Uncertain Data Management
Reminders on Relational Algebra and Calculus

Antoine Amarilli[1], Silviu Maniu[2]

[1] Télécom ParisTech

[2] LRI

November 21st, 2016

Thanks to Pierre Senellart for proofreading these slides.

Table of contents

Relations

- Relation name and arity
- Attribute names (optional)

Relation Class, arity 5

date	teacher	resp	name	num

Relations

- Relation name and arity
- Attribute names (optional)

Relation Class, arity 5

date	**teacher**	**resp**	**name**	**num**
2015-11-23	Antoine	Fabian	Uncert. Data Mgmt	1
2015-11-30	Antoine	Fabian	Uncert. Data Mgmt	2
2015-12-07	Antoine	Fabian	Uncert. Data Mgmt	3
2015-11-14	Silviu	Fabian	Uncert. Data Mgmt	4

- Set of rows (tuples) on a domain (no duplicates)

Relational signature

- Signature σ: set of relation names and attributes
  - Example: Class(**date**, **teacher**, **resp**, **name**, **num**), Student(**id**, **name**), Member(**student**, **classname**)
- Database instance: one relation for each name in σ

Relational signature

- Signature σ: set of relation names and attributes
  - Example: Class(**date**, **teacher**, **resp**, **name**, **num**), Student(**id**, **name**), Member(**student**, **classname**)
- Database instance: one relation for each name in σ

$\rightarrow$ Query:

- Input: database
- Output: relation

Languages

Two languages to write queries:

- The relational algebra:
  - operational way to define queries
  - based on operators to construct new relations
- The relational calculus:
  - declarative way to define queries
  - based on first-order logic

Languages

Two languages to write queries:

- The relational algebra:
  - operational way to define queries
  - based on operators to construct new relations
- The relational calculus:
  - declarative way to define queries
  - based on first-order logic

→ Codd's theorem: both have the same expressive power

Languages

~~Two~~ Three languages to write queries:

- The relational algebra:
  - operational way to define queries
  - based on operators to construct new relations
- The relational calculus:
  - declarative way to define queries
  - based on first-order logic

$\rightarrow$ Codd's theorem: both have the same expressive power

- SQL, the practical language used by databases

Table of contents

Relational algebra

- Basic relations:
  - the relation names in the signature
  - constant relations, e.g., the empty relation
- Projection Π
- Selection σ
- Renaming ρ
- Union $\cup$
- Product $\times$ and join $\bowtie$
- Difference $-$

Projection

Keep a subsequence of the attributes:

Class

date	teacher	resp	name	num
2015-11-23	Antoine	Fabian	Uncert. Data Mgmt	1
2015-11-30	Antoine	Fabian	Uncert. Data Mgmt	2
2015-12-07	Antoine	Fabian	Uncert. Data Mgmt	3
2015-11-14	Silviu	Fabian	Uncert. Data Mgmt	4

Projection

Keep a subsequence of the attributes:

Class

date	teacher	resp	name	num
2015-11-23	Antoine	Fabian	Uncert. Data Mgmt	1
2015-11-30	Antoine	Fabian	Uncert. Data Mgmt	2
2015-12-07	Antoine	Fabian	Uncert. Data Mgmt	3
2015-11-14	Silviu	Fabian	Uncert. Data Mgmt	4

$\Pi_{\mathbf{teacher,resp}}(\text{Class})$

teacher	resp

Projection

Keep a subsequence of the attributes:

Class

date	**teacher**	**resp**	**name**	**num**
2015-11-23	Antoine	Fabian	Uncert. Data Mgmt	1
2015-11-30	Antoine	Fabian	Uncert. Data Mgmt	2
2015-12-07	Antoine	Fabian	Uncert. Data Mgmt	3
2015-11-14	Silviu	Fabian	Uncert. Data Mgmt	4

$\Pi_{\mathbf{teacher,resp}}(\text{Class})$

teacher	**resp**
Antoine	Fabian
Silviu	Fabian

Projection

Keep a subsequence of the attributes:

Class

date	teacher	resp	name	num
2015-11-23	Antoine	Fabian	Uncert. Data Mgmt	1
2015-11-30	Antoine	Fabian	Uncert. Data Mgmt	2
2015-12-07	Antoine	Fabian	Uncert. Data Mgmt	3
2015-11-14	Silviu	Fabian	Uncert. Data Mgmt	4

$\Pi_{\mathbf{teacher,resp}}(\text{Class})$

teacher	resp
Antoine	Fabian
Silviu	Fabian

→ Duplicates are removed

Selection

- Keep a subset of the tuples

Class

date	teacher	resp	name	num
2015-11-23	Antoine	Fabian	Uncert. Data Mgmt	1
2015-11-30	Antoine	Fabian	Uncert. Data Mgmt	2
2015-12-07	Antoine	Fabian	Uncert. Data Mgmt	3
2015-11-14	Silviu	Fabian	Uncert. Data Mgmt	4

Selection

- Keep a subset of the tuples

Class

date	**teacher**	**resp**	**name**	**num**
2015-11-23	Antoine	Fabian	Uncert. Data Mgmt	1
2015-11-30	Antoine	Fabian	Uncert. Data Mgmt	2
2015-12-07	Antoine	Fabian	Uncert. Data Mgmt	3
2015-11-14	Silviu	Fabian	Uncert. Data Mgmt	4

$\sigma_{\mathbf{teacher}=\text{“Silviu”}}(\text{Class})$

date	**teacher**	**resp**	**name**	**num**

Selection

- Keep a subset of the tuples

Class

date	**teacher**	**resp**	**name**	**num**
2015-11-23	Antoine	Fabian	Uncert. Data Mgmt	1
2015-11-30	Antoine	Fabian	Uncert. Data Mgmt	2
2015-12-07	Antoine	Fabian	Uncert. Data Mgmt	3
2015-11-14	Silviu	Fabian	Uncert. Data Mgmt	4

$\sigma_{\mathbf{teacher}=\text{“Silviu”}}(\text{Class})$

date	**teacher**	**resp**	**name**	**num**
2015-11-14	Silviu	Fabian	Uncert. Data Mgmt	4

Rename

- Change the name of attributes

Class

date	teacher	resp	name	num
2015-11-23	Antoine	Fabian	Uncert. Data Mgmt	1
2015-11-30	Antoine	Fabian	Uncert. Data Mgmt	2
2015-12-07	Antoine	Fabian	Uncert. Data Mgmt	3
2015-11-14	Silviu	Fabian	Uncert. Data Mgmt	4

Rename

- Change the name of attributes

Class

date	teacher	resp	name	num
2015-11-23	Antoine	Fabian	Uncert. Data Mgmt	1
2015-11-30	Antoine	Fabian	Uncert. Data Mgmt	2
2015-12-07	Antoine	Fabian	Uncert. Data Mgmt	3
2015-11-14	Silviu	Fabian	Uncert. Data Mgmt	4

$\rho_{\mathbf{resp}\to\mathbf{boss}}(\text{Class})$

Rename

- Change the name of attributes

Class

date	teacher	resp	name	num
2015-11-23	Antoine	Fabian	Uncert. Data Mgmt	1
2015-11-30	Antoine	Fabian	Uncert. Data Mgmt	2
2015-12-07	Antoine	Fabian	Uncert. Data Mgmt	3
2015-11-14	Silviu	Fabian	Uncert. Data Mgmt	4

$\rho_{\mathbf{resp}\to\mathbf{boss}}(\text{Class})$

date	teacher	boss	name	num
2015-11-23	Antoine	Fabian	Uncert. Data Mgmt	1
2015-11-30	Antoine	Fabian	Uncert. Data Mgmt	2
2015-13-07	Antoine	Fabian	Uncert. Data Mgmt	3
2015-11-14	Silviu	Fabian	Uncert. Data Mgmt	4

Union

- Take tuples occurring in one of the input tables
- Applies to two tables with the same attributes

Union

- Take tuples occurring in one of the input tables
- Applies to two tables with the same attributes

Students1

id	name
1	Arthur Dent
2	Ford Prefect

Union

- Take tuples occurring in one of the input tables
- Applies to two tables with the same attributes

Students1

id	name
1	Arthur Dent
2	Ford Prefect

$\cup$

Union

- Take tuples occurring in one of the input tables
- Applies to two tables with the same attributes

Students1

id	name
1	Arthur Dent
2	Ford Prefect

$\cup$

S2

id	name
42	Zaphod B.

Union

- Take tuples occurring in one of the input tables
- Applies to two tables with the same attributes

Students1

id	name
1	Arthur Dent
2	Ford Prefect

$\cup$

S2

id	name
42	Zaphod B.

$=$

Union

- Take tuples occurring in one of the input tables
- Applies to two tables with the same attributes

Students1

id	name
1	Arthur Dent
2	Ford Prefect

$\cup$

S2

id	name
42	Zaphod B.

$=$

Students1 $\cup$ S2

id	name
1	Arthur Dent
2	Ford Prefect
42	Zaphod B.

Product

- Take all combinations of the input tables

Product

- Take all combinations of the input tables

Students

id	name
1	Arthur Dent
2	Ford Prefect

Product

- Take all combinations of the input tables

Students

id	name
1	Arthur Dent
2	Ford Prefect

×

Product

- Take all combinations of the input tables

Students

id	name
1	Arthur Dent
2	Ford Prefect

$\times$

Rooms

room
E200
E242

Product

- Take all combinations of the input tables

Students

id	name
1	Arthur Dent
2	Ford Prefect

$\times$

Rooms

room
E200
E242

$=$

Product

- Take all combinations of the input tables

Students

id	name
1	Arthur Dent
2	Ford Prefect

$\times$

Rooms

room
E200
E242

$=$

Students $\times$ Rooms

id	name	room
1	Arthur Dent	E200
1	Arthur Dent	E242
2	Ford Prefect	E200
2	Ford Prefect	E242

Join

→ Product is useful to express joins:

Join

→ Product is useful to express joins:

Member

id	class
1	UDM
2	UDM

Join

→ Product is useful to express joins:

Member

id	class
1	UDM
2	UDM

$\bowtie$

Join

→ Product is useful to express joins:

Member

id	class
1	UDM
2	UDM

$\bowtie$

Class

class	date
UDM	Nov 23
ABC	Nov 27
UDM	Nov 30

Join

→ Product is useful to express joins:

Member

id	class
1	UDM
2	UDM

$\bowtie$

Class

class	date
UDM	Nov 23
ABC	Nov 27
UDM	Nov 30

$=$

Join

→ Product is useful to express joins:

Member

id	class
1	UDM
2	UDM

$\bowtie$

Class

class	date
UDM	Nov 23
ABC	Nov 27
UDM	Nov 30

$=$

Member $\bowtie$ Class

id	class	date
1	UDM	Nov 23
1	UDM	Nov 30
2	UDM	Nov 23
2	UDM	Nov 30

Join

→ Product is useful to express joins:

Member

id	class
1	UDM
2	UDM

$\bowtie$

Class

class	date
UDM	Nov 23
ABC	Nov 27
UDM	Nov 30

$=$

Member $\bowtie$ Class

id	class	date
1	UDM	Nov 23
1	UDM	Nov 30
2	UDM	Nov 23
2	UDM	Nov 30

Express Member $\bowtie$ Class with the previous operators:

Join

→ Product is useful to express joins:

Member

id	class
1	UDM
2	UDM

$\bowtie$

Class

class	date
UDM	Nov 23
ABC	Nov 27
UDM	Nov 30

$=$

Member $\bowtie$ Class

id	class	date
1	UDM	Nov 23
1	UDM	Nov 30
2	UDM	Nov 23
2	UDM	Nov 30

Express Member $\bowtie$ Class with the previous operators:

Member $\times$ Class

Join

→ Product is useful to express joins:

Member

id	class
1	UDM
2	UDM

$\bowtie$

Class

class	date
UDM	Nov 23
ABC	Nov 27
UDM	Nov 30

$=$

Member $\bowtie$ Class

id	class	date
1	UDM	Nov 23
1	UDM	Nov 30
2	UDM	Nov 23
2	UDM	Nov 30

Express Member $\bowtie$ Class with the previous operators:

$$\rho_{\textbf{class}\to\textbf{class2}}(\text{Member}) \times \text{Class}$$

Join

→ Product is useful to express joins:

Member

id	class
1	UDM
2	UDM

$\bowtie$

Class

class	date
UDM	Nov 23
ABC	Nov 27
UDM	Nov 30

$=$

Member $\bowtie$ Class

id	class	date
1	UDM	Nov 23
1	UDM	Nov 30
2	UDM	Nov 23
2	UDM	Nov 30

Express Member $\bowtie$ Class with the previous operators:

$$\sigma_{\mathbf{class=class2}} \left(\rho_{\mathbf{class \to class2}}(\text{Member}) \times \text{Class} \right)$$

Join

→ Product is useful to express joins:

Member

id	class
1	UDM
2	UDM

$\bowtie$

Class

class	date
UDM	Nov 23
ABC	Nov 27
UDM	Nov 30

$=$

Member $\bowtie$ Class

id	class	date
1	UDM	Nov 23
1	UDM	Nov 30
2	UDM	Nov 23
2	UDM	Nov 30

Express Member $\bowtie$ Class with the previous operators:

$$\Pi_{\text{id,class,date}}\Big(\ \sigma_{\textbf{class}=\textbf{class2}}\ \big(\ \rho_{\textbf{class}\to\textbf{class2}}(\text{Member}) \times \text{Class}\big)\Big)$$

Difference

- Take tuples that are in one table but not in the other
- Applies to two tables with same attributes

Difference

- Take tuples that are in one table but not in the other
- Applies to two tables with same attributes

Students1

id	name
1	Arthur Dent
2	Ford Prefect

Difference

- Take tuples that are in one table but not in the other
- Applies to two tables with same attributes

Students1

id	name
1	Arthur Dent
2	Ford Prefect

—

Difference

- Take tuples that are in one table but not in the other
- Applies to two tables with same attributes

Students1

id	name
1	Arthur Dent
2	Ford Prefect

—

S3

id	name
1	Arthur Dent
42	Zaphod B.

Difference

- Take tuples that are in one table but not in the other
- Applies to two tables with same attributes

Students1

id	name
1	Arthur Dent
2	Ford Prefect

$-$

S3

id	name
1	Arthur Dent
42	Zaphod B.

$=$

Difference

- Take tuples that are in one table but not in the other
- Applies to two tables with same attributes

Students1

id	name
1	Arthur Dent
2	Ford Prefect

$-$

S3

id	name
1	Arthur Dent
42	Zaphod B.

$=$

Students1 $-$ S3

id	name
2	Ford Prefect

Table of contents

Basic relations

```
CREATE TABLE Students(id INT(6), name VARCHAR(30));
INSERT INTO Students VALUES (1, 'Arthur Dent');
INSERT INTO Students VALUES (2, 'Ford Prefect');
```

Basic relations

```
CREATE TABLE Students(id INT(6), name VARCHAR(30));
INSERT INTO Students VALUES (1, 'Arthur Dent');
INSERT INTO Students VALUES (2, 'Ford Prefect');

SELECT * FROM Students;
```

Basic relations

```
CREATE TABLE Students(id INT(6), name VARCHAR(30));
INSERT INTO Students VALUES (1, 'Arthur Dent');
INSERT INTO Students VALUES (2, 'Ford Prefect');

SELECT * FROM Students;
```

```
+------+--------------+
| id   | name         |
+------+--------------+
|    1 | Arthur Dent  |
|    2 | Ford Prefect |
+------+--------------+
2 rows in set (0.00 sec)
```

Projection and rename

```
SELECT name FROM Students;
```

Projection and rename

```
SELECT name FROM Students;
```

```
+--------------+
| name         |
+--------------+
| Arthur Dent  |
| Ford Prefect |
+--------------+
```

Projection and rename

```
SELECT name FROM Students;
```

```
+--------------+
| name         |
+--------------+
| Arthur Dent  |
| Ford Prefect |
+--------------+
```

```
SELECT name, id, id AS identifier FROM Students;
```

Projection and rename

```
SELECT name FROM Students;
```

```
+--------------+
| name         |
+--------------+
| Arthur Dent  |
| Ford Prefect |
+--------------+
```

```
SELECT name, id, id AS identifier FROM Students;
```

```
+--------------+------+------------+
| name         | id   | identifier |
+--------------+------+------------+
| Arthur Dent  |    1 |          1 |
| Ford Prefect |    2 |          2 |
+--------------+------+------------+
```

Selection

```
SELECT * FROM Students;
```

Selection

```
SELECT * FROM Students;
```

```
+------+--------------+
| id   | name         |
+------+--------------+
|    1 | Arthur Dent  |
|    2 | Ford Prefect |
+------+--------------+
```

Selection

```
SELECT * FROM Students;
```

```
+------+--------------+
| id   | name         |
+------+--------------+
|    1 | Arthur Dent  |
|    2 | Ford Prefect |
+------+--------------+
```

```
SELECT * FROM Students WHERE id='2';
```

Selection

```
SELECT * FROM Students;
```

```
+------+--------------+
| id   | name         |
+------+--------------+
|    1 | Arthur Dent  |
|    2 | Ford Prefect |
+------+--------------+
```

```
SELECT * FROM Students WHERE id='2';
```

```
+------+--------------+
| id   | name         |
+------+--------------+
|    2 | Ford Prefect |
+------+--------------+
```

Union

```
SELECT * FROM Students;
```

```
+------+--------------+
| id   | name         |
+------+--------------+
|    1 | Arthur Dent  |
|    2 | Ford Prefect |
+------+--------------+
```

Union

```
SELECT * FROM Students;
```

```
+------+--------------+
| id   | name         |
+------+--------------+
|    1 | Arthur Dent  |
|    2 | Ford Prefect |
+------+--------------+
```

```
SELECT * FROM S2;
```

```
+------+----------+
| id   | name     |
+------+----------+
|   42 | Zaphod B |
+------+----------+
```

Union

```
SELECT * FROM Students;
```

```
+------+--------------+
| id   | name         |
+------+--------------+
|    1 | Arthur Dent  |
|    2 | Ford Prefect |
+------+--------------+
```

```
SELECT * FROM S2;
```

```
+------+----------+
| id   | name     |
+------+----------+
|   42 | Zaphod B |
+------+----------+
```

```
(SELECT * FROM Students)
UNION
(SELECT * FROM S2);
```

Union

```
SELECT * FROM Students;
```

```
+------+--------------+
| id   | name         |
+------+--------------+
|    1 | Arthur Dent  |
|    2 | Ford Prefect |
+------+--------------+
```

```
SELECT * FROM S2;
```

```
+------+----------+
| id   | name     |
+------+----------+
|   42 | Zaphod B |
+------+----------+
```

```
(SELECT * FROM Students)
UNION
(SELECT * FROM S2);
```

```
+------+--------------+
| id   | name         |
+------+--------------+
|    1 | Arthur Dent  |
|    2 | Ford Prefect |
|   42 | Zaphod B     |
+------+--------------+
```

Product

```
SELECT * FROM Students;
```

```
+------+--------------+
| id   | name         |
+------+--------------+
|    1 | Arthur Dent  |
|    2 | Ford Prefect |
+------+--------------+
```

Product

```
SELECT * FROM Students;
```

```
+------+--------------+
| id   | name         |
+------+--------------+
|    1 | Arthur Dent  |
|    2 | Ford Prefect |
+------+--------------+
```

```
SELECT * FROM Rooms;
```

```
+------+
| room |
+------+
| E200 |
| E242 |
+------+
```

Product

```
SELECT * FROM Students;
```

```
+------+--------------+
| id   | name         |
+------+--------------+
|    1 | Arthur Dent  |
|    2 | Ford Prefect |
+------+--------------+
```

```
SELECT * FROM Rooms;
```

```
+------+
| room |
+------+
| E200 |
| E242 |
+------+
```

```
SELECT * FROM Students, Rooms;
```

Product

```
SELECT * FROM Students;
```

```
+------+--------------+
| id   | name         |
+------+--------------+
|    1 | Arthur Dent  |
|    2 | Ford Prefect |
+------+--------------+
```

```
SELECT * FROM Rooms;
```

```
+------+
| room |
+------+
| E200 |
| E242 |
+------+
```

```
SELECT * FROM Students, Rooms;
```

```
+------+--------------+------+
| id   | name         | room |
+------+--------------+------+
|    1 | Arthur Dent  | E200 |
|    2 | Ford Prefect | E200 |
|    1 | Arthur Dent  | E242 |
|    2 | Ford Prefect | E242 |
+------+--------------+------+
```

Join

```
SELECT * FROM Member;
```

```
+------+-------+
| id   | class |
+------+-------+
|    1 | UDM   |
|    2 | UDM   |
+------+-------+
```

Join

```
SELECT * FROM Member;
```

```
+------+-------+
| id   | class |
+------+-------+
|    1 | UDM   |
|    2 | UDM   |
+------+-------+
```

```
SELECT * FROM Classes;
```

```
+-------+--------+
| class | date   |
+-------+--------+
| UDM   | Nov 23 |
| ABC   | Nov 27 |
| UDM   | Nov 30 |
+-------+--------+
```

Join

```
SELECT * FROM Member;
```

```
+------+-------+
| id   | class |
+------+-------+
|    1 | UDM   |
|    2 | UDM   |
+------+-------+
```

```
SELECT * FROM Classes;
```

```
+-------+--------+
| class | date   |
+-------+--------+
| UDM   | Nov 23 |
| ABC   | Nov 27 |
| UDM   | Nov 30 |
+-------+--------+
```

```
SELECT * FROM
  Member NATURAL JOIN Classes;
```

Join

```
SELECT * FROM Member;
```

```
+------+-------+
| id   | class |
+------+-------+
|    1 | UDM   |
|    2 | UDM   |
+------+-------+
```

```
SELECT * FROM Classes;
```

```
+-------+--------+
| class | date   |
+-------+--------+
| UDM   | Nov 23 |
| ABC   | Nov 27 |
| UDM   | Nov 30 |
+-------+--------+
```

```
SELECT * FROM
  Member NATURAL JOIN Classes;
```

```
+-------+------+--------+
| class | id   | date   |
+-------+------+--------+
| UDM   |    1 | Nov 23 |
| UDM   |    2 | Nov 23 |
| UDM   |    1 | Nov 30 |
| UDM   |    2 | Nov 30 |
+-------+------+--------+
```

Difference

```
SELECT * FROM Students;
```

```
+------+--------------+
| id   | name         |
+------+--------------+
|    1 | Arthur Dent  |
|    2 | Ford Prefect |
+------+--------------+
```

Difference

```
SELECT * FROM Students;
```

```
+------+--------------+
| id   | name         |
+------+--------------+
|    1 | Arthur Dent  |
|    2 | Ford Prefect |
+------+--------------+
```

```
SELECT * FROM S3;
```

```
+------+-------------+
| id   | name        |
+------+-------------+
|    1 | Arthur Dent |
|   42 | Zaphod B.   |
+------+-------------+
```

Difference

```
SELECT * FROM Students;
```

```
+------+--------------+
| id   | name         |
+------+--------------+
|    1 | Arthur Dent  |
|    2 | Ford Prefect |
+------+--------------+
```

```
SELECT * FROM S3;
```

```
+------+-------------+
| id   | name        |
+------+-------------+
|    1 | Arthur Dent |
|   42 | Zaphod B.   |
+------+-------------+
```

```
SELECT * FROM Students
WHERE (id, name) NOT IN
  (SELECT * FROM S3);
```

Difference

```
SELECT * FROM Students;
```

```
+------+--------------+
| id   | name         |
+------+--------------+
|    1 | Arthur Dent  |
|    2 | Ford Prefect |
+------+--------------+
```

```
SELECT * FROM S3;
```

```
+------+------------+
| id   | name       |
+------+------------+
|    1 | Arthur Dent |
|   42 | Zaphod B.  |
+------+------------+
```

```
SELECT * FROM Students
WHERE (id, name) NOT IN
  (SELECT * FROM S3);
```

```
+------+--------------+
| id   | name         |
+------+--------------+
|    2 | Ford Prefect |
+------+--------------+
```

Composing operations

Our translation of:

```
SELECT * FROM
  Member NATURAL JOIN Classes;
```

can be expressed as:

Composing operations

Our translation of:

```
SELECT * FROM
  Member NATURAL JOIN Classes;
```

can be expressed as:

```
SELECT id, class, date FROM
  (SELECT * FROM
    (SELECT id, class AS class2 FROM Member) sub1,
    Classes
  ) sub2
  WHERE class = class2;
```

Composing operations

Our translation of:

```
SELECT * FROM
  Member NATURAL JOIN Classes;
```

can be expressed as:

```
SELECT id, class, date FROM
  (SELECT * FROM
    (SELECT id, class AS class2 FROM Member) sub1,
    Classes
  ) sub2
  WHERE class = class2;
```

→ SQL can express the relational algebra

Composing operations

Our translation of:

```
SELECT * FROM
  Member NATURAL JOIN Classes;
```

can be expressed as:

```
SELECT id, class, date FROM
  (SELECT * FROM
    (SELECT id, class AS class2 FROM Member) sub1,
    Classes
  ) sub2
  WHERE class = class2;
```

→ SQL can express the relational algebra

- ... but please, never write queries like this!

Table of contents

Signatures and instances

Remember the signature σ:

- relations having a name and an arity
  - e.g., Class has arity 5
- no more attribute names; attributes are $(1, 2, 3, 4, 5)$

Signatures and instances

Remember the signature σ:

- relations having a name and an arity
  - e.g., Class has arity 5
- no more attribute names; attributes are $(1, 2, 3, 4, 5)$

An interpretation $\mathcal{I}$:

- domain $\mathcal{D}$ of values
- for each relation name R with arity n, a set $R^{\mathcal{I}}$ of n-tuples
- we write each n-tuple as a fact: $R(a, b, c)$

Signatures and instances

Remember the signature σ:

- relations having a name and an arity
  - e.g., Class has arity 5
- no more attribute names; attributes are $(1, 2, 3, 4, 5)$

An interpretation $\mathcal{I}$:

- domain $\mathcal{D}$ of values
- for each relation name R with arity n, a set $R^{\mathcal{I}}$ of n-tuples
- we write each n-tuple as a fact: $R(a, b, c)$

Example: $\text{Class}^{\mathcal{I}} = \{$

$\text{Class}(2015\text{-}11\text{-}23, \text{Antoine}, \text{Fabian}, \text{UDM}, 1),$

$\text{Class}(2015\text{-}11\text{-}30, \text{Antoine}, \text{Fabian}, \text{UDM}, 2),$

$\text{Class}(2015\text{-}12\text{-}07, \text{Antoine}, \text{Fabian}, \text{UDM}, 3),$

$\text{Class}(2015\text{-}12\text{-}14, \text{Silviu}, \text{Fabian}, \text{UDM}, 4),$

$\}$

First-order logic

First-order logic (FO) on a signature σ:

- atoms $R(x, y, z)$
  - each x, y, z can be a variable
  - each x, y, z can be a constant (e.g., "Silviu")
- Boolean connectives: AND, OR, NOT
- existential quantification $\exists x\ \phi(x)$
- universal quantification $\forall x\ \phi(x)$
- variables can be free or bound

Satisfaction of a formula

- Signature σ
- Interpretation $\mathcal{I}$
- FO formula ϕ

We can say that $\mathcal{I}$ satisfies ϕ:

Satisfaction of a formula

- Signature σ
- Interpretation $\mathcal{I}$
- FO formula ϕ

We can say that $\mathcal{I}$ satisfies ϕ:

$\rightarrow$ e.g., $\{R(a, b), R(b, c)\}$ satisfies $\exists x\, y\; R(x, y)$

Satisfaction of a formula

- Signature σ
- Interpretation $\mathcal{I}$
- FO formula ϕ

We can say that $\mathcal{I}$ satisfies ϕ:

→ e.g., $\{R(a,b), R(b,c)\}$ satisfies $\exists x\, y\; R(x,y)$

→ e.g., in $\mathcal{I} = \{S(a,b,c)\}$, for $\phi : \exists z\; S(x,y,z)$, $\mathcal{I}$ satisfies $\phi(a,b)$.

Satisfaction of a formula (formally)

- Signature σ
- Interpretation $\mathcal{I}$
- FO formula ϕ

Formally, by induction:

Satisfaction of a formula (formally)

- Signature σ
- Interpretation $\mathcal{I}$
- FO formula ϕ

Formally, by induction:

- if $\phi(x, y, z)$ is $R(x, y, z)$ and $R^{\mathcal{I}}$ contains $R(a, b, c)$ then $\mathcal{I}$ satisfies $\phi(a, b, c)$

Satisfaction of a formula (formally)

- Signature σ
- Interpretation $\mathcal{I}$
- FO formula ϕ

Formally, by induction:

- if $\phi(x, y, z)$ is $R(x, y, z)$ and $R^{\mathcal{I}}$ contains $R(a, b, c)$ then $\mathcal{I}$ satisfies $\phi(a, b, c)$
- if ϕ is $\psi \wedge \psi'$ and $\mathcal{I}$ satisfies ψ and ψ' then $\mathcal{I}$ satisfies ϕ

Satisfaction of a formula (formally)

- Signature σ
- Interpretation $\mathcal{I}$
- FO formula ϕ

Formally, by induction:

- if $\phi(x, y, z)$ is $R(x, y, z)$ and $R^{\mathcal{I}}$ contains $R(a, b, c)$ then $\mathcal{I}$ satisfies $\phi(a, b, c)$
- if ϕ is $\psi \wedge \psi'$ and $\mathcal{I}$ satisfies ψ and ψ' then $\mathcal{I}$ satisfies ϕ
- if $\phi(y, z)$ is $\exists x\, \psi(x, y, z)$ and $\mathcal{I}$ satisfies $\psi(a, b, c)$ for some a then $\mathcal{I}$ satisfies $\phi(b, c)$

Satisfaction of a formula (formally)

- Signature σ
- Interpretation $\mathcal{I}$
- FO formula ϕ

Formally, by induction:

- if $\phi(x, y, z)$ is $R(x, y, z)$ and $R^{\mathcal{I}}$ contains $R(a, b, c)$ then $\mathcal{I}$ satisfies $\phi(a, b, c)$
- if ϕ is $\psi \wedge \psi'$ and $\mathcal{I}$ satisfies ψ and ψ' then $\mathcal{I}$ satisfies ϕ
- if $\phi(y, z)$ is $\exists x\, \psi(x, y, z)$ and $\mathcal{I}$ satisfies $\psi(a, b, c)$ for some a then $\mathcal{I}$ satisfies $\phi(b, c)$
- etc.

FO vs queries

- An FO formula ϕ with free variables x, y, z can be seen as defining a query
- We can evaluate the query on an interpretation $\mathcal{I}$ and obtain a relation

FO vs queries

- An FO formula ϕ with free variables x, y, z can be seen as defining a query
- We can evaluate the query on an interpretation $\mathcal{I}$ and obtain a relation

$\rightarrow$ For each 3-tuple (a, b, c) in the domain, (a, b, c) is in the relation iff

FO vs queries

- An FO formula ϕ with free variables x, y, z can be seen as defining a query
- We can evaluate the query on an interpretation $\mathcal{I}$ and obtain a relation

$\rightarrow$ For each 3-tuple (a, b, c) in the domain, (a, b, c) is in the relation iff $\phi(a, b, c)$ holds in $\mathcal{I}$

FO vs queries

- An FO formula ϕ with free variables x, y, z can be seen as defining a query
- We can evaluate the query on an interpretation $\mathcal{I}$ and obtain a relation

$\rightarrow$ For each 3-tuple (a, b, c) in the domain, (a, b, c) is in the relation iff $\phi(a, b, c)$ holds in $\mathcal{I}$

$\rightarrow$ Let's translate the relational algebra operators!

Base relations and renames

- For a relation Student with arity 2, take the formula ϕ : Student(x, y)
  $\rightarrow$ free variables: x, y

Base relations and renames

- For a relation Student with arity 2,
  take the formula ϕ : Student(x, y)
  $\rightarrow$ free variables: x, y

- We no longer have attribute names, only positions
- We can swap variables around:
  $\rightarrow$ If $\phi(x, y)$ defines a relation, we can define $\psi(x, y)$ as $\phi(y, x)$

Selections

If $\phi(x, y, z)$ defines a relation, we can do:

- $\sigma_{1=2}$: take $\phi(x, x, z)$ with free variables x, z
- $\sigma_{1=\text{“Silviu”}}$: take $\phi(\text{“Silviu”}, y, z)$
  - free variables y, z
  - “Silviu” is a constant

Projections

- Take $\phi(x, y, z)$ that defines a relation
- We want to project to positions 1 and 2...

Projections

- Take $\phi(x, y, z)$ that defines a relation
- We want to project to positions 1 and 2...

$$\psi(x, y) := \exists z\ \phi(x, y, z)$$

Projections

- Take $\phi(x, y, z)$ that defines a relation
- We want to project to positions 1 and 2...

$$\psi(x, y) := \exists z\ \phi(x, y, z)$$

$\rightarrow$ This defines a relation of arity 2

$\rightarrow$ $\psi(a, b)$ holds iff $\phi(a, b, c)$ holds for some c

Union

- Take $\phi(x, y, z)$ and $\phi'(x, y, z)$ that define relations
- Assume that they have the same arity
- Say we want to take their union...

Union

- Take $\phi(x, y, z)$ and $\phi'(x, y, z)$ that define relations
- Assume that they have the same arity
- Say we want to take their union...

$$\psi(x, y, z) := \phi(x, y, z) \vee \phi'(x, y, z)$$

Union

- Take $\phi(x, y, z)$ and $\phi'(x, y, z)$ that define relations
- Assume that they have the same arity
- Say we want to take their union...

$$\psi(x, y, z) := \phi(x, y, z) \vee \phi'(x, y, z)$$

$\rightarrow$ This defines a relation with same arity

$\rightarrow$ $\psi(a, b, c)$ holds for all (a, b, c)
such that $\phi(a, b, c)$ holds or $\phi'(a, b, c)$ holds

Product

- Take $\phi(x, y)$ and $\phi'(z, w)$ that define relations
- How to take the product?

Product

- Take $\phi(x, y)$ and $\phi'(z, w)$ that define relations
- How to take the product?

$$\psi(x, y, z, w) := \phi(x, y) \wedge \phi'(z, w)$$

Product

- Take $\phi(x, y)$ and $\phi'(z, w)$ that define relations
- How to take the product?

$$\psi(x, y, z, w) := \phi(x, y) \wedge \phi'(z, w)$$

$\rightarrow$ The arity is the sum of the original arities

$\rightarrow$ $\psi(a, b, c, d)$ holds iff $\phi(a, b)$ and $\phi'(b, c)$ hold

Join

- We can't define the join based on attribute names.

Join

- We can't define the join based on attribute names.
- Join position 2 of $\phi(x, y)$ with position 1 of $\phi'(x, y)$:

Join

- We can't define the join based on attribute names.
- Join position 2 of $\phi(x, y)$ with position 1 of $\phi'(x, y)$:

$$\psi(x, y, z) := \phi(x, y) \wedge \phi'(y, z)$$

Difference

- Take $\phi(x, y)$ and $\phi'(x, y)$ with same arity.
- Let's define the difference

Difference

- Take $\phi(x, y)$ and $\phi'(x, y)$ with same arity.
- Let's define the difference

$$\psi(x, y) := \phi(x, y) \wedge \neg\phi'(x, y)$$

Difference

- Take $\phi(x, y)$ and $\phi'(x, y)$ with same arity.
- Let's define the difference

$$\psi(x, y) := \phi(x, y) \wedge \neg\phi'(x, y)$$

$\rightarrow$ In general, restrictions on when to use $\neg$

Table of contents

Conjunctive queries

- Relational algebra: disallow $\cup$ and $-$
  $\rightarrow$ only rename, select, project, product

Conjunctive queries

- Relational algebra: disallow $\cup$ and $-$
  $\rightarrow$ only rename, select, project, product
- SQL: essentially

```
SELECT [fields] FROM [tables] WHERE [condition]
```

Conjunctive queries

- Relational algebra: disallow $\cup$ and $-$
  $\rightarrow$ only rename, select, project, product
- SQL: essentially
  `SELECT [fields] FROM [tables] WHERE [condition]`
- Relational calculus:
  existentially quantified conjunction of atoms

Conjunctive queries

- Relational algebra: disallow $\cup$ and $-$
  $\rightarrow$ only rename, select, project, product
- SQL: essentially
  `SELECT [fields] FROM [tables] WHERE [condition]`
- Relational calculus:
  existentially quantified conjunction of atoms

Relation Class, arity 5

date	teacher	resp	name	num

$$\phi(d, n) : \exists r\, i\ \text{Class}(d, \text{“Silviu”}, r, n, i)$$

Conjunctive queries

- Relational algebra: disallow $\cup$ and $-$
  $\rightarrow$ only rename, select, project, product
- SQL: essentially
  `SELECT [fields] FROM [tables] WHERE [condition]`
- Relational calculus:
  existentially quantified conjunction of atoms

Relation Class, arity 5

date	teacher	resp	name	num

$$\phi(d, n) : \exists r\, i\ \text{Class}(d, \text{“Silviu”}, r, n, i)$$

“What are the date–name triples of classes taught by Silviu?”

Positive relational algebra

- Relational algebra: disallow the difference operator $-$

Positive relational algebra

- Relational algebra: disallow the difference operator $-$
- SQL: disallow `NOT IN`, `NOT EXISTS`, etc.

Positive relational algebra

- Relational algebra: disallow the difference operator $-$
- SQL: disallow `NOT IN`, `NOT EXISTS`, etc.
- Relational calculus: unions of conjunctive queries

$$\phi(\mathbf{x}) : \bigvee \exists \mathbf{y} \; R_1(\mathbf{x}, \mathbf{y}) \wedge \cdots \wedge R_n(\mathbf{x}, \mathbf{y})$$

Positive relational algebra

- Relational algebra: disallow the difference operator $-$
- SQL: disallow `NOT IN`, `NOT EXISTS`, etc.
- Relational calculus: unions of conjunctive queries

$$\phi(\mathbf{x}) : \bigvee \exists \mathbf{y}\ R_1(\mathbf{x}, \mathbf{y}) \wedge \cdots \wedge R_n(\mathbf{x}, \mathbf{y})$$

Relation Class, arity 5

date	**teacher**	**resp**	**name**	**num**

$$\phi(n, i) : (\exists t\, r\ \text{Class}(\text{“2015-11-23”}, t, r, n, i)) \vee (\exists t\, r\ \text{Class}(\text{“2015-11-30”}, t, r, n, i))$$

Positive relational algebra

- Relational algebra: disallow the difference operator $-$
- SQL: disallow `NOT IN`, `NOT EXISTS`, etc.
- Relational calculus: unions of conjunctive queries

$$\phi(\mathbf{x}) : \bigvee \exists \mathbf{y}\ R_1(\mathbf{x}, \mathbf{y}) \wedge \cdots \wedge R_n(\mathbf{x}, \mathbf{y})$$

Relation Class, arity 5

date	**teacher**	**resp**	**name**	**num**

$$\phi(n, i) : (\exists t\, r\ \text{Class}(\text{“2015-11-23”}, t, r, n, i)) \vee (\exists t\, r\ \text{Class}(\text{“2015-11-30”}, t, r, n, i))$$

“What are the classes taught today and next Monday?”

Boolean queries

- Relational calculus: queries with no free variables
  - → The formula is either satisfied or not

Boolean queries

- Relational calculus: queries with no free variables
  - → The formula is either satisfied or not
  - → Exemple: Boolean conjunctive query

Relation Class, arity 5

date	teacher	resp	name	num

$$\phi() : \exists d\, r\, n\, i\ \text{Class}(d, r, r, n, i)$$

Boolean queries

- Relational calculus: queries with no free variables
  - → The formula is either satisfied or not
  - → Exemple: Boolean conjunctive query

Relation Class, arity 5

date	**teacher**	**resp**	**name**	**num**

$$\phi() : \exists d\, r\, n\, i\ \text{Class}(d, r, r, n, i)$$

"Is there a class taught by the person responsible for the course?"

Boolean queries

- Relational calculus: queries with no free variables
  - → The formula is either satisfied or not
  - → Exemple: Boolean conjunctive query

Relation Class, arity 5

date	teacher	resp	name	num

$$\phi() : \exists d\, r\, n\, i\ \text{Class}(d, r, r, n, i)$$

"Is there a class taught by the person responsible for the course?"

- Relational algebra: table with arity 0:
  - → Either it is empty or it contains the empty tuple

Boolean queries

- Relational calculus: queries with no free variables
  - → The formula is either satisfied or not
  - → Exemple: Boolean conjunctive query

Relation Class, arity 5

date	teacher	resp	name	num

$$\phi() : \exists d\, r\, n\, i\ \text{Class}(d, r, r, n, i)$$

"Is there a class taught by the person responsible for the course?"

- Relational algebra: table with arity 0:
  - → Either it is empty or it contains the empty tuple
- SQL: can be approximated:

```
SELECT DISTINCT 1 FROM ([subquery])
```

  - → Pointless

Boolean queries

- Relational calculus: queries with no free variables
  - → The formula is either satisfied or not
  - → Exemple: Boolean conjunctive query

Relation Class, arity 5

date	teacher	resp	name	num

$$\phi() : \exists d\, r\, n\, i\ \text{Class}(d, r, r, n, i)$$

"Is there a class taught by the person responsible for the course?"

- Relational algebra: table with arity 0:
  - → Either it is empty or it contains the empty tuple
- SQL: can be approximated:

```
SELECT DISTINCT 1 FROM ([subquery])
```

  - → Pointless

→ Easier to reason about Boolean queries in the calculus

Table of contents

Standard query evaluation

Using the relational calculus, but it works the same for the algebra.

- We have an interpretation $\mathcal{I}$ ("database")
- We have a conjunctive query Q
- Our task:

  Boolean. Does $\mathcal{I}$ satisfies Q? (Yes/no, written $\mathcal{I} \models Q$)

  Non-Bool. For which values a, b, c does $\mathcal{I}$ satisfy $Q(a, b, c)$?

Standard query evaluation

Using the relational calculus, but it works the same for the algebra.

- We have an interpretation $\mathcal{I}$ ("database")
- We have a conjunctive query Q
- Our task:

  Boolean. Does $\mathcal{I}$ satisfies Q? (Yes/no, written $\mathcal{I} \models Q$)

  Non-Bool. For which values a, b, c does $\mathcal{I}$ satisfy $Q(a, b, c)$?

- Goal: be as efficient as possible

  $\rightarrow$ computational complexity

Complexity definition

Several notions of complexity:

Data. The query is fixed, the input is the instance

Query. The instance is fixed, the input is the query (rare)

Combined. The input is the instance and query

Complexity definition

Several notions of complexity:

Data. The query is fixed, the input is the instance

Query. The instance is fixed, the input is the query (rare)

Combined. The input is the instance and query

By default: data complexity: for each query Q we ask

- given $\mathcal{I}$ as input
- what is the complexity of checking if $\mathcal{I} \models Q$
- ... or of computing all a, b, c such that $\mathcal{I} \models Q(a, b, c)$

Complexity definition

Several notions of complexity:

Data. The query is fixed, the input is the instance

Query. The instance is fixed, the input is the query (rare)

Combined. The input is the instance and query

By default: data complexity: for each query Q we ask

- given $\mathcal{I}$ as input
- what is the complexity of checking if $\mathcal{I} \models Q$
- ... or of computing all a, b, c such that $\mathcal{I} \models Q(a, b, c)$

$\rightarrow$ Queries are usually small and data usually large

$\rightarrow$ Scaling in the data more important than in the query

Complexity results: data complexity

- Data complexity: fixed Boolean CQ Q, input instance $\mathcal{I}$
  - $\rightarrow$ Evaluation is in polynomial time

Complexity results: data complexity

- Data complexity: fixed Boolean CQ Q, input instance $\mathcal{I}$
  $\rightarrow$ Evaluation is in polynomial time
- Rewrite Q in the relational algebra
- For each operator, the complexity is polynomial in the input

Complexity results: data complexity

- Data complexity: fixed Boolean CQ Q, input instance $\mathcal{I}$
  $\rightarrow$ Evaluation is in polynomial time
- Rewrite Q in the relational algebra
- For each operator, the complexity is polynomial in the input

$\rightarrow$ Induction: evaluating a fixed query is polynomial

Complexity results: data complexity

- Data complexity: fixed Boolean CQ Q, input instance $\mathcal{I}$
  $\rightarrow$ Evaluation is in polynomial time
- Rewrite Q in the relational algebra
- For each operator, the complexity is polynomial in the input

$\rightarrow$ Induction: evaluating a fixed query is polynomial

$\rightarrow$ Highly parallelizable: with polynomial number of processors execution is in constant time

Complexity results: query and combined complexity

- Query complexity: input Boolean CQ Q and fixed instance I

Complexity results: query and combined complexity

- Query complexity: input Boolean CQ Q and fixed instance I
  - NP-complete for some I

Complexity results: query and combined complexity

- Query complexity: input Boolean CQ Q and fixed instance I
  - NP-complete for some I
  - Reduction from 3-colorability

Complexity results: query and combined complexity

- Query complexity: input Boolean CQ Q and fixed instance I
  - NP-complete for some I
  - Reduction from 3-colorability
- Combined complexity: input Boolean CQ Q and instance I

Complexity results: query and combined complexity

- Query complexity: input Boolean CQ Q and fixed instance I
  - NP-complete for some I
  - Reduction from 3-colorability
- Combined complexity: input Boolean CQ Q and instance I
  - NP-complete as well

References I


Abiteboul, S., Hull, R., and Vianu, V. (1995).
Foundations of Databases.
Addison-Wesley.
http://webdam.inria.fr/Alice/pdfs/all.pdf.


ISO (2008).
ISO 9075:2008: SQL.