CDCS
CENTER FOR DATA AND COMPUTING
IN NATURAL SCIENCES

Databases Under The Hood

An Introduction For The Curious User

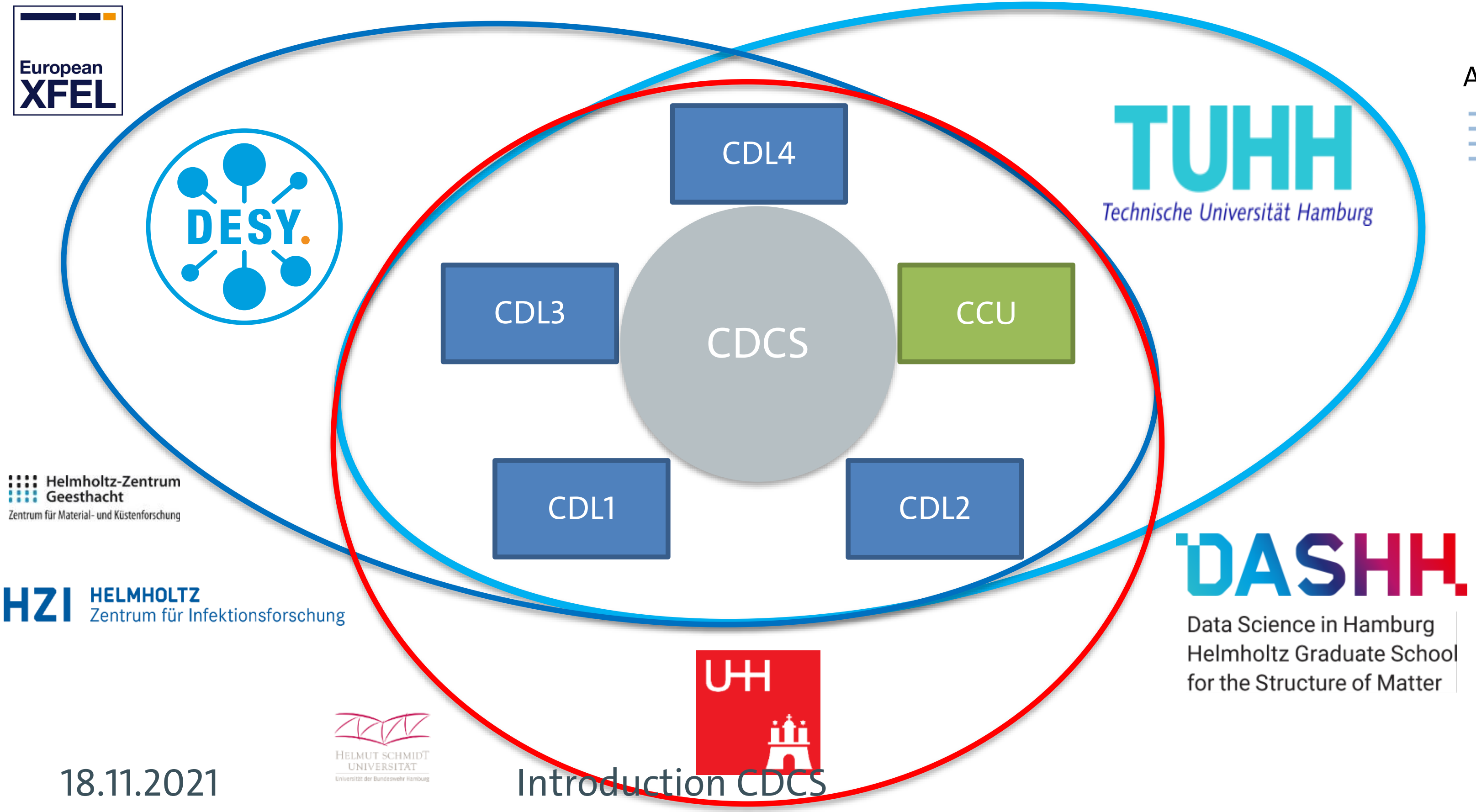
Annett Ungethüm, 16.12.2021

Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

DESY.

TUHH
Technische Universität Hamburg-Harburg

CDCS Hamburg-X Project (BWFGB)



Associated Partner

HAW HAMBURG

CDCS Structure

UHH Project Coordinator

Prof. Dr. Matthias Rarey
UHH Computer Science
Spokesperson CDCS

DESY Project Coordinator

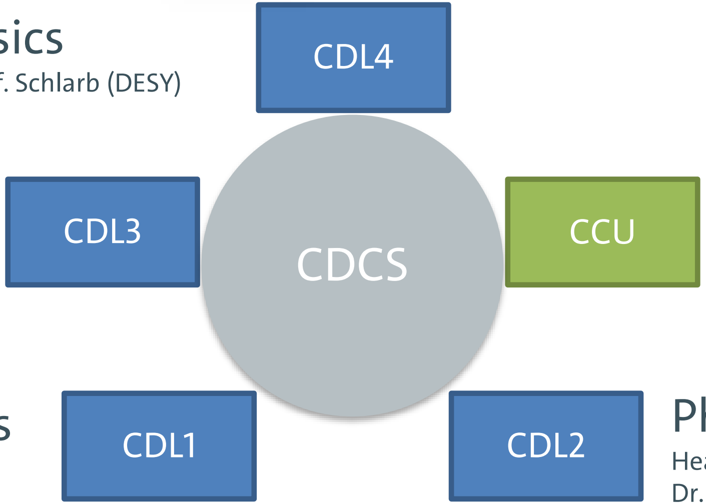
Prof. Dr. Nina Rohringer
DESY / UHH Physics

TUHH Project Coordinator

Prof. Dr. Sabine Le Borne
TUHH Mathematics

Accelerator Physics
Head: Prof. Fey (TUHH) / Prof. Schlarb (DESY)

Systems Biology
Head: Prof. Grünewald (UHH)/
Prof. Baumbach (UHH)



Computational Core Unit
Head: Prof. Rarey (UHH)/ Prof. Knopp (TUHH)

Astro & Particle Physics
Head: Prof. Schleper (DESY)/
Dr. Gaede (UHH)

Photon Science
Head: Dr. Barty (DESY)/
Dr. White (DESY)

CDCS and CDLs in Detail

Accelerator Physics
Head: Prof. Fey (TUHH) / Prof. Schlarb (DESY)

CDL4


Dr. Antonin Sulc

Dr. Ahmad Al-Zoubi

Systems Biology
Head: Prof. Grünewald (UHH)/
Prof. Baumbach (UHH)

CDL3


Dr. Karen M.-Cantos


Dr. Khalique Newaz

CDCS
Miriam Döring
Thomas Merling

CCU

Computational Core Unit
Head: Prof. Rarey (UHH)/ Prof. Knopp (TUHH)


Dr. Marie Tolkiehn
Scientific-administrative
Management


Dr. Florian Griese


Dr. Annett Ungethüm

CDL1

CDL2

Astro & Particle Physics
Head: Prof. Schleper (DESY)/
Prof. Gaede (UHH)


Dr. Patrick Connor


Dr. Janis Kummer


Dr. Lennart Rustige

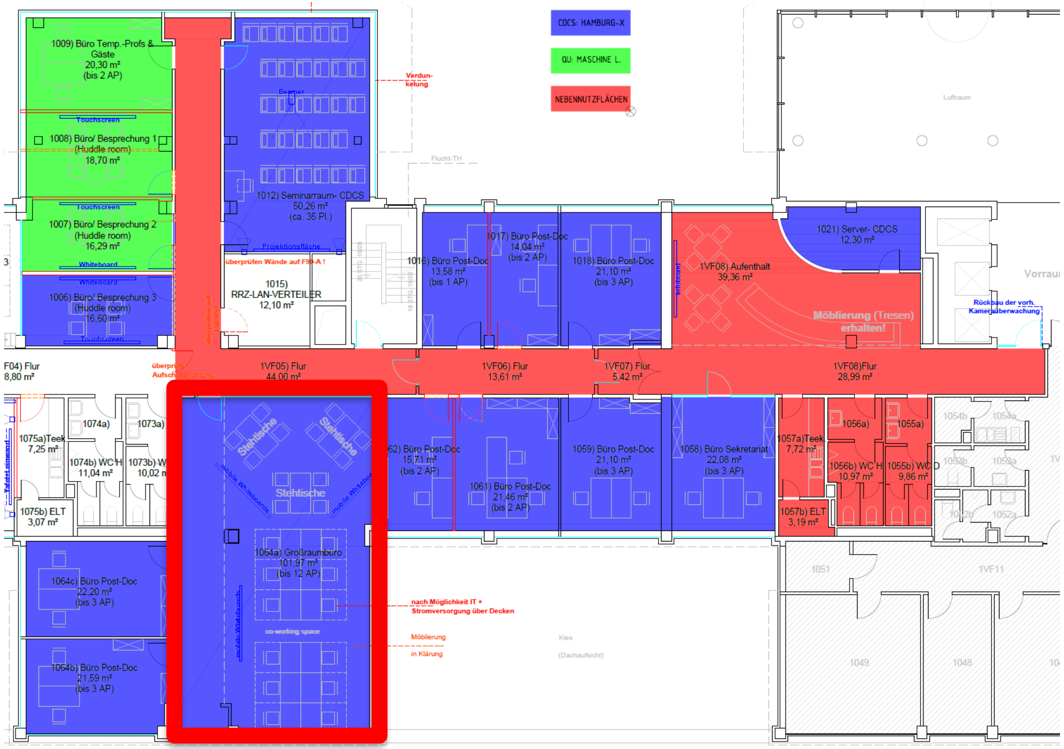
Photon Science
Head: Dr. Barty (DESY)/
Dr. White (DESY)


Dr. Mads Jakobsen

N.N

The CDCS Office Space

CDCS
CENTER FOR DATA AND COMPUTING
IN NATURAL SCIENCES



As a DASHH student you can get a transponder to the CDCS hot desk office space (room 1064)

Ask our secretary Miriam Döring: miriam.doering@uni-hamburg.de

Data Science Thursdays: Database Timeline

CDCS
CENTER FOR DATA AND COMPUTING
IN NATURAL SCIENCES

Topic of the month: Databases

You might want to send us your questions in advance to get more sophisticated answers

16 Dec 2021
Why and how should I use a database and why it is different from an excel sheet

23 Dec 2021
Getting help with your first queries (upon request)

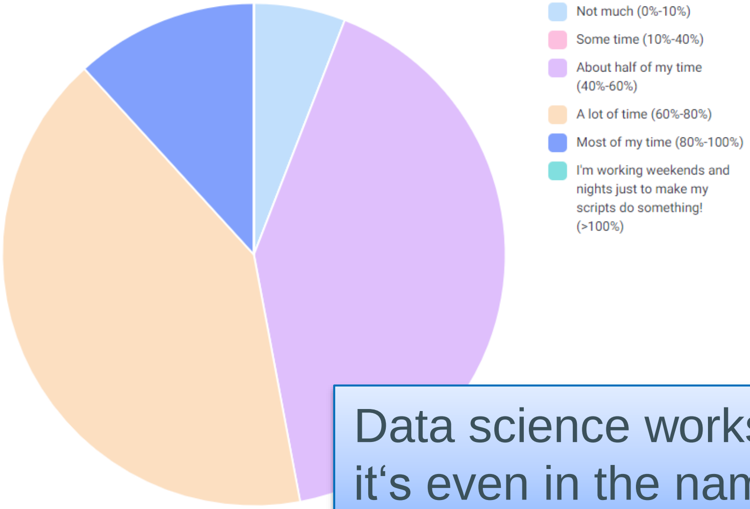
06 Jan 2021
Relational DBs, document stores, key-value-stores: There's a system for every use-case

13 Jan 2021
Get your research data into a database!

Survey Results

CDCS
CENTER FOR DATA AND COMPUTING
IN NATURAL SCIENCES

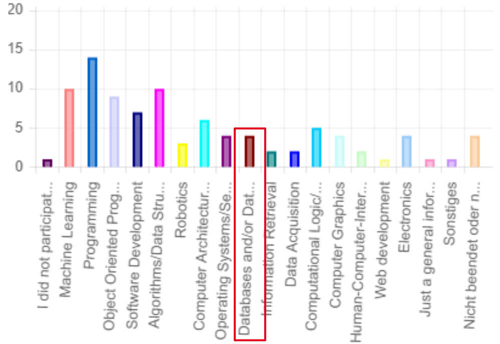
How much of your working time do you spend with computer science issues (scripting, debugging, setting up workflows,...)?



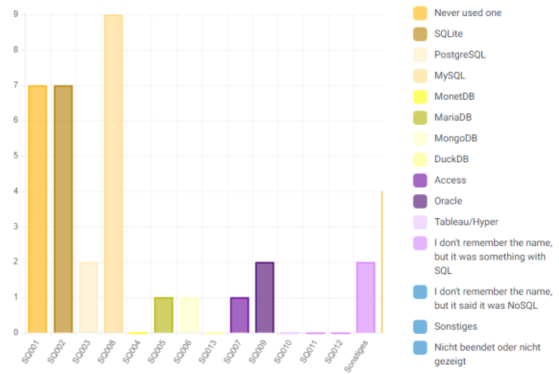
Data science works with data, it‘s even in the name

Databases are made for managing and analyzing data

Have you had any computer science courses before, e.g. during your undergrad? If yes, which kind of courses.



Have you ever used a database system? If yes, which one?



Why use a DB?

CDCS
CENTER FOR DATA AND COMPUTING
IN NATURAL SCIENCES

	Database	CSV, Excel sheets	Other file formats: Hdf5, Binary,…	Libraries: dplyr, pandas,…
Fast loading and parsing	✓	✗	ℹ *Possible but depends on your tools and knowledge*	ℹ *If you optimize it yourself*
Automatic parallellization	✓ **most DB systems*	ℹ *Only with PowerBI; No parallelism with a vanilla Excel*	ℹ *Depends on your tools*	✗
Cares for data validity, e.g. consistent transactions	✓	✗	✗ *Only if you combine it with a database, e.g. hdf5 + Hadoop*	ℹ *Depends on your tools*
Optimizes your queries	✓ **most DB systems*	✗	✗	✗
Optimized join of data	✓ **most DB systems*	✗	✗	✗
Offers a turing complete query language	✓ *Accidentally since SQL2003*	✓ *If you really want to use VBA*	ℹ *Depends on your tools*	✓ *Many programming languages are already turing complete*

By „most“ I mean all systems but SQLite. We will get to that point later.

You can implement all of this yourself…However, generations of PhD students in systems architecture will tell you to run if your supervisor ever asks you to do this.

These are basically query execution engines, but they are slow engines.

It‘s simple! … Really!

Let‘s assume we have to tables:
Table_A and *Table_B*.

Column_1	Column_2
3	5
4	6

Column_a	Column_b
5	2
7	1

Let‘s further assume we want all entries where Column_2 equals Column_a.

Column_1	Column_2	Column_a	Column_b
3	**5**	**5**	2

dplyr (R)

inner_join(Table_A, Table_B, by = c(“*Column_2*“ = “*Column_a*“))

pandas (python)

pd.merge (Table_A, Table_B, left_on=‘*Column_2*‘, right_on=‘*Column_a*‘, how=‘inner‘)

SQL

SELECT * FROM Table_A, Table_B WHERE *Table_A.Column_2=Table_B.Column_a*;

→ It won‘t win a prize for literature, but it‘s close to a spoken language.

Join tables with Excel

Merge Tables Wizard - quick way to join tables by matching columns

*requires an add-on

If you are not very comfortable with Excel formulas yet, nor do you have time to figure out the arcane quirks of Power Query, our Merge Tables Wizard could be your time-saver. Below I will show three most popular uses cases.

Example 1. Combine two tables by multiple columns

If you find the array formula for columns match hard to remember, rely on our add-in to do the job quickly and perfectly.

For this example, we will be using the already familiar tables and join them based on 2 columns, *Seller* and *Product*. Please note that the lookup table has 2 more columns than the main table:

Main table

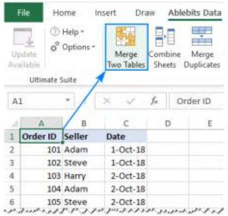
Order ID	Seller	Product	Amount
101	Adam	Bananas	
102	Harry	Oranges	
103	Harry	Apples	
104	Adam	Lemons	
105	Harry	Bananas	
106	Adam	Apples	

Lookup table

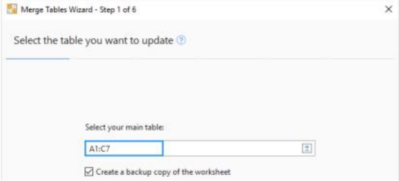
Seller	Product	Amount
Adam	Bananas	$510
Harry	Apples	$520
Harry	Oranges	$550
Adam	Apples	$600
Harry	Lemons	$605
Harry	Bananas	$605
Adam	Lemons	$705
Adam	Oranges	$735

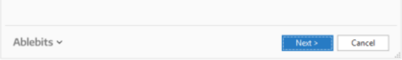
With the Merge Tables Wizard added to your Excel ribbon, here's what you need to do:

1. Select any cell within your **main table** and click the **Merge Two Tables** button on the *Ablebits Data* tab:

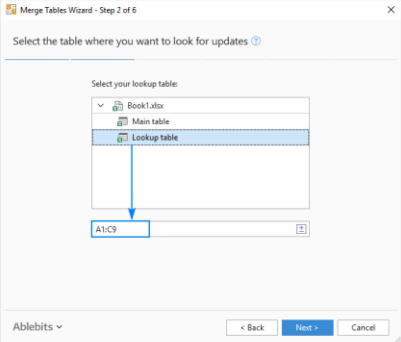


2. Make sure the add-in got the range right, and click *Next*:

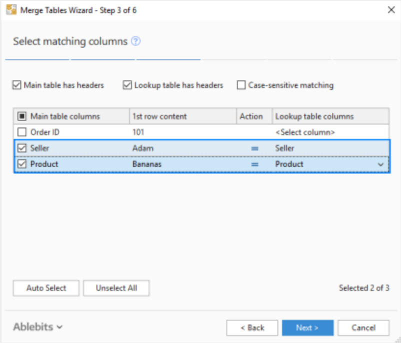




3. Select the lookup table, and click *Next*:

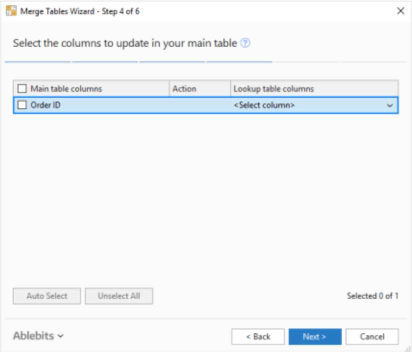


4. Specify the column pairs to match, *Seller* and *Product* in our case, and click *Next*:

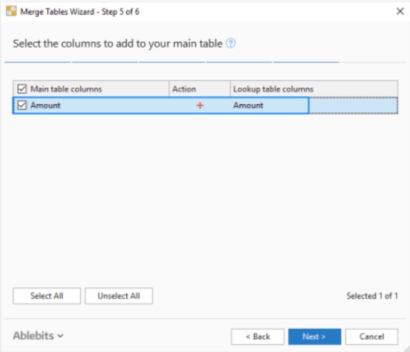


Tip. If the text case in the key columns matters, check the *Case-sensitive matching* box to treat uppercase and lowercase as different characters.

5. Optionally, choose the columns to update with the values from the lookup table. Since there is nothing to update in the *Order IDs* column, we leave it unselected, and simply click *Next*.

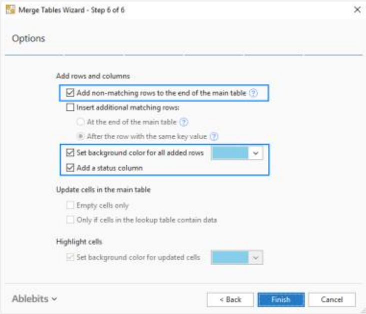


6. Select the columns to add to the main table and click *Next*.

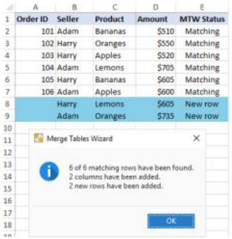


7. In this step, you tell the wizard how exactly you want the tables to be merged. All the options have descriptive labels, so I won't go into long explanations. If you are unsure about a certain option, click the question mark next to it, and a small diagram will show you how the tables are going to be combined.

The default options work just fine in our case, so we click *Finish* without changing anything:



Allow the wizard a few seconds for processing and review the result:



Order ID	Seller	Product	Amount	MTW Status
101	Adam	Bananas	$510	Matching
102	Harry	Oranges	$550	Matching
103	Harry	Apples	$520	Matching
104	Adam	Lemons	$705	Matching
105	Harry	Bananas	$605	Matching
106	Adam	Apples	$600	Matching
	Harry	Lemons	$605	New row
	Adam	Oranges	$735	New row

As you can see in the screenshot above, the wizard has done the following:

1. Added the *Amount column* by matching the seller name and product in both tables.
2. Added the *Status* column that allows you to easily filter matching and new rows. If you don't want it, clear the corresponding box in the final step.
3. New rows that were present only in the lookup table were copied to the end and highlighted in blue.
   - If you don't want to highlight new rows, unselect *Set background color for all added rows* in the last step.
   - If you don't want to add new rows, unselect *Add non-matching rows to the end of the main table* in the last step.

source: ablebits.com

It‘s an all in one solution!

- Optimized data storage and reader
  - ➢ Does not kill your file system with thousands of small files
  - ➢ Loads only what is necessary, i.e. not always the whole file
  - ➢ Indexes your data (more or less automatically)
- Comes with a standardized query language (SQL)
- Optimized operators (e.g. join, merge, and aggregation are operators)
- Query optimizer (the thing that schedules your operators)
- Additional features often included: compression, encoding, user management, out-of-memory execution (in case your files are really big),…

End of commercial break

Let‘s start simple!

A Data Filter – What‘s for lunch?

CDCS
CENTER FOR DATA AND COMPUTING
IN NATURAL SCIENCES

MensaMeals

Meal	Price
Pizza	6,50
Pasta	4,90
Pie	1,20
Potato Salad	5,80
Pannfisch	7,90

The typical broke student problem:

Which meals are cheaper than 5 €?

Which meals are cheaper than 5€?

To answer this question, we need:

Data

MensaMeals

Meal	Price
Pizza	6,50
Pasta	4,90
Pie	1,20
Potato Salad	5,80
Pannfisch	7,90

Query

Select all meals where the price is lower than 5

A plan

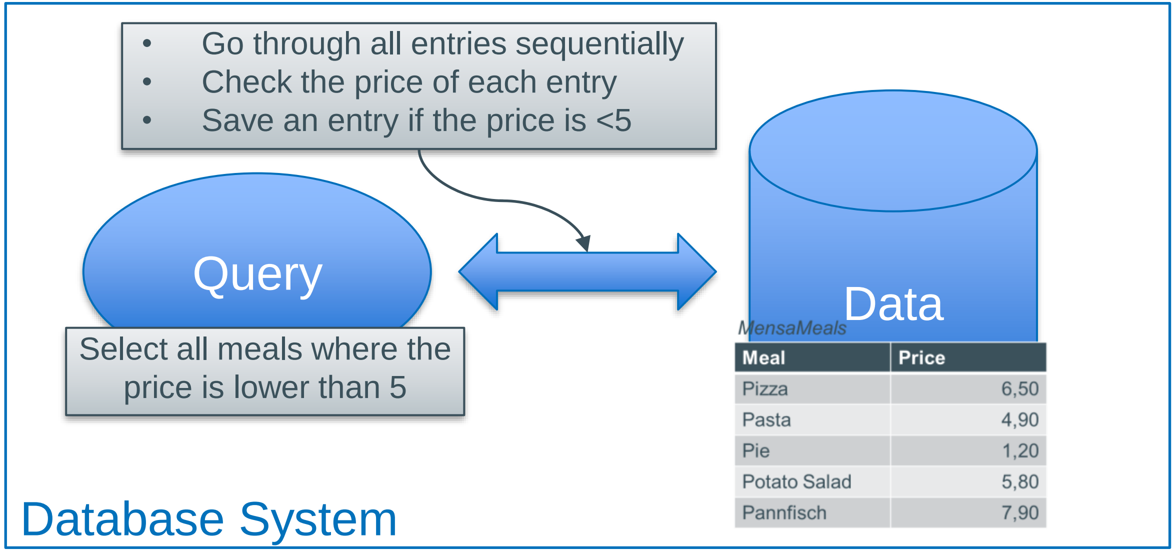
- Go through all entries sequentially
- Check the price of each entry
- Save an entry if the price is <5

System

Something that implements everything

Roadmap

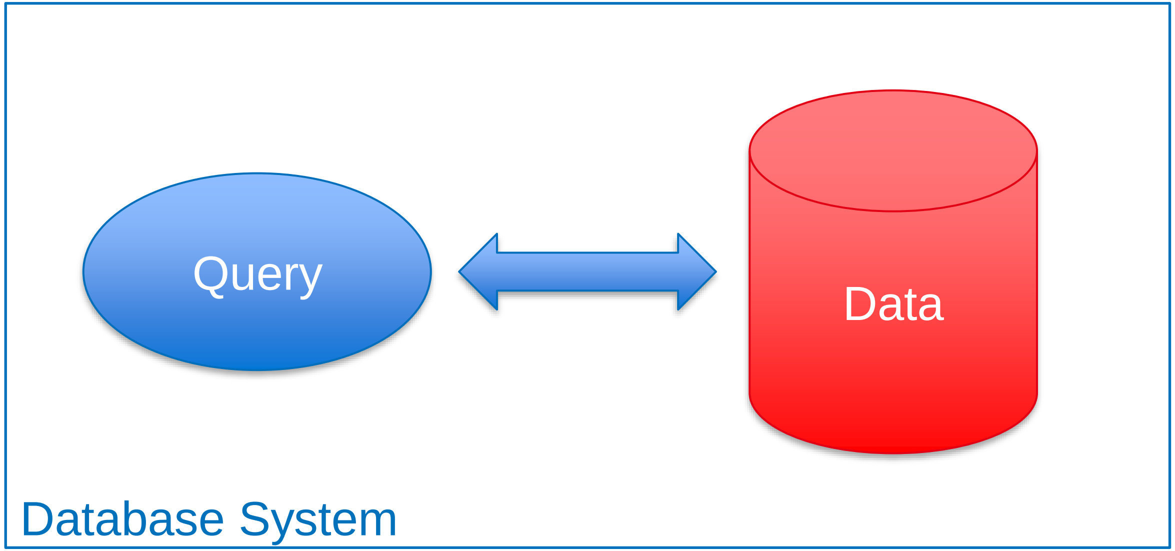
CDCS
CENTER FOR DATA AND COMPUTING
IN NATURAL SCIENCES



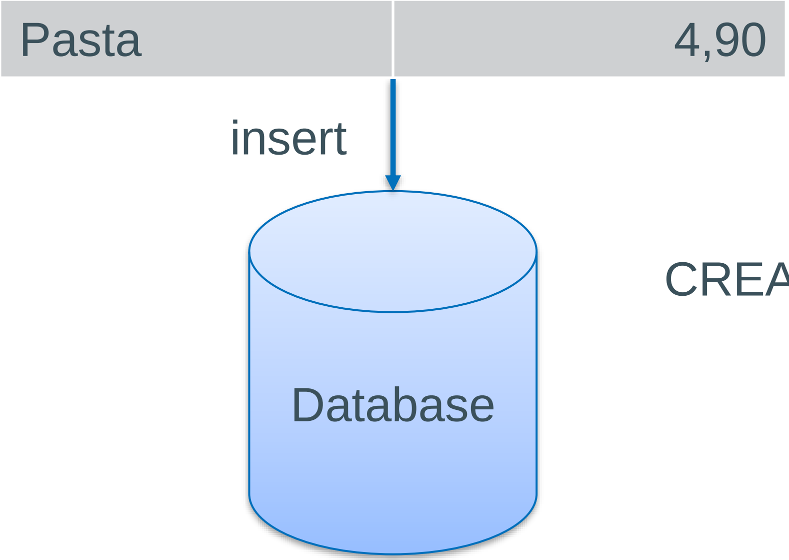
Meal	Price
Pizza	6,50
Pasta	4,90
Pie	1,20
Potato Salad	5,80
Pannfisch	7,90

Something that implements everything

Roadmap



Get data into a virgin database



1. Create an empty table

CREATE TABLE MensaMeals (Meal TEXT, Prize REAL);

Name of table

Name and type of columns

End of statement

2. Insert rows

INSERT INTO MensaMeals VALUES (‘Pasta‘, 4.90);

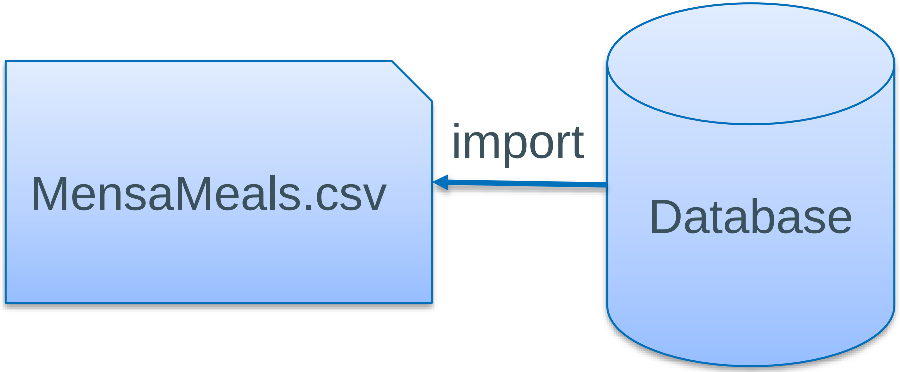
Name of table

Values

Show contents of your table:
SELECT * FROM MensaMeals;

Get data into a virgin database

CDCS
CENTER FOR DATA AND COMPUTING
IN NATURAL SCIENCES



Database Systems offer import functions

→ Supported formats differ (csv, parquet, db, ...)

→ Syntax differs

→ Auto-detection of data types and delimiters may or may not work

Examples

PostgreSQL

COPY MensaMeals FROM ‘home/itsme/MensaMeals.csv‘ DELIMITER ‘, ‘ CSV HEADER;

CREATE TABLE MensaMeals AS SELECT * FROM ‘MensaMeals.csv‘; DuckDB

COPY INTO MensaMeals FROM ‘home/itsme/MensaMeals.csv‘; MonetDB

Let‘s keep it simple!

…And do some theory while you are still listening.

Data in Relational Databases

This is a **relation**, defined as *MensaMeals(Meal,Price)*

**That's why table based DBs are called relational Databases*

MensaMeals

Meal	Price
Pizza	6,50
Pasta	4,90
Pie	1,20
Potato Salad	5,80
Pannfisch	7,90

Meal and *Price* are the **attributes** of the relation *MensaMeals*

This is a **tuple**, which belongs to the relation *MensaMeals*

Relation and *table* often used as synonyms
but

- A relation can be defined without tuples, i.e. without being a 'real' table
- A table is only an illustration of your data

Storage Layouts

Relations are usually **illustrated** as tables → This tells us nothing about the storage layout
(cf. a matrix that can be stored differently → row- or column-major)

2 main layouts to store your table

MensaMeals

Meal	Price
Pizza	6,50
Pasta	4,90
Pie	1,20
Potato Salad	5,80
Pannfisch	7,90

Row-Store (tuple-wise)

This is what your traditional relational SQL database does

Pizza	6,50	Pasta	4,90	Pie	1,20	Potato Salad	5,80	Pann-fisch	7,90

Memory address →

Column-Store (attribute-wise)

This is what all (not so traditional) column-oriented databases do

Pizza	Pasta	Pie	Potato Salad	Pann-fisch

Memory address →

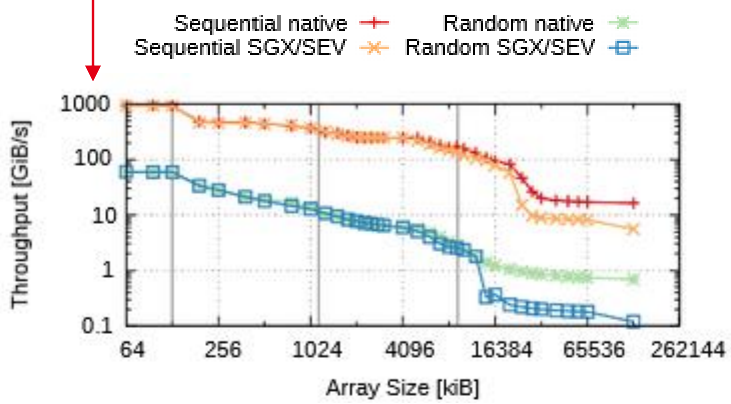
6,50	4,90	1,20	5,80	7,90

Memory address →

Why should you care?

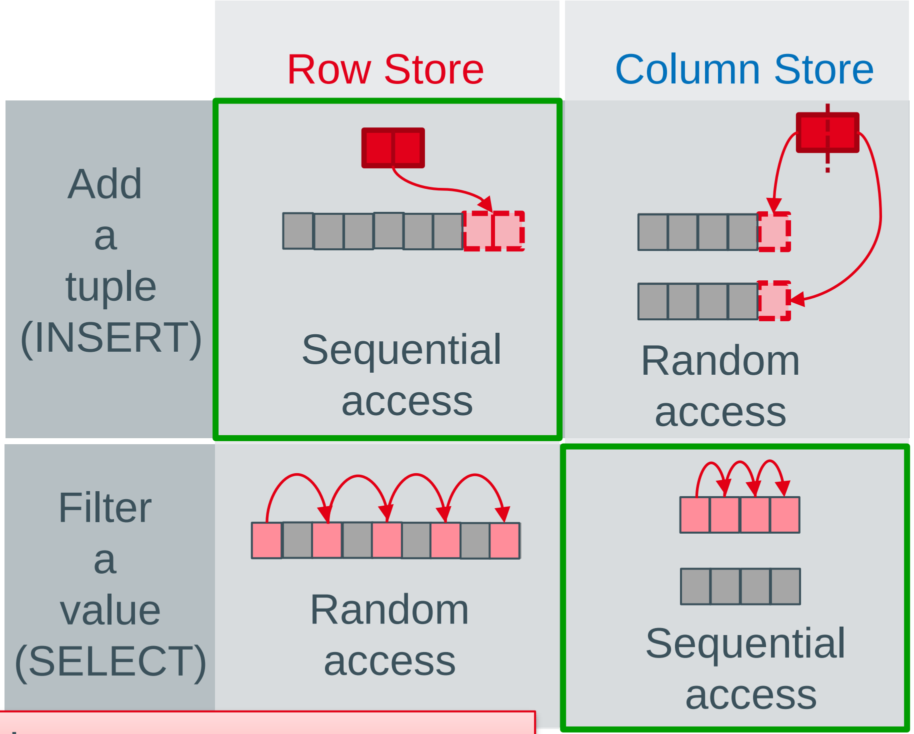
Memory access is expensive!

Note the log scale



*Throughput on an Intel Xeon E3-1275

Gottel, Christian & Pires, Rafael & Rocha, Isabelly & Vaucher, Sébastien & Felber, Pascal & Pasin, Marcelo & Schiavoni, Valerio. (2018). SRDS 2018

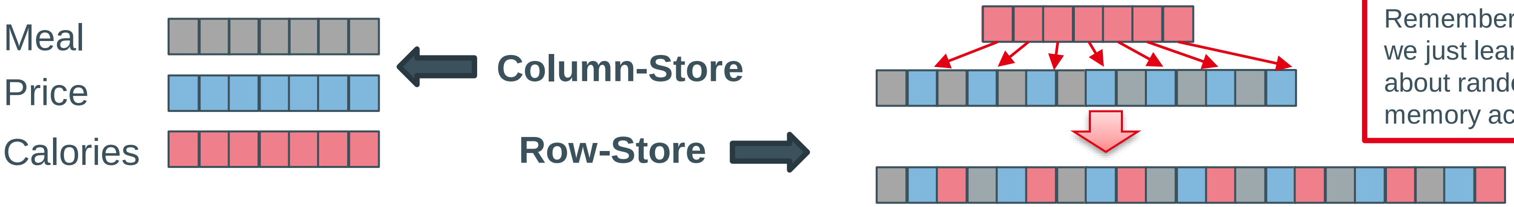


Your ideal layout depends on your use-case.
Different systems use different layouts, so choose wisely!

NoSQL and column-oriented DBs: Frequent misunderstandings

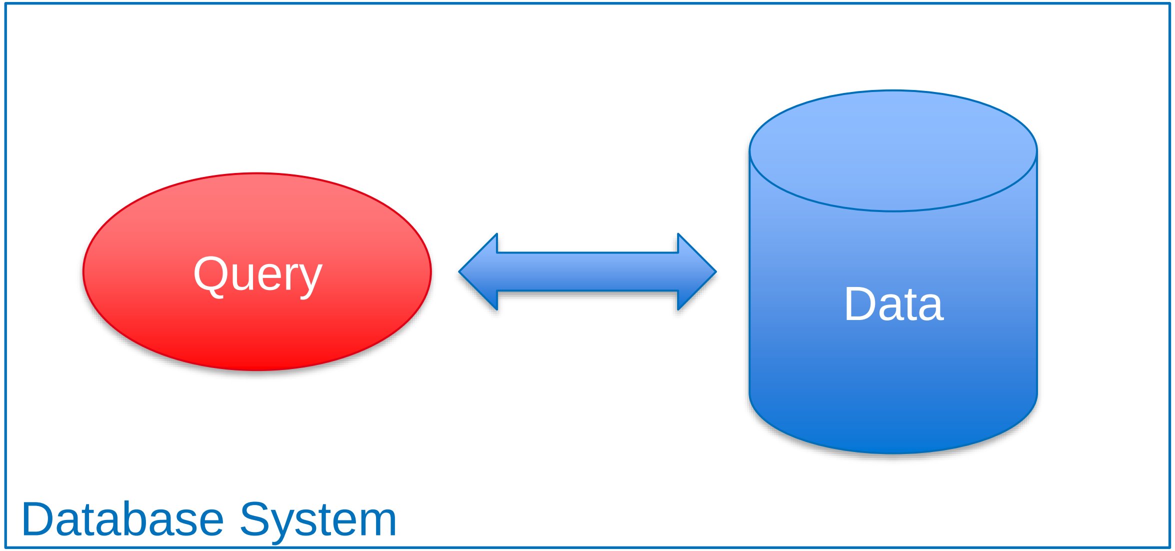
- NoSQL stands for **N**ot **o**nly **SQL**
- Wide-column DBs (NoSQL) and column-stores (SQL) are not the same, but both often referenced as column-oriented
  - ➢ We will use it to reference column-stores
- Usually, column-oriented databases can be queried using SQL and allow the definition of relations
  - ➢ Convenience of SQL, and performance and flexibility of column-stores
  - ➢ Example: Fast and easy addition/deletion of attributes

ALTER TABLE *MensaMeals* ADD *Calories* INT NULL;



Remember what we just learned about random memory access

Roadmap



Queries

- Queries consist of **operators** and can be formally described with **query languages**, e.g. relational algebra (RA), SQL
- **SQL** is a keyboard-friendly query language while **RA** is used for internal representation

Examples: Select Operator

SQL: SELECT * FROM MensaMeals WHERE Price < 5;

Relational Algrebra: $\sigma_{\text{Price} < 5}$ (MensaMeals)

Operator
Parameter
Relation

Result:

Meal	Price
Pasta	4,90
Pie	1,20

MensaMeals

Meal	Price
Pizza	6,50
Pasta	4,90
Pie	1,20
Potato Salad	5,80
Pannfisch	7,90

Operator Examples

Project Operator

Show only the names of all meals where the price is lower than 5€.

$\Pi_{Meal}(\sigma_{Price < 5}(MensaMeals))$

Meal	Price
Pasta	4,90
Pie	1,20

SELECT Meal FROM MensaMeals
WHERE Price < 5;

Result:

Meal
Pasta
Pie

Join Operator

Where can I get the meals which cost less than 5€?

DailyOffers

Mensa	Meal
Campus Mensa	Pizza
Mensa Cafe	Pie
Garden Mensa	Pasta
Old Mensa	Potato Salad

$\Pi_{Mensa}(\sigma_{Price < 5}(MensaMeals \bowtie_{MensaMeals.Meal=DailyOffers.Meal} DailyOffers))$

SELECT Mensa FROM
MensaMeals JOIN DailyOffers
ON MensaMeals.Meal=DailyOffers.Meal
WHERE Price < 5;

Result:

Mensa
Mensa Cafe
Garden Mensa

Why should you care?

CDCS
CENTER FOR DATA AND COMPUTING IN NATURAL SCIENCES

- With RA you can do everything, you can do with other algebras, e.g. prove that two queries produce the same results
- Restructure a query for better performance or reusability of subqueries
- Understand the output of the query optimizer (later today)

Price < 5 → θ_0

MensaMeals.Meal=DailyOffers.Meal → θ_1

$$\Pi_{Mensa}\,(\sigma_{\theta_0}\,(MensaMeals \bowtie_{\theta_1} DailyOffers))$$

Subquery Join is executed first

equivalent transformation

$$\Pi_{Mensa}\,((\sigma_{\theta_0}\,(MensaMeals)) \bowtie_{\theta_1} DailyOffers)$$

Subquery Selection is executed first

```
SELECT Mensa FROM
(Select Meal FROM MensaMeals WHERE
Price < 5) food
JOIN DailyOffers
ON food.Meal=DailyOffers.Meal;
```

Subquery can be saved and reused

A comprehensible list of transformations can be found here: https://www.postgresql.org/message-id/attachment/32513/EquivalenceRules.pdf

Reuse Subqueries with views

Reusability of queries and query results

- ➔ Queries and Subqueries (Views) can be stored and referenced → nicer queries
- ➔ The result of views can be stored → higher performance for frequently used queries and remote data

Store (materialize) the view

Create a view called CheapFood

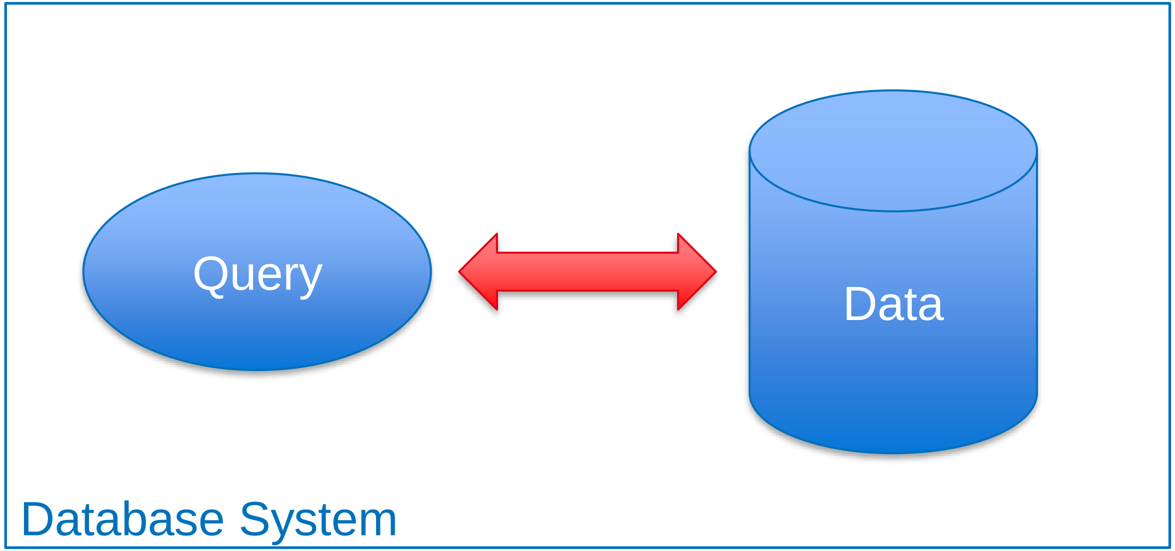
Refresh the view

CREATE MATERIALIZED VIEW *CheapFood* AS
SELECT *Meal* FROM *MensaMeals* WHERE
Price < 5;

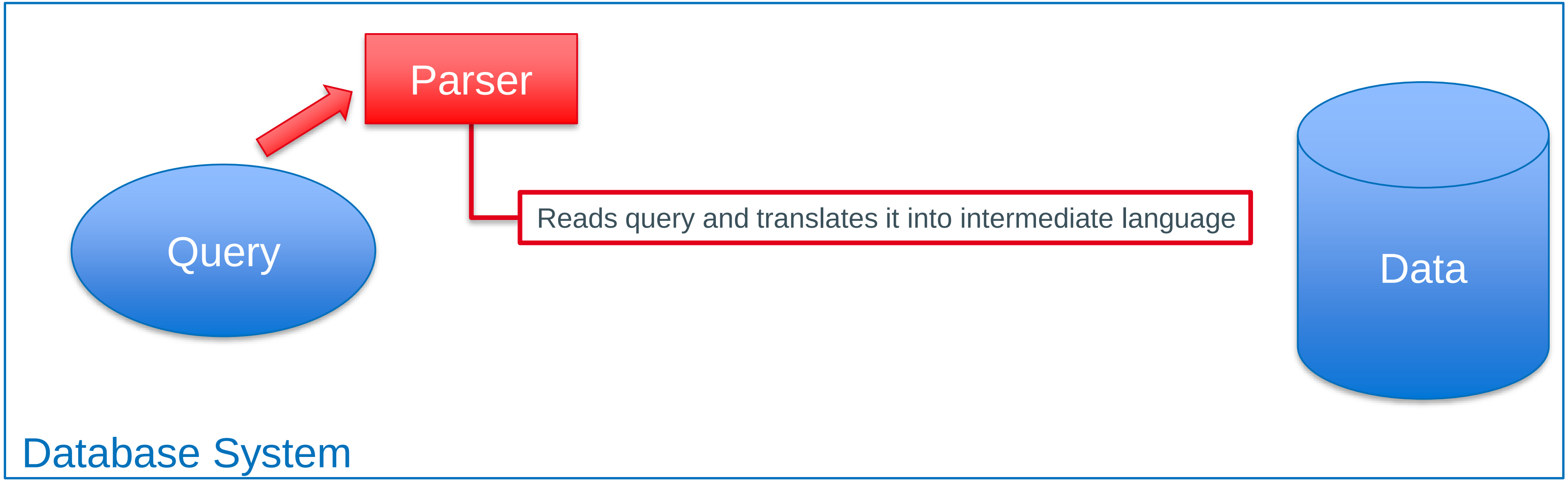
REFRESH MATERIALIZED
VIEW *CheapFood*;

- Refresh the view after updates in your base data
- Not supported by all database systems

Roadmap

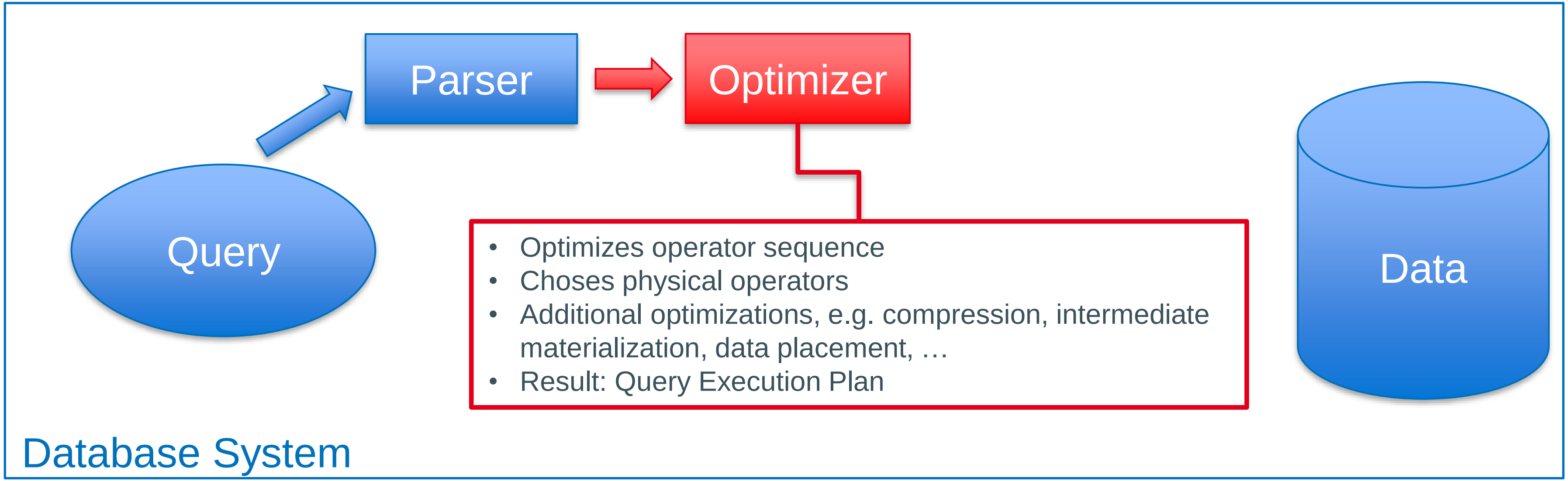


Roadmap



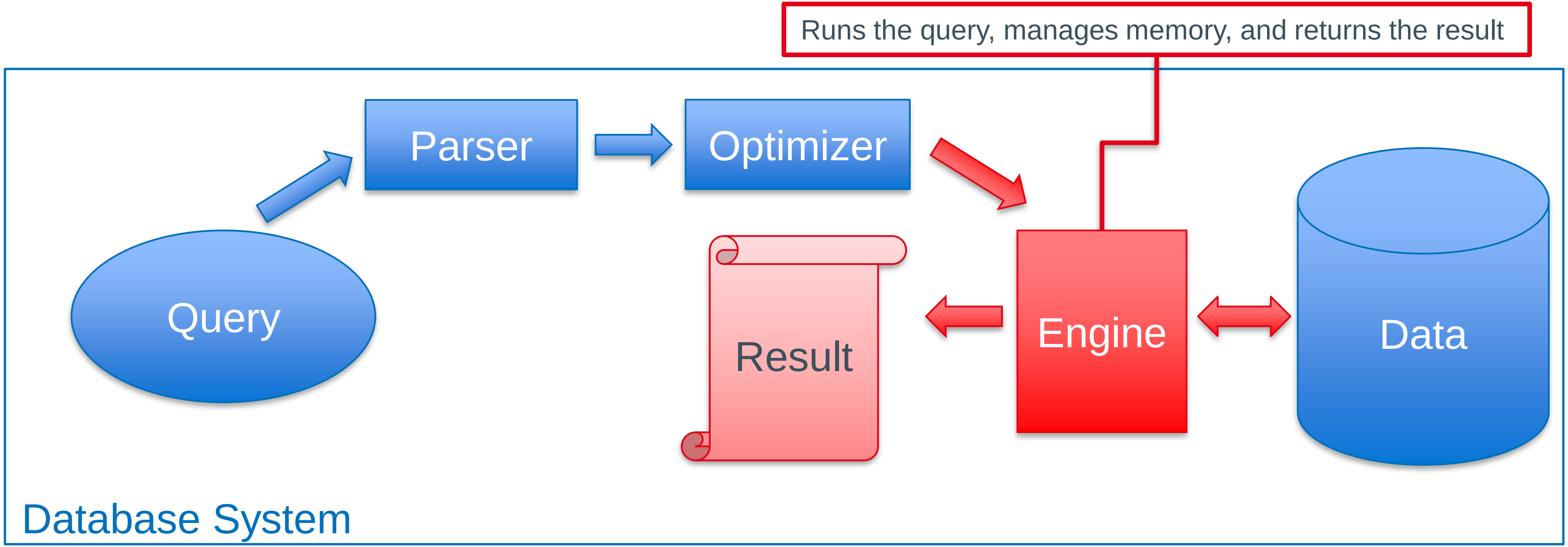
*strongly simplified

Roadmap



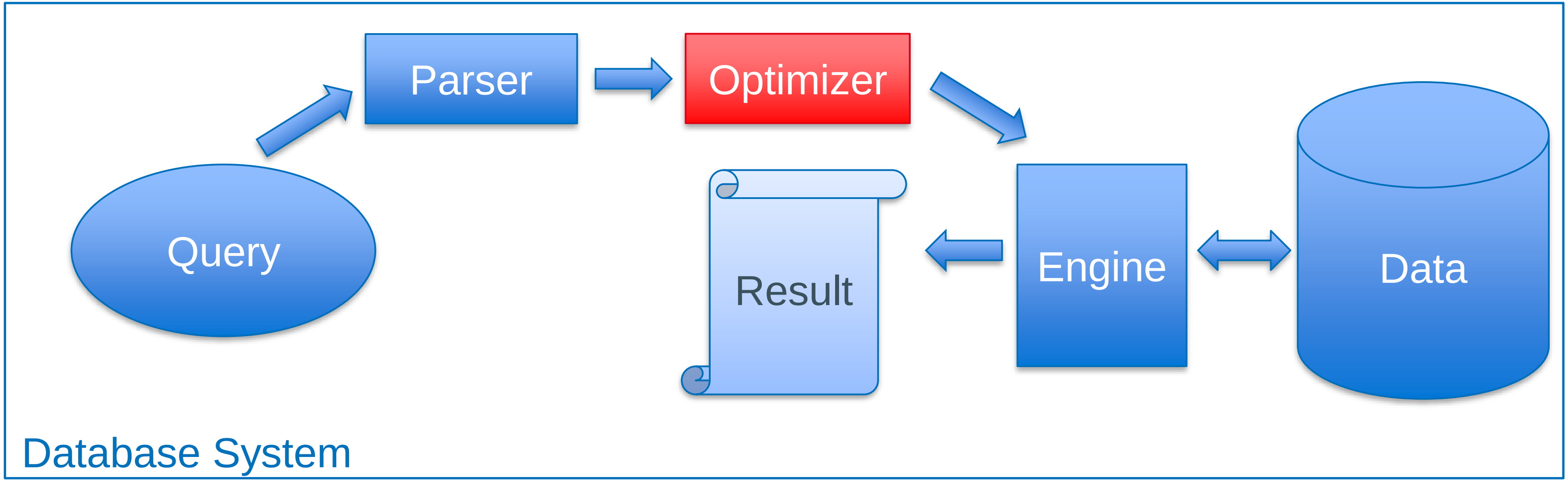
*strongly simplified

Roadmap



*strongly simplified

Roadmap



*strongly simplified

Optimizer: Query Execution Plan Optimization

SELECT Mensa FROM MensaMeals JOIN DailyOffers ON MensaMeals.Meal=DailyOffers.Meal WHERE Price < 5;

Plan A: $\Pi_{Mensa}(\sigma_{\theta_0}(MensaMeals \bowtie_{\theta_1} DailyOffers))$

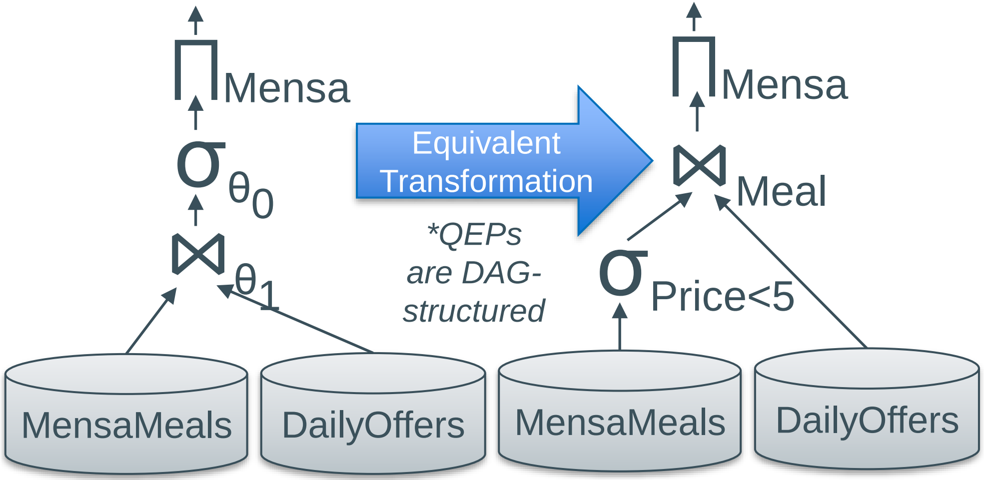
Plan B: $\Pi_{Mensa}((\sigma_{\theta_0}(MensaMeals)) \bowtie_{\theta_1} DailyOffers)$

- Database Systems use a relational algebra for internal representation
- Optimizers try to automatically find the most efficient sequence of operators

➔ Conventional approach: Reduce data as early and as cheap as possible

➔ Tool: Cardinality/Selectivity estimation

- The chosen sequence of operators is the final Query Execution Plan (QEP)



Further Reading

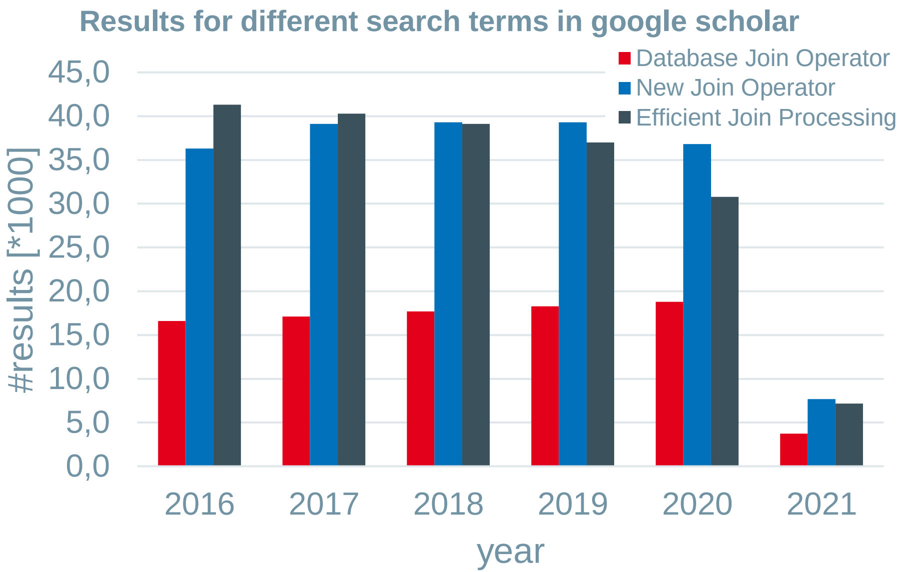
Foundations for operator order optimization: https://www.researchgate.net/publication/2916321_Bringing_Order_to_Query_Optimization

Survey on different cardinality estimation techniques: http://www.vldb.org/pvldb/vol11/p499-harmouch.pdf

Optimizer: Physical Operator Selection

- For each **logical operator** (e.g. join), there can be different **physical operators** (e.g. hash-join, nested-loop-join), i.e. the same operator can be implemented in different ways
- Joins are a bottleneck in most queries → Join optimization is a much-noticed field of research
- Choice of physical operator depends on exact use case. Examples from PostgreSQL:

➔ Nested-Loop: full join, one very small table, condition is not an equality
➔ Hash Join: similarity joins, small expected hash table
➔ Merge Join: sorted data, large tables



Further Reading

More on join order optimization: *Query optimization through the looking glass, and what we found running the Join Order Benchmark*, V.Leis et al.

Overview on Popular Join algorithms and an alternative: *New algorithms for join and grouping operations*, G. Graefe

Why should I care?

- A look at the query plan can help you identify the bottleneck of your query
- The **Explain** keyword is supported by many systems and shows the query plan, the physical operator, sometimes the cost (i.e. the runtime) of the operators, and some more or less useful additional information (e.g. the size of the relations and intermediates)

> - Output can look different depending on DB system,
> - Options might be available, e.g. analyze, timing on/off, buffers

```
EXPLAIN (analyze) SELECT Mensa
FROM MensaMeals JOIN DailyOffers
ON MensaMeals.Meal=DailyOffers.Meal
WHERE Price < 5;
```

- Example output for join operator (PostgreSQL):

```
Hash Join (cost=0.00..5.37 rows=3 width=2) (actual time=0.00..2.222 rows=2 loops=1)
    -> Hash Cond: (MensaMeals.Meal=DailyOffers.Meal)
    …
```

Output of EXPLAIN

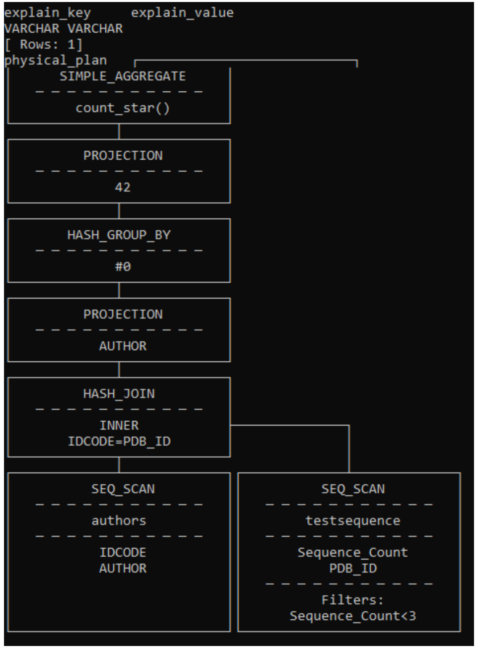
The mensa example is too small to generate interesting output
→ Switch to the Protein Database (PDB)
→ Create a more complex query

```
SELECT count(*) FROM (SELECT 1 FROM testsequence, authors WHERE
Sequence_Count < 3 AND testsequence.PDB_ID = authors.IDCODE
GROUP BY authors.author) foo;
```

Output of EXPLAIN

CDCS
CENTER FOR DATA AND COMPUTING
IN NATURAL SCIENCES

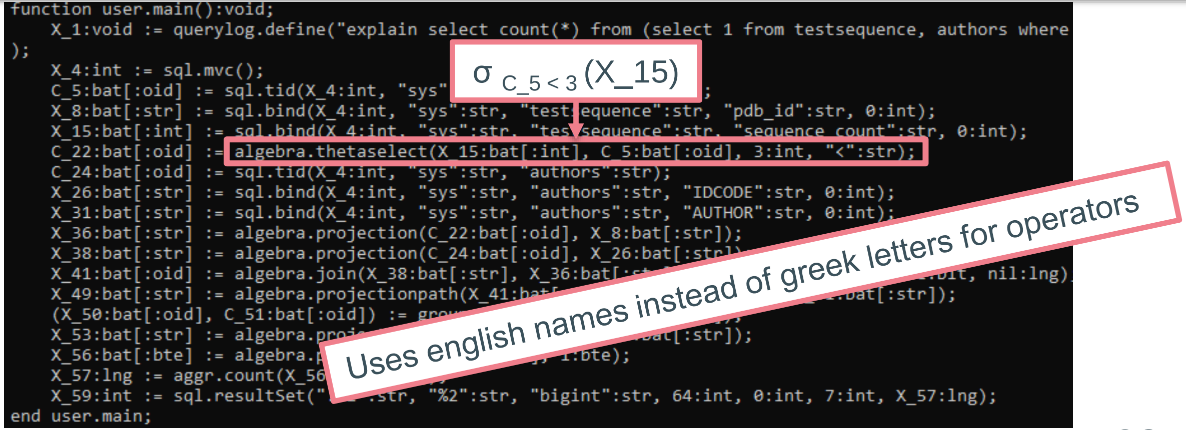
Some show a graph (duckdb)…



…some show an ugly graph (sqlite)…

```
QUERY PLAN
|--CO-ROUTINE 1
|  |--SCAN TABLE testsequence
|  |--SEARCH TABLE authors USING AUTOMATIC COVERING INDEX (IDCODE=?)
|  `--USE TEMP B-TREE FOR GROUP BY
`--SCAN SUBQUERY 1
```

…and some show a formatted version of their internal RA representation (e.g. MonetDB, PostgreSQL)
→ This is where you are lost without Relational Algebra



Output of EXPLAIN

CDCS
CENTER FOR DATA AND COMPUTING
IN NATURAL SCIENCES

DuckDB

```
explain_key      e
VARCHAR VARCHAR
[ Rows: 1]
physical_plan
   SIMPLE_AGGREGATE
   - - - - - - - - - - -
      count_star()

      PROJECTION
   - - - - - - - - - - -
          42

     HASH_GROUP_BY
   - - - - - - - - - - -
          #0

      PROJECTION
   - - - - - - - - - - -
        AUTHOR

       HASH_JOIN
   - - - - - - - - - - -
         INNER
     IDCODE=PDB_ID

       SEQ_SCAN                    SEQ_SCAN
   - - - - - - - - - - -     - - - - - - - - - - -
        authors                  testsequence
   - - - - - - - - - - -     - - - - - - - - - - -
        IDCODE                  Sequence_Count
        AUTHOR                      PDB_ID
                             - - - - - - - - - - -
                                   Filters:
                               Sequence_Count<3
```

SQLite

```
QUERY PLAN
|--CO-ROUTINE 1
|  |--SCAN TABLE testsequence
|  |--SEARCH TABLE authors USING AUTOMATIC COVERING INDEX (IDCODE=?)
|  `--USE TEMP B-TREE FOR GROUP BY
`--SCAN SUBQUERY 1
```

Additional Projection operator only needed in column-stores
→Find the data of the affected tuples (items with the same idx) in the arrays which store the remaining columns
→In row-stores, tuples are stored together, no lookup needed

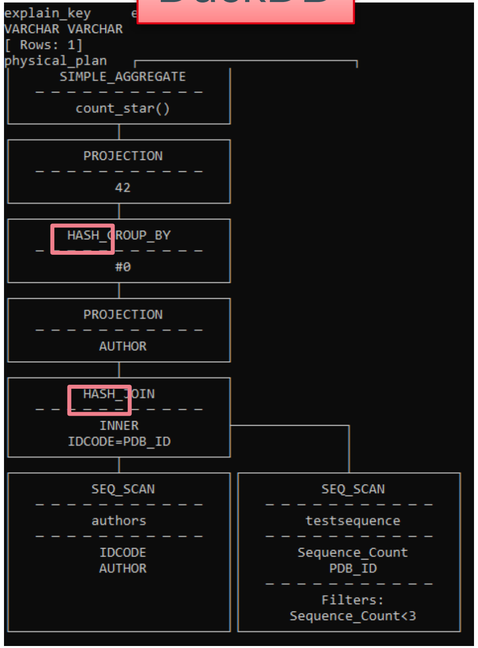
MonetDB

```
function user.main():void;
    X_1:void := querylog.define("explain select count(*) from (select 1 from testsequence, authors where
);
    X_4:int := sql.mvc();
    C_5:bat[:oid] := sql.tid(X_4:int, "sys":str, "testsequence":str);
    X_8:bat[:str] := sql.bind(X_4:int, "sys":str, "testsequence":str, "pdb_id":str, 0:int);
    X_15:bat[:int] := sql.bind(X_4:int, "sys":str, "testsequence":str, "sequence_count":str, 0:int);
    C_22:bat[:oid] := algebra.thetaselect(X_15:bat[:int], C_5:bat[:oid], 3:int, "<":str);
    C_24:bat[:oid] := sql.tid(X_4:int, "sys":str, "authors":str);
    X_26:bat[:str] := sql.bind(X_4:int, "sys":str, "authors":str, "IDCODE":str, 0:int);
    X_31:bat[:str] := sql.bind(X_4:int, "sys":str, "authors":str, "AUTHOR":str, 0:int);
    X_36:bat[:str] := algebra.projection(C_22:bat[:oid], X_8:bat[:str]);
    X_38:bat[:str] := algebra.projection(C_24:bat[:oid], X_26:bat[:str]);
    X_41:bat[:oid] := algebra.join(X_38:bat[:str], X_36:bat[:str], nil:BAT, nil:BAT, false:bit, nil:lng)
    X_49:bat[:str] := algebra.projectionpath(X_41:bat[:oid], C_24:bat[:oid], X_31:bat[:str]);
    (X_50:bat[:oid], C_51:bat[:oid]) := group.groupdone(X_49:bat[:str]);
    X_53:bat[:str] := algebra.projection(C_51:bat[:oid], X_49:bat[:str]);
    X_56:bat[:bte] := algebra.project(X_53:bat[:str], 1:bte);
    X_57:lng := aggr.count(X_56:bat[:bte]);
    X_59:int := sql.resultSet(".%2":str, "%2":str, "bigint":str, 64:int, 0:int, 7:int, X_57:lng);
end user.main;
```

Output of EXPLAIN

CDCS
CENTER FOR DATA AND COMPUTING
IN NATURAL SCIENCES

DuckDB



SQLite

```
QUERY PLAN
|--CO-ROUTINE 1
|  |--SCAN TABLE testsequence
|  |--SEARCH TABLE authors USING AUTOMATIC COVERING INDEX (IDCODE=?)
|  `--USE TEMP B-TREE FOR GROUP BY
`--SCAN SUBQUERY 1
```

B-Trees and Hashes are index structures
→ Takes time to build
→ Makes lookups faster

MonetDB

```
function user.main():void;
    X_1:void := querylog.define("explain select count(*) from (select 1 from testsequence, authors where
);
    X_4:int := sql.mvc();
    C_5:bat[:oid] := sql.tid(X_4:int, "sys":str, "testsequence":str);
    X_8:bat[:str] := sql.bind(X_4:int, "sys":str, "testsequence":str, "pdb_id":str, 0:int);
    X_15:bat[:int] := sql.bind(X_4:int, "sys":str, "testsequence":str, "sequence_count":str, 0:int);
    C_22:bat[:oid] := algebra.thetaselect(X_15:bat[:int], C_5:bat[:oid], 3:int, "<":str);
    C_24:bat[:oid] := sql.tid(X_4:int, "sys":str, "authors":str);
    X_26:bat[:str] := sql.bind(X_4:int, "sys":str, "authors":str, "IDCODE":str, 0:int);
    X_31:bat[:str] := sql.bind(X_4:int, "sys":str, "authors":str, "AUTHOR":str, 0:int);
    X_36:bat[:str] := algebra.projection(C_22:bat[:oid], X_8:bat[:str]);
    X_38:bat[:str] := algebra.projection(C_24:bat[:oid], X_26:bat[:str]);
    X_41:bat[:oid] := algebra.join(X_38:bat[:str], X_36:bat[:str], nil:BAT, nil:BAT, false:bit, nil:lng)
    X_49:bat[:str] := algebra.projectionpath(X_41:bat[:oid], C_24:bat[:oid], X_31:bat[:str]);
    (X_50:bat[:oid], C_51:bat[:oid]) := group.groupdone(X_49:bat[:str]);
    X_53:bat[:str] := algebra.projection(C_51:bat[:oid], X_49:bat[:str]);
    X_56:bat[:bte] := algebra.project(X_53:bat[:str], 1:bte);
    X_57:lng := aggr.count(X_56:bat[:bte]);
    X_59:int := sql.resultSet(".%2":str, "%2":str, "bigint":str, 64:int, 0:int, 7:int, X_57:lng);
end user.main;
```

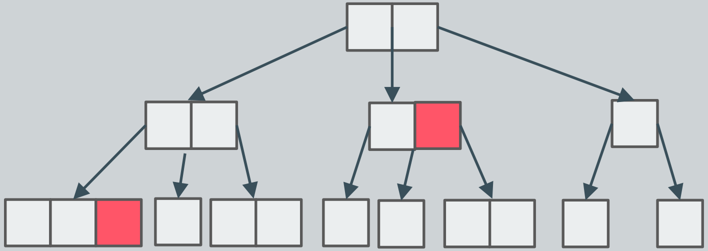
Index structures

Task: Find all red entries

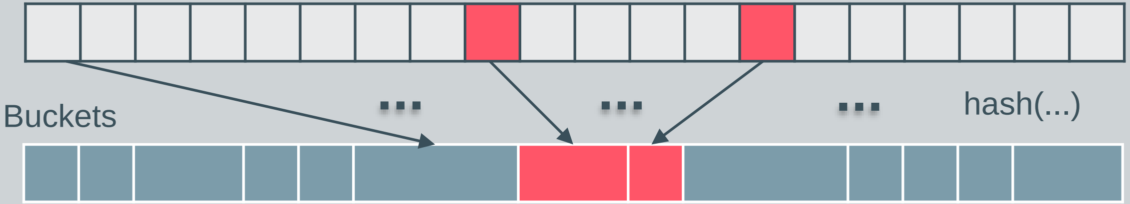
Trivial solution: Scan the whole dataset

Structures for faster searching:

Sorting



Tree structures



Hash-based structures (position defined by hash function)

Index structures

CDCS
CENTER FOR DATA AND COMPUTING
IN NATURAL SCIENCES

A good database system takes care for you of the index structures

But

- It it … owing (e.g. if an index would come in handy for future queries)
- Not all systems are good sy…

→ You can create an index yourself with CREA…

CREATE INDEX countindex ON testsequence (Sequence_…

If I lost you here, just join next week and we will try this out ☺

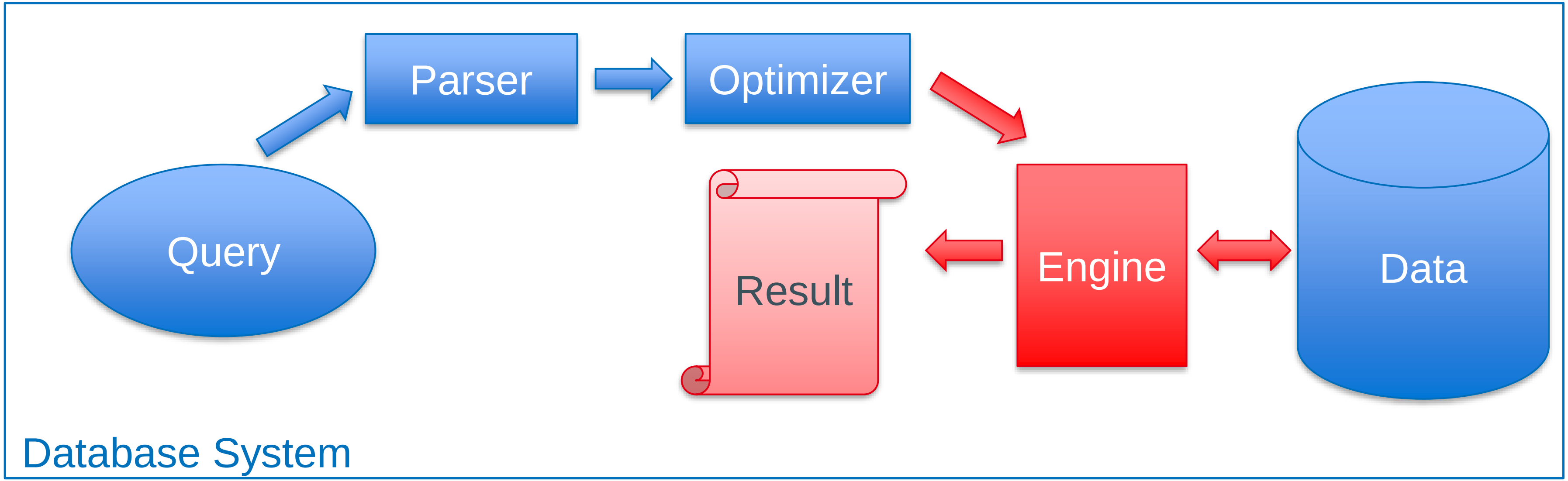
User-defined indexes can be ignored by the DBS (looking at you: MonetDB)

An Index can be nested, e.g.

CREATE INDEX countindex ON testsequence (PDB_ID, Sequence_Count);

- Your query plan may or may not provide useful information on which attributes it is using in which sequence
- Nested indexes only work for the exact sequence they are made for, i.e. PDB_ID, or PDB_ID, Sequence_Count, but not Sequence_Count

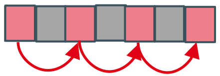
Roadmap



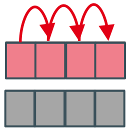
*strongly simplified

Data Processing Models

Row Store



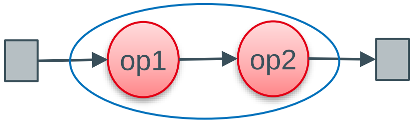
Column Store



Analytical queries usually read only a small number of columns, but all elements of these columns

For parallel or pipelined execution, data must be split

Tuple-at-a-time

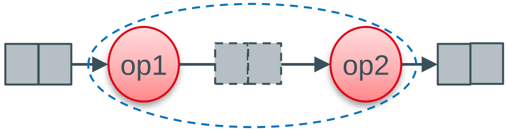


- Intermediate tuples not stored, but passed directly to next operator

→Operators can be fused

- Limited applicability of other optimizations, e.g. prefetching, vectorization, compression,…

Vector/Block-at-a-time

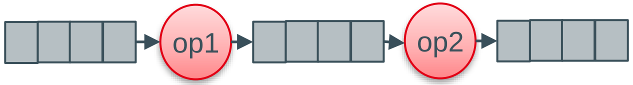


- A part (vector/block) of the column processed at once

→Operator fusion only for small blocks

- Trade-off between operator fusion and memory access performance

Operator-at-a-time



- Whole operator (all elements of the column) processed at once
- Intermediates materialized

→No operator fusion, only coarse-grained parallelization

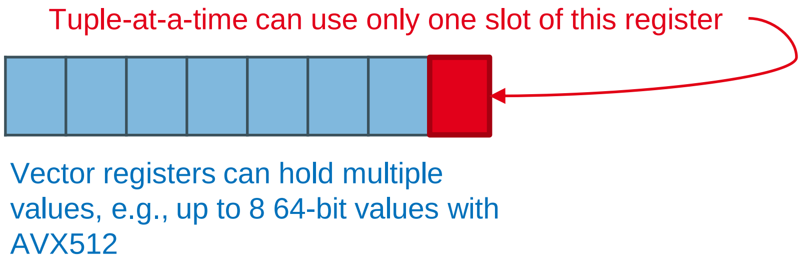
- High potential for optimization of memory reads

Why should I care?

- Different optimizations work with different processing models
- Your hardware limits your optimization space

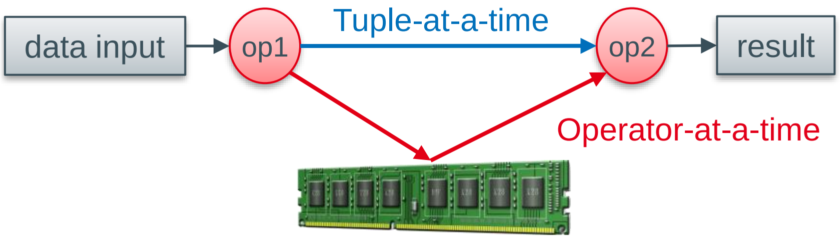
Example A: You have a new intel server with the AVX512 instruction set for vectorization (under linux, *lscpu* tells you if you have it; no root required)

➔ A system which implements only tuple-at-a-time is not able to use this instruction set



Example B: You do not have much main memory and writing to it is slow

➔ Materializing your intermediates becomes a bottleneck and might not work at all with operator-at-a-time or large blocks (block-at-a-time)



The Effect of Optimizers and Engines

CDCS
CENTER FOR DATA AND COMPUTING
IN NATURAL SCIENCES

SQLite

Ancient and extremely popular

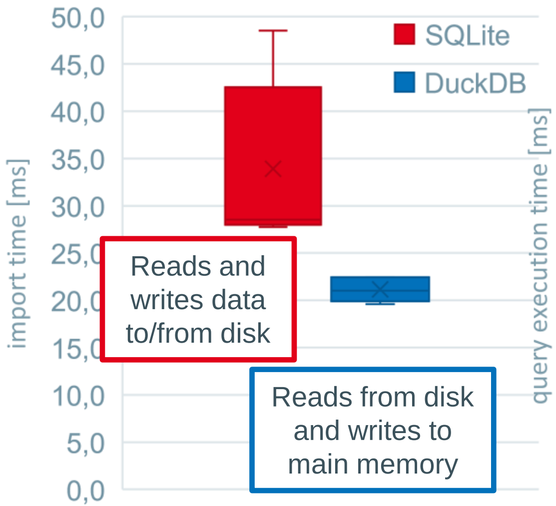
- Row-Store
- Disc-centric
- Tuple-at-a-time processing
- Only 1 Join implementation
- Next to no query optimization

DuckDB

New and completely fameless

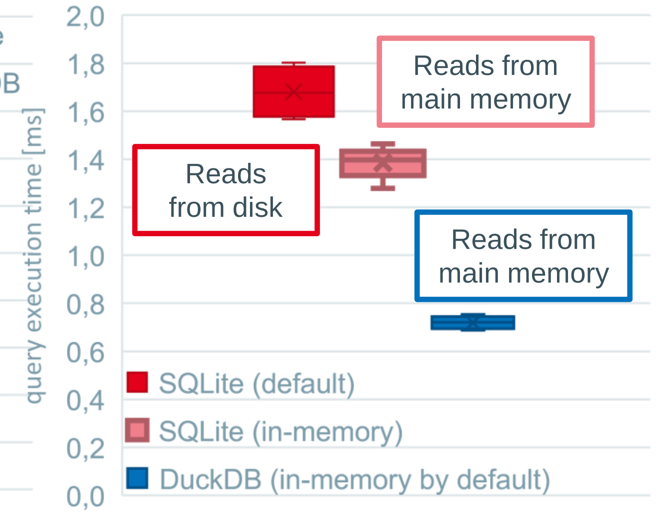
- Column-Store
- In-Memory with out-of-memory option
- Vector-at-a-time processing
- Different Join implementations
- Optimizer actually does stuff (join order optimization, eliminate common subqueries,…)

Load data from csv file (<300kB)



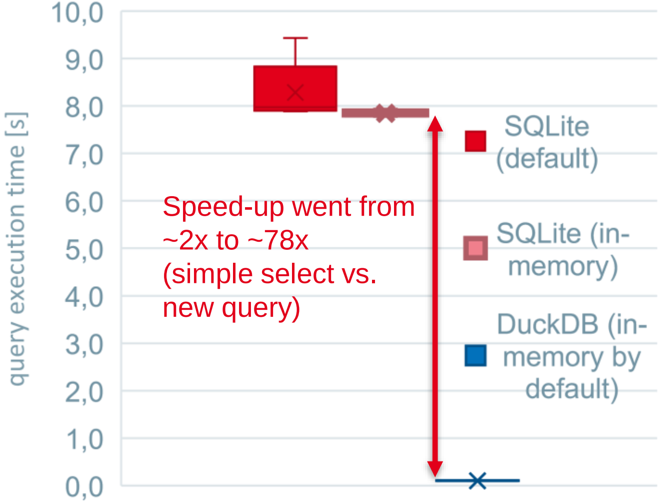
Simple query on small data (<300kB)

SELECT * FROM testsequence WHERE Sequence_Count<3;

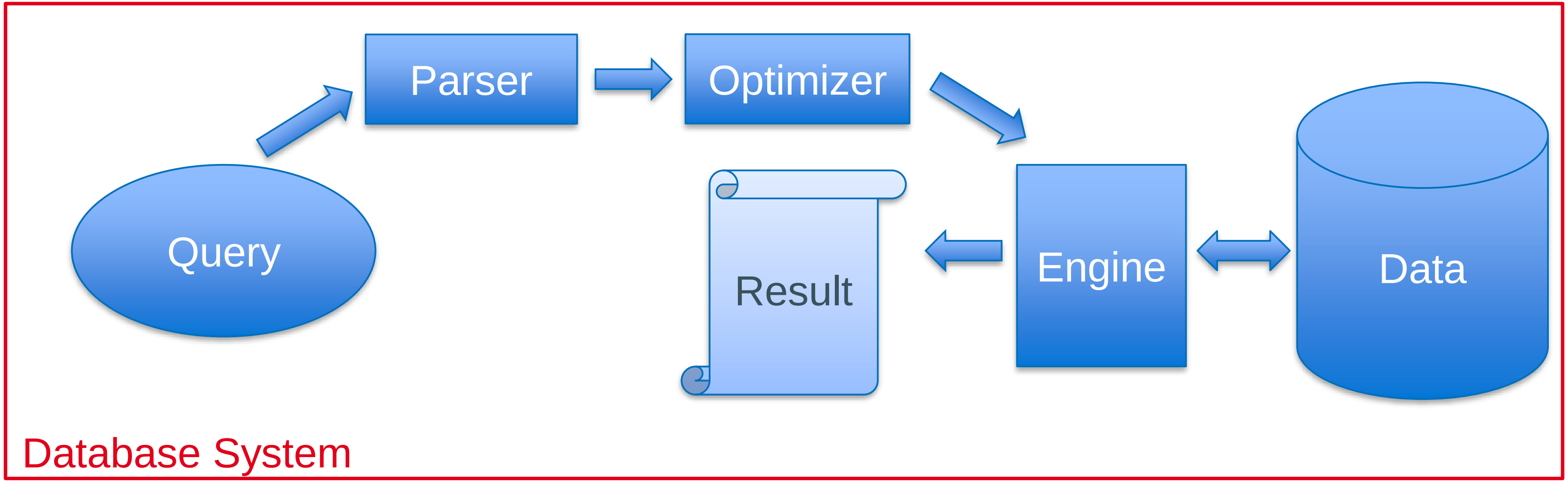


Complex query on slightly more data (~20MB)

SELECT count(*) FROM (SELECT 1 FROM testsequence, authors WHERE Sequence_Count < 3 AND testsequence.PDB_ID = authors.IDCODE GROUP BY authors.author) foo;



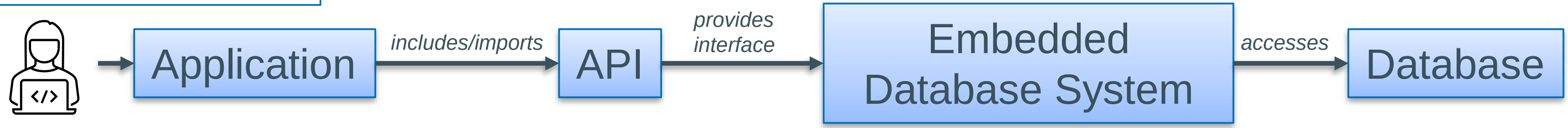
Roadmap



*strongly simplified

The 2 Flavours of Database Systems

Embedded DBS



You Python, R, C++,… project

e.g. Python module, C++ header,…

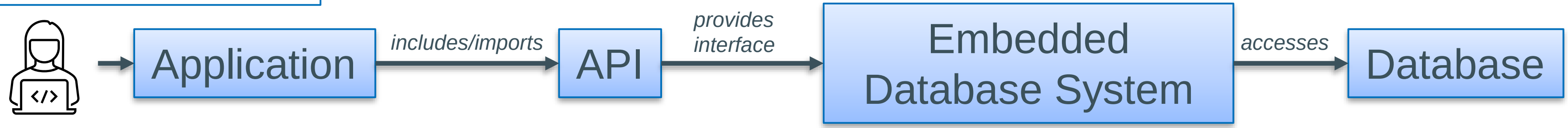
Usually a library which does not need special permissions to be installed, e.g. a *.so file

One or more files containing your data, format depends on your DB system

- No special permissions required
- No bells and whistles (no user management, distributed processing, multiple databases at once,…)
- Runs (almost) everywhere
- Porting to other platforms relatively simple (of course, a container makes it even easier)

The 2 Flavours of Database Systems

Embedded DBS



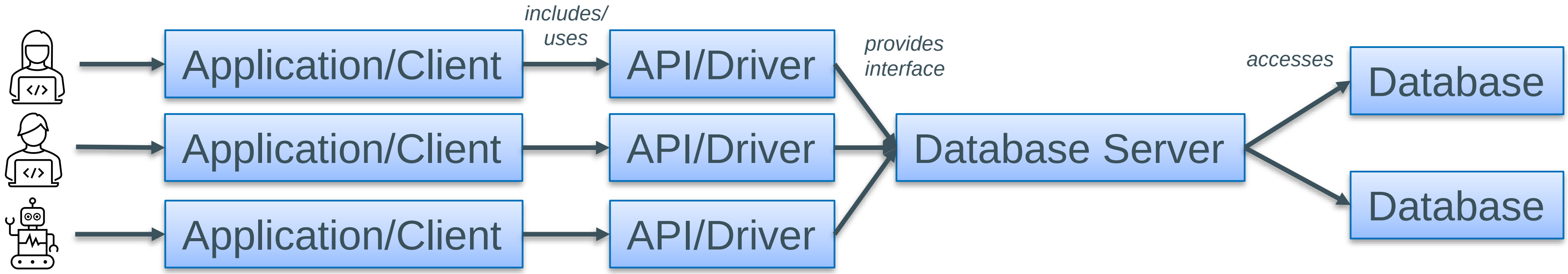
You Python, R, C++,… project

e.g. Python module, C++ header,…

Usually a library which does not need special permissions to be installed, e.g. a *.so file

One or more files containing your data, format depends on your DB system

Client-Server DBS



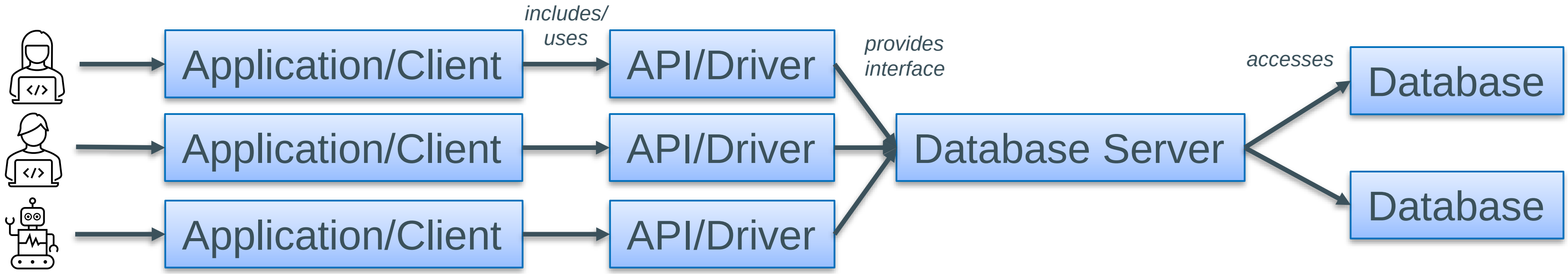
The 2 Flavours of Database Systems

- Usually requires installation as root
- Shipping the whole package (DB, DBS, Application) is challenging

→ Try a container

- Offers more features than Embedded DBS, e.g. user management, data partitioning, drivers for a standardized interface,…

Client-Server DBS



Database Systems

There is no one fits all. Choose wisely!

	Storage	Column Processing	Physical Join Operators	Free?	Open source?	Embedded/ Client-Server
SQLite	Row store	-	Nested Loop	yes	yes	Embedded
PostgreSQL	Row store, Column store available as extension	depends on extension	(Indexed) Nested Loop, Hash Join, Merge Join	yes	yes	Client-Server, 3rd party projects for embedding
MySQL	Row Store	-	Different combinations of Block-based, Indexed, Nested Loop, Hash Join	yes (community version)	yes	Client-Server, Embedded (commercial)
MonetDB	Column store	Operator-at-a-time	Different combinations & variations of Partitioned, Indexed, Nested Loop, Hash Join	yes	yes	Client-Server
MariaDB	Column store + hybrid (multiple versions)	Block-at-a-time	Different combinations of Indexed, Block-based, Nested Loop, Hash Join	yes (community version)	yes	Client-Server
DuckDB	Column store (data blocks)	Block-at-a-time	(Indexed, Block-based) Nested Loop, Merge Join	yes	yes	Embedded

**Information comes from a wild combination of different sources including documentations, papers, and source code. It might be incomplete.*

Database Systems

CDCS
CENTER FOR DATA AND COMPUTING
IN NATURAL SCIENCES

	Use-Cases
SQLite	It's better than not having a database at all.
PostgreSQL	Can do almost everything if you know which extension(s) you need. Might be overkill for what you need.
MySQL	For frequent transactions (e.g. insert new tuples), some features might not be free.
MonetDB	Good all-in-one solution for most analytical use-cases. Runs reliably on all sizes of machines, even on a laptop (and on my mobile phone).
MariaDB	Like MonetDB but with more features, e.g. different storages. Might be a total overkill for your project.
DuckDB	If you don't need bells and whistles for your analytics (e.g. no user management, no server, limited variety of APIs).
Oracle	A database system on steroids, can deal with almost everything including geo-spatial data and data files in the PB range. Costs and arm and a leg, but DESY seems to have the money.

source: https://www.oracle.com/assets/technology-price-list-070617.pdf

Database Systems

CDCS
CENTER FOR DATA AND COMPUTING
IN NATURAL SCIENCES

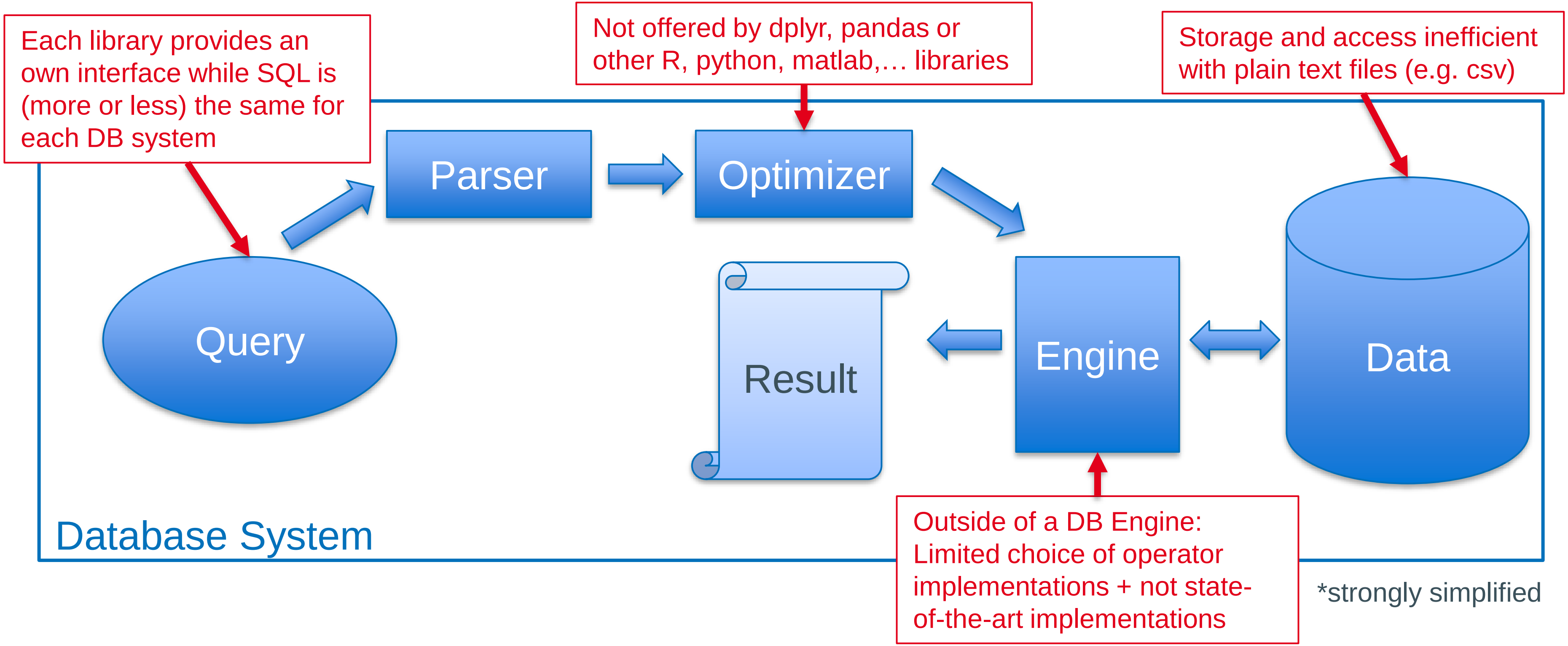
SQLite	It's
PostgreSQL	Car
MySQL	For
MonetDB	Goo … op (and on r
MariaDB	Like
DuckDB	If yo … APIs).
Oracle	A d … ne PB ran Cos

	Named User Plus	Software Update License & Support	Processor License	Software Update License & Support
Database Products				
Oracle Database				
Standard Edition 2	350	77.00	17,500	3,850.00
Enterprise Edition	950	209.00	47,500	10,450.00
Personal Edition	460	101.20	-	-
Mobile Server	-	-	23,000	5,060.00
NoSQL Database Enterprise Edition	200	44	10,000	2,200.00
Enterprise Edition Options:				
Multitenant	350	77.00	17,500	3,850.00
Real Application Clusters	460	101.20	23,000	5,060.00
Real Application Clusters One Node	200	44.00	10,000	2,200.00
Active Data Guard	230	50.60	11,500	2,530.00
Partitioning	230	50.60	11,500	2,530.00
Real Application Testing	230	50.60	11,500	2,530.00
Advanced Compression	230	50.60	11,500	2,530.00
Advanced Security	300	66.00	15,000	3,300.00
Label Security	230	50.60	11,500	2,530.00
Database Vault	230	50.60	11,500	2,530.00
OLAP	460	101.20	23,000	5,060.00
TimesTen Application-Tier Database Cache	460	101.20	23,000	5,060.00
Database In-Memory	460	101.20	23,000	5,060.00
Database Enterprise Management				
Diagnostics Pack	150	33.00	7,500	1,650.00
Tuning Pack	100	22.00	5,000	1,100.00
Database Lifecycle Management Pack	240	52.80	12,000	2,640.00
Data Masking and Subsetting Pack	230	50.60	11,500	2,530.00
Cloud Management Pack for Oracle Database	150	33.00	7,500	1,650.00

source: https://www.oracle.com/assets/technology-price-list-070617.pdf

Summary

CDCS
CENTER FOR DATA AND COMPUTING
IN NATURAL SCIENCES



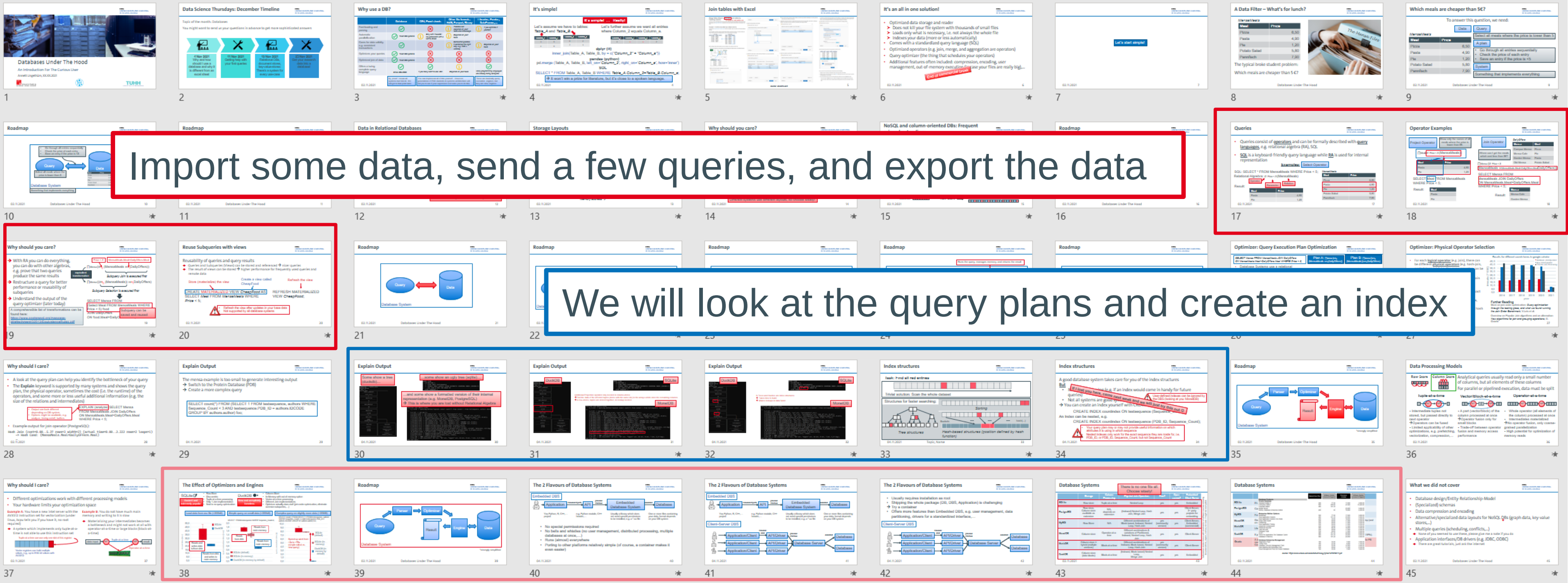
*strongly simplified

What we did not cover

- Database design/Entity-Relationship-Model
- (Specialized) schemas, normalization
- Data compression and encoding
- NoSQL DBs (graph data, key-value stores,...) → **January**
- Concurrent queries (scheduling, conflicts, anomalies...)
- Application interfaces/DB drivers (e.g. JDBC, ODBC)
  → There are great tutorials, just ask the internet
- User-defined functions → for everything you don‘t want to express with SQL
- Anything with a little more depth

Next Week

CDCS
CENTER FOR DATA AND COMPUTING
IN NATURAL SCIENCES



Import some data, send a few queries, and export the data

We will look at the query plans and create an index

We will use DuckDB (because it will likely run on your laptop and is comparatively fast)

CDCS
CENTER FOR DATA AND COMPUTING
IN NATURAL SCIENCES



Databases Under The Hood

An Introduction For The Curious User

Annett Ungethüm, 16.12.2021

Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

DESY.

TUHH
Technische Universität Hamburg-Harburg