Do Programming Languages need Query Languages?

Jelle Hellings

Department of Computing and Software
McMaster University
1280 Main St. W., Hamilton, ON L8S 4L7, Canada

Do Programming Languages need Query Languages?

(spoiler alert)

Yes they do!

Programming and data processing

Claims

1. Data processing plays a central role in programming.

Programming and data processing

Claims

1. Data processing plays a central role in programming.

Examples: standard support libraries

1. *Collections* to store data
   binary search trees, hash tables, tuples, dynamic arrays, ...
2. *Algorithms* to operate on data
   sorting, filtering, transforming, set operations, ...

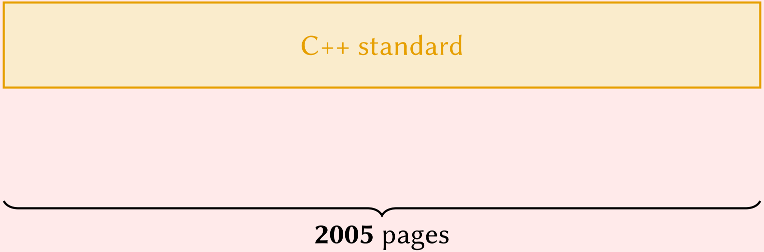
Programming and data processing

Claims

1. Data processing plays a central role in programming.

Working Draft of the C++ standard

(Document Number: N4964, Date 2023-10-15.)



C++ standard

2005 pages

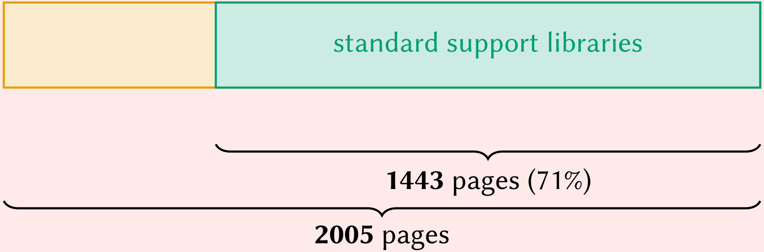
Programming and data processing

Claims

1. Data processing plays a central role in programming.

Working Draft of the C++ standard

(Document Number: N4964, Date 2023-10-15.)



standard support libraries

1443 pages (71%)

2005 pages

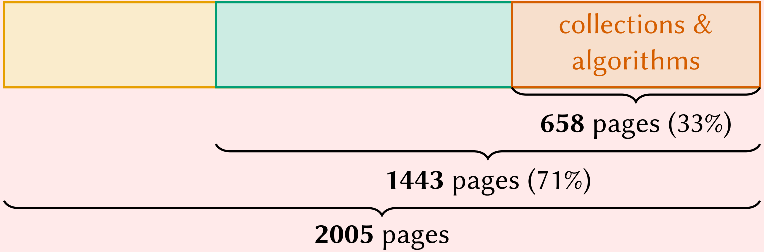
Programming and data processing

Claims

1. Data processing plays a central role in programming.

Working Draft of the C++ standard

(Document Number: N4964, Date 2023-10-15.)



collections & algorithms

658 pages (33%)

1443 pages (71%)

2005 pages

Not counting: numeric, time, formatting, IO, threads, ...

An example: How to *program* relational algebra in C++

An example: How to *program* relational algebra in C++

Data storage: `std::tuple` (single row), `std::set` (rows).

An example: How to *program* relational algebra in C++

Data storage: `std::tuple` (single row), `std::set` (rows).

- **Projection** (π_{columns}): `std::transform`.
- **Selection** ($\sigma_{\text{conditions}}$): `std::copy_if`, `std::views::filter`.
- **Joins** ($\times$, $\bowtie$): loops, `std::views::cartesian_product`.
- **Set operations** ($\cap$, $\cup$, $\setminus$): `std::set_union`, ….

An example: How to *program* relational algebra in C++

Data storage: `std::tuple` (single row), `std::set` (rows).

- **Projection** (π_{columns}): `std::transform`.
- **Selection** ($\sigma_{\text{conditions}}$): `std::copy_if`, `std::views::filter`.
- **Joins** ($\times$, $\bowtie$): loops, `std::views::cartesian_product`.
- **Set operations** ($\cap$, $\cup$, $\setminus$): `std::set_union`, ….

- **Sorting**: `std::sort`, `std::stable_sort`.
- **Deduplication**: `std::unique_copy`.
- **Aggregation**: `std::accumulate` and `std::reduce`, `std::ranges::fold_left`.

An example: How to *program* relational algebra in C++

Data storage: `std::tuple` (single row), `std::set` (rows).

- **Projection** (π_{columns}): `std::transform`.
- **Selection** ($\sigma_{\text{conditions}}$): `std::copy_if`, `std::views::filter`.
- **Joins** ($\times$, $\bowtie$): loops, `std::views::cartesian_product`.
- **Set operations** ($\cap$, $\cup$, $\setminus$): `std::set_union`, ….

- **Sorting**: `std::sort`, `std::stable_sort`.
- **Deduplication**: `std::unique_copy`.
- **Aggregation**: `std::accumulate` and `std::reduce`, `std::ranges::fold_left`.

- **Parallelization**: execution policies (algorithms), `std::async` (evaluation), ….
- **Pipelined execution**: `std::ranges`, coroutines using `std::generator`, ….

An example: How to *program* relational algebra in C++

$$\pi_{R.\text{parent}}(\sigma_{R.\text{child}=S.\text{name}}(\rho_R(\text{parentOf}) \times \sigma_{S.\text{place}=\text{“Hamilton”}}(\rho_S(\text{person}))))$$

An example: How to *program* relational algebra in C++

$$\pi_{R.\text{parent}}(\sigma_{R.\text{child}=S.\text{name}}(\rho_R(\text{parentOf}) \times \sigma_{S.\text{place}=\text{“Hamilton”}}(\rho_S(\text{person}))))$$

```
using namespace std::views;

auto where_pred = [](auto l) { return l.place == "Hamilton"; };

auto product_pred = [](auto t) {
  auto [po, p] = t; return po.child == p.name; };

auto join = cartesian_product(parents, persons | filter(where_pred));
for (auto [po, p] : join | filter(product_pred)) {
    std::cout << po.parent << std::endl;
}
```

Programming and data processing

Claims

1. Data processing plays a central role in programming.
2. Programming languages are *bad* at data processing.

Programming and data processing

Claims

1. Data processing plays a central role in programming.
2. Programming languages are *bad* at data processing.

Efficient data processing

"*Complex*" data processing algorithms even for *simple* data processing tasks.
E.g., join algorithms, join ordering, index usage, selection push down, ….

This complexity is delegated to the programmer.

An example: A *more-efficient* query in C++

```
/* parents is a set, ordered on (parent, child).
 * persons is a set, ordered on (name, place). */

auto where_pred = [](auto l) { return l.place == "Hamilton"; };
auto filtered = persons | filter(where_pred)
                        | std::ranges::to<std::vector>();

for (auto& [pname, cname] : parents) {
  person_t pcname, "Hamilton";
  bool has_child = std::binary_search(filtered.begin(),
                                      filtered.end(), p);
  if (has_child) {
    std::cout << pname << std::endl;
  }
}
```

An example: A *more-efficient* query in C++

```
/* parents is a set, ordered on (parent, child).
 * persons is a set, ordered on (name, place). */

for (auto& [pname, cname] : parents) {
  person_t pcname, "Hamilton";
  bool has_child = persons.contains(p);
  if (has_child) {
    std::cout << pname << std::endl;
  }
}
```

An example: A *more-efficient* query in C++

	Runtime complexity	Memory usage
Original	$\Theta(\lvert\text{parents}\rvert \cdot \lvert\text{persons}\rvert)$	$\Theta(1)$
First variant	$\Theta(\lvert\text{parents}\rvert \log \lvert\text{filtered}\rvert + \lvert\text{persons}\rvert)$	$\Theta(\lvert\text{filtered}\rvert)$
Second variant	$\Theta(\lvert\text{parents}\rvert \log \lvert\text{persons}\rvert)$	$\Theta(1)$

Both “improved” versions can be significantly improved to reduce overheads!

An example: A *more-efficient* query in C++

	Runtime complexity	Memory usage
Original	$\Theta(\lvert\text{parents}\rvert \cdot \lvert\text{persons}\rvert)$	$\Theta(1)$
First variant	$\Theta(\lvert\text{parents}\rvert \log \lvert\text{filtered}\rvert + \lvert\text{persons}\rvert)$	$\Theta(\lvert\text{filtered}\rvert)$
Second variant	$\Theta(\lvert\text{parents}\rvert \log \lvert\text{persons}\rvert)$	$\Theta(1)$

Both “improved” versions can be significantly improved to reduce overheads!

What if we want *unique* parents?

An example: A *more-efficient* query in SQL

```
SELECT pname
FROM parents R, persons S
WHERE R.child = S.name AND S.place = "Hamilton";
```

An example: A *more-efficient* query in SQL

```
SELECT DISTINCT pname
FROM parents R, persons S
WHERE R.child = S.name AND S.place = "Hamilton";
```

An example: A *more-efficient* query in SQL

```
SELECT DISTINCT pname
FROM parents R, persons S
WHERE R.child = S.name AND S.place = "Hamilton";
```

Some database systems need a helping hand...

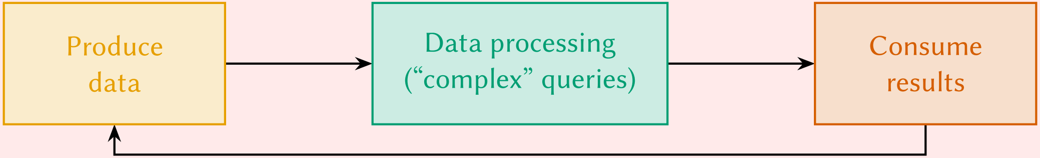
```
SELECT DISTINCT pname
FROM parents
WHERE child IN (SELECT name
                FROM persons
                WHERE place = "Hamilton");
```

Programming and data processing

Claims

1. Data processing plays a central role in programming.
2. Programming languages are *bad* at data processing.
3. We need database technology in our programming languages.

Potential solution?


Produce data

Data processing (“complex” queries)

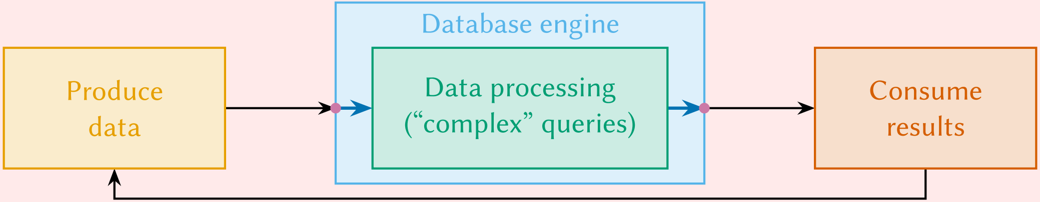
Consume results

Programming and data processing

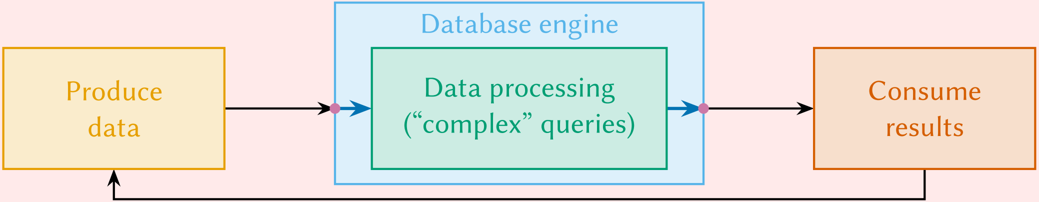
Claims

1. Data processing plays a central role in programming.
2. Programming languages are *bad* at data processing.
3. We need database technology in our programming languages.

Potential solution?



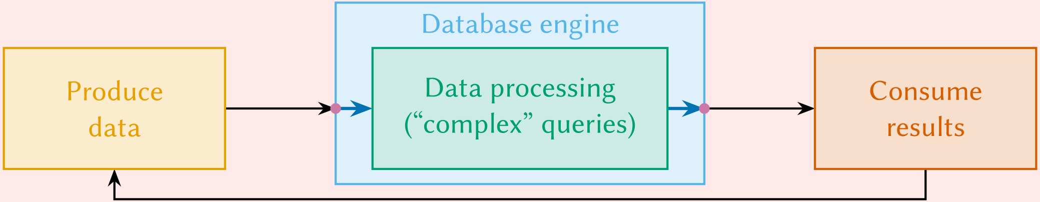
Database systems are *not* the solution



What if we use a database system

For example, PostgreSQL (is free!)

Database systems are *not* the solution



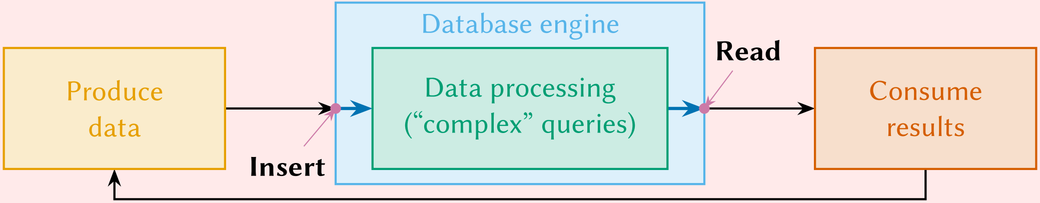
What if we use a database system

For example, PostgreSQL (is free!)

Not all data is always in a database system.

- There might not be a database system.

Database systems are *not* the solution



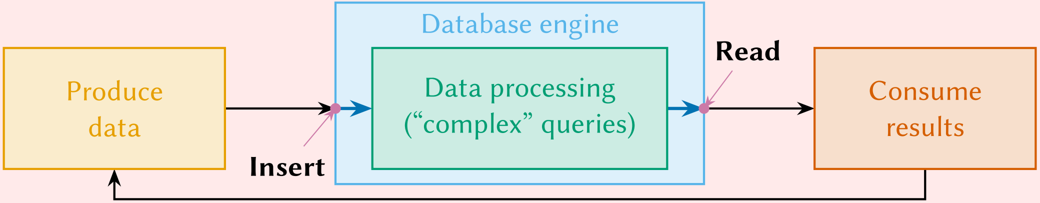
What if we use a database system

For example, PostgreSQL (is free!)

Not all data is always in a database system.

- There might not be a database system.
- Inserting data into and reading data from the system is *not free*:
  Data transfers, necessary code to "translate" between types, ...

Database systems are *not* the solution



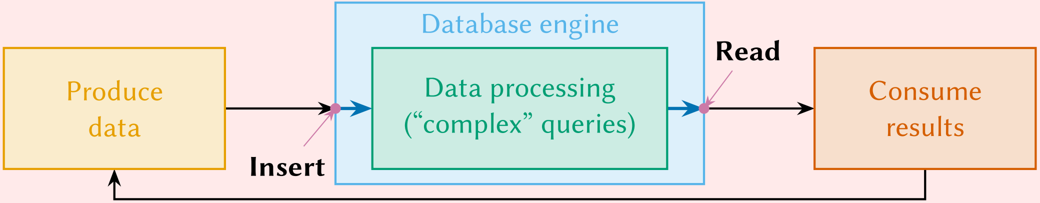
What if we use a database system

For example, PostgreSQL (is free!)

Not all data is always in a database system.

- There might not be a database system.
- Inserting data into and reading data from the system is *not free*:
  Data transfers, necessary code to “translate” between types, ….
- Conceptual mismatches: database code does not mix well with other code!
  For example, type safety (program) versus no type safety (query results).

Database systems are *not* the solution



What if we use a database system

For example, ~~PostgreSQL~~ SQLLite (is free!)

Not all data is always in a database system.

- ~~There might not be a database system~~.
- Inserting data into and reading data from the system is *not free*:
  Data transfers, necessary code to "translate" between types, ….
- Conceptual mismatches: database code does not mix well with other code!
  For example, type safety (program) versus no type safety (query results).

Proposed solution

What if…

```
query<"%(parent) :- parents(parent, c),"
      "              persons(c, 'Hamilton')">(dataset);
```

is valid C++ code in which **`query`** implements an *optimized* C++ algorithm.

Proposed solution

What if…

```
query<"%(parent) :- parents(parent, c),"
      "              persons(c, 'Hamilton')">(dataset);
```

is valid C++ code in which **query** implements an *optimized* C++ algorithm.

How to do so

Create a C++ library that

- Provides a *domain specific query language* for C++.
- At C++ compile time, the library *compiles these queries* into C++ algorithms: derive result types, derive query evaluation strategy, ….
- At runtime: these queries are *normal C++ functions*.

Proposed solution: Feasibility and status

- C++ compile-time support for *domain specific languages*:
  Almost feature complete (optimistic: publication in 2024?).
- C++ compile-time query compiler:
  Proof-of-concept complete → we can turn high-level Datalog into algorithms.
- Compile-time query planner and query optimizer: *more work needed*.

Proposed solution: Feasibility and status

- C++ compile-time support for *domain specific languages*:
  Almost feature complete (optimistic: publication in 2024?).
- C++ compile-time query compiler:
  Proof-of-concept complete → we can turn high-level Datalog into algorithms.
- Compile-time query planner and query optimizer: *more work needed*.

Challenges

```
std::set<my_fancy_type, my_fancy_ordering> my_fancy_dataset;
```

- Which fields does `my_fancy_type` define?
- What ordering does `my_fancy_ordering` provide? Keys? Index?

Proposed solution: Feasibility and status

- C++ compile-time support for *domain specific languages*:
  Almost feature complete (optimistic: publication in 2024?).
- C++ compile-time query compiler:
  Proof-of-concept complete → we can turn high-level Datalog into algorithms.
- Compile-time query planner and query optimizer: *more work needed*.

Challenges

```
std::set<my_fancy_type, my_fancy_ordering> my_fancy_dataset;
```

- Which fields does `my_fancy_type` define?
- What ordering does `my_fancy_ordering` provide? Keys? Index?

Solution. Sensible defaults where possible, else provide such information via *traits classes*.

Proposed solution: Feasibility and status

- C++ compile-time support for *domain specific languages*:
  Almost feature complete (optimistic: publication in 2024?).
- C++ compile-time query compiler:
  Proof-of-concept complete → we can turn high-level Datalog into algorithms.
- Compile-time query planner and query optimizer: *more work needed*.

Challenges

```
std::set<my_fancy_type, my_fancy_ordering> my_fancy_dataset;
```

- Which fields does `my_fancy_type` define?
- What ordering does `my_fancy_ordering` provide? Keys? Index?

Solution. Sensible defaults where possible, else provide such information via *traits classes*.

Reflection (potential future C++ standards) might partly help.

Proposed solution: Feasibility and status

Source: https://xkcd.com/303/ by Randall Munroe.

Query compilation and evaluation

We can use a database-style query planner and query optimizer.
A compile time environment is different, however.

Zero-cost principle

Our *abstraction* of a domain specific query language should not introduce costs (when compared to a handwritten algorithm).

Query compilation and evaluation

We can use a database-style query planner and query optimizer.
A compile time environment is different, however.

Zero-cost principle

Our *abstraction* of a domain specific query language should not introduce costs (when compared to a handwritten algorithm).

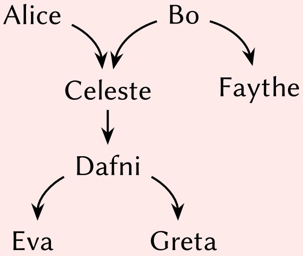
- *Only-once* query optimization (one compile time, many executions).
- Query optimization *without* access to the data.
- At runtime: algorithms can make choices (e.g., `if-else`),
  But there is only limited information: C++ containers do not have heuristics.

Demonstration—If time allows

Base relations

- *person*(id, name) assigns names to nodes.
- *parentOf*(parent, child) relates persons.

Example dataset


Alice
Bo
Celeste
Faythe
Dafni
Eva
Greta

Demonstration—If time allows

Example queries

```
grandparentOf(0, 1) :- parentOf(0, 2), parentOf(2, 1);

grandparentOnly(0) :-  parentOf(0, 1), parentOf(1, 2);

ancestorOf(0, 1) :- parentOf(0, 1);
ancestorOf(0, 1) :- parentOf(0, 2), ancestorOf(2, 1);

ancestorNamed(0, 1) :- ancestorOf(2, 3), person(2, 0), person(3, 1);

siblingOf(0, 1) :- parentOf(2, 0), parentOf(2, 1);
```

Evaluation: Two perspectives

First perspective: How do we compare with existing database products.

Evaluation: Two perspectives

First perspective: How do we compare with existing database products.

Second perspective: How do we compare with (C++) *programmers*?

- Performance?
- Readability?
- Ease-of-use?

Evaluation: Two perspectives

First perspective: How do we compare with existing database products.

Second perspective: How do we compare with (C++) *programmers*?

- Performance?
- Readability?
- Ease-of-use?

Central question

Should we switch to *declarative languages* even in traditional *procedural* source code?

Research Volunteers Needed

To evaluate our research, we will compare with data processing programs written in C++.

You can provide these C++ programs via an online questionnaire.

- Letter of Information `https://jhellings.nl/q/loi.pdf`.
- Questionnaire at `https://jhellings.nl/q/`.

The questionnaire takes 30min–120min and participation is entirely voluntarily.

This study has been reviewed by and received ethics clearance from the
McMaster Research Ethics Board (#6885).

Conclusion

Programming Languages do need Query Languages!

Questionnaire at `https://jhellings.nl/q/`.