Lecture 6

Type	Lecture

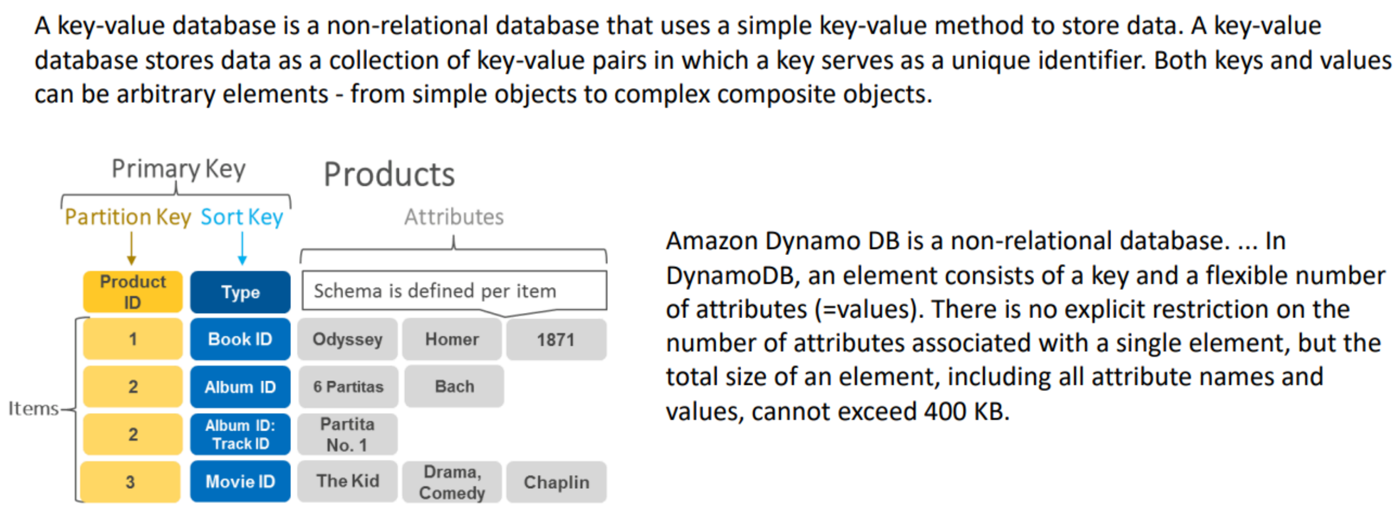
NoSQL databases

NoSQL includes key-value, document, wide-column, and graph databases.

- Designed for scalability, high performance, and flexible data models.

Amazon DynamoDB

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



Relational Database – Key-Value Database

Relational databases often use artificial primary keys and support schema-based searches.

In a relational database, when you create a table, you typically start by defining a primary key (PK)— an attribute that uniquely identifies each row in the table.

```
CREATE TABLE IF NOT EXISTS users (
  user_id INT PRIMARY KEY AUTO_INCREMENT,
  username VARCHAR(50),
  email VARCHAR(100)
);
```

Key-Value Database

Key in a Key-Value Database

Keys must be meaningful and descriptive because they are the only access method. Without knowing the key, the data is unreachable (no `WHERE` clauses or joins).

Key-Value Databases

Key-Value (KV) databases are the simplest form of NoSQL systems. Data is stored as pairs: a unique key and its associated value.

Important:

- No fixed structure (schema-free).
- Cannot search by value — only by key.
- No joins, constraints, or relationships between pairs.

- The application handles all logic related to how values are interpreted.
- Examples include Redis, Riak, and DynamoDB.

Good Practices:

- Make keys descriptive and unique.
- Use consistent naming conventions (e.g., `user:123:name`).
- Avoid storing sensitive data in the key.

Access to Data

- 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

put(key, value) get(key, value) value=get(key); delete(key)

Use Case — Shopping Cart

High write operations (add/update/remove).

Important: KV stores are ideal for temporary, frequently changing data.

Use Case — Real-Time Counters

Tracks scores or votes instantly.

Important: High speed and minimal conflict resolution required.

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

Session ID or UUID is the key.

Important: Enables personalization by quickly fetching session data.

Use Case: Sensor Data

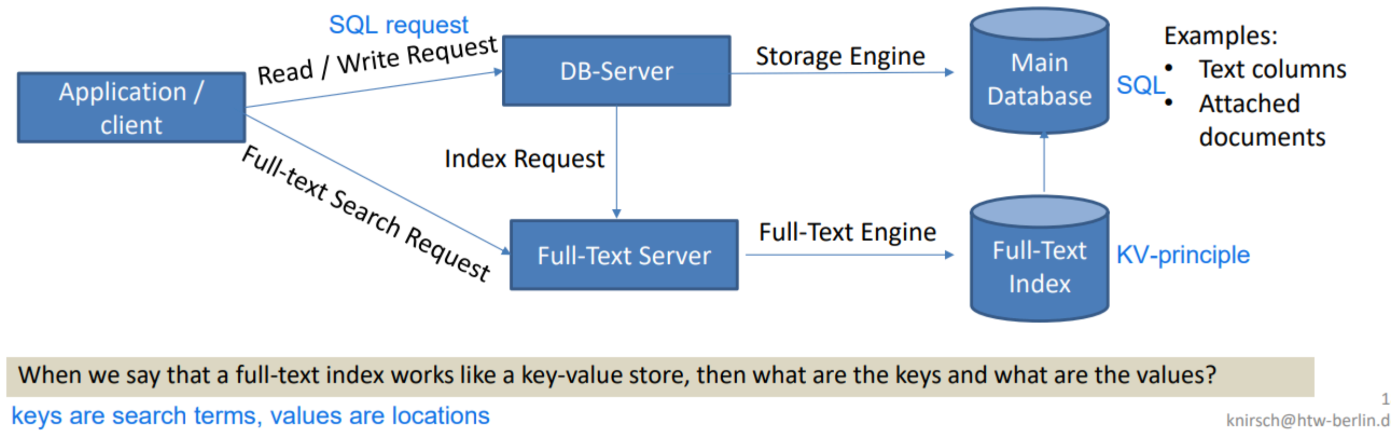
Periodic updates like temperature readings.

Important: Suited for IoT and monitoring systems.

Use Case: Full-Text Index Concepts

Full-text engines use key-value logic to map terms (keys) to documents (values).

Important: Often integrated with or run alongside main DBs.



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

Inverted Index

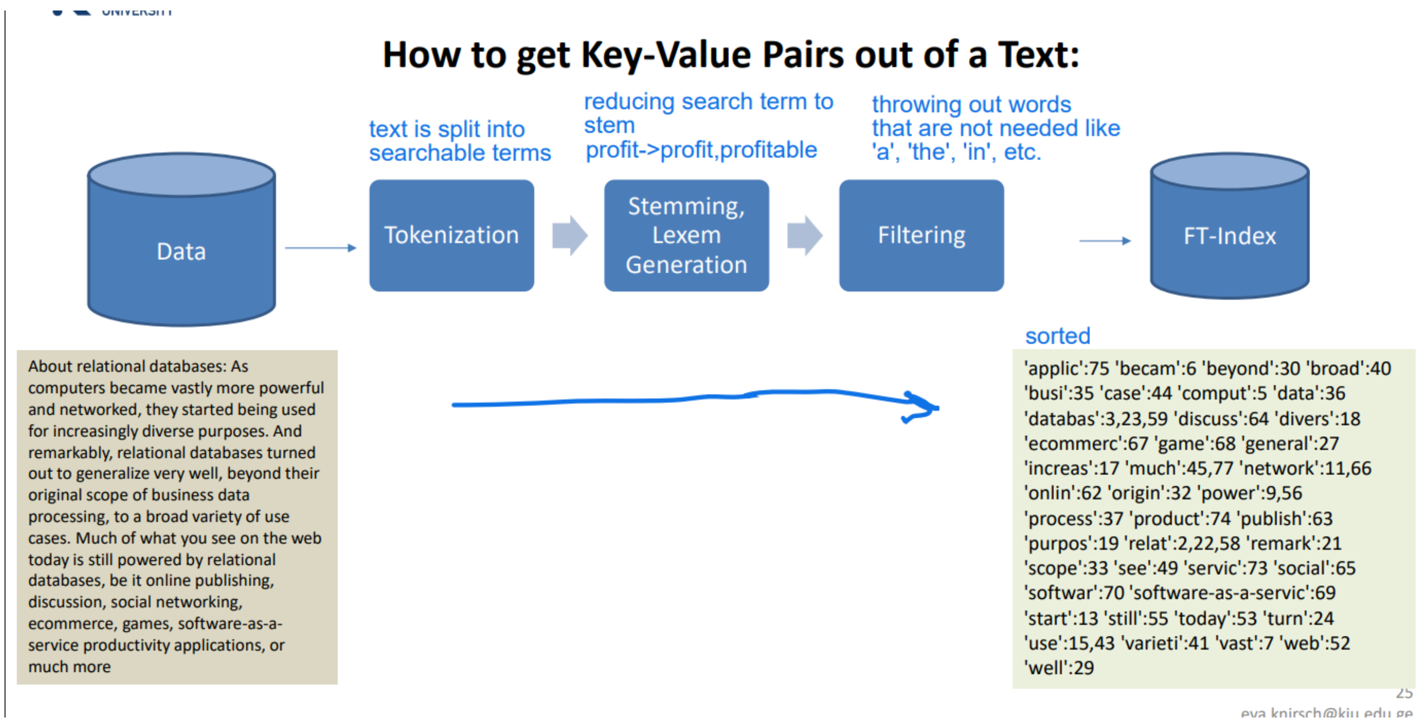
Each unique key (searchable term) is associated with a list of locations (pointers) where that term appears in the text or texts.

`tsvector` and GIN indexes build the full-text search.

Important: Efficient for indexing and searching large text data.

Tokenization and Text Processing

Text is broken down into searchable parts (tokens, stems, lexemes). Filtering removes common words, improving search quality.



To get from text to ts-vector format:

Tokenizer: Parsing text into tokens

Stemmer: Converting tokens into stems (lexems)

Filter: Eliminating stop words

26

PostgreSQL Fulltext

Use `to_tsvector()` and `to_tsquery()` to search text columns. GIN indexes provide fast lookup for matching terms.

INTERNATIONAL UNIVERSITY

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');

29