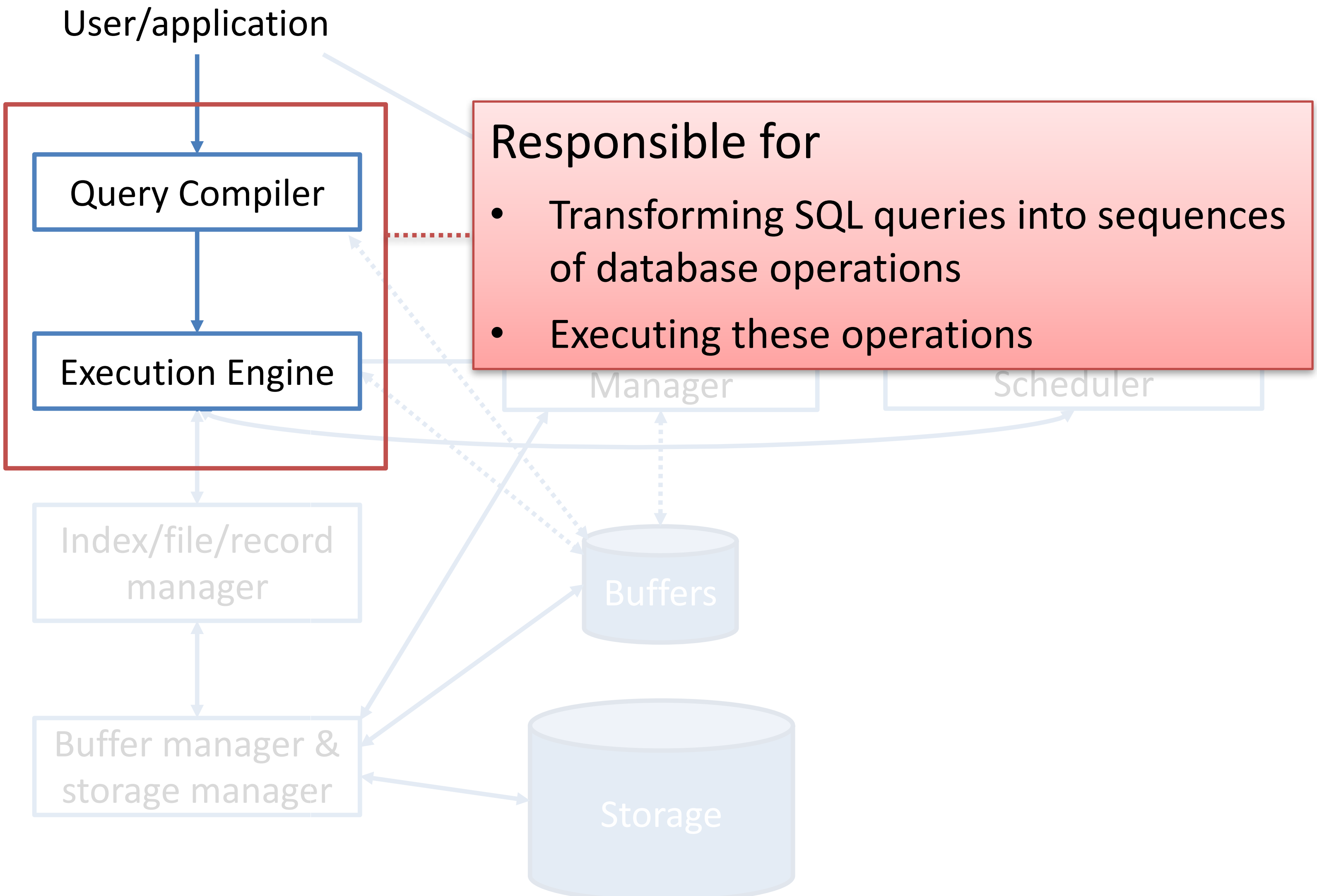
Now for something completely different: Chapter 3

Where we are…



Responsible for

- Transforming SQL queries into sequences of database operations
- Executing these operations

Relational Model Terminology

- Relations:

Attributes

Relation name → **Students**

name	id	programme
Anna	20171989	G402
John	20174378	G702
…	…	…

Tuple (or row)

- Schema: description of all tables in the database

Students(name, id, programme)

- Order of attributes matters!

SQL Queries

Here typically SELECT statements

- **SQL query:** SQL SELECT/INSERT/UPDATE/DELETE statement

```
SELECT id AS student_id
FROM Students
WHERE programme = 'G402';
```

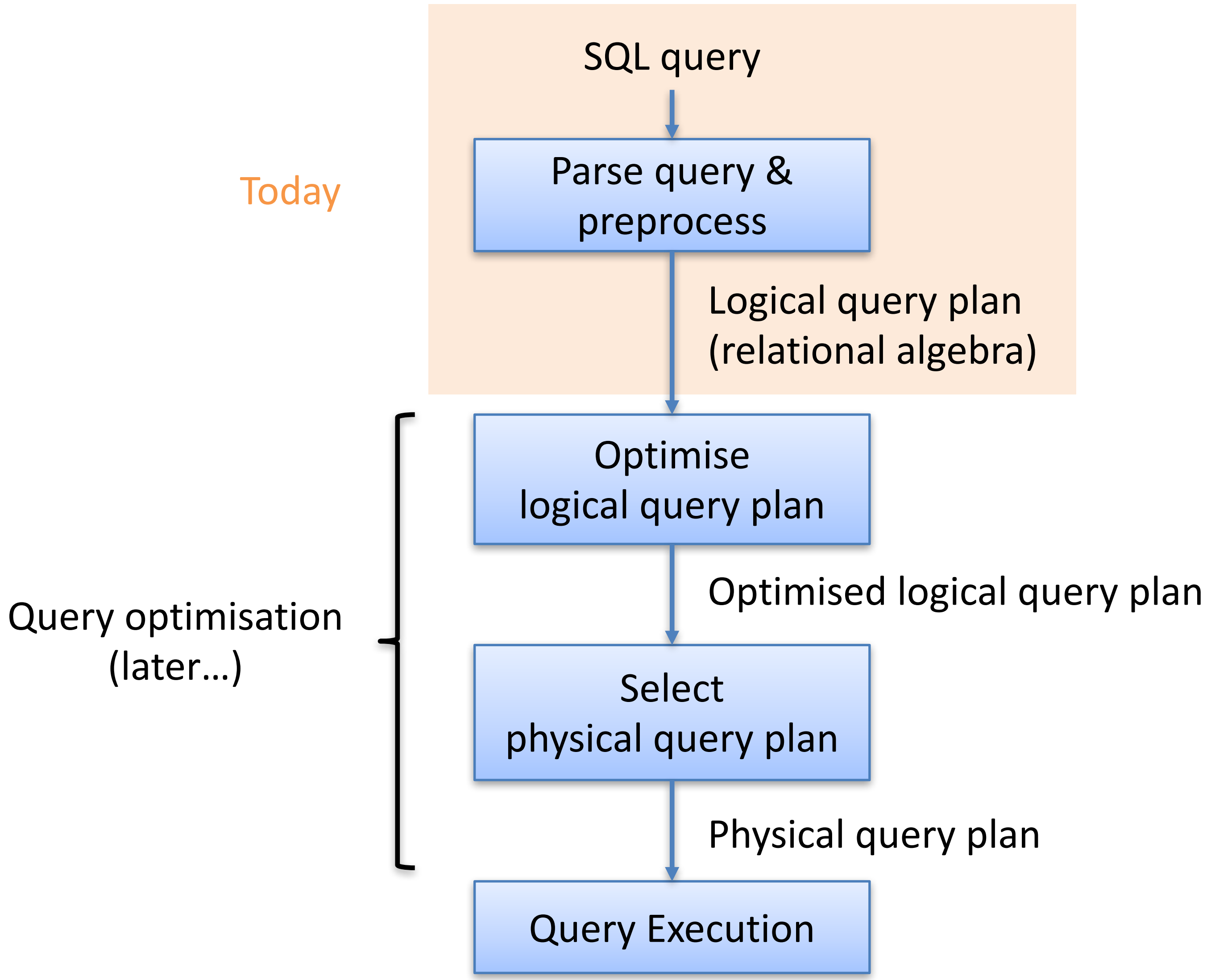
Students(name, id, programme)

Marks(student_id, module, mark)

```
SELECT name, avg(mark)
FROM Students, Marks
WHERE id = student_id AND module = 'COMP207'
GROUP BY name;
```

- Declarative: tells the DBMS **what we want, *not* how to get it**
- **DBMS selects a good sequence of database operations** to execute the query

How Does Query Processing Work?

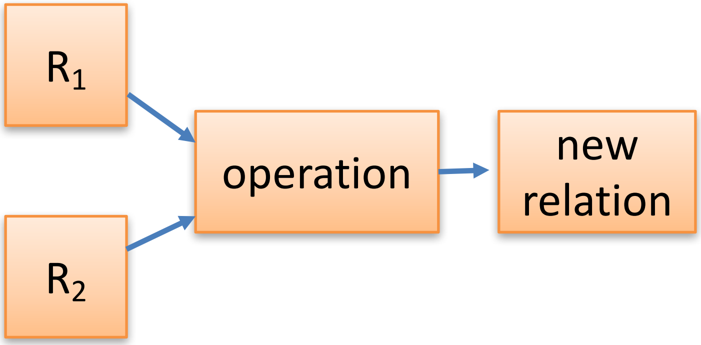


From SQL to Relational Algebra

Relational Algebra

- Set of **operations** that can be applied to **relations** to **compute new relations**

- Basic relational algebra:
  - Selection (σ)
  - Projection (π)
  - Cartesian product ($\times$)
  - Union ($\cup$)
  - Difference ($-$)
  - Renaming (ρ)



- Many others can be defined from these, e.g.:
  - Natural join ($\bowtie$)
  - Semijoin ($\ltimes$)
  - see COMP107/CSE103
  - was also discussed in lecture 2

Selection (σ)

- $\sigma_{\text{condition}}(R)$ = set of all tuples in R that satisfy the condition

The SELECT keyword in SQL has nothing to do with the selection operator!

SQL query

```
SELECT *
FROM Modules
WHERE year = 2 AND sem = 1;
```

translates into

$\sigma_{\text{year=2 AND sem=1}}(\text{Modules})$ =

module	year	sem
COMP201	2	1
COMP207	2	1

Relational algebra expression

Modules

module	year	sem
COMP105	1	1
COMP201	2	1
COMP202	2	2
COMP207	2	1

Projection (π)

- $\pi_{\text{attribute list}}(R)$ = restricts **R** to the attributes in **attribute list**

SQL query

```
SELECT sem, module
FROM Modules;
```

translates into

Attribute order matters

$\pi_{\text{sem, module}}(\text{Modules})$ =

Relational algebra expression

Modules

module	year	sem
COMP105	1	1
COMP201	2	1
COMP202	2	2
COMP207	2	1

sem	module
1	COMP105
1	COMP201
2	COMP202
1	COMP207

Cartesian Product (×)

- $R_1 \times R_2$ = pairs each tuple in R_1 with each tuple in R_2

SQL query

```
SELECT *
FROM Modules, Lecturers;
```

translates into

Modules × Lecturers

Relational algebra expression

Modules

code	year
COMP105	1
COMP201	2

Lecturers

name	module
J. Fearnley	COMP105
S. Coope	COMP201

=

code	year	name	module
COMP105	1	J. Fearnley	COMP105
COMP105	1	S. Coope	COMP201
COMP201	2	J. Fearnley	COMP105
COMP201	2	S. Coope	COMP201

Renaming (ρ)

- $\rho_{A1 \rightarrow B1, A2 \rightarrow B2, \ldots}(R)$ = renames attribute **A1** to **B1**, attribute **A2** to **B2**, …

SQL query

```
SELECT module AS module_code
FROM Modules;
```

translates into

$$\rho_{module \rightarrow module\_code}(Modules)$$

Relational algebra expression

Modules

module	year	sem
COMP105	1	1
COMP201	2	1

=

module_code	year	sem
COMP105	1	1
COMP201	2	1

Combining Operators

- Operators can be combined:

$$\pi_{\text{module,name}}(\sigma_{\text{code=module AND year=2}}(\textbf{Modules} \times \textbf{Lecturers}))$$

Modules

code	year
COMP105	1
COMP201	2
…	…

Lecturers

name	module
J. Fearnley	COMP105
S. Coope	COMP201
…	…

Resulting relation

~~code~~	~~year~~	name	module
~~COMP105~~	~~1~~	~~J. Fearnley~~	~~COMP105~~
~~COMP105~~	~~1~~	~~S. Coope~~	~~COMP201~~
~~COMP201~~	~~2~~	~~J. Fearnley~~	~~COMP105~~
~~COMP201~~	~~2~~	S. Coope	COMP201
~~…~~	~~…~~	…	…

```
SELECT module, name
FROM Modules, Lecturers
WHERE code=module AND year=2;
```

From SQL to Relational Algebra

- For simple SELECT-FROM-WHERE queries:

SELECT $A_1, \ldots, A_m$
FROM $R_1, R_2, \ldots, R_n$
WHERE **condition**;

$\downarrow$

$$\pi_{A1,\ldots,Am}(\sigma_{condition}(R_1 \times R_2 \times \ldots \times R_n))$$

- Similar for more complex SQL queries:
  - With renaming, aggregates, union, etc.
  - Nested queries

Joins

- Joins form one of the most important operators of relational algebra
- Are also one of the most expensive to compute
- Many types of joins:
  – Cartesian product ($\times$)
  – Natural join ($\bowtie$)
  – Equijoin ($\bowtie_{A=B}$) and Theta-joins ($\bowtie_{\theta}$)
  – Semi-join ($\ltimes$)
  – Outer joins …
- Definable in terms of σ, π, and $\times$

Natural Join ($\bowtie$)

```
SELECT *
FROM Modules
NATURAL JOIN
Lecturers;
```

- $R_1 \bowtie R_2$ = pairs matching tuples in R_1 and R_2

Two tuples match
if they have the same values
for all common attributes

Modules

module	year
COMP105	1
COMP201	2

Lecturers

name	module
J. Fearnley	COMP105
S. Coope	COMP201

Modules $\bowtie$ **Lecturers** =

module	year	name
COMP105	1	J. Fearnley
COMP201	2	S. Coope

- Can be expressed by the other operators:

Modules $\bowtie$ **Lecturers** =

$$\pi_{\text{module,year,name}}(\sigma_{\text{module=module}'}(\text{Modules} \times \rho_{\text{module}\to\text{module}'}(\text{Lecturers})))$$

Semijoin ($\ltimes$)

- $R_1 \ltimes R_2$ = tuples from R_1 matching tuples in R_2

Two tuples match
if they have the same values
for all common attributes

Modules

Lecturers

```
SELECT *
FROM Modules
WHERE EXISTS    (SELECT 1
                 FROM Lecturers
                 WHERE
                 Modules.module =
                 Lecturers.module)
```

module	year
COMP105	1

- Can be expressed by the other operators:

Modules $\ltimes$ Lecturers =

$$\pi_{\text{module,year}}(\sigma_{\text{module=module'}}(\text{Modules} \times \rho_{\text{module}\rightarrow\text{module'}}(\text{Lecturers})))$$

Careful!

SQL Doesn't Eliminate Duplicates

Employee

name	salary
Anna	£45,000
Ben	£40,000
Chloe	£40,000

SELECT salary **FROM** Employee;

salary
£45,000
£40,000
£40,000

Duplicates removed on request only

SELECT DISTINCT salary **FROM** Employee;

salary
£45,000
£40,000

Basic Relational Algebra Eliminates Duplicates

Relations are sets!

Employee

name	salary
Anna	£45,000
Ben	£40,000
Chloe	£40,000

↓

π_{salary}(Employee) = **SELECT DISTINCT** salary **FROM** Employee;

↓

salary
£45,000
£40,000

From Sets to Multisets

- DBMS work with a variant … algebra that views relations as multisets instead of sets
  - Means: the same tuple may occur multiple times in a relation
  - Straightforward extension of most relational algebra operators

Even lists with ORDER BY

- Working with this variant of relational algebra can be subtle
- In this module: **we continue to use the relational algebra that operates on sets**
  (… but be aware of this issue)

Query Plans

(a.k.a. Query Trees)

Query Plans

- A **relational algebra expression** that is obtained from an SQL query is also called a **(logical) query plan**

```
SELECT module, name
FROM Modules, Lecturers
WHERE code=module AND year=2;
```

SQL query

$$\pi_{\text{module,name}}(\sigma_{\text{code=module AND year=2}}(\text{Modules} \times \text{Lecturers}))$$

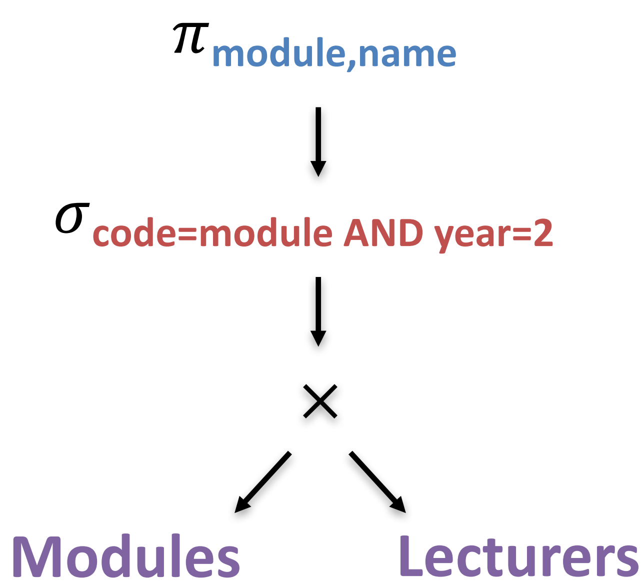
Query plan for the query

- Query plans are typically **represented as trees**

Query Plans As Trees

$$\pi_{\text{module,name}}(\sigma_{\text{code=module AND year=2}}(\text{Modules} \times \text{Lecturers}))$$

- Tree representation:



- Inner nodes = operators
- Leaves = input relations

- Such trees are evaluated from the leaves to the root

Exercise

- Represent the following query plan as a tree:

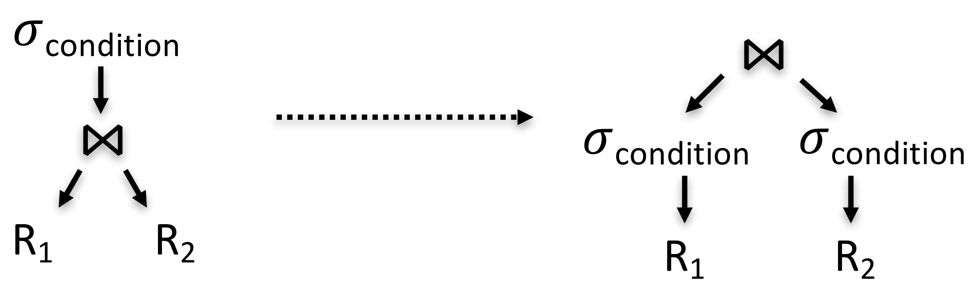
$$\sigma_{\text{code=module}}(\pi_{\text{code}}(\textbf{Modules}) \times \pi_{\text{module,name}}(\textbf{Lecturers}))$$

$$\begin{tikzcd}
& \sigma_{\text{code=module}} \arrow[d] & \\
& \times \arrow[dl] \arrow[dr] & \\
\pi_{\text{code}} \arrow[d] & & \pi_{\text{module,name}} \arrow[d] \\
\textbf{Modules} & & \textbf{Lecturers}
\end{tikzcd}$$

Equivalent Query Plans

- There are typically **many different query plans**
- DBMSs aim to **select a best possible query plan**
- Relational algebra is better suited than SQL for this
  - Can use **equivalence laws** of relational algebra to generate a query plan for the same query that can be executed faster!
  - Example:
    - $\sigma_{\text{condition}}(R_1 \bowtie R_2) = \sigma_{\text{condition}}(R_1) \bowtie \sigma_{\text{condition}}(R_2)$

Details will come later…



Summary

- DBMS translate SQL queries into relational algebra expressions, also called **(logical) query plans**
- The DBMS will then
  - Optimise the logical query plan by using equivalence laws (later…)
  - Select suitable algorithms for computing each operator in the logical query plan (later…)
- Next lecture: algorithms for computing operators of relational algebra