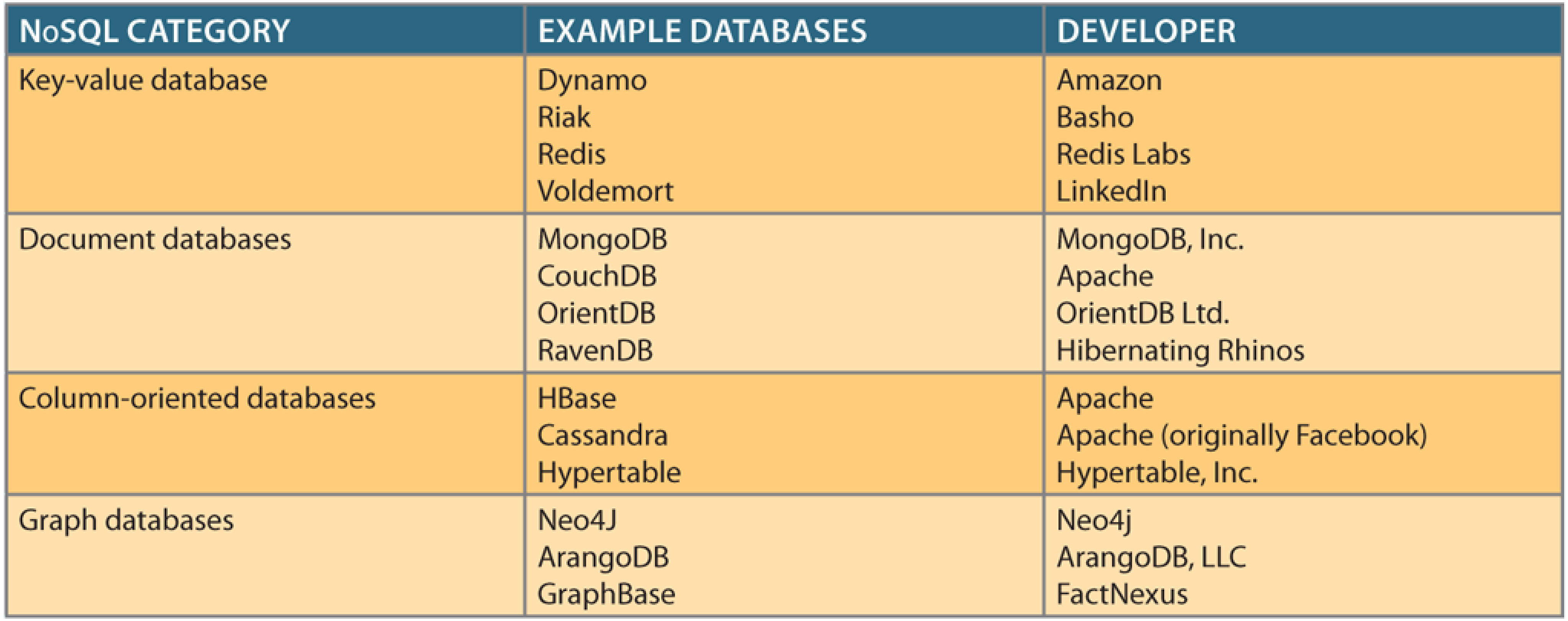
5
Data and Storage Models
Key-Value Databases
Fulltext Search Engines

Reading: [Me] chapter 7
[Ha], chapter 10, p. 158 ff
[Kl], chapter 3
[Pe], chapter 7

NoSQL databases

https://db-engines.com/en/ranking

Dynamo: "Prototype" that implemented NoSQL technology for the first time



NoSQL CATEGORY	EXAMPLE DATABASES	DEVELOPER
Key-value database	Dynamo Riak Redis Voldemort	Amazon Basho Redis Labs LinkedIn
Document databases	MongoDB CouchDB OrientDB RavenDB	MongoDB, Inc. Apache OrientDB Ltd. Hibernating Rhinos
Column-oriented databases	HBase Cassandra Hypertable	Apache Apache (originally Facebook) Hypertable, Inc.
Graph databases	Neo4J ArangoDB GraphBase	Neo4j ArangoDB, LLC FactNexus

Source: Coronel,Morris, Data Base Systems, ch. 14

Amazon DynamoDB

DynamoDB is not a classical key-value database, rather a key-value database with some wide-column database characteristics.

- If you want to learn about DynamoDB: create an Amazon DynamoDB educational account (no costs incurred). You will fiind a lot of learning material.
- If you want to play around with DynamoDB: create an Anmazon DynamoDB acccount
  Attention: you need to provide a credit card number! This service is – in principle – not free. However, you can set a cost threshold (e.g. 0 $) and trigger a warning message if you are about to exceed the limit.

Relational Database – Key-Value Database

When you create a table in a relational database, what attribute do you typically start with?

```
CREATE TABLE IF NOT EXISTS ......... customer
(
    ........................ customer_id integer/serial/autoincrement PK
    first_name
    last_name
    etc
```

you know the scheme and can query on values.

Key in a Key-Value Database

Often, the PK in a relational table is an artificial PK (e.g. autoincrement integer), that does not carry any meaning.

You can choose an artificial PK without any meaning because you have ample means to find the demanded values / rows without knowing the key:

- you know the scheme behind the table (datatypes, domains)
  → you know which data is in the columns
- you use where conditions on specific value or values ranges to get the needed data

With a key-value database, you have to think and start differently: There is no scheme and no searches with where conditions and the only way to retrieve specific values is the key. If you do not know the key or cannot deduct the key, you cannot read out the value.

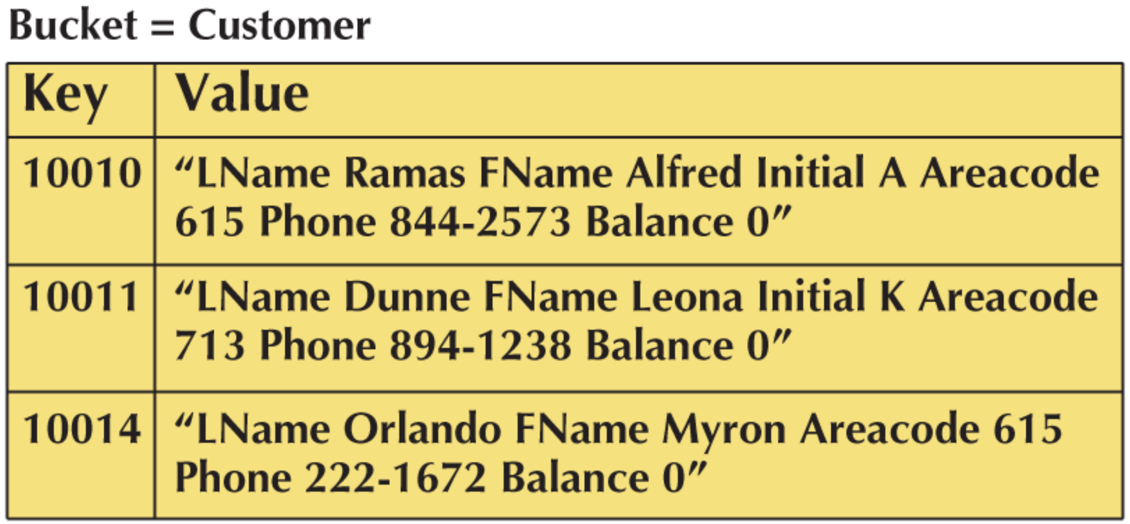
→ the key has a crucial importance for retrieving values.

→ it needs to be meaningful and descriptive.

→ it possibly may contain different components to make it more descriptive.

Key Value databases

Example1 :
Source: Coronel, Morris, Data base systems, ch. 14



Bucket = Customer

Key	Value
10010	"LName Ramas FName Alfred Initial A Areacode 615 Phone 844-2573 Balance 0"
10011	"LName Dunne FName Leona Initial K Areacode 713 Phone 894-1238 Balance 0"
10014	"LName Orlando FName Myron Areacode 615 Phone 222-1672 Balance 0"

Best Practice example?

NO!

we can't find Leona with the name, just the key

In the example it looks as if key and values are stored in a table again. However, this is not the case. This representation helps to understand the concept but does not represent the storage.

Key-Value Databases

- Simplest concept of NoSQL databases:
  - data is stored as key-value pairs
  - a data object (value) is assigned another data object (key) in order to be able to be referenced.
- Access on a value is not possible. A value can be accessed via the key only.
- Complex queries are not possible.
- A value does not need to have a certain format. It can be whatever the database allows: an atomic value, a text, a list, an array, a JSON document or a graphical element, …
- The application is responsible for managing the values (the contents) of the database. The database itself is ignorant about the contents. in contrast to SQL databases where the database knows the content
- In general, there are no relationships (and thus no constraints) between the key-value pairs.
  (RIAK, however, allows a link from one KV pair to another.)
- Some systems allow for iterating through the keys.
- There are no secondary indexes (search keys).

Advantages of this concept?

write operations are much faster -> velocity, volume

a lot of data comes in and needs to be stored somewhere, whether it stores in memory or disk, it's still much faster.

horizontal scaling

fast reads but no complex queries

Access to Data

Instead of a query language, a simple set of operations can be used to access the data. Basically, there are 3 operations in key-value databases:

- Set / Put / Post — insert / update
  If the key does not exist yet, a pair will be inserted. If the key exists, the value will be overwritten.
- Get — reads the value of a pair
- Delete — deletes an entire key-value pair

What is wrong with the example? Correct it.

- put(key, value)
- get(key, value) where value = "…" value=get(key)
- delete(key)

Key-Value Databases

we will do this right part of slide when we get to distributed databases, right now skip

RDBMS	Key-Value
database	cluster
table	bucket
tuple	key-value
PK / rowID	key

Creating / storing a key-value pair (Riak):

```
riakObj.setContentType('text/plain');
riakObj.setValue('German Shepherd');
client.storeValue({
   bucketType: 'animals', bucket: 'dogs', key: 'rufus',
   value: riakObj
}
```

```
Creating a bucket (Riak):
Bucket bucket = connection
.createBucket(bucketName)
.allow_mult(true / false)
.n_Val(numberOfReplicationCopies, default 3)
.last_write_wins(default false)
.w(numberOfNodesToRespondToWrite)
.r(numberOfNodesToRespondToRead)
.execute();
```

.allow_mult(): allows multiple possible values for an object. A single key can store multiple sibling values, caused by concurrent writes on the same or on different nodes.

→ by definition, sibling values conflict with each other

→ The application will need to develop a strategy for conflict resolution, i.e. the application will need to decide which value is more correct depending on the use case.

Key-Value Databases

RDBMS	Key-Value
insert into dogs ( ) values ( )	store(bucketType:animals, bucket:dogs, key:Stuppi, value:"..."
select * from dogs	does not exist
select * from table where PK = 'rufus'	get(key='rufus')
select * from table where breed = 'German Shepherd'	does not exist
update dogs set value= 'DoB 2023-12-01' where PK='rufus'	same as 1st command with different value
delete from dogs where value = ''DoB 2023-12-01'	does not exist
delete from dogs where PK = 'rufus'	delete(key='rufus')

Data is accessed using key-lookups. No locks, no joins, no constraints, no unions, no transactional support, ...

Key values databases

Example2
Source: Meier and Kaufmann, 2016, p. 203

```
SET User:U17547:firstname John
SET User:U17547:lastname Doe
SET User:U17547:email john.doe@blue_planet.net
SET User:U17547:pwhash D75872C818DC63BC1D87EA12
SET User:U17548:firstname Mina
SET User:U17548:lastname Maier
SET User:U17548:photo Mina.Maier.jpg
. . .
GET User:U17547:email
> john.doe@blue_planet.net
```

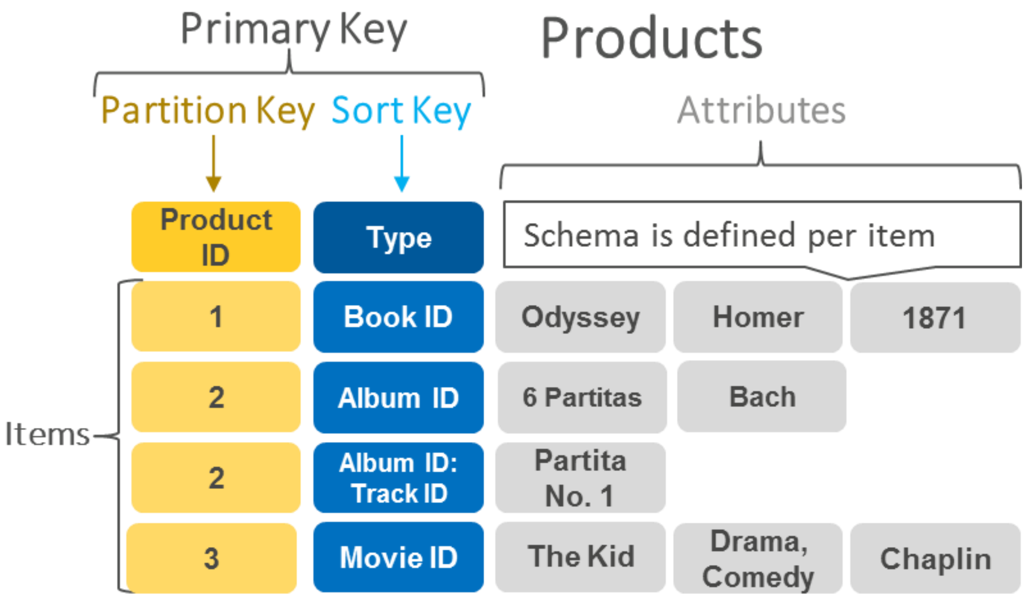
What is the key?

first key	User:U17547:firstname
second key	User:U17547:lastname
third key	User:U17547:email

Redis: Keys are strings that can hold any characters. Colon ':' is often used as a separator to make a key more descriptive. It is simplky a naming convention.

DynamoDB (Amazon)

A key-value database is a non-relational database that uses a simple key-value method to store data. A key-value database stores data as a collection of key-value pairs in which a key serves as a unique identifier. Both keys and values can be arbitrary elements - from simple objects to complex composite objects.



Primary Key

Products

Partition Key

Sort Key

Attributes

Items

Product ID	Type	Schema is defined per item		
1	Book ID	Odyssey	Homer	1871
2	Album ID	6 Partitas	Bach	
2	Album ID: Track ID	Partita No. 1		
3	Movie ID	The Kid	Drama, Comedy	Chaplin

Amazon Dynamo DB is a non-relational database. ... In DynamoDB, an element consists of a key and a flexible number of attributes (=values). There is no explicit restriction on the number of attributes associated with a single element, but the total size of an element, including all attribute names and values, cannot exceed 400 KB.

Use Case: Shopping Cart

Shopping cart applications carry a heavy write load: Users continously add, update and remove items – and a large number of users do this concurrently.

→ high velocity that SQL databases struggle with.

A KV database is much more performant: under the key, the shopping cart items are stored as value.

- What could the key be in this case? session_id
- When the value (shopping cart content) is updated (item added, updated or removed), following operations are done:
    - value = get(key)
    - put(key, updatedValue)

- Redis even has a list datatype and operations that directly add / subtract from the list.
- Once the user moves to checkout, the shopping cart is transferred to another - typically relational – database. Now, user needs to log in so that relational database application gets contact and payment data. Shopping cart items are inserted into a relational table and the purchase itself is done by an ACID ompliant database / application.

Use Case: Real-Time Counters (Gaming or Voting Apps)

- tracking of scores, votes or counters for a large number of concurrent players, voters,...
- what could the key be? user_id
- excellent velocity

- real-time counter update – based on old value:
    - value = get(key)
    - put(key, value)

- real-time counter update:
    - put(key, value)

Use Case: Storing Authentification or Session Information (Preferences, Personalized Marketing)

(Universally Unique Identifier) – A unique long identifier

- Key: session_id, UUID – stored e.g. in a cookie and in a KV
- Key could also be IP address

Sometimes used but not reliable (changes frequently)

- value holds authentification information
- value collects clicks, time on page, etc.
- when user returns: key is read out of cookie. Key gets value stored in KV → personalization

key is sent to the user's browser as a cookie

website updates the stored data for example what they clicked

server looks up the key in the KV store, retrieves stored data for
example login status, preferences, history...
And site personalizes content for the user.

Use Case: Sensor Data

- Example: controlling temperature in data centers
- key: location_id
- value: temperature
- update: every 10 seconds
- put(key, temperature value)

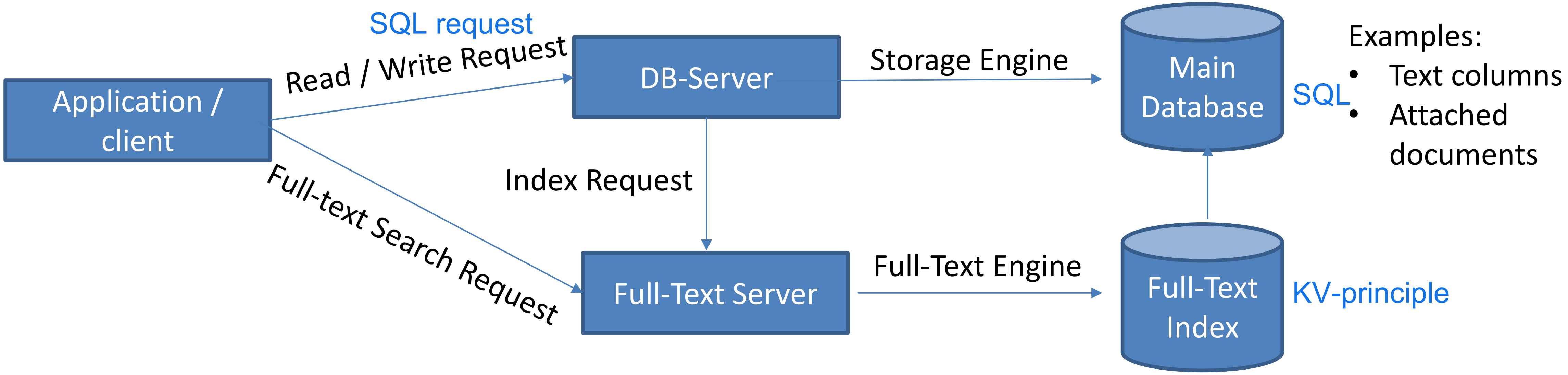
Summary: Use Cases for Key Values Stores

In general: Business cases with high velocity where the keys are enough to work with the data.

- Our Private Lesson Business model could not be implemented with a key-value store in a meaningful way. Why?

Use Case: Full Text Index

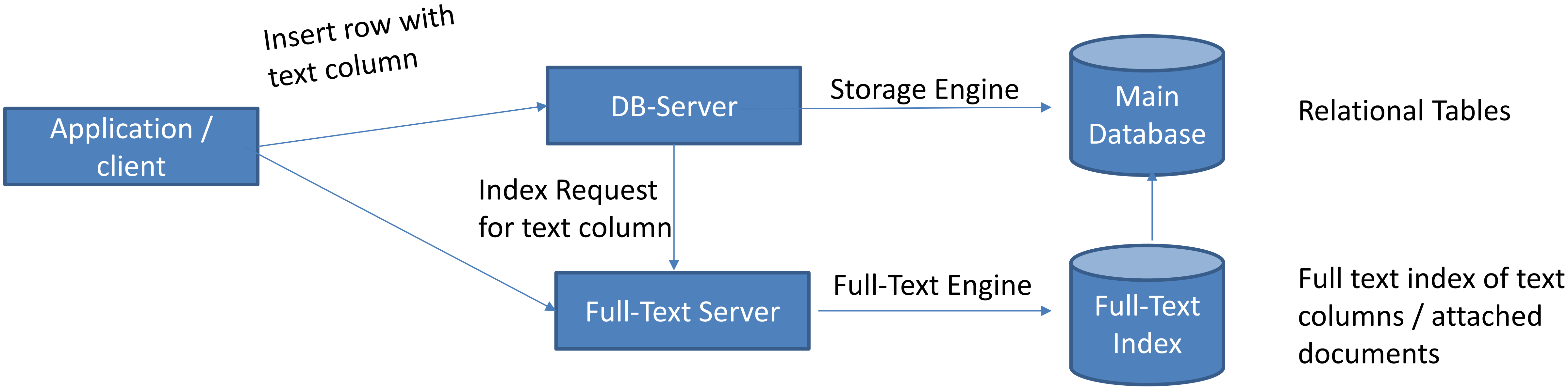
- Full-text indexer engines are based on the key-value concept.
- One could say that fulltext-indexer engines are a special – very complex – case of key-value stores.
- Some databases have a full-text engine / full-text storage, sometimes rudimentary. Other databases / applications partner up with Index and Search Engines.
  https://db-engines.com/de/ranking
- In the latter case, the full-text index usually is a separate database and a separate storage along the main database.



When we say that a full-text index works like a key-value store, then what are the keys and what are the values?

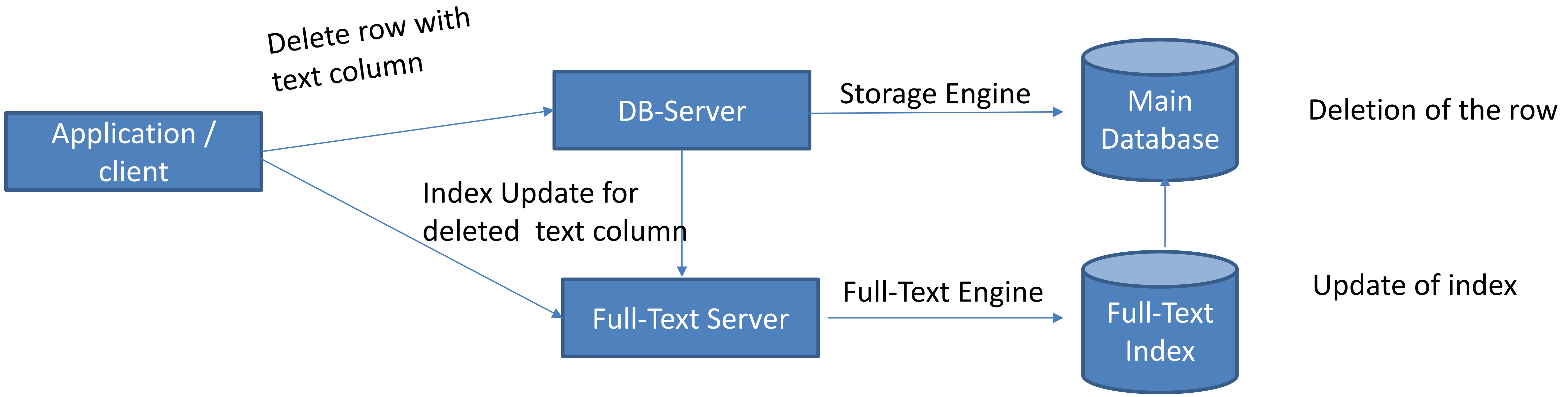
keys are search terms, values are locations

Use Case: Full Text Index



Example: insert of a row into teacher table with a long text in column remark (datatype: text)

Use Case: Full Text Index

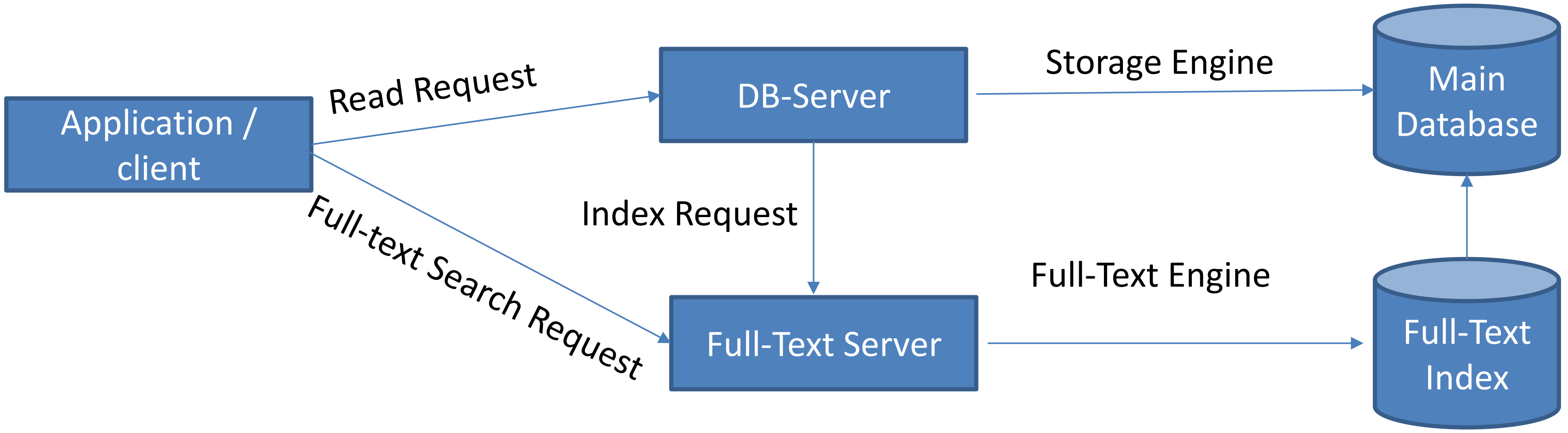


Example: Delete a row out of teacher table thatn has a long text in column remark (datatype: text)

Full Text Index

SQL Query: select * from teacher where postalcode = 4600

FTS: Return all rows that contin 'experience' AND 'success' in column remark



full text never searches in main database, it would be too slow.
it always searches in this inverted list.

Full-Text Index and Search Engine

- Objects that need to be full-text indexed are passed to the full-text indexer machine.
- The full-text indexer creates an index (inverted file / inverted index) of the searchable terms in the term dictionary (full text index).
- The inverted index is the keys.
- Each term in the full text index keeps a list (postings list) that contains all documents / objects and positions in which the term appears.
- Key-Value Pair: term (key) – postings list (value)

- Whenever new objects need to be indexed and added to the full text index, the full text index must be updated. This can be done periodically or immediately.
- Widely-used indexer / search engines: Apache Lucene, Elasticsearch, Sphinx, Mindbreeze

  https://db-engines.com/de/ranking

Inverted Index

- Each unique key (searchable term) is associated with a list of locations (pointers) where that term appears in the text or texts.
  PostgrerSQL:
  - In order to fulltext index a text column, another column of type ts_vector needs to be added to the table.

    inverted list
  - The ts_vector column needs to be populated with the key-value pairs (the indexed form of the row text) for each row. Ts_vector is essentially a transformed version of the text data.
  - A GIN index (Generalized Inverted Index) is created on the ts_vector column.
    It stores a set of (key, posting list) pairs, where a *posting list* is a set of row IDs in which the key occurs. The same row ID can appear in multiple posting lists. ... Each key is stored only once, so a GIN index is very compact for cases where the same key appears many times.

since we want to search whole table and not row by row we use GIN index.

- Internally a GIN is implemented as ....BTree in postgres...

Inverted Index

- teacher table has a text column (t_remark) that we want to be fulltext indexed.
- We populate the column t_remark with some texts.
- ALTER TABLE teacher ADD COLUMN **t_remark_tsvector** tsvector;
  - column is of type tsvector
  - Contains **for each row** the key-value pair representation of the field t_remark
- UPDATE teacher SET t_remark_tsvector = to_tsvector('english', t_remark);
  Populate the column of type tsvector with key-value pairs built out of column t_remark, using function to_tsvector
- CREATE INDEX t_remark_gin on teacher using GIN(t_remark_tsvector); This will allow fast search queries on the t_remark_tsvector column.
- What does this command do? How to undo?
  UPDATE teacher SET t_remark_tsvector = to_tsvector('english', 'About relational databases: ')
  WHERE t_id = 3;

we will lose what's written in t_id=3 ts_vector and override it with indecies of 'About relational databases:'

to undo we repeat original query

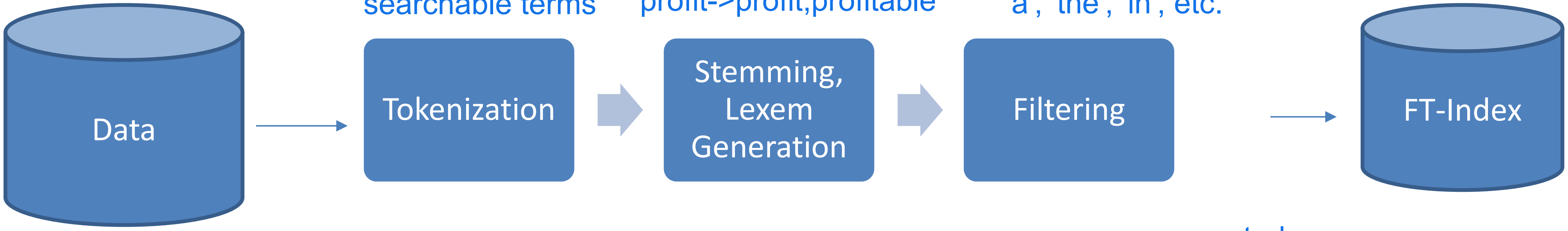
to undo the update and restore the original tsvector for that row we can re-run the original query for that specific row.

How to get Key-Value Pairs out of a Text:

text is split into searchable terms

reducing search term to stem profit->profit,profitable

throwing out words that are not needed like 'a', 'the', 'in', etc.

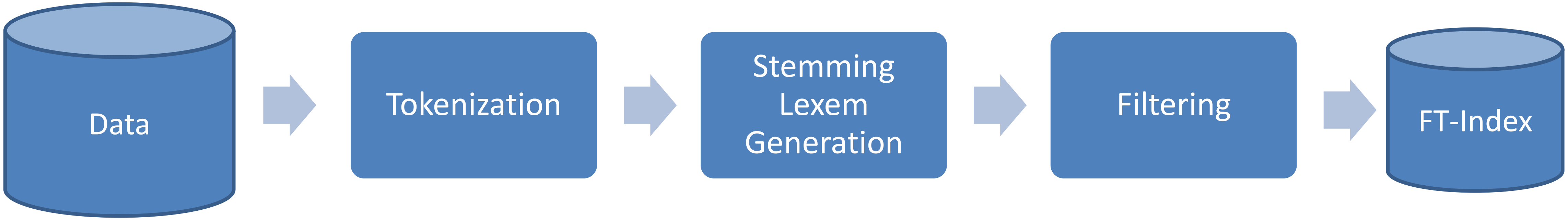


About relational databases: As computers became vastly more powerful and networked, they started being used for increasingly diverse purposes. And remarkably, relational databases turned out to generalize very well, beyond their original scope of business data processing, to a broad variety of use cases. Much of what you see on the web today is still powered by relational databases, be it online publishing, discussion, social networking, ecommerce, games, software-as-a-service productivity applications, or much more

sorted

```
'applic':75 'becam':6 'beyond':30 'broad':40
'busi':35 'case':44 'comput':5 'data':36
'databas':3,23,59 'discuss':64 'divers':18
'ecommerc':67 'game':68 'general':27
'increas':17 'much':45,77 'network':11,66
'onlin':62 'origin':32 'power':9,56
'process':37 'product':74 'publish':63
'purpos':19 'relat':2,22,58 'remark':21
'scope':33 'see':49 'servic':73 'social':65
'softwar':70 'software-as-a-servic':69
'start':13 'still':55 'today':53 'turn':24
'use':15,43 'varieti':41 'vast':7 'web':52
'well':29
```

How to get Key-Value Pairs out of a Text:



To get from text to ts-vector format:

Tokenizer: Parsing text into tokens

Stemmer: Converting tokens into stems (lexems)

Filter: Eliminating stop words

Full-Text Index Engine

Here is a small collection of documents that need to be full-text indexed:

- docID1:
  Titanic was the most profitable blockbuster ever. Kate Winslet and Leonardo di Caprio leaning against the railing has become the symbol for a truly romantic moment.
- docID2:
  Leonardo di Caprio says in the making-of “The Great Gatsby” that he played as good as in Titanic, but the film never got around to become a true blockbuster. But then, is it necessary for a film to be a blockbuster?

What would the term dictionary and the postings lists look like?

MongoDB Tokenization

Tokenization Delimiters

For tokenization, version 3 text index uses the delimiters categorized under Dash, Hyphen, Pattern_Syntax, Quotation_Mark, Terminal_Punctuation, and White_Space in Unicode 8.0

For example, if given a string "Il a dit qu'il «était le meilleur joueur du monde»", the text index treats «, », and spaces as delimiters.

Previous versions of the index treat « as part of the term "«était" and » as part of the term "monde»".

https://www.mongodb.com/docs/manual/core/index-text/

PostgreSQL Fulltext

- PostgreSQL also includes fulltext indexer and fulltext search.
- PostgreSQL by default stores the fulltext index per row in the tables as extra key-value column of type ts_vector
- A GIN indedx (B-tree) on the ts_vector column then builds the fulltext index out of all ts_vector rows as fulltext index of the whole table or for multiple tables.

To perform a fulltext search:

Functions to_tsvector() and to_tsquery are used together with matching operator @@

Ad-hoc full text search:

SELECT * FROM table WHERE to_tsvector(text column) @@ to_tsquery('searchterm');

Full text search on existing column of type ts_vector
SELECT * FROM table WHERE ts_vector_column @@ to_tsquery('searchterm');