DB fundamentals overview

Denis Miginsky

Data modification

Data inside the DB are usually not read-only.

It can be appended (new rows can be inserted into the DB), updated (existing rows can be modified), removed.

In SQL INSERT, UPDATE, MERGE, DELETE queries can do this.

The new problems:

- How to keep data in the **consistent** state?
- How to organize data on the disk drive?

Consistency

Low-level consistency:

- All the values correspond to their types (INT, TEXT, etc.)
- All the low-level data representation requirements are met (like low/big-endian, encoding, etc.)
- All the references should point to existing objects

High-level consistency:

- All the PKs are unique
- All other DB constraints are met (unique, non-null, FK, etc.)
- All other domain constraints are met, even if they cannot be directly supported by the DB

Durability

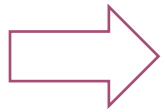
Durability (or object persistence) – the property of the data to survive the restart of the software managing the data.

Problems:

- Data could be much larger than the RAM size, the durability must be applicable to the fragments of the data.
- Data fragments must be accessible separately from each other on the disk drive, with O(1) complexity
- The disk space consumption must be as close as possible to the original data size

Low-level data organization

WARE	CLASS
Meat	Food
Meat	Raw food
Steak	Food
Steak	Ready food



0x00	M	e	a	t	\0	F	o	o	d	\0	M	e	a	t	\0	R
0x10	a	w	_	f	o	o	d	\0	S	t	e	a	k	\0	F	o
0x20	o	d	\0	S	t	e	a	k	\0	R	a	w	_	f	o	o
0x30	d	\0														

```
UPDATE CATEGORY SET CLASS='Any food'
WHERE CLASS='Food'
```

Any food

0x00	M	e	a	t	\0						M	e	a	t	\0	R
0x10	a	w	_	f	o	o	d	\0	S	t	e	a	k	\0		
0x20				S	t	e	a	k	\0	R	a	w	_	f	o	o
0x30	d	\0														

Q: How to organize data in the memory?

Problem: fragmentation

Problem: access complexity

Memory consumption

Even the original data can be much larger than the RAM size and cannot fit it. And with additional indices, intermediate results, etc. …

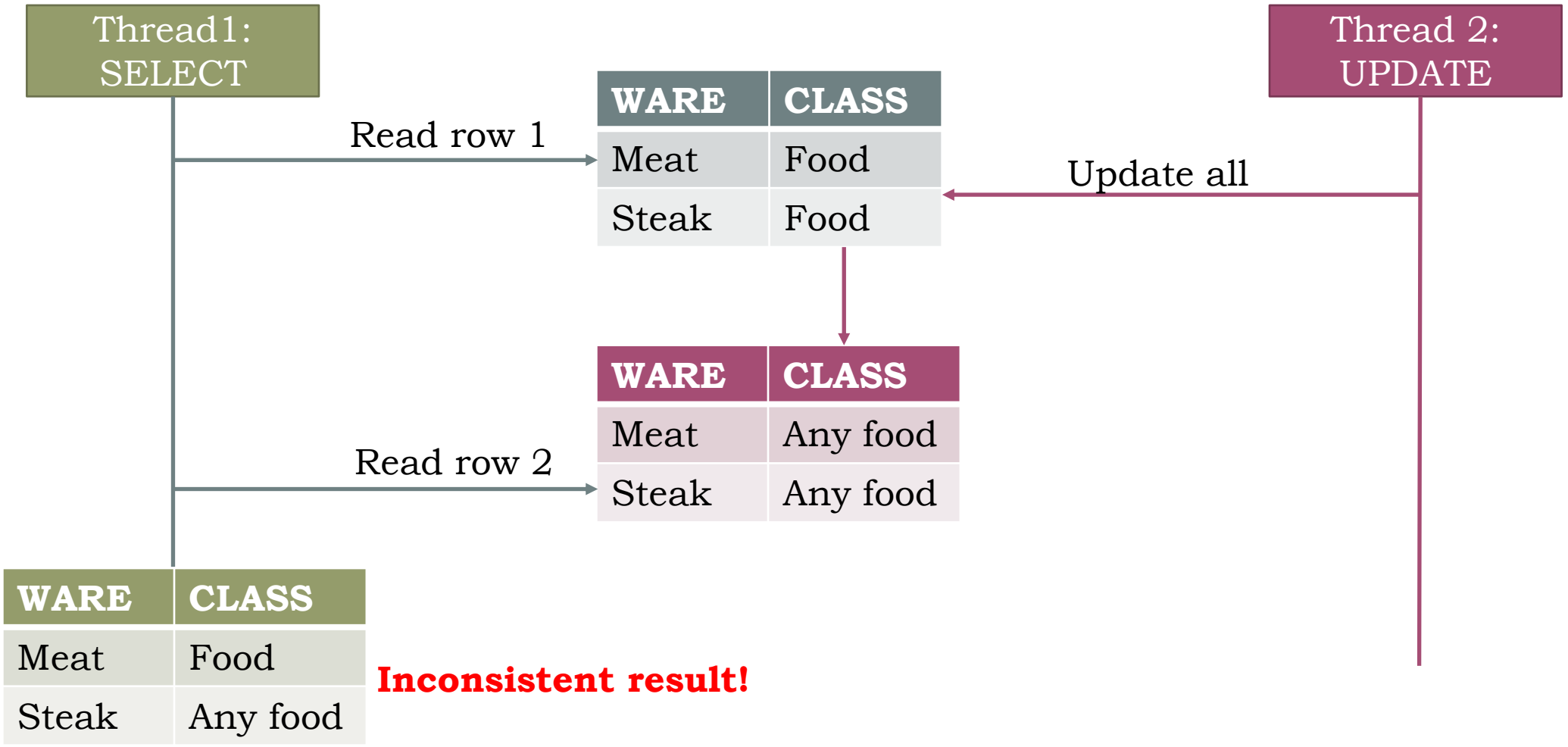
On the other hand, RAM must be used as much as possible due to the significantly higher performance in comparison to disk drive.

Problem: what shall we put into the memory?

Solutions:

- Laziness for the intermediate results where possible
- Different caching strategies for the initial and other durable data

Concurrency



Transactions

A transaction in general is a group of operations that either shall be completed entirely or nothing shall be completed at all.

Properties of transactions (ACID):

- **Atomicity** – all the operations in the transaction work as a single unit
- **Consistency** – transaction shall bring the system from one consistent state to another one
- **Isolation** – the effect of some **concurrent** transactions shall be equivalent to the application of them in the sequential way (in some order).
- **Durability** – as before. This property is essential for the DB transactions, but optional in general case.

Performance

	Pros	Cons
Normalization	Reduces redundancy, usually improves performance	Sometimes reduces performance
Indices	Improves performance of SELECTs	Reduces the modification performance, more complex maintenance
Advanced algorithms	Improves performance	Limited functionality, makes the planner AI more complex
Database statistics	Helps the query planner	Makes the planner AI much more complex
Transactions	Essential for consistency and concurrency	Complex implementation, reduces performance

Query planner

Artificial intelligence!!!

Alternative data models

The most notable data models:

- Relational
- Key-value
- Hierarchical
- Graph

They are equivalent to each other: if the data can be represented in one model, it can be converted to any other. However, the convenience, redundancy and efficiency could vary.

OLAP and OLTP

Main DB usage patterns:

- **OLAP** (On-Line Analytical Processing) – DB with almost static data and complex read queries.
- **OLTP** (On-Line Transaction Processing) – DB with many simple read and write queries.
- **Hybrid**

Distributed DBs

What for?

- Larger space
- Scalability and high availability

Problems:

- How to distribute the data across the nodes?
- How to achieve good performance on complex queries?
- How to implement ACID?

Key-value

Just a **map** (or multimap)

Pros	Cons
The simples model, easy to implement and use	Bad for OLAP
Good for massive OLTP	Its tricky to use for data with the complex structure
The best model for the distributed DB	

Implementations: BerkleyDB, Amazon Dynamo, Apache Cassandra

Hierarchical

A forest (in general) with possible value(-s) in each tree's node. Queries can traverse trees up and down and extract values from nodes.

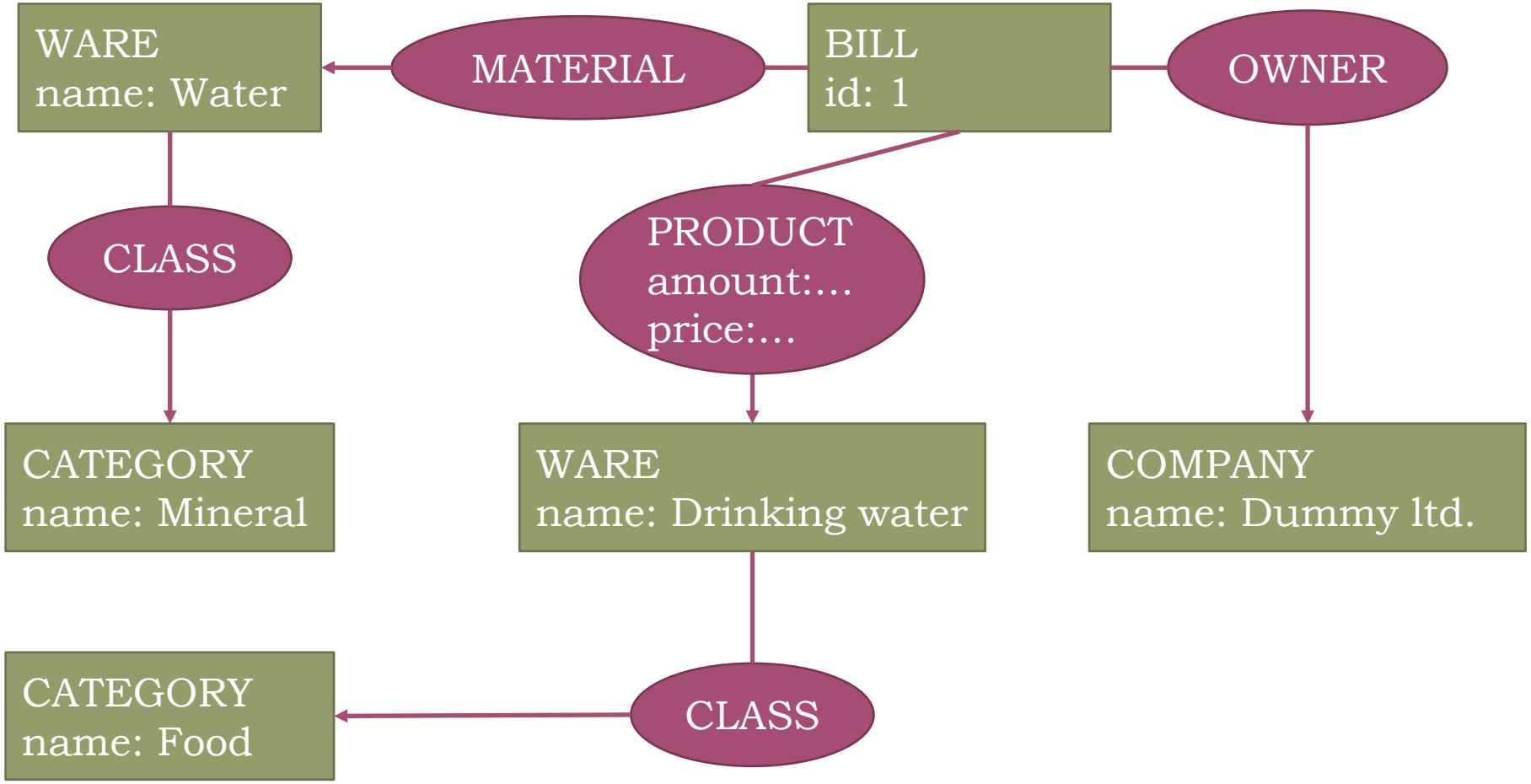
The important subset are document-oriented DBs (for XML, JSON, etc.).

Pros	Cons
More complex structure and queries supported	Still bad for OLAP
Good for OLTP	Still tricky to use for data with the complex structure
Not bad for distributed DB (depends on the forest structure)	

Implementations: MongoDB, OrientDB, CouchDB

Any others?

Property graph: example



Property graph DB

Pros	Cons
General purposes DB (in theory) as well as relational DB	Bad for the distributed DB with analytical queries.
Complex structure that is usually more convenient than relational	For now the performance is worse than in relational DBs.
Variety of different implementations with different query languages and either static or dynamic schemas	No formal criteria for normalization yet

Implementations: Neo4j, DataStax, Amazon Neptune

Query languages: Cypher, Gremlin

Example: Cypher

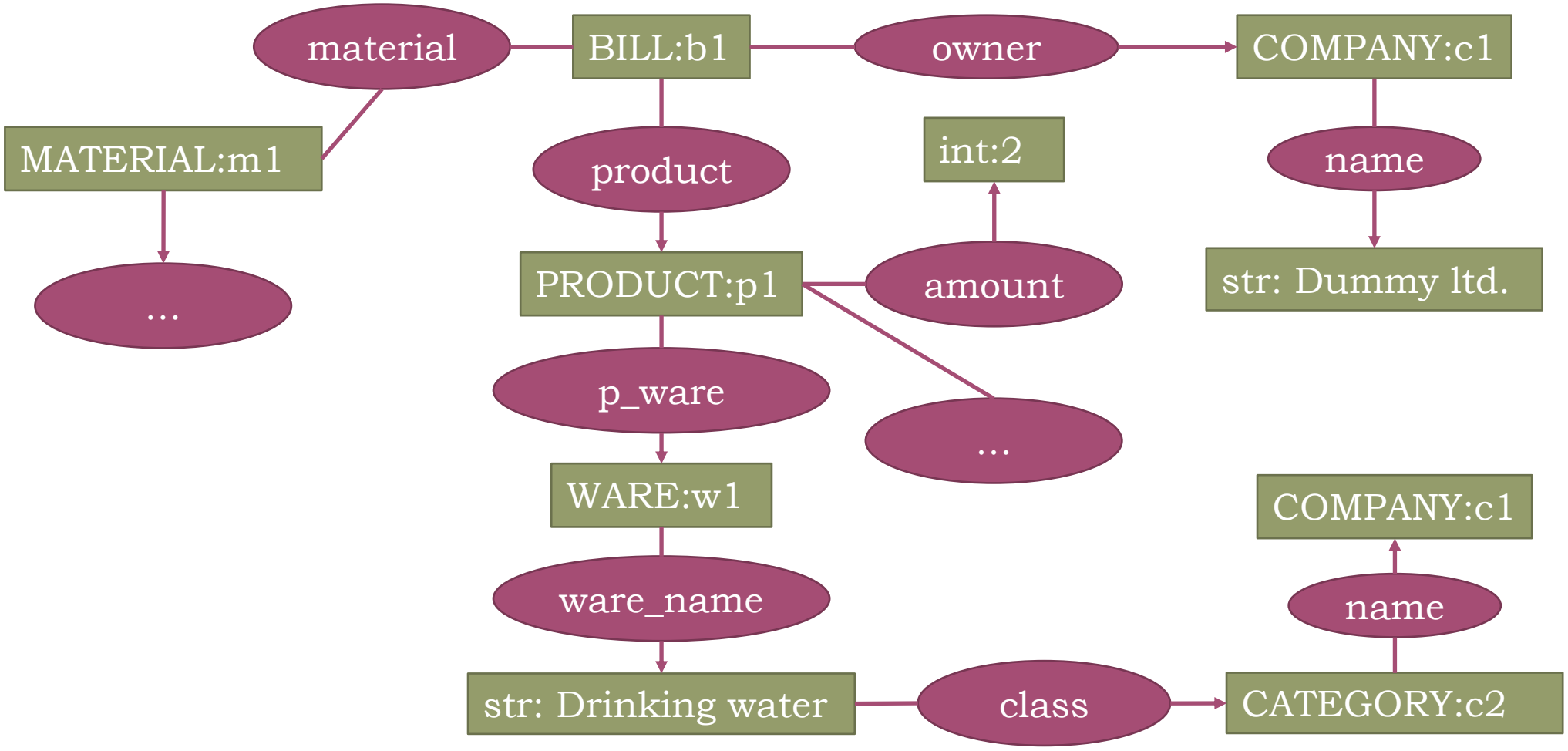
SQL:

```
SELECT DISTINCT p.WARE, m.COMPANY
FROM MANUFACTURER m, PRODUCT p, CATEGORY c
WHERE c.CLASS='Raw food' AND m.BILL_ID=p.BILL_ID
  AND c.WARE=p.WARE
```

Cypher:

```
MATCH (c:COMPANY)<-[:OWNER]-(:BILL)-[:PRODUCT]->
      (p:WARE)-[:CLASS]->(ct:CATEGORY)
WHERE ct.name='Raw food'
RETURN DISTINCT p,c
```

RDF graph/triple store: example



Triple store

Pros	Cons
Regular and very compact model	Performance is worse than for property graphs
Flexible dynamic schema	
Good for formal ontologies and reasoning	

Implementations: OpenLink Virtuoso, AlegroGraph , Amazon Neptune

Query languages: SPARQL

Related courses

- **Introduction to artificial intelligence** (3rd sem.) – how to write your own planner?
- **Operating systems** (3rd sem.) – low-level concurrency, resource management, file systems.
- **Team project** (2nd year) – practice (possibly with DBs)
- **Concurrency** (4th sem.) – math models for the concurrency.
- **Introduction to networking** (4th sem.) – interactions in distributed applications (DBs: main app – DB, distributed DB)
- **Data processing and storage** (3rd year) – various topics about DBs and high-level concurrency
- **Software design** (3rd year) – application design and development (possibly with DBs)