Hasso Plattner Institut
IT Systems Engineering | Universität Potsdam

Build your own Database

Week 5

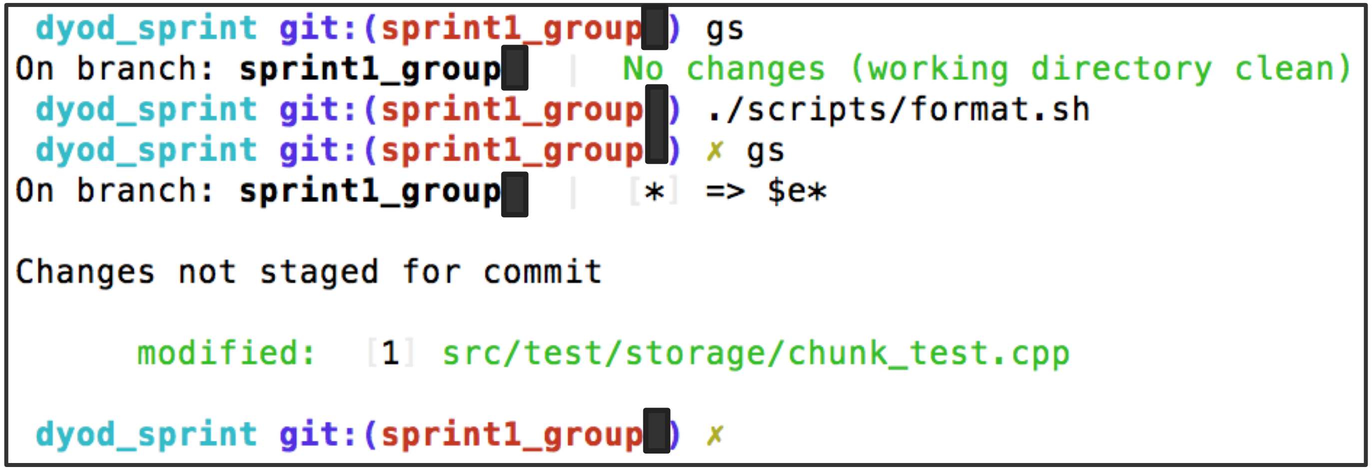
Agenda

- A few minor things
- std::optional
- Lambdas
- Relational Algebra and Operators in Opossum
- Presentation of Sprint 3

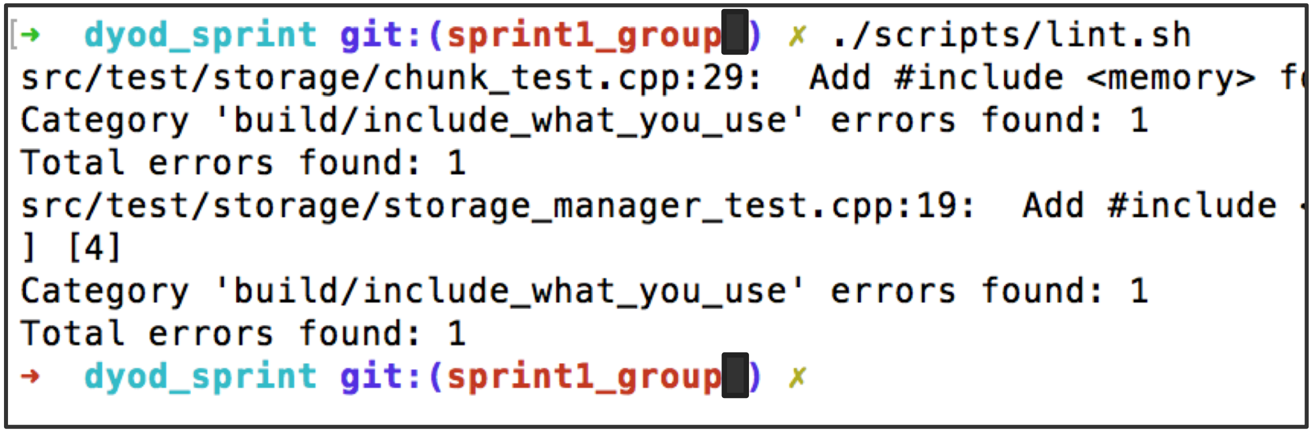
Sprint 2

Reviews: see Piazza

Formatting / Linting

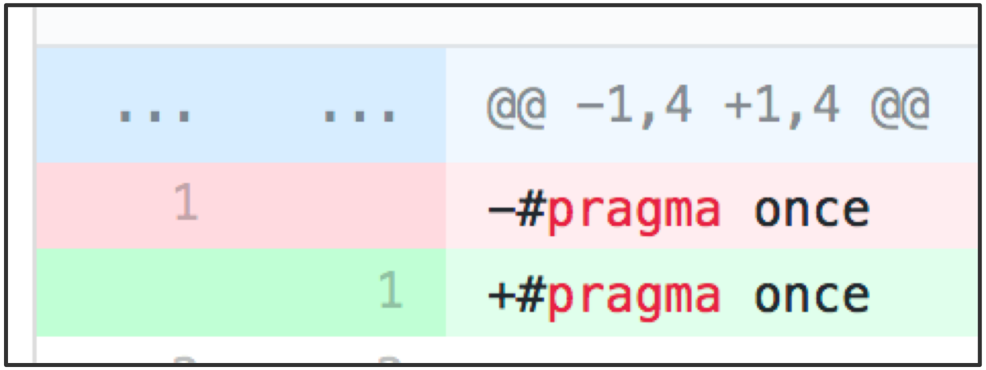


```
 dyod_sprint git:(sprint1_group[redacted]) gs
On branch: sprint1_group[redacted]  |   No changes (working directory clean)
 dyod_sprint git:(sprint1_group[redacted]) ./scripts/format.sh
 dyod_sprint git:(sprint1_group[redacted]) ✗ gs
On branch: sprint1_group[redacted]  |   [*] => $e*

Changes not staged for commit

      modified:   [1] src/test/storage/chunk_test.cpp

 dyod_sprint git:(sprint1_group[redacted]) ✗
```

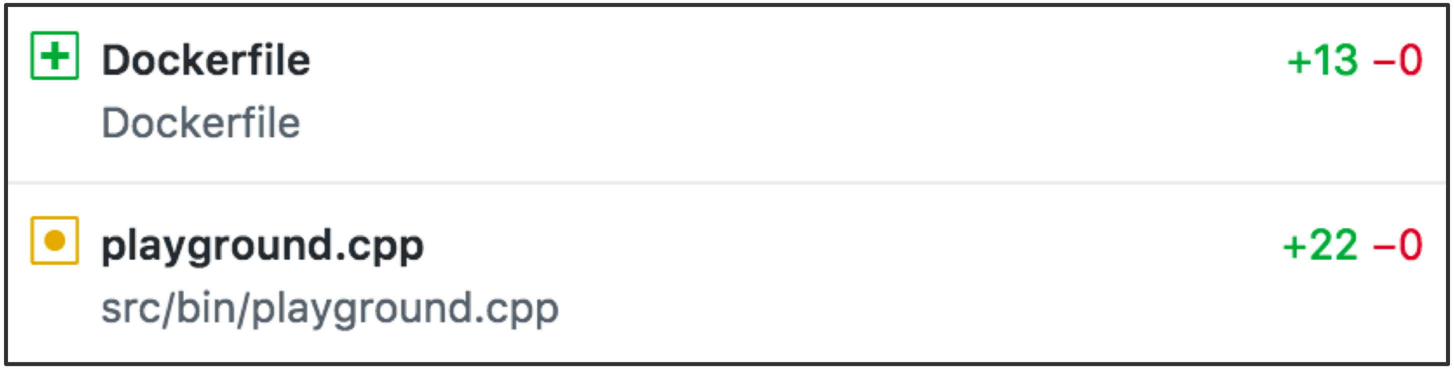


```
[➜  dyod_sprint git:(sprint1_group[redacted]) ✗ ./scripts/lint.sh
src/test/storage/chunk_test.cpp:29:  Add #include <memory> f
Category 'build/include_what_you_use' errors found: 1
Total errors found: 1
src/test/storage/storage_manager_test.cpp:19:  Add #include
] [4]
Category 'build/include_what_you_use' errors found: 1
Total errors found: 1
➜  dyod_sprint git:(sprint1_group[redacted]) ✗
```

Clean Commits



```
...     ...     @@ -1,4 +1,4 @@
  1             -#pragma once
          1     +#pragma once
```



Dockerfile +13 −0
Dockerfile

playground.cpp +22 −0
src/bin/playground.cpp

Conceptual Things

```cpp
std::vector<std::string> StorageManager::table_names() const {
  std::vector<std::string> names;
  auto get_name = [](const auto& entry) { return entry.first; };
  std::transform(m_tables.begin(), m_tables.end(), std::back_inserter(names), get_name);
  return names;
}
```

269 characters, lambdas, std::transform, std::back_inserter

```cpp
std::vector<std::string> StorageManager::table_names() const {
  std::vector<std::string> names;
  for (const auto& table_item : _tables) {
    names.emplace_back(table_item.first);
  }
  return names;
}
```

209 characters

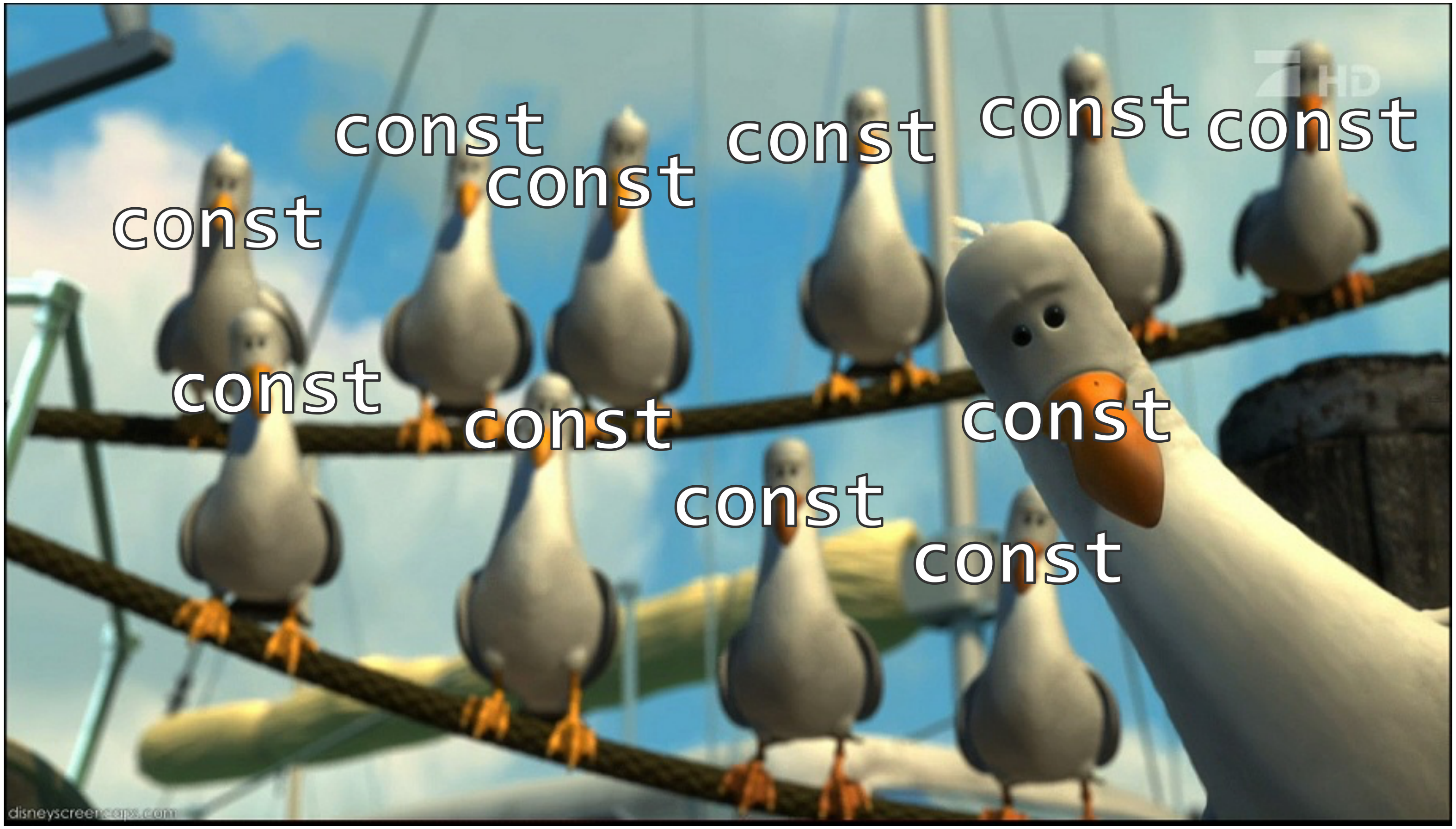
C++ things

Let’s play a different game – what did we *like* about this?

```
std::vector<std::string> StorageManager::table_names() const {
  std::vector<std::string> names;
  names.reserve(m_tables.size());
  // […]
```

```
for (const auto& chunk : _chunks) {
  count += chunk.size();
}
```

Const


const
const
const
const
const
const
const
const
const
const
const
7 HD
disneyscreencaps.com

Optionals

- *„Manages an optional contained value, i.e., a value that may or may not be present.“*
- Example use case: Null values or a table scan that supports between and, therefore, needs two search value parameters

```
#include <optional>
// Templated object of type std::optional<T>
std::optional<AllTypeVariant> opt;
std::optional<AllTypeVariant> opt2 = std::nullopt;
std::optional<AllTypeVariant> opt3 = 17;
if (opt) {
    do_something(*opt);
}
```

Optionals

Any ideas how to implement that?

~~std::pair<T,bool>~~

```
template <typename T>
class optional {
  bool _initialized;
  T _storage;
};
```

What is the result of `sizeof(std::optional<uint32_t>)`?

Lambdas

Constructs a closure: an unnamed function object capable of capturing variables in scope.

Lambda Expressions

A simplified table scan…

```
for (auto idx = 0; idx < value_segment.size(); ++idx) {
  switch (_scan_type) {
    case ScanType::OpEquals: {
      return value_segment.get(idx) == search_value;
      break;
    }
    case ScanType::OpNotEquals: {
      return value_segment.get(idx) != search_value;
      break;
    }
    case ScanType::OpLessThan: {
      return value_segment.get(idx) < search_value;
      break;
// [...]
```

Lambda Expressions

With lambda expressions

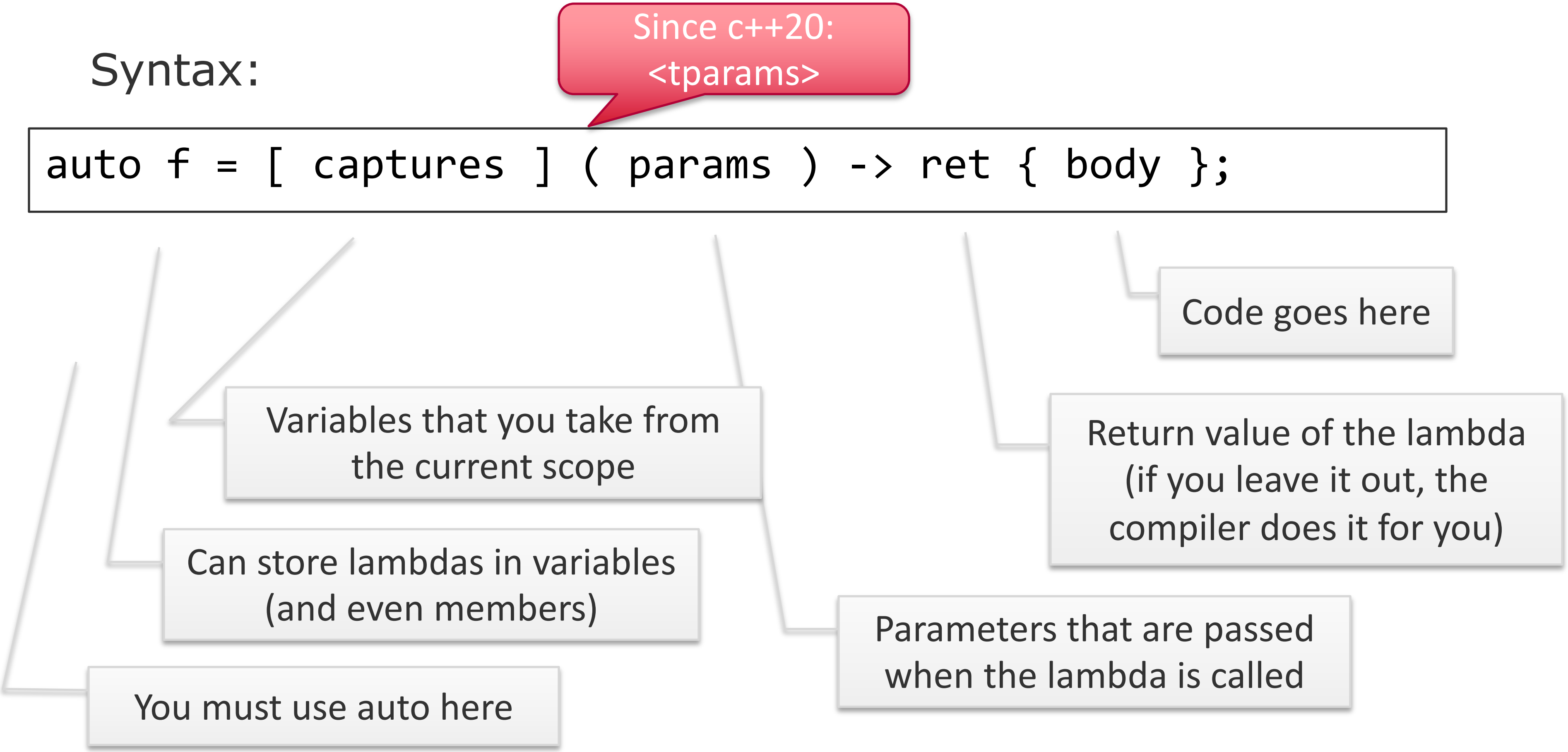
```
auto comparator = get_comparator(_scan_type);
for (auto idx = 0; idx < value_segment.size(); ++idx) {
  return comparator(value_segment.get(idx), search_value);
}
```

```
auto get_comparator(ScanType type) {
  switch (type) {
    case ScanType::OpEquals: {
      _return = [](auto left, auto right) { return left == right; };
      break;
    }
    case ScanType::OpNotEquals: {
      _return = [](auto left, auto right) { return left != right; };
      break;
    }
    // [...]
  }
}
```

+ separation of concerns
+ checks only once
+ reuse

Lambda Expressions

Syntax:



Since c++20:
<tparams>

```
auto f = [ captures ] ( params ) -> ret { body };
```

Code goes here

Variables that you take from the current scope

Return value of the lambda (if you leave it out, the compiler does it for you)

Can store lambdas in variables (and even members)

Parameters that are passed when the lambda is called

You must use auto here

Lambda Expressions

`auto f = [ captures ] ( params ) ->` ~~`ret`~~ `{ body };`

```
int main() {
  auto f = []() {
    std::cout << "Hallo Welt" << std::endl;
  };

  f();
}
```

Lambda Expressions

`auto f = [ captures ] ( params ) ->` ~~`ret`~~ `{ body };`

```
int main() {
  auto f = [](const std::string& name) {
    std::cout << "Hallo " << name << std::endl;
  };

  f("Alexander");
}
```

Lambda Expressions

```
auto f = [ captures ] ( params ) -> ret { body };
```

```
int main() {
  std::string my_name{"Larry"};

  auto f = [my_name](const std::string& name) {
    std::cout << "Hallo " << name << ", ich bin "
              << my_name << std::endl;
  };

  f("Alexander");
}
```

Lambda Expressions

```
auto f = [ captures ] ( params ) -> ~~ret~~ { body };
```

```
int main() {
  std::string my_name{"Larry"};


  auto f = [&my_name](const std::string& name) {
    std::cout << "Hallo " << name << ", ich bin "
              << my_name << std::endl;
  };

  f("Alexander");
}
```

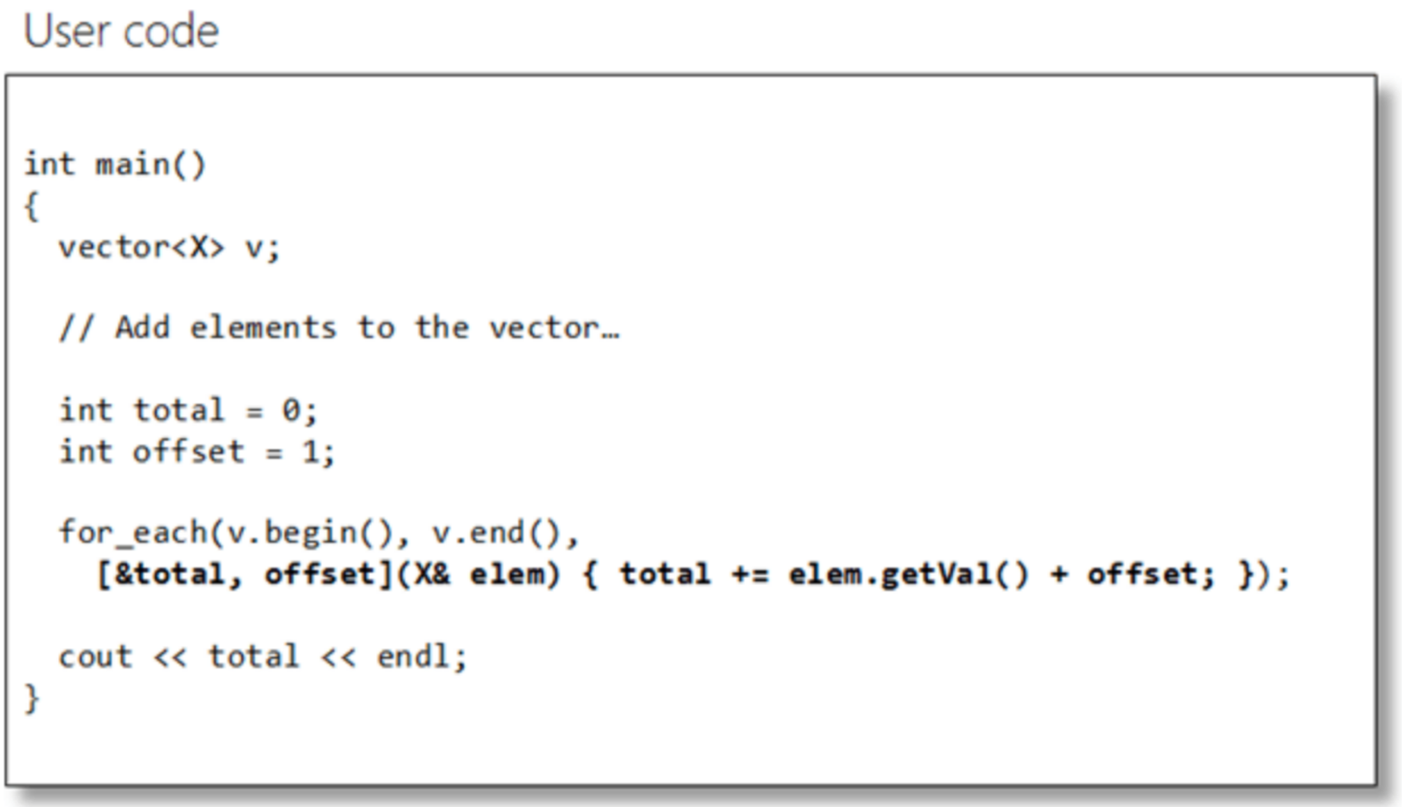
Lambda Expressions

```
auto get_lambda() {
  std::string my_name{"Larry"};
  return [my_name]() {
    std::cout << "Ich bin " << my_name << std::endl;
  };
}

int main() {
  f = get_lambda();

  // my_name is undefined here

  f();
}
```

Lambda Expressions

```
auto get_lambda() {
  std::string my_name{"Larry"};

  return [&my_name]() {
    std::cout << "Ich bin " << my_name << std::endl;
  };
}

int main() {
  f = get_lambda();

  // my_name is undefined here

  f();
}
```

Lambda Expressions

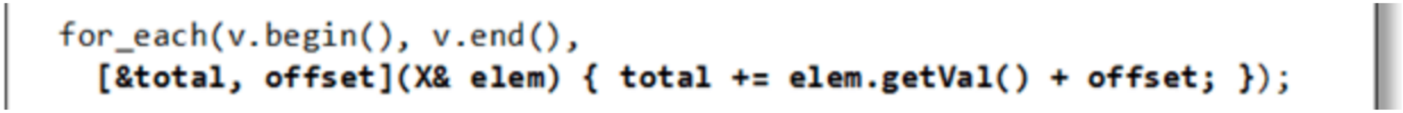


User code

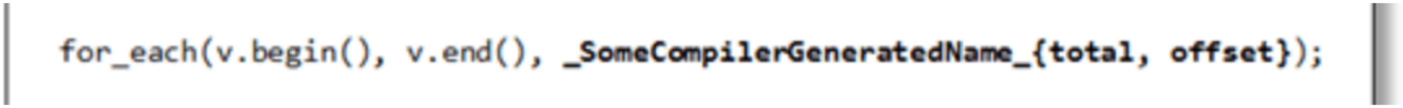
```
int main()
{
  vector<X> v;

  // Add elements to the vector…

  int total = 0;
  int offset = 1;

  for_each(v.begin(), v.end(),
    [&total, offset](X& elem) { total += elem.getVal() + offset; });

  cout << total << endl;
}
```

From https://blog.feabhas.com/2014/03/demystifying-c-lambdas/
A great resource if you want to learn more about lambdas

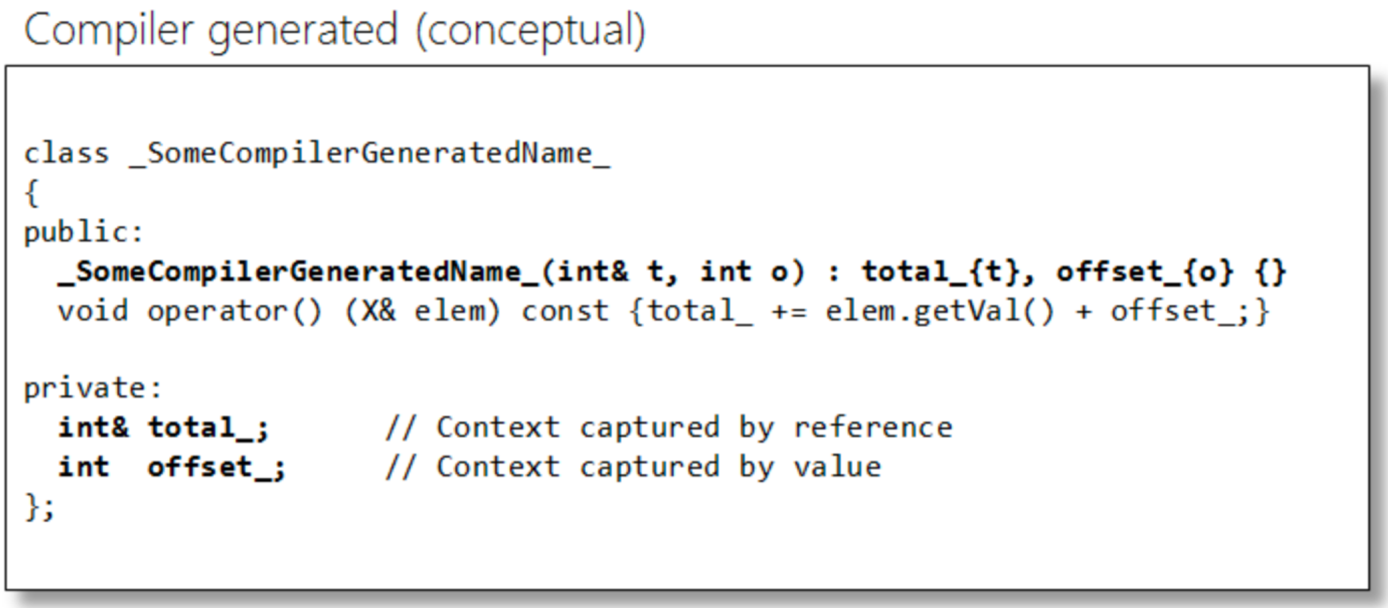
Lambda Expressions



```
for_each(v.begin(), v.end(),
  [&total, offset](X& elem) { total += elem.getVal() + offset; });
```



```
for_each(v.begin(), v.end(), _SomeCompilerGeneratedName_{total, offset});
```



Compiler generated (conceptual)

```
class _SomeCompilerGeneratedName_
{
public:
  _SomeCompilerGeneratedName_(int& t, int o) : total_{t}, offset_{o} {}
  void operator() (X& elem) const {total_ += elem.getVal() + offset_;}

private:
  int& total_;       // Context captured by reference
  int  offset_;      // Context captured by value
};
```

THE RELATIONAL MODEL

Based on “Database Systems - The Complete Book”
(H. Garcia-Molina, J. D. Ullman, J. Widom)

MOTIVATION FOR THE RELATIONAL MODEL

- Previously, databases tightly coupled logical and physical layers which impeded maintainability
- No conceptual idea of which operators are required
- Ted Codd proposed the **relational model** in the 1970s
    - Abstraction model using simple data structures and high-level operators
    - Implementation and physical storage is up to vendor

RELATIONAL DATABASES

- Database - organized collection of data
- Database Management System (DBMS) - the program that manages the database
- Relational database is based on relational **data model**
    1. Structure of the data
        - Conceptual model
        - (Physical model)
    2. Operations on the data
        - Modifications - change the database
        - Queries - retrieve information
    3. Constraints on the data

RELATIONAL MODEL - CONCEPTUAL DATA MODEL

- *Data* - two-dimensional table, called relation
  - Set or bag (multiset)
- *Attribute* - name of a column
- *Schema* - name of relations and set of attributes and constraints
- *Tuple* - row (except header) of a relation

- Further concepts:
  equality, relational instance, domain/data type, NULL

RELATIONAL MODEL – OPERATIONS

- Relational algebra is the basis for how the relational model is implemented in practice
    - Theoretical foundation for relational databases and SQL
- Operations
    - Take one or more relations as input(s) and output new relation
    - Can be chained to form more complex **queries**
- Classes of traditional operations:
    - Operations that **remove** parts of a relation: selection, and projection
    - Operations that **combine** tuples of two relations: cartesian product, and join
    - **Renaming**: relations and attributes
    - **Set operations**: union, intersection, and difference

RELATIONAL MODEL - PROJECTION

- Projection of R produces a new relation with a subset of R's columns
  - πA1, ..., An(R)
  - In the relational algebra of sets, duplicate tuples are eliminated

RELATIONAL MODEL – PROJECTION

Relation R

First Name	Last Name	Country	Year of Birth
Paul	Smith	Australia	1986
Lena	Jones	USA	1990
Hanna	Schulze	Germany	1942
Hanna	Schulze	USA	2000

$\pi_{\text{First Name, Last Name}}(R)$

```
SELECT DISTINCT
  FirstName, LastName FROM R
```

First Name	Last Name
Paul	Smith
Lena	Jones
Hanna	Schulze

RELATIONAL MODEL – SELECTION

- Selections of R produces a new relation with a subset of R's tuples (those that satisfy a condition)
  - σAθB (R) **or** σAθValueConstant (R)
  - θ = {<, ≤, =, >, ≥}

RELATIONAL MODEL – SELECTION

Relation R

First Name	Last Name	Country	Year of Birth
Paul	Smith	Australia	1986
Lena	Jones	USA	1990
Hanna	Schulze	Germany	1942
Hanna	Schulze	USA	2000

σCountry=‘USA’(R)

```
SELECT * FROM R WHERE
  Country = ‘USA’
```

First Name	Last Name	Country	Year of Birth
Lena	Jones	USA	1990
Hanna	Schulze	USA	2000

RELATIONAL MODEL - OPERATIONS THAT COMBINE TUPLES OF TWO RELATIONS

- Cartesian product ((cross-)product) of R and S is the set of pairs formed by choosing the first element to be any element of R and the second any element of S
    - The schema of the new relation is the union of schemas for R and S (Exception: R and S have attribute A in common -> use new name R.A and S.A)

RELATIONAL MODEL – CROSS PRODUCT

Relation R

First Name	Last Name	Country	Year of Birth
Paul	Smith	Australia	1986
Lena	Jones	USA	1990
Hanna	Schulze	Germany	1942
Hanna	Schulze	USA	2000

Relation S

Country	Capital
Germany	Berlin
USA	Washington

R x S

```
SELECT * FROM R, S
```

First Name	Last Name	R.Country	Year of Birth	S.Country	Capital
Paul	Smith	Australia	1986	Germany	Berlin
Paul	Smith	Australia	1986	USA	Washington
Lena	Jones	USA	1990	Germany	Berlin
Lena	Jones	USA	1990	USA	Washington
Hanna	Schulze	Germany	1942	Germany	Berlin
Hanna	Schulze	Germany	1942	USA	Washington
Hanna	Schulze	USA	2000	Germany	Berlin
Hanna	Schulze	USA	2000	USA	Washington

RELATIONAL MODEL - OPERATIONS THAT COMBINE TUPLES OF TWO RELATIONS

- Cartesian product ((cross-)product) of R and S is the set of pairs formed by choosing the first element to be any element of R and the second any element of S
    - The schema of the new relation is the union of schemas for R and S (Exception: R and S have attribute A in common -> use new name R.A and S.A)
- Join of R and S pairs tuples that match in some way
    - **Dangling tuple**: tuple with no match
    - Natural join: match in common attributes of R and S

RELATIONAL MODEL – NATURAL JOIN

Relation R

First Name	Last Name	Country	Year of Birth
Paul	Smith	Australia	1986
Lena	Jones	USA	1990
Hanna	Schulze	Germany	1942
Hanna	Schulze	USA	2000

Relation S

Country	Capital
Germany	Berlin
USA	Washington

$R \bowtie S$

```
SELECT * FROM R NATURAL JOIN S
```

First Name	Last Name	Country	Year of Birth	Capital
Lena	Jones	USA	1990	Washington
Hanna	Schulze	Germany	1942	Berlin
Hanna	Schulze	USA	2000	Washington

RELATIONAL MODEL - OPERATIONS THAT COMBINE TUPLES OF TWO RELATIONS

- Cartesian product ((cross-)product) of R and S is the set of pairs formed by choosing the first element to be any element of R and the second any element of S
  - The schema of the new relation is the union of schemas for R and S (Exception: R and S have attribute A in common -> use new name R.A and S.A)
- Join of R and S pairs tuples that match in some way
  - **Dangling tuple**: tuple with no match
  - Natural join: match in common attributes of R and S
  - Theta/Equi join: match based on arbitrary condition C
    - Product of R and S, filtered by condition C
  -

RELATIONAL MODEL - EQUI JOIN

Relation R

First Name	Last Name	Country	Year of Birth
Paul	Smith	Australia	1986
Lena	Jones	USA	1990
Hanna	Schulze	Germany	1942
Hanna	Schulze	USA	2000

Relation S

Country	Capital
Germany	Berlin
USA	Washington

$$R \bowtie_{\text{Country} = \text{Country}} S$$

```sql
SELECT * FROM R [INNER] JOIN S
  ON R.Country = S.Country
```

First Name	Last Name	Country	Year of Birth	Capital
Lena	Jones	USA	1990	Washington
Hanna	Schulze	Germany	1942	Berlin
Hanna	Schulze	USA	2000	Washington

RELATIONAL MODEL – OPERATIONS THAT COMBINE TUPLES OF TWO RELATIONS

- Cartesian product ((cross-)product) of R and S is the set of pairs formed by choosing the first element to be any element of R and the second any element of S
    - The schema of the new relation is the union of schemas for R and S (Exception: R and S have attribute A in common -> use new name R.A and S.A)
- Join of R and S pairs tuples that match in some way
    - **Dangling tuple**: tuple with no match
    - Natural join: match in common attributes of R and S
    - Theta/Equi join: match based on arbitrary condition C
        - Product of R and S, filtered by condition
    - Semi join of R and S is the set of tuples in R that match the join condition

RELATIONAL MODEL – SEMI JOIN

Relation R

First Name	Last Name	Country	Year of Birth
Paul	Smith	Australia	1986
Lena	Jones	USA	1990
Hanna	Schulze	Germany	1942
Hanna	Schulze	USA	2000

Relation S

Country	Capital
Germany	Berlin
USA	Washington

$R \ltimes S$

```
SELECT FirstName, LastName,
  Country, YearOfBirth
  FROM R NATURAL JOIN S
```

First Name	Last Name	Country	Year of Birth
Lena	Jones	USA	1990
Hanna	Schulze	Germany	1942
Hanna	Schulze	USA	2000

RELATIONAL MODEL - SET OPERATIONS

- Union of R and S is the set of elements that are in R or S or both
- Intersection of R and S is the set of elements that are in both R and S
- Difference of R and S is the set of that are in R but not in S
  - R - S is different from S - R

- Conditions for R and S:
  - R and S must have schemas with identical attributes and domains

RELATIONAL MODEL – UNION

Relation R

Country	Capital
Norway	Oslo
USA	Washington
Poland	Warsaw

Relation S

Country	Capital
Germany	Berlin
USA	Washington

R ∪ S

```
SELECT * FROM R
UNION
SELECT * FROM S
```

Country	Capital
Norway	Oslo
USA	Washington
Poland	Warsaw
Germany	Berlin

RELATIONAL MODEL – UNION

Relation R

Country	Capital
Norway	Oslo
USA	Washington
Poland	Warsaw

Relation S

Country	Capital
Germany	Berlin
USA	Washington

R ∪ S

```
SELECT * FROM R
UNION ALL
SELECT * FROM S
```

Country	Capital
Norway	Oslo
USA	Washington
Poland	Warsaw
Germany	Berlin
USA	Washington

RELATIONAL MODEL - INTERSECT

Relation R

Country	Capital
Norway	Oslo
USA	Washington
Poland	Warsaw

R ∩ S

```
SELECT * FROM R
INTERSECT
SELECT * FROM S
```

Country	Capital
USA	Washington

Relation S

Country	Capital
Germany	Berlin
USA	Washington

RELATIONAL MODEL – DIFFERENCE

Relation R

Country	Capital
Norway	Oslo
USA	Washington
Poland	Warsaw

Relation S

Country	Capital
Germany	Berlin
USA	Washington

R \ S

```
SELECT * FROM R
EXCEPT
SELECT * FROM S
```

Country	Capital
Norway	Oslo
Poland	Warsaw

S \ R

```
SELECT * FROM S
EXCEPT
SELECT * FROM R
```

Country	Capital
Germany	Berlin

RELATIONAL MODEL - MINIMAL RELATIONAL ALGEBRA?

- Union, intersection, difference, projection, selection, cartesian product, natural join, theta join, semi join, renaming

RELATIONAL MODEL - MINIMAL RELATIONAL ALGEBRA

- Union, ~~intersection~~, difference, projection, selection, cartesian product, ~~natural join, theta join, semi join~~, renaming

RELATIONAL MODEL – WHAT IS MISSING?

- Bag semantic (+ duplicate elimination)
- Aggregation (and grouping)
- Sort
- Extended projection
- Outer join

RELATIONAL MODEL – BAG SEMANTIC

- Bags are multi sets (allow duplicates)
  - Redefinition of set operations necessary
- Some relational operations are more efficient with the bag model (without duplicate elimination)
  - Union
  - Projection
- Duplicate-elimination operator turns bag into set by eliminating all but one copy of each tuple

RELATIONAL MODEL - AGGREGATION

- Aggregations summarize or “aggregate” the values in one column
  - Examples: SUM, AVG, MIN, MAX, COUNT
  - Groupings allow aggregations of tuple groups that correspond to the value of one or multiple columns
  - $\gamma_{A1, \ldots, Am, AVG(Au), COUNT(Av), MIN(Aw), MAX(Ax), SUM(Ay)}\ \mathbf{R}$

RELATIONAL MODEL – AGGREGATION

Relation R

First Name	Last Name	Country	Year of Birth
Paul	Smith	Australia	1986
Lena	Jones	USA	1990
Hanna	Schulze	Germany	1942
Hanna	Schulze	USA	2000

γMin(Year of Birth) (R)

```
SELECT MIN(YearOfBirth)
FROM R;
```

MIN(Year of Birth)
1942

RELATIONAL MODEL – AGGREGATION

Relation R

First Name	Last Name	Country	Year of Birth
Paul	Smith	Australia	1986
Lena	Jones	USA	1990
Hanna	Schulze	Germany	1942
Hanna	Schulze	USA	2000

γCountry Max(Year of Birth) (R)

```
SELECT Country, MIN(YearOfBirth)
FROM R GROUP BY Country;
```

Country	MIN(Year of Birth)
Australia	1986
USA	1990
Germany	1942

RELATIONAL MODEL - SORT

- Turns unordered container, e.g., set, bag, into an ordered one, e.g., list
  - Only useful as last operator of a **relational query**, because following operators turn list into set or bag
  - Of importance for finding efficient **physical query plans** (an operator implementation may require sorted inputs)

RELATIONAL MODEL - EXTENDED PROJECTION

- Besides renamings, extended projections allow arbitrary expressions
  - Constants
  - Arithmetic operators
  - String operators

RELATIONAL MODEL - OUTER JOIN

- Outer join is the union of the natural join and all dangling tuples from R and S; dangling tuples of R and S must be padded with NULLs for missing attributes
  - Full (⟗), left (⟕), and right (⟖) outer join
  - Theta join versions of outer join operate analogous
  - Inner join is a synonym of “normal” join

RELATIONAL MODEL – FULL OUTER JOIN

Relation R

First Name	Last Name	Country	Year of Birth
Paul	Smith	Australia	1986
Lena	Jones	USA	1990
Hanna	Schulze	Germany	1942
Hanna	Schulze	USA	2000

Relation S

Country	Capital
Germany	Berlin
USA	Washington
Norway	Oslo

R ⟗ S

```
SELECT R.Country, S.Capital
  FROM R
  FULL OUTER JOIN S ON
  R.Country=S.Country;
```

R.Country	S.Capital
Australia	
USA	Washington
Germany	Berlin
USA	Washington
	Oslo

RELATIONAL MODEL – LEFT OUTER JOIN

Relation R

First Name	Last Name	Country	Year of Birth
Paul	Smith	Australia	1986
Lena	Jones	USA	1990
Hanna	Schulze	Germany	1942
Hanna	Schulze	USA	2000

Relation S

Country	Capital
Germany	Berlin
USA	Washington
Norway	Oslo

R ⟕ S

```
SELECT R.Country, S.Capital
  FROM R
  LEFT [OUTER] JOIN S ON
  R.Country=S.Country;
```

R.Country	S.Capital
Australia	
USA	Washington
Germany	Berlin
USA	Washington

RELATIONAL MODEL – RIGHT OUTER JOIN

Relation R

First Name	Last Name	Country	Year of Birth
Paul	Smith	Australia	1986
Lena	Jones	USA	1990
Hanna	Schulze	Germany	1942
Hanna	Schulze	USA	2000

Relation S

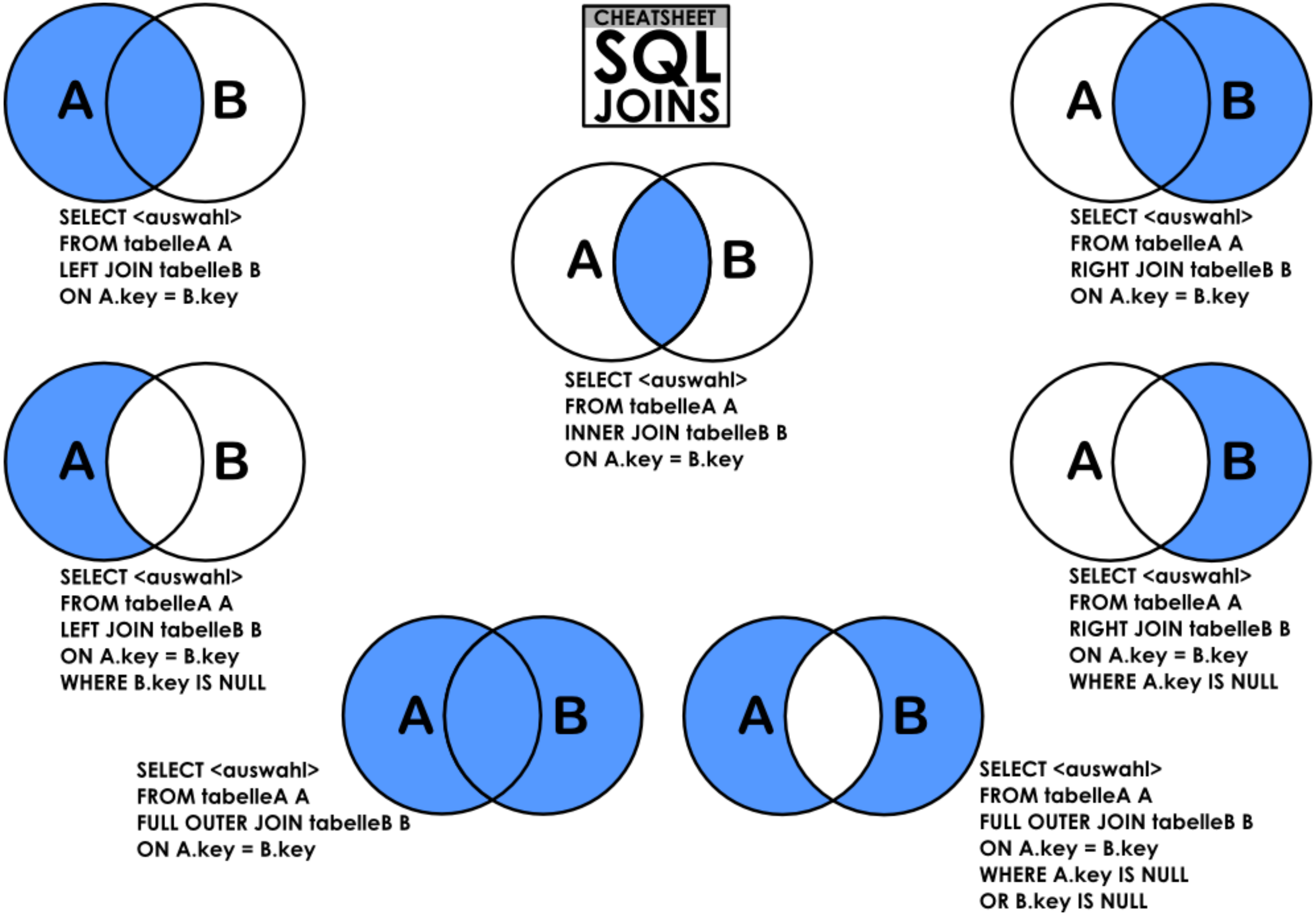
Country	Capital
Germany	Berlin
USA	Washington
Norway	Oslo

R ⟖ S

```
SELECT R.Country, S.Capital
  FROM R
  RIGHT [OUTER] JOIN S ON
  R.Country=S.Country;
```

R.Country	S.Capital
USA	Washington
Germany	Berlin
USA	Washington
	Oslo

JOINS



```
SELECT <auswahl>
FROM tabelleA A
LEFT JOIN tabelleB B
ON A.key = B.key
```

```
SELECT <auswahl>
FROM tabelleA A
INNER JOIN tabelleB B
ON A.key = B.key
```

```
SELECT <auswahl>
FROM tabelleA A
RIGHT JOIN tabelleB B
ON A.key = B.key
```

```
SELECT <auswahl>
FROM tabelleA A
LEFT JOIN tabelleB B
ON A.key = B.key
WHERE B.key IS NULL
```

```
SELECT <auswahl>
FROM tabelleA A
RIGHT JOIN tabelleB B
ON A.key = B.key
WHERE A.key IS NULL
```

```
SELECT <auswahl>
FROM tabelleA A
FULL OUTER JOIN tabelleB B
ON A.key = B.key
```

```
SELECT <auswahl>
FROM tabelleA A
FULL OUTER JOIN tabelleB B
ON A.key = B.key
WHERE A.key IS NULL
OR B.key IS NULL
```

Source: https://stackoverflow.com/questions/24257890/why-on-clause-used-on-full-outer-join

JOINS

JAVA, SQL AND JOOQ.

Best Practices and Lessons Learned from Writing Awesome Java and SQL Code. Get some hands-on insight on what's behind developing jOOQ.

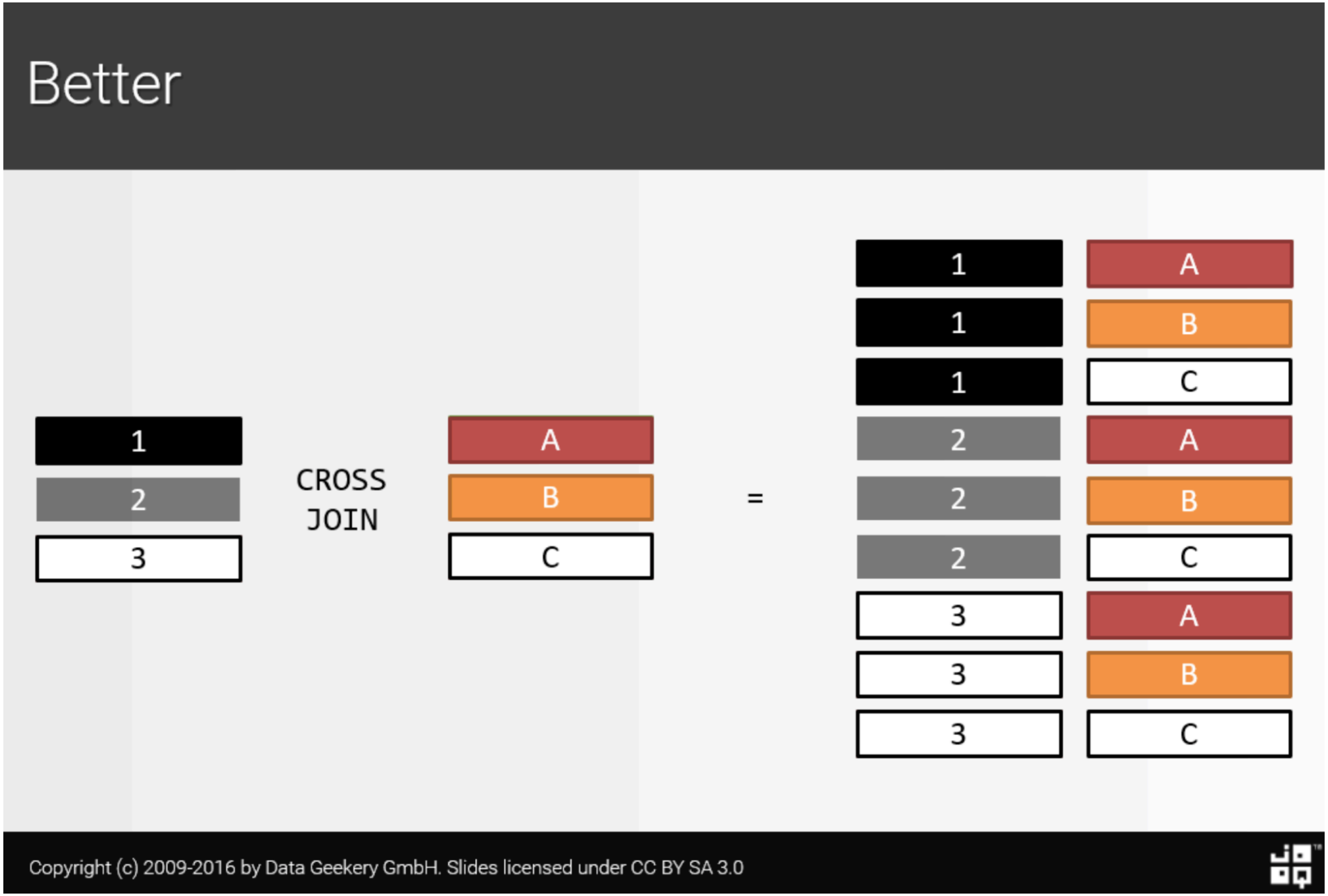
Say NO to Venn Diagrams When Explaining JOINs

Posted on July 5, 2016 by lukaseder

Like this blog? Check out our product:

Source: https://blog.jooq.org/2016/07/05/say-no-to-venn-diagrams-when-explaining-joins/

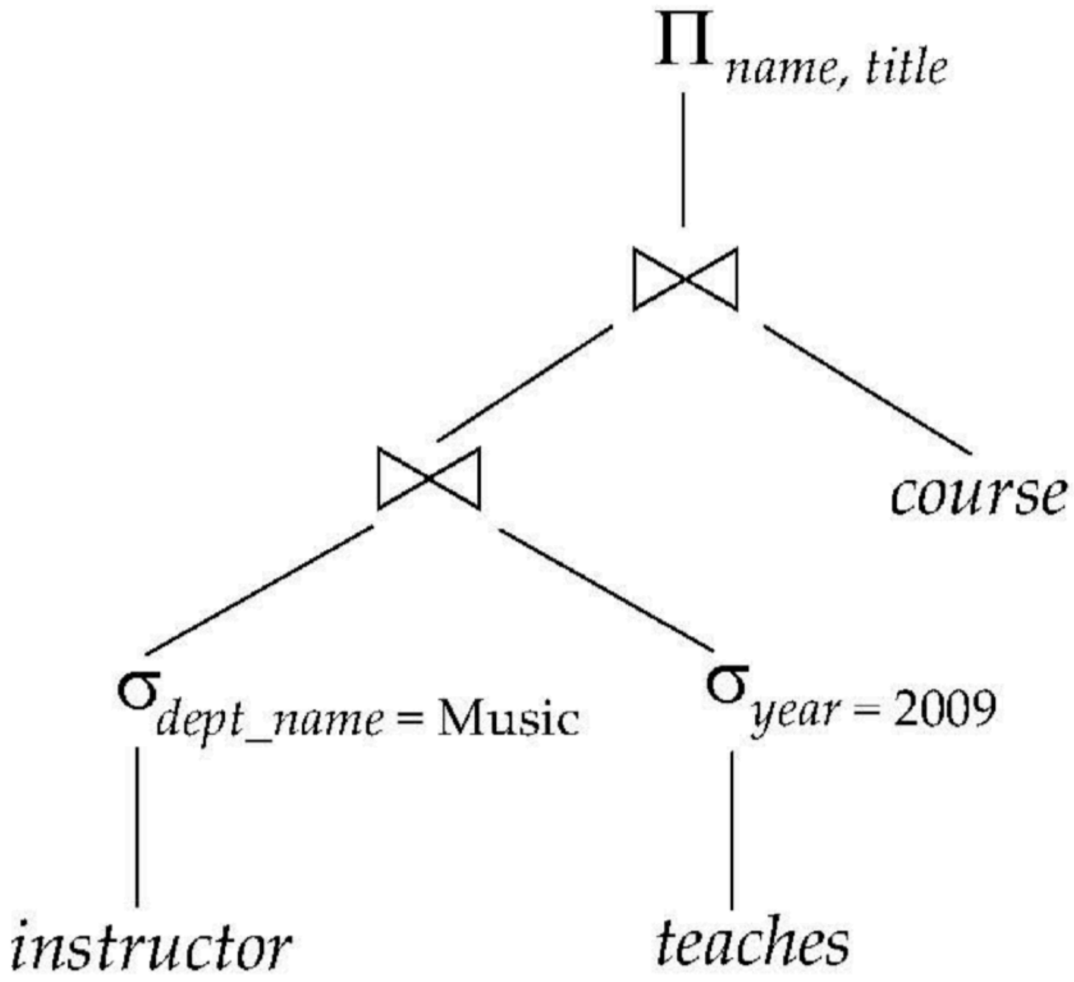
JOINS



Better

Copyright (c) 2009-2016 by Data Geekery GmbH. Slides licensed under CC BY SA 3.0

RELATIONAL MODEL – OPERATOR PLAN



Source: http://www.cbcb.umd.edu/confcour/Spring2014/CMSC424/query_optimization.pdf

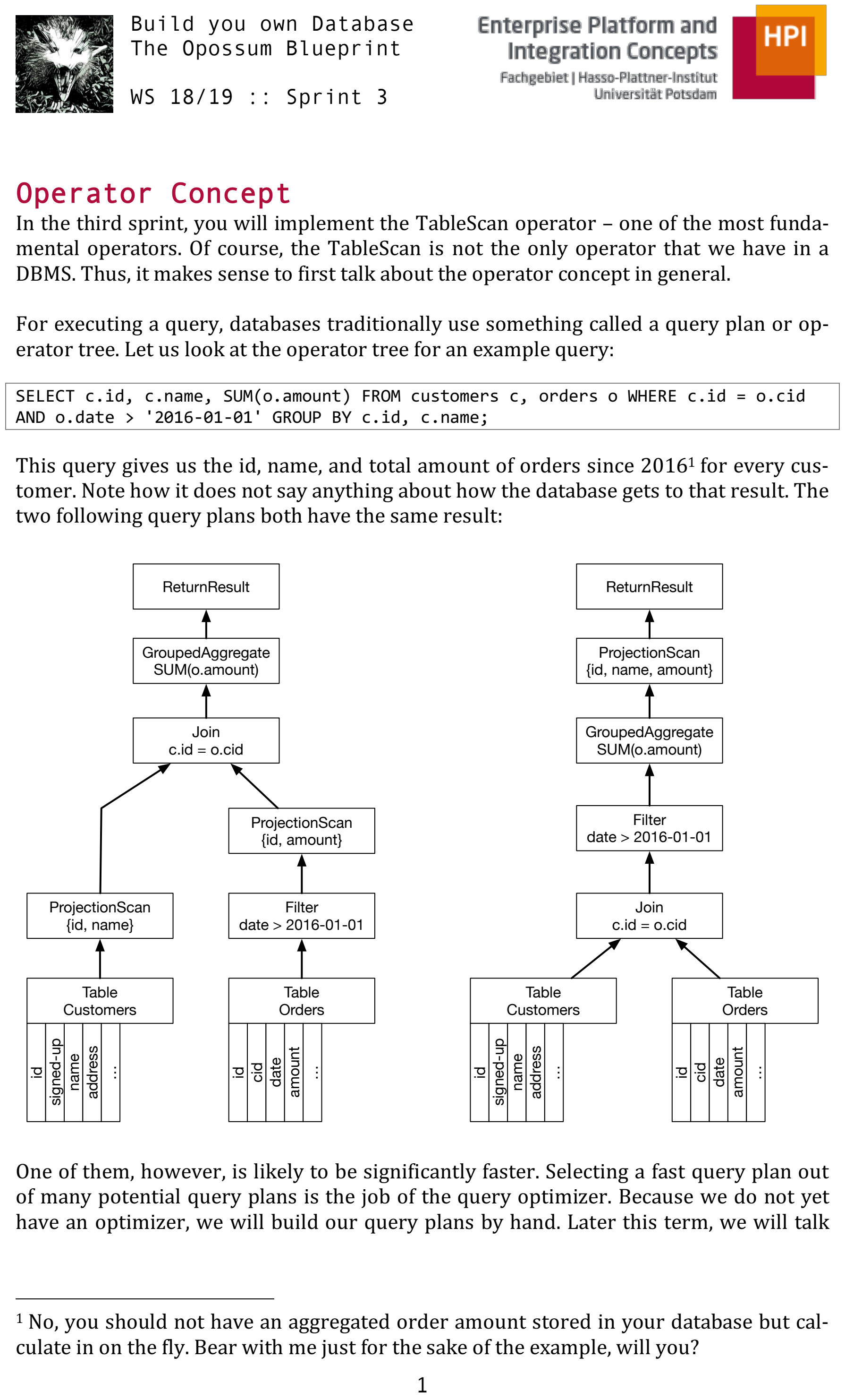
SQL – THE DATABASE LANGUAGE

- Structured Query Language
  - Express queries of relational algebra (declaratively)
  - Statements for modifying the database
  - Declaring the database schema
  - Further concepts: constraints, views, indexes, …

OPOSSUM'S OPERATOR CONCEPT

- Opossum implements operators that loosely resemble the relational algebra
    - Queries can be formulated as DAG of multiple operators
    - Usually, the first operator is the GetTable operator
    - Operators take none to two other operators as input
    - The result of an operator is passed as table to the next operator
- Efficiency is crucial in database systems
    - Operators itself need to be implemented in efficient ways
    - Order of query operators offers large optimization potential

SPRINT 3



Build you own Database
The Opossum Blueprint

WS 18/19 :: Sprint 3

Enterprise Platform and Integration Concepts
Fachgebiet | Hasso-Plattner-Institut
Universität Potsdam

Operator Concept

In the third sprint, you will implement the TableScan operator – one of the most fundamental operators. Of course, the TableScan is not the only operator that we have in a DBMS. Thus, it makes sense to first talk about the operator concept in general.

For executing a query, databases traditionally use something called a query plan or operator tree. Let us look at the operator tree for an example query:

```
SELECT c.id, c.name, SUM(o.amount) FROM customers c, orders o WHERE c.id = o.cid
AND o.date > '2016-01-01' GROUP BY c.id, c.name;
```

This query gives us the id, name, and total amount of orders since 2016[1] for every customer. Note how it does not say anything about how the database gets to that result. The two following query plans both have the same result:

One of them, however, is likely to be significantly faster. Selecting a fast query plan out of many potential query plans is the job of the query optimizer. Because we do not yet have an optimizer, we will build our query plans by hand. Later this term, we will talk

---

[1] No, you should not have an aggregated order amount stored in your database but calculate in on the fly. Bear with me just for the sake of the example, will you?

1

ORGANIZATION

- Sprint 3 Deadline: 26.11.2019 - 23:59:59 CET
- Next Week
    - Sprint 2 Feedback
    - Group Topic Presentation
    - NULL Values, Virtual Method Calls, Chunks…

THAT’S IT.