Programmierpraktikum: Datensysteme
01 Kickoff and Introduction

Prof. Dr. Matthias Boehm
Technische Universität Berlin
Berlin Institute for the Foundations of Learning and Data
Big Data Engineering (DAMS Lab)

 Last update: Apr 15, 2024

About Me

- **Since 09/2022 TU Berlin**, Germany
  - University professor for Big Data Engineering (DAMS)
- **2018-2022 TU Graz**, Austria
  - BMK endowed chair for data management + research area manager
  - **Data management for data science** (DAMS), **SystemDS & DAPHNE**
- **2012-2018 IBM Research – Almaden**, CA, USA
  - Declarative large-scale machine learning
  - Optimizer and runtime of **Apache SystemML**
- **2007-2011 PhD TU Dresden**, Germany
  - Cost-based optimization of integration flows
  - Time series forecasting / in-memory indexing & query processing

DB group

Agenda

- Course Organization
- Background Data Management
- #1 Query Processing on Raw Data (DAMS)
- #2 Efficient Duplicate Detection (D2IP)
- Course Selection/Enrolment

Course Organization

Basic Course Organization

- **Language**
  - Lectures and slides: **English** (German if preferred)
  - Communication and presentations: **English**/**German**
  - **Informal language** (first name is fine)
  - Offline **Q&A in forum**, answered by teaching assistants

- **Course Format**
  - **6 ECTS** (4 SWS) bachelor computer science / information systems
  - **Every-other-week lectures** (**Mon 4.15pm sharp**, including **Q&A**), **attendance optional**

- **Prerequisites**
  - Basic programming skills in languages such as **C, C++**, **Java**, Rust, etc
  - Basic understanding of data management SQL / RA (or willingness to fill gaps)

Course Goals and Structure

- **Objectives**
  - **Apply basic programming skills** to more complex problem (in self-organized team work)
  - Technical focus on data management and data systems
  - Holistic programming projects: **prototyping, design, versioning, tests, experiments, benchmarks**

- **Grading: Pass/Fail**
  - **Project Implementation** (project source code) [**45%**]
  - **Component and Functional Tests** (test source code) [**10%**]
  - **Runtime Experiments** (achieve performance target) [**15%**]
  - **Documentation** (design document up to 5 pages / code documentation) [**15%**]
  - **Result Presentation** (10min talk) [**15%**]

- **Academic Honesty / No Plagiarism** (incl LLMs like ChatGPT)

Sub-Course Offerings

- **#1 Query Processing on Raw Data**
  - Capacity: 36+/48
  - Organized by **DAMS** group
  - Broad technical focus
  - Lectures every-other-week in **H 0111**

- **#2 Efficient Duplicate Detection**
  - Capacity: 12/48
  - Organized by **D2IP** group
  - Focus on entity resolution
  - Lectures in **TEL-12?** seminar room

➔ **Admitted Students:**
  - 28 + ~5 via email + 73 on ISIS (incl duplicates)
  - **Total registrations: up to 48**
    → 12 teams, 4 students each

Background Data Management

History 1970/1980s
Relational Database Systems

SQL Standard
(SQL-**86**)

↑

SEQUEL

Oracle, IBM DB2, Informix, Sybase
→ MS SQL

QUEL - - - → **SEQUEL**

Ingres @ UC Berkeley
(Stonebraker et al.,
Turing Award '14)

System R @ IBM
Research – Almaden
(Jim Gray et al.,
Turing Award '98)

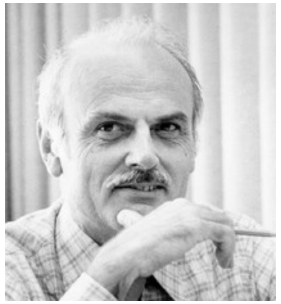
Tuple Calculus

Relational Algebra

Relational Model

Goal: Data Independence
(physical data independence)

- Ordering Dependence
- Indexing Dependence
- Access Path Dependence



Edgar F. "Ted" Codd @ IBM Research (Turing Award '81)

[E. F. Codd: A Relational Model of Data for Large Shared Data Banks. Comm. ACM 13(6), **1970**]

Success of SQL / Relational Model

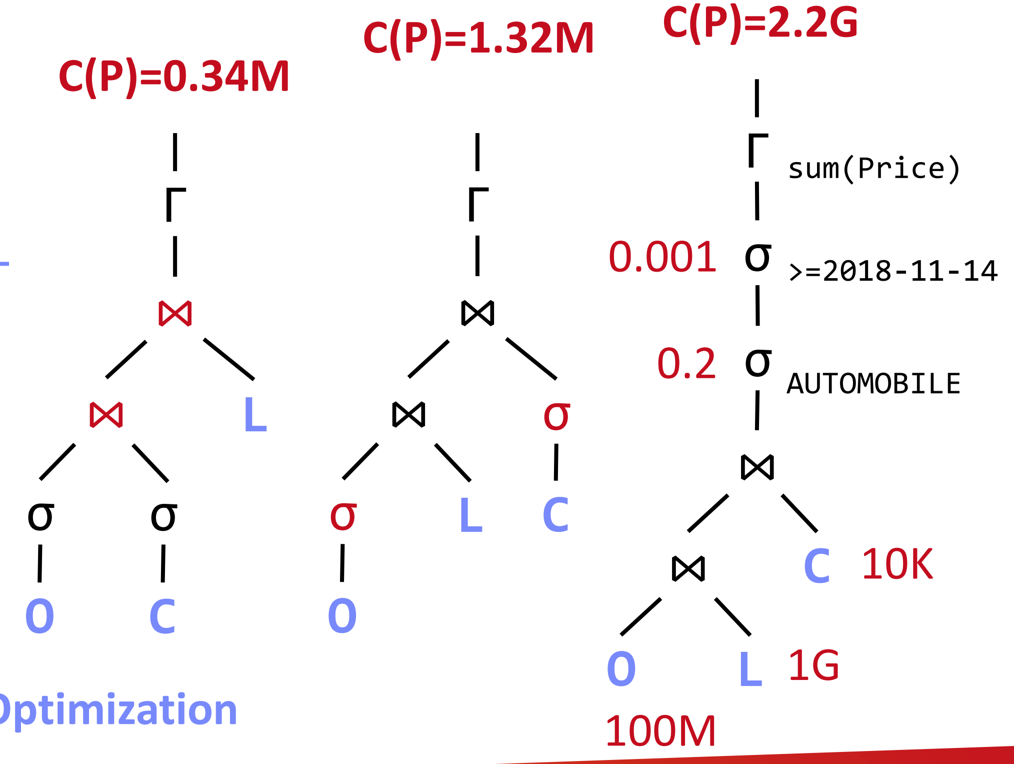
#1 Declarative: what not how

#2 Flexibility: closure property → composability

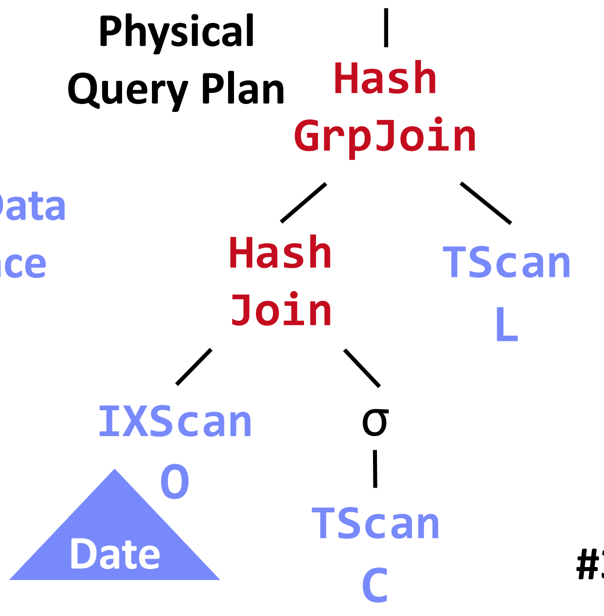
Query:

```
SELECT O_OID, sum(L_Price)
FROM Orders, Lineitem, Customer
WHERE O_OID = L_OID AND O_CID = C_CID
   AND O_Odate >= '2018-11-14'
   AND C_Msegment = 'AUTOMOBILE'
GROUP BY O_OID
```

Logical Query Plans



#4 Physical Data Independence



#3 Automatic Optimization

Query Processing – Iterator Model

[Goetz Graefe: Volcano - An Extensible and Parallel Query Evaluation System. **IEEE Trans. Knowl. Data Eng. 1994**]

- **Volcano Iterator Model**
  - **Open-Next-Close** (ONC) interface
  - Query execution from root node (pull-based) → **Pipelined**

Scalable (small memory)
High CPI measures

- **Example**
  $\sigma_{A=7}(R)$

```
void open() { R.open(); }

void close() { R.close(); }

Record next() {
  while( (r = R.next()) != EOF )
    if( p(r) ) //A==7
      return r;
  return EOF;
}
```

```
open()
  next()
    next() → EOF
      close()
        |
open()
next()
next()      σA=7  → EOF
close()
              |
open()
next()
next()        R
next()
next()              → EOF
close()
```

PostgreSQL: `Init()`, `GetNext()`, `ReScan()`, `MarkPos()`, `RestorePos()`, `End()`

- **Blocking Operators**
  - Sorting, grouping/aggregation, build-phase of (simple) hash joins

#1 Query Processing on Raw Data (DAMS)

Additional Course Logistics

- **Staff**
  - **Lecturer:** Prof. Dr. Matthias Boehm
  - **Teaching Assistants:** Christina Dionysio, David Justen



Each teams gets a mentor
Q&A sessions on demand

- **Next Dates/Lectures**
  - Apr 22: Course Selection; team preferences, otherwise assignment
  - Apr 29: **Background Relational Algebra**
  - May 13: **Background Query Processing**
  - May 27: **Background Query Optimization**
  - Jun 10: **Experiments and Reproducibility**
  - **Jul 01:** Project submissions (**performance target:** 4x faster than reference implementation)
  - **Jul 08:** Project presentations (10min per team, mandatory attendance)

- **Infrastructure**
  - Setup your own private Github/Gitlab repository

Query Processing on Raw Data – Motivation

- **DBMS for Exploratory Data Analysis**
  - **#1: Define a schema** for the data
  - **#2: Load the data**
  - **#3: Tune the system** for the expected workload

"To DB, or Not to DB"

- **Vision**

[Stratos Idreos, Ioannis Alagiannis, Ryan Johnson, Anastasia Ailamaki: **Here are my Data Files. Here are my Queries. Where are my Results? CIDR 2011**]

- **Initial System**

[Ioannis Alagiannis, Renata Borovica, Miguel Branco, Stratos Idreos, Anastasia Ailamaki: NoDB: Efficient Query Execution on Raw Data Files. **SIGMOD 2012**]

SIGMOD'22 Test-of-Time Award

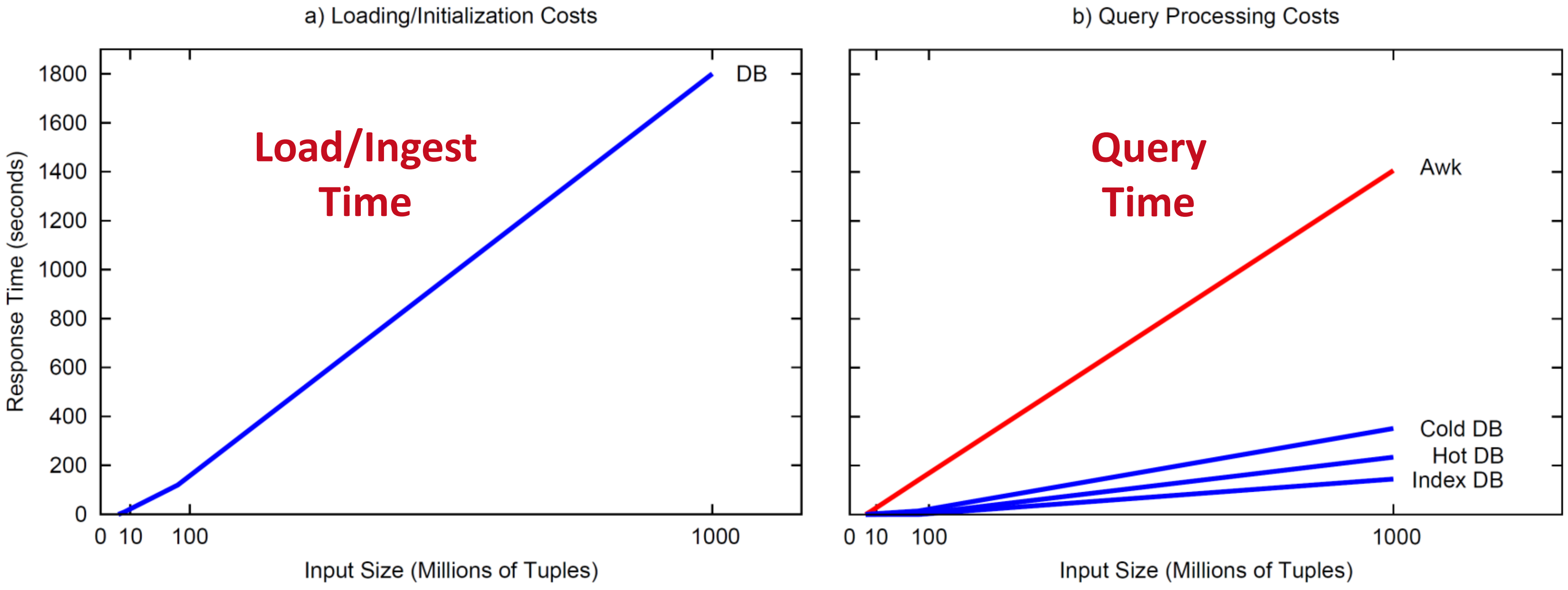
- **Lots of Follow-up Work**
  - Heterogeneous data sources
  - RAW Labs – The NoDB Company: https://www.raw-labs.com/

[Manos Karpathiotakis, Ioannis Alagiannis, Anastasia Ailamaki: Fast Queries Over Heterogeneous Data Through Engine Customization. **PVLDB 9(12), 2016**]

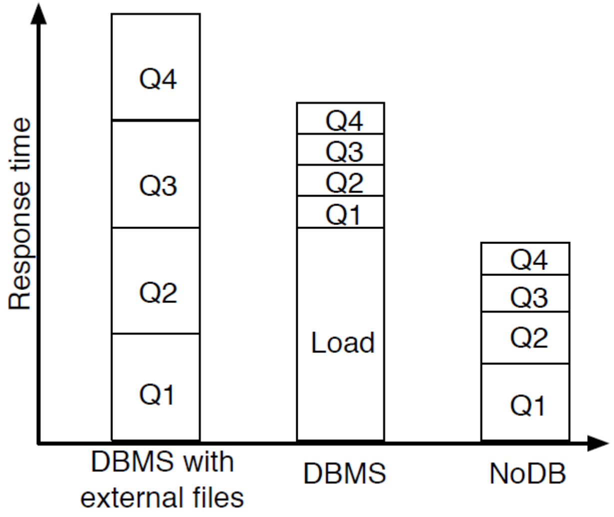
Query Processing on Raw Data – Vision and Goals

- **Initial Experiments**
  - DBMS: MonetDB
  - Tables w/ 4 int columns

```
SELECT sum(a1), min(a4), max(a3),avg(a2)
  FROM R
  WHERE a1>v1 AND a1<v2 AND a2>v3 AND a2<v4
```

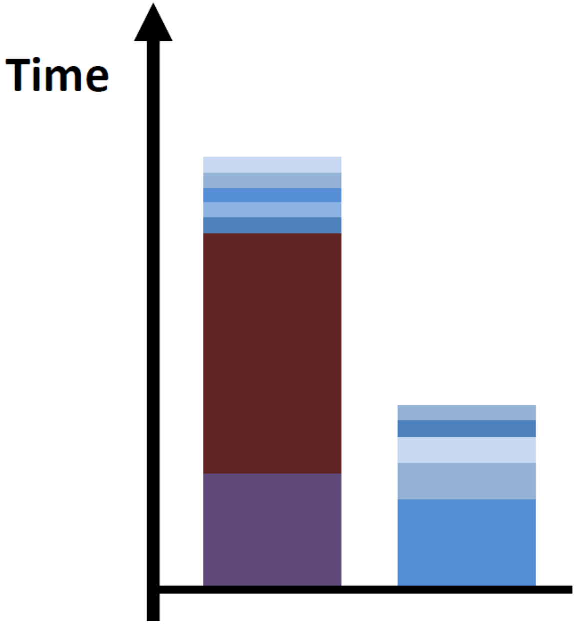


➔ **Vision:** Hybrid system



Query Processing on Raw Data – The NoDB Philosophy

- **Key Principles of NoDB Philosophy**



No data loading

Instant gateway to data

Raw files first-class citizen

Driven by the workload

- **Directions**
  - **#1** Minimizing the cost of raw data access (special data structures)
  - **#2** Selectively eliminating raw data access (caching, scheduled raw access)

Query Processing on Raw Data – Efficient CSV Parsing

- **Parsing and Tokenization**
  - Needed when accessing raw data
  - Parsing: identify row boundaries (\n) → tuple
  - Tokenization: identify tuple attributes (delimiter), and convert strings to types

Assumes non-quoted tokens

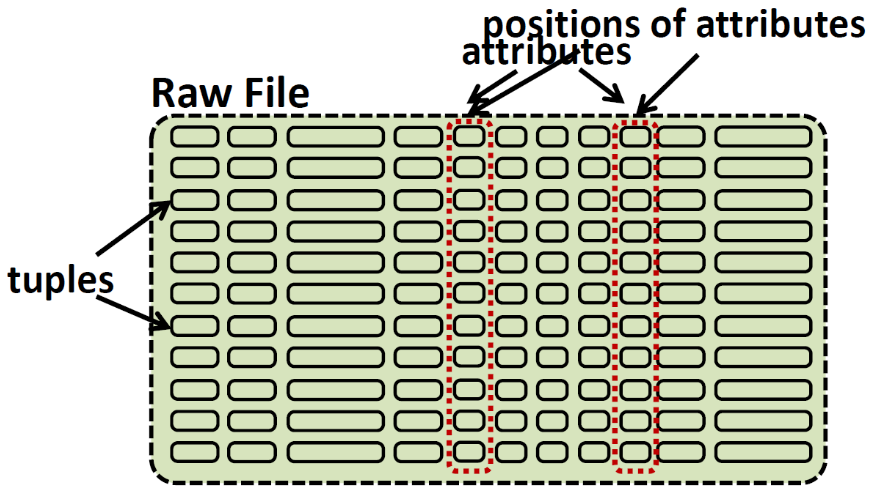
- **#1 Selective Tokenization**
  - If query needs 4th and 8th attribute, stop tokenization at 8th attribute
    → no I/O reduction, reduced tokenization effort

In PostgreSQL, any physical op can act as projection / selection

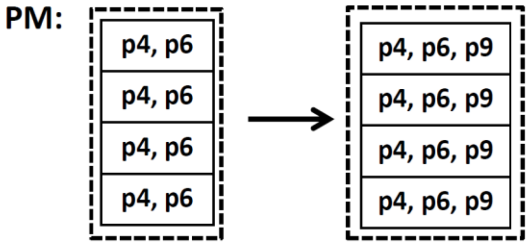
- **#2 Selective Parsing**
  - If query needs 4th and 8th attribute, convert only 4th and 8th attribute
  - Defer parsing 8th attribute if 4th and 8th attribute are used in conjunctive filters
- **#3 Selective Tuple Formation**
  - Tuple construction after selections (only qualifying) tuples

Query Processing on Raw Data – Efficient CSV Parsing, cont.

- **Positional Map**
  - Store metadata on structure of CSV files to navigate and retrieve raw data faster
  - **Goal:** Learn as much as possible from data already touched by other queries
  - Example: Pos of 4th and 8th attributes
  - Allows direct or "close" access



- **Similar to Database Cracking**
  - Q1 accesses a4 and a6; Q2 accesses a4 and a9



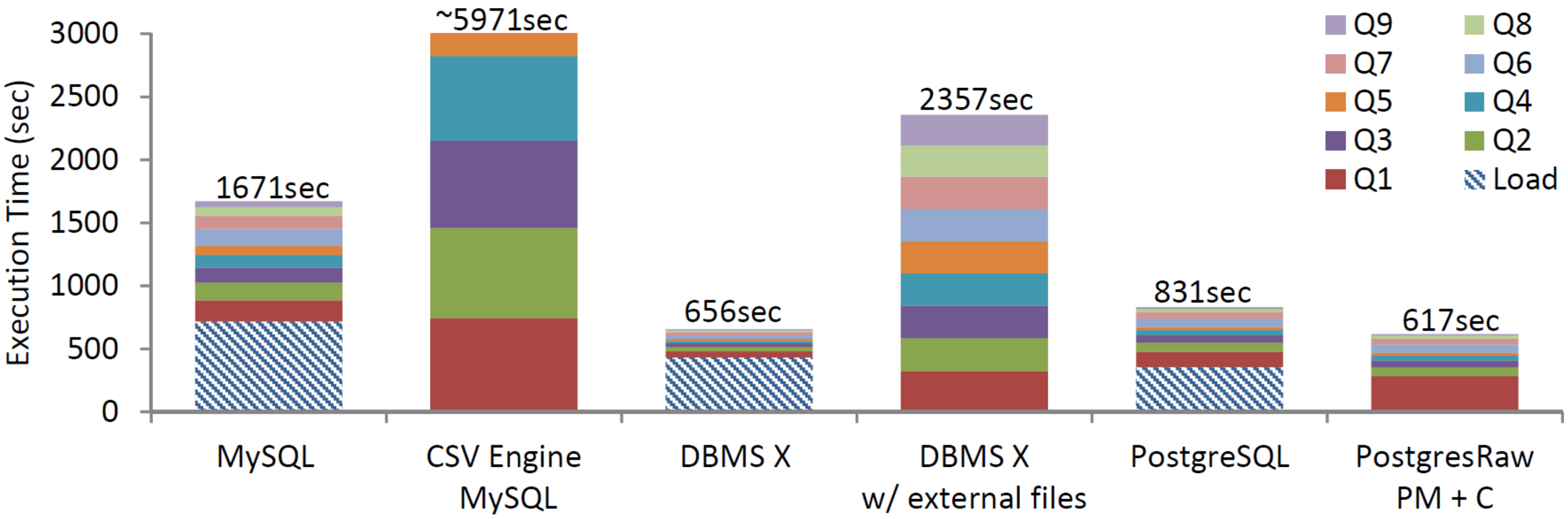
[Stratos Idreos, Martin L. Kersten, Stefan Manegold: Database Cracking. **CIDR 2007**]

Make raw data access progressively cheaper

Positional Map can be larger than original data

Query Processing on Raw Data – End-to-end Experiments

- **Setup**
  - 7.5M rows x 150 columns, 9 queries with different selectivity
  - Q1: 100% rows/cols, Q2-5: decrease rows by 20%, Q6-9: decrease cols by 20%



- **Additional experiments:** TPC-H, FITS format, Stats Impact

API, Reference Implementation, and Task Description

[https://mboehm7.github.io/teaching/ss24_ppds/index.htm]

- **Application Programming Interface (API)**
  - Provided **C++** and **Java** APIs
  - Includes **basic tests and benchmark**
  - Other language perfectly fine

```java
public interface ONCIterator {
    /**
     * Initializes the iterator, and allocates necessary resources.
     */
    public void open();

    /**
     * Returns the next qualifying record or null to indicate end-of-file (EOF).
     * The returned records can be internally reused on any subsequent next call.
     *
     * @return next record, null for EOF
     */
    public Record next();

    /**
     * Closes the iterator, and frees any resources allocated during
     * invocations of open or next.
     */
    public void close();
}
```

```java
public abstract class QueryProcessor {
    /**
     * This method compiles a physical query execution plan (QEP)
     * (directly executable) from a given logical query plan. The QEP
     * should be composed of operators implementing the ONCIterator
     * interface
     *
     * @param node root node of the logical query plan.
     * @return root iterator of the query execution plan.
     */
    public abstract ONCIterator compileQuery(PlanNode node);


    /**
     * This method compiles and executes a given logical query plan,
     * and returns the results as a materialized list.
     *
     * @param node root node of the logical query plan.
     * @return query results
     */
    public List<Record> executeQuery(PlanNode node) {
        //step 1: compile logical plan to physical QEP
        ONCIterator iter = compileQuery(node);

        //step 2: execute query and buffer results
        List<Record> ret = new ArrayList<>();
        Record r = null;
        iter.open();
        while( (r = iter.next()) != null ) {
            //copy record because iterators might reuse
            ret.add(new Record(r));
        }
        iter.close();
        return ret;
    }
}
```

- **Reference Implementation in C++**

- **Task Description for unclear Submission Details**
  - Published: Apr 13

- **Test System**
  - **HW:** Two Intel Xeon Gold 6338 CPUs@2.2-3.2 GHz (64 physical/128 virtual cores), 1 TB DDR4 RAM, 16x SATA SSDs (in RAID-0), 2x A40 GPUs.
  - **OS:** Ubuntu 20.04, **C/C++ Compiler:** gcc/g++ 11, **Java-11:** Openjdk 11.0.20

#2 Efficient Duplicate Detection (D2IP)

Course Selection/Enrolment

Select Your Course

- **#1 Query Processing on Raw Data (DAMS)**
  - Capacity: 36+/48
- **#2 Efficient Duplicate Detection (D2IP)**
  - Capacity: 12/48

https://tinyurl.com/5c5cbedm

Summary & QA

- Course Organization
- Background Data Management
- #1 Query Processing on Raw Data (DAMS)
- #2 Efficient Duplicate Detection (D2IP)
- Course Selection/Enrolment by Apr 22 EOD

Thanks

https://tinyurl.com/5c5cbedm