D	
43	Kern	10045	kern@immer_da.xx	1977-06-11	m	Ph.D	
44	Schuster	10004	schusterl@flieg.xx	1975-03-27	f	Master	
45	Newman	10010	new@kriech.xx	1995-11-20	m	Ph.D	

heap file storage

Header, 41, Krawinkel, 10045, krawinkel@wir_wissen_alles.xx ,1978-09-15,f,Master, Header, 42, Kaiser, 10045, kaiser@zirkus.xx ,2001-11-30,f,Ph.D, Header, 43, Kern 4,3, 10045,_kern@immer_da.xx, 1977-06-11, m, Ph.D., Header, 44, Schuster 10004, schusterl@flieg.xx , 1975-03-27,f,Master, Header, 45, Newman, 10010, new@kriech.xx, 1995-11-20,m, Ph.D. Header, 41, Krawinkel, 10047,..

Column Store

teacherID
41
42
43
44
45

name
Krawinkel
Kaiser
Kern
Schuster
Neumann

zip
10045
10045
10045
10004
10010

rating
4
5
3
5
5

[41,42,43,44,45]
[Krawinkel,Kaiser,Kern,Schuster,Neumann]
[10045,10045,10045,10004,10010]
[4,5,3,5,5]

select * from teacher where teacherID = 44

How is this query executed?

array position

Why not store a TID with each column value?

Column Compression

profID
41
42
43
44
45

name
Krawinkel
Kaiser
Kern
Schuster
Neumann

zip
10045
10045
10045
10004
10010

rating
4
5
3
5
5

41,42,43,44,45;
Krawinkel,Kaiser,Kern,Schuster,Neumann;
10045,10045,10045,10004,10010;
4,5,3,5,5

- Columns often hold redundant values and / or a very limited number of different values. Those columns are suitable for compression.
- Popular compression method for columnar storage are bitmaps: each possible value gets a bitmap assigned. Each bitmap has same length as the column vector.
- Run-length encoding is used to further compact the bitmaps.

column

Bitmaps + Run-Length Encoding

- bitmap index on column rating with values {1,2,3,4,5}.
- The column vector has length 5 (5 tuples, Ids 41 - 45).
- Each rating value {1,2,3,4,5} gets a bitmap with length 5
- For each row, a 1 is filled into the bitmap if the row has that value; a 0 if it does not have the value.

Bitmap

	1	2	3	4	5
41	0	0	0	1	0
42	0	0	0	0	1
43	0	0	1	0	0
44	0	0	0	0	1
45	0	0	0	0	1

rating
4
5
3
5
5

What advantge does compression have?

Run_length Encoding

	5	5	2,1	0,1	1,1,1,2
	5zeros	5 zeros	2 zeros, 1 one, rest zero	0 zeros, 1 one rest z	1 zero, 1 one, 1 zero, 2 ones

for compaction. advantage of the compression is performance.

NoSQL

Column-Family Databases

RDBMS versus Column-Family Concepts

RDBMS:

schema
referential integrity
normalization --> minimize
redundancy --> consistency
join
ACID

"miniworld
"

Column-Family: P2P – token ring

schema
no referential integrity (no fks)
denormalized, redundant data
no joins --> the query needs to be answered by one table

Designing for optimal storage
Sorting is a design decision

CAP – AP

https://cassandra.apache.org/doc/latest/cassandra/developing/data-modeling/data-modeling_rdbms.html

Defining Application Queries

keyspace lesson

Q1: Return all information about a given, specific teacher

John

Q2: Return the students of a given, specific teacher

Ana

Q3: Return a teacher list by given, specific postalcode

How are the queries mapped on the components of the Cassandra data model:
→ a table for each query

- Rowkeys (=partition keys)
- column-families (attribute group)
- Columns (attribute – value)

controls the distribution onto the ring

Create Table Command for RDBMS (Postgres) and Column-Family (Cassandra)

Postgres:

```
CREATE TABLE IF NOT EXISTS public.teacher ((
   t_id integer NOT NULL DEFAULT
nextval('"teacher_tID_seq"'::regclass),
   t_name character varying(30) COLLATE pg_catalog."default" NOT
NULL,
   t_mail character varying(50) COLLATE pg_catalog."default" NOT
NULL,
   t_postalcode integer NOT NULL,
   t_gender gender,
   t_education education NOT NULL,
   t_payment integer NOT NULL DEFAULT 0,
CONSTRAINT teacher_pkey PRIMARY KEY (t_id)
)
```

Cassandra:

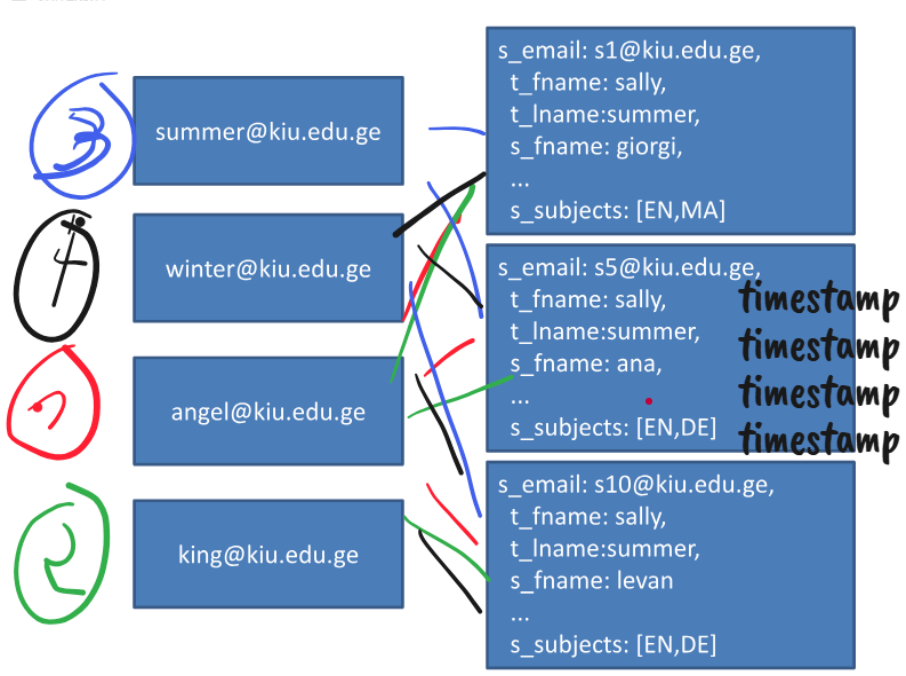
Table answers 1st query

```
CREATE TABLE lesson.teachers(
 t_email text,
 t_name text,
 t_phone text,
 address frozen<address>,
 t_subjects set<text> ,
 t_education text,
 PRIMARY KEY (t_email)
 WITH comment = 'Q1. Find information about a teacher';
```

Table definition in Postgres and Cassandra looks similar. That is the problem.
Concepts and behavior are very different → data design needs to be approached differently,
application development is different.

Partition key

Students



- Assume that these partiton keys all are hashed to the same node
- Sssume that the teachers have the same students
- What would the sorting in the memtable be?

flushed to disk

Use Case Examples

Storing user history of a service (e.g. streaming service, driving service)

```
CREATE TABLE history_by_user (
   user_id UUID,
   usage_timestamp TIMESTAMP, show_id TEXT,
   title TEXT,
   genre TEXT,
   duration_watched INT,
   location TEXT,
   PRIMARY KEY ((user_id), view_timestamp)   usage_timestamp
) WITH CLUSTERING ORDER BY (usage_timestamp DESC);
```

Other use cases:
- Global Press Agencies storing their articles / press releases by date / topic
- Social media: feeds by followed persons per user