Relational Algebra

Gordon Royle

School of Mathematics & Statistics
University of Western Australia

Relational Algebra

The theory underlying relational databases is called *relational algebra*, which is (unsurprisingly) the study of the algebra of *relations* — think of the word *algebra* as meaning *symbolic manipulation*.

Solving equations like $2 + 3x = 12y$ is algebra where the variables, x and y, are numbers, but in relational algebra, the “variables” are relations!

This content is covered in Jennifer Widom’s “mini-course”

- Databases: DB4 Relational Algebra

from Coursera (`https://www.coursera.org`).

Relations

If you open any introductory book on Pure Mathematics, you will find a definition such as this:

DEFINITION: *A relation of arity n is a subset*

$$S \subseteq A_1 \times A_2 \times \cdots \times A_n$$

where

$$A_1 \times A_2 \times \cdots \times A_n$$

denotes the Cartesian product of the sets $A_1, A_2, \ldots, A_n$.

Sets

We won't be too formal about sets — essentially a set is an *unordered collection* of "objects" *with no repeats*.

- A set of numbers

$$A = \{1, 2, 3, 4, 5\}$$

- A set of colours

$$B = \{\text{red}, \text{blue}, \text{green}\}$$

- A set of names

$$C = \{\text{Alice}, \text{Bob}, \text{Chloë}\}$$

Cartesian product

The *Cartesian product* of two sets S and T is the set

$$S \times T = \{(s,t) : s \in S, t \in T\}.$$

More informally, $S \times T$ is the set of 2-*tuples* such that the first component is from S, and the second component is from T.

For tuples, the order *does matter*.

Some examples

Using our earlier examples, if

$$A = \{1, 2, 3, 4, 5\} \qquad B = \{\text{red}, \text{blue}, \text{green}\}$$

then

$$\begin{aligned} A \times B = \{ & (1, \text{red}), (2, \text{red}), (3, \text{red}), (4, \text{red}), (5, \text{red}) \\ & (1, \text{blue}), (2, \text{blue}), (3, \text{blue}), (4, \text{blue}), (5, \text{blue}) \\ & (1, \text{green}), (2, \text{green}), (3, \text{green}), (4, \text{green}), (5, \text{green})\} \end{aligned}$$

Databases

How does all this relate to *Databases*?

Each *type* can be viewed as a set — namely the set of *all legal values* of that particular type.

For example, the type `INT` is the set consisting of all integers (i.e. whole numbers) x such that

$$-2147483648 \leq x \leq 2147483647.$$

In other words, you can store *any whole number* between these bounds in a column of type `INT`, and nothing else.

A 2-column table

Suppose we have a table with *two columns*, similar to `Country`:

```
+------+------------+
| Code | Population |
+------+------------+
| ABW  |     103000 |
| AFG  |   22720000 |
| AGO  |   12878000 |
| AIA  |       8000 |
..
```

The set of *all legal values* for `Code` is all 3-character strings

$$\{\texttt{AAA}, \texttt{AAB}, \texttt{AAC}, \ldots, \texttt{ZZZ}\}$$

and the set of *all legal values* for `Population` is a range of numbers.

The Cartesian product

The Cartesian product of the two sets `CHAR(3)` and `INT` is then all the *possible tuples* that form *legitimate rows* for the relation.

```
(AAA, 1)
(AAA, 2)
.
.
.
(ZZZ, 2147483647)
```

At any given moment, the *actual* set of rows — that is, the *instance* of the relation — will be a *subset* of the Cartesian product, namely the collection of the legitimate tuples *currently contained* by the table.

Higher arity

A relation of arity 2 is called a *binary relation*.

If there are more than 2 sets, say A, B and C, then we define the Cartesian product in the natural way as the set of *triples*

$$A \times B \times C = \{(a, b, c) : a \in A, b \in B, c \in C\}.$$

A relationship of arity 3 is sometimes called a *ternary relation*, and so on, but eventually the individual names run out.

Example relations

Consider a relation `Student` with three attributes

- `id` of type `CHAR(8)`
- `name` of type `VARCHAR(64)`
- `gender` of type `ENUM("M", "F", "X")`

and a relation `Grade` also with three attributes

- `id` of type `CHAR(8)`
- `unit` of type `CHAR(8)`
- `grade` of type `INT`

Example relations

id	name	gender
12345678	Ebenezer Scrooge	M
12345682	Jane Austen	F
12345689	Martin Chuzzlewit	M

id	unit	grade
12345678	CITS1402	88
12345678	CITS2211	75
12345682	CITS1402	91
12345682	CITS2211	71
12345689	CITS1402	55

Two Greek Symbols

Mathematics (and theoretical computer science) make heavy use of the Greek alphabet, and we need two symbols in particular — *"sigma"* and *"pi"*.

The *lower-case* versions of these two symbols are

$$\sigma \qquad \pi$$

while the *upper-case* versions are

$$\Sigma \qquad \Pi$$

Relational Algebra

Relational algebra is the *mathematical language* describing the *manipulation of relations*, while SQL is an approximation to relational algebra.

There are two fundamental operators:

- *Selection* denoted by σ (sigma)
  This operator *selects* a subset of the rows satisfying some condition
- *Projection* denoted by π (pi)
  This operator *projects* the tuples onto a subset of the columns

Terminology warning

In `SQL` the keyword `SELECT` is used to specify *which columns* to be output — this is what the *projection operator* π does in relational algebra.

In `SQL` the keyword `WHERE` is used to specify *which rows* are to be processed — this is what the *selection operator* σ does in relational algebra.

Purpose	In `SQL`	In rel. alg
Choose cols	`SELECT`	π
Choose rows	`WHERE`	σ

Selection

If R is a relation instance and c is a *boolean condition* (i.e. an expression that is either *true* or *false*) then the value of the expression

$$\sigma_c(R)$$

is the relation containing only the rows of R *that satisfy the condition* c.

Sometimes, expressions leave off brackets if they are not necessary

$$\sigma_c\ R$$

(This is like writing $\cos x$ instead of $\cos(x)$.)

Selection

Consider the relational algebra expression:

$$\sigma_{\texttt{grade}>80}\ (\texttt{Grade})$$

This should be viewed as a *function* applied to the relation `Grade` whose *value* is another relation.

id	unit	grade
12345678	CITS1402	88
12345678	CITS2211	75
12345682	CITS1402	91
12345682	CITS2211	71
12345689	CITS1402	55

Projection

Now consider the expression

$$\pi_{\texttt{id}}(\texttt{Student})$$

This goes through each row, and only keeps the *specified columns*.

The result is *another relation* with fewer columns but — in this case — the same number of rows.

id
12345678
12345682
12345689

MySQL - CREATE TABLE

First we *create* the (empty) tables:

```
CREATE TABLE Student (id CHAR(8),
                      name VARCHAR(64),
                      gender ENUM("M","F","X"));

CREATE TABLE Grade (id CHAR(8),
                    unit CHAR(8),
                    grade INT);
```

MySQL - INSERT INTO

Next we *insert* the initial data:

```
INSERT INTO Student VALUES('12345678', 'Ebenezer Scrooge', 'M');
INSERT INTO Student VALUES('12345682', 'Jane Austen', 'F');
INSERT INTO Student VALUES('12345689', 'Martin Chuzzlewit', 'M');

INSERT INTO Grade VALUES('12345678', 'CITS1402', 88);
INSERT INTO Grade VALUES('12345678', 'CITS2211', 75);
INSERT INTO Grade VALUES('12345682', 'CITS1402', 91);
INSERT INTO Grade VALUES('12345682', 'CITS2211', 71);
INSERT INTO Grade VALUES('12345689', 'CITS1402', 55);
```

MySQL - SELECT *

In relational algebra, an entire relation can be referred to just by its name:

```
Grade
```

In MySQL this is not a legal expression, and we must explicitly state that we want *all the columns* from a table.

```
mysql> SELECT * from Grade;
+----------+----------+-------+
| id       | unit     | grade |
+----------+----------+-------+
| 12345678 | CITS1402 |    88 |
| 12345678 | CITS2211 |    75 |
| 12345682 | CITS1402 |    91 |
| 12345682 | CITS2211 |    71 |
| 12345689 | CITS1402 |    55 |
+----------+----------+-------+
5 rows in set (0.00 sec)
```

Selection in MySQL

In MySQL a *selection* is accomplished by adding a `WHERE` clause containing the conditions.

```
SELECT *
FROM Grade
WHERE grade > 80;
+----------+----------+-------+
| id       | unit     | grade |
+----------+----------+-------+
| 12345678 | CITS1402 |    88 |
| 12345682 | CITS1402 |    91 |
+----------+----------+-------+
2 rows in set (0.00 sec)
```

Projection in MySQL

In MySQL a *projection* is accomplished by explicitly listing the columns you want to keep.

```
SELECT id
FROM Student;
+----------+
| id       |
+----------+
| 12345678 |
| 12345682 |
| 12345689 |
+----------+
3 rows in set (0.00 sec)
```

Select and Project in MySQL

In relational algebra we can combine operations

$$\pi_{\texttt{id}}(\sigma_{\texttt{grade}>\texttt{80}}(\texttt{Grade}))$$

This *first* operation selects the rows with `grade > 80` and the second *then* projects onto the `id` column only.

```
SELECT id
FROM Grade
WHERE grade > 80;
+----------+
| id       |
+----------+
| 12345678 |
| 12345682 |
+----------+
2 rows in set (0.00 sec)
```

Relations are sets . . .

While MySQL *approximates* relational algebra, it doesn't do it perfectly.

$$\pi_{\texttt{id}}(\texttt{Grade})$$

should produce

id
12345678
12345682
12345689

because a relation is defined to be a *set* of tuples, so repeats are not allowed.

. . . but not in MySQL . . .

```
mysql> SELECT id FROM Grade;
+----------+
| id       |
+----------+
| 12345678 |
| 12345678 |
| 12345682 |
| 12345682 |
| 12345689 |
+----------+
5 rows in set (0.00 sec)
```

. . . unless you force it

```
mysql> SELECT DISTINCT id FROM Grade;
+----------+
| id       |
+----------+
| 12345678 |
| 12345682 |
| 12345689 |
+----------+
3 rows in set (0.00 sec)
```

Boolean expressions

A expression like `grade > 80` is called a *Boolean expression* because when it is evaluated it takes the value `true` or `false`.

Boolean expressions can be combined using the `AND` and `OR` operators, which are usually written $\wedge$ and $\vee$ respectively.

In fancy Maths books,

- `AND` ($\wedge$) is called *conjunction*,
- `OR` ($\vee$) is called *disjunction*.

The word *Boolean* and the phrase *boolean algebra* are named to honour George Boole (1815–1864) who developed the idea of representing and manipulating logical expressions symbolically.

Head's letter

Suppose that the Head sends letters of congratulations to students who get more than 80 in any unit, or more than 70 in `CITS2211`.

What *relational algebra expression* yields a relation containing just the student ids for all students who should receive a letter?

- Boolean expression to test if any grade is more than 80:

Head's letter

Suppose that the Head sends letters of congratulations to students who get more than 80 in any unit, or more than 70 in `CITS2211`.

What *relational algebra expression* yields a relation containing just the student ids for all students who should receive a letter?

- Boolean expression to test if any grade is more than 80:

  `grade > 80`
- Boolean expression to test if a CITS2211 grade is more than 70:

Head's letter

Suppose that the Head sends letters of congratulations to students who get more than 80 in any unit, or more than 70 in `CITS2211`.

What *relational algebra expression* yields a relation containing just the student ids for all students who should receive a letter?

- Boolean expression to test if any grade is more than 80:

  `grade > 80`

- Boolean expression to test if a CITS2211 grade is more than 70:

  $(\texttt{unit} =' \texttt{CITS2211}') \wedge (\texttt{grade} > \texttt{70})$

The final condition

The overall boolean expression is the AND of these two

$$(\texttt{grade} > \texttt{80}) \vee \left((\texttt{unit} =' \texttt{CITS2211}') \wedge (\texttt{grade} > \texttt{70}) \right)$$

Thus the relational algebra expression whose value is the relation consisting of all the rows of Grade meeting this condition is

$$\sigma_{(\texttt{grade}>\texttt{80}) \vee (\texttt{grade}>70 \wedge \texttt{unit}='\texttt{CITS2211}')} (\texttt{Grade})$$

The final expression

The final expression that produces the desired relation is a *projection* of the relation onto the `id` column

$$\pi_{\texttt{id}}\left(\sigma_{(\texttt{grade}>\texttt{80})\vee(\texttt{grade}>70\wedge\texttt{unit}='\texttt{CITS2211}')}\,(\texttt{Grade})\right)$$

In SQL

```
SELECT *
FROM Grade
WHERE (grade > 80) OR
      (grade > 70 AND unit = 'CITS2211');
+----------+----------+-------+
| id       | unit     | grade |
+----------+----------+-------+
| 12345678 | CITS1402 |    88 |
| 12345678 | CITS2211 |    75 |
| 12345682 | CITS1402 |    91 |
| 12345682 | CITS2211 |    71 |
+----------+----------+-------+
4 rows in set (0.00 sec)
```

In SQL

```
SELECT id
FROM Grade
WHERE (grade > 80) OR
      (grade > 70 AND unit = 'CITS2211');
+----------+
| id       |
+----------+
| 12345678 |
| 12345678 |
| 12345682 |
| 12345682 |
+----------+
4 rows in set (0.00 sec)
```

In SQL

```
SELECT DISTINCT(id)
FROM Grade
WHERE (grade > 80) OR
      (grade > 70 AND unit = 'CITS2211');
+----------+
| id       |
+----------+
| 12345678 |
| 12345682 |
+----------+
4 rows in set (0.00 sec)
```

More columns

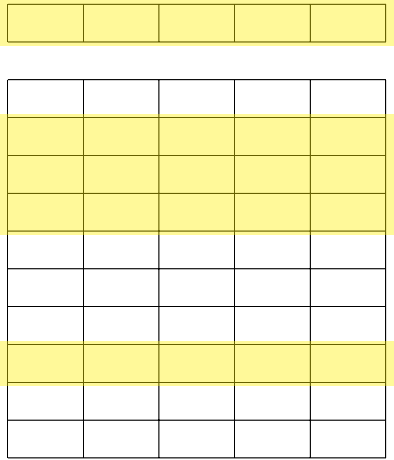
In relational algebra, the projection can pick out any number of columns

$$\pi_{\texttt{id,name}}(\texttt{Student})$$

```
SELECT id, name
FROM Student;
+----------+--------------------+
| id       | name               |
+----------+--------------------+
| 12345678 | Ebenezer Scrooge   |
| 12345682 | Jane Austen        |
| 12345689 | Martin Chuzzlewit |
+----------+--------------------+
3 rows in set (0.00 sec)
```

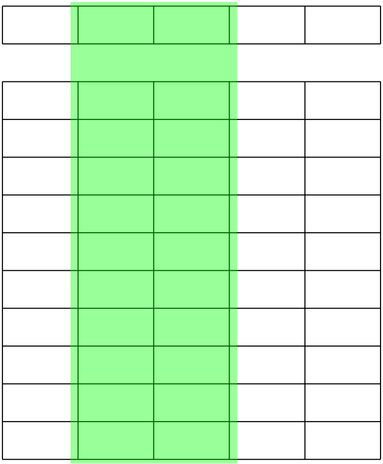
Reminder - selection

The *select* operator σ selects *rows* of a table (inlcuding the header).



Reminder - Projection

The *project* operator π selects *columns* of a table, including the header.



Products and Joins

The *Cartesian product* of relational algebra

$$\texttt{Student} \times \texttt{Grade}$$

creates a new relation with 6 attributes, namely

`id`, `name`, `gender`, `id`, `unit`, `grade`

and with $3 \times 5 = 15$ rows obtained by gluing together a tuple from `Student` and a tuple from `Grade` in *every possible way*.

Cartesian product in MySQL

```
mysql> SELECT * FROM Student, Grade;
+----------+--------------------+--------+----------+----------+-------+
| id       | name               | gender | id       | unit     | grade |
+----------+--------------------+--------+----------+----------+-------+
| 12345678 | Ebenezer Scrooge   | M      | 12345678 | CITS1402 |    88 |
| 12345682 | Jane Austen        | F      | 12345678 | CITS1402 |    88 |
| 12345689 | Martin Chuzzlewit  | M      | 12345678 | CITS1402 |    88 |
| 12345678 | Ebenezer Scrooge   | M      | 12345678 | CITS2211 |    75 |
| 12345682 | Jane Austen        | F      | 12345678 | CITS2211 |    75 |
| 12345689 | Martin Chuzzlewit  | M      | 12345678 | CITS2211 |    75 |
| 12345678 | Ebenezer Scrooge   | M      | 12345682 | CITS1402 |    91 |
| 12345682 | Jane Austen        | F      | 12345682 | CITS1402 |    91 |
| 12345689 | Martin Chuzzlewit  | M      | 12345682 | CITS1402 |    91 |
| 12345678 | Ebenezer Scrooge   | M      | 12345682 | CITS2211 |    71 |
| 12345682 | Jane Austen        | F      | 12345682 | CITS2211 |    71 |
| 12345689 | Martin Chuzzlewit  | M      | 12345682 | CITS2211 |    71 |
| 12345678 | Ebenezer Scrooge   | M      | 12345689 | CITS1402 |    55 |
| 12345682 | Jane Austen        | F      | 12345689 | CITS1402 |    55 |
| 12345689 | Martin Chuzzlewit  | M      | 12345689 | CITS1402 |    55 |
+----------+--------------------+--------+----------+----------+-------+
15 rows in set (0.00 sec)
```

Matching them up

What we *really want* is for each row to combine the `Student` information and the `Grade` information for the *same student*.

In relational algebra

$$\sigma_{\texttt{Student.id}=\texttt{Grade.id}}(\texttt{Student} \times \texttt{Grade})$$

This forms the Cartesian product, and then selects only the rows where the two occurrences of `id` match.

Matching in MySQL

```
SELECT *
FROM Student, Grade
WHERE Student.id = Grade.id;
+----------+-------------------+--------+----------+----------+-------+
| id       | name              | gender | id       | unit     | grade |
+----------+-------------------+--------+----------+----------+-------+
| 12345678 | Ebenezer Scrooge  | M      | 12345678 | CITS1402 |    88 |
| 12345678 | Ebenezer Scrooge  | M      | 12345678 | CITS2211 |    75 |
| 12345682 | Jane Austen       | F      | 12345682 | CITS1402 |    91 |
| 12345682 | Jane Austen       | F      | 12345682 | CITS2211 |    71 |
| 12345689 | Martin Chuzzlewit | M      | 12345689 | CITS1402 |    55 |
+----------+-------------------+--------+----------+----------+-------+
5 rows in set (0.00 sec)
```

Natural Join

In relational algebra, the *natural join* operator automatically matches *all columns* with the *same name*, and then removes one of each duplicate pair.

The symbol for a natural join is the "bowtie" symbol

$$\bowtie$$

So if R and S are relations, then

$$R \bowtie S$$

denotes their natural join.

Sample natural join

Therefore, in relational algebra

$$\texttt{Student} \bowtie \texttt{Grade}$$

yields a relation with *five* columns.

In MySQL

```
SELECT *
FROM Student NATURAL JOIN Grade;
+----------+-------------------+--------+----------+-------+
| id       | name              | gender | unit     | grade |
+----------+-------------------+--------+----------+-------+
| 12345678 | Ebenezer Scrooge  | M      | CITS1402 |    88 |
| 12345678 | Ebenezer Scrooge  | M      | CITS2211 |    75 |
| 12345682 | Jane Austen       | F      | CITS1402 |    91 |
| 12345682 | Jane Austen       | F      | CITS2211 |    71 |
| 12345689 | Martin Chuzzlewit | M      | CITS1402 |    55 |
+----------+-------------------+--------+----------+-------+
5 rows in set (0.00 sec)
```

The rename operator

Relational algebra also has an operator ρ (rho) for *renaming* tables and attributes.

The syntax of this operator is not fully standardised, so you may see a number of variations, but we'll stick to one of the simplest.

Suppose that R is a relation with attributes $r_1, r_2, \ldots, r_n$. Then the value of the expression

$$\rho_{S(s_1, s_2, \ldots, s_n)}(R)$$

is a relation called S with attributes $s_1, s_2, \ldots, s_n$ but with exactly the same *contents* as R.

Renaming

r_1	r_2	r_3	r_4

R

$$\rho_{S(s_1,s_2,\ldots,s_n)}(R)$$

s_1	s_2	s_3	s_4

S

Why do we need rename?

Renaming is mostly for convenience, but it is essential for *self-joins* — this is when a table is joined to (another copy of) itself.

For example, suppose we want to find the students who have grades for more than one unit.

(This can be done by using some of the “counting operators” of MySQL but we’ll do it with joins first.)

Self-joins

We really need to analyse *two distinct rows* of the `Grade` table, but we can't do this because SQL is a "row-processing machine".

So we have to convert "two distinct rows" to "a single row of twice the size".

```
mysql> SELECT * FROM Grade, Grade;
ERROR 1066 (42000): Not unique table/alias: 'Grade'
```

Self-joins

We'll rename each copy of the table.

```
SELECT *
FROM Grade G1, Grade G2;
+----------+----------+-------+----------+----------+-------+
| id       | unit     | grade | id       | unit     | grade |
+----------+----------+-------+----------+----------+-------+
| 12345678 | CITS1402 |    88 | 12345678 | CITS1402 |    88 |
| 12345678 | CITS2211 |    75 | 12345678 | CITS1402 |    88 |
| 12345682 | CITS1402 |    91 | 12345678 | CITS1402 |    88 |
| 12345682 | CITS2211 |    71 | 12345678 | CITS1402 |    88 |
| 12345689 | CITS1402 |    55 | 12345678 | CITS1402 |    88 |
| 12345678 | CITS1402 |    88 | 12345678 | CITS2211 |    75 |
...
| 12345689 | CITS1402 |    55 | 12345689 | CITS1402 |    55 |
+----------+----------+-------+----------+----------+-------+
25 rows in set (0.00 sec)
```

Self-joins

But now we have to fix the usual `JOIN` problem that the two halves sometimes make no sense.

```
SELECT *
FROM Grade G1, Grade G2
WHERE G1.id = G2.id;
+----------+----------+-------+----------+----------+-------+
| id       | unit     | grade | id       | unit     | grade |
+----------+----------+-------+----------+----------+-------+
| 12345678 | CITS1402 |    88 | 12345678 | CITS1402 |    88 |
| 12345678 | CITS2211 |    75 | 12345678 | CITS1402 |    88 |
| 12345678 | CITS1402 |    88 | 12345678 | CITS2211 |    75
```

Self-joins

Each row should refer to enrolments in two *different* units.

```
SELECT *
FROM Grade G1, Grade G2
WHERE G1.id = G2.id
      AND G1.unit <> G2.unit;
+----------+----------+-------+----------+----------+-------+
| id       | unit     | grade | id       | unit     | grade |
+----------+----------+-------+----------+----------+-------+
| 12345678 | CITS2211 |    75 | 12345678 | CITS1402 |    88 |
| 12345678 | CITS1402 |    88 | 12345678 | CITS2211 |    75 |
| 12345682 | CITS2211 |    71 | 12345682 | CITS1402 |    91 |
| 12345682 | CITS1402 |    91 | 12345682 | CITS2211 |    71 |
+----------+----------+-------+----------+----------+-------+
4 rows in set (0.00 sec)
```

Self-joins

Now we can pull out just what we want.

```
SELECT DISTINCT G1.id AS overloaded
FROM Grade G1, Grade G2
WHERE G1.id = G2.id
      AND G1.unit <> G2.unit;
+------------+
| overloaded |
+------------+
| 12345678   |
| 12345682   |
+------------+
```

A small peek ahead

Relational algebra is *relation-closed* — the result of any expression involving relations is a relation itself.

This means that wherever a *relation* occurs in an expression, the relation can be a *derived relation* rather than an actual relation.

Similarly in SQL, a table used in a query need not be an *actual table*, but can instead be the result of *another query*.

Names of overloaded students

```
SELECT *
FROM   student S,
       (SELECT DISTINCT G1.id AS overloaded
        FROM   grade G1,
               grade G2
        WHERE  G1.id = G2.id
               AND G1.unit <> G2.unit) AS T
WHERE  S.id = T.overloaded;
+----------+------------------+--------+------------+
| id       | name             | gender | overloaded |
+----------+------------------+--------+------------+
| 12345678 | Ebenezer Scrooge | M      | 12345678   |
| 12345682 | Jane Austen      | F      | 12345682   |
+----------+------------------+--------+------------+
2 rows in set (0.00 sec)
```

Set notation

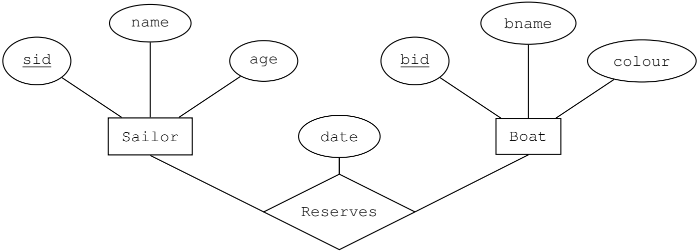
Recall some basic *set theory terminology*:

If A and B are sets, then

- The *union* of A and B, denoted $A \cup B$ is the set containing everything that is in *either* A *or* B (or both).
- The *intersection* of A and B, denoted $A \cap B$ is the set containing everything that is in *both* A *and* B.
- The *set difference* of A and B, denoted $A - B$ is the set containing everything that *is in* A *but not in* B.

In-class examples

Consider the following conceptual schema that is related to a boat-rental operation.



This example is based on one in the book *Database Management Systems* by Ramakrishnan & Gehrke.

Sample Boat

```
mysql> SELECT * FROM Boat;
+-----+-----------+--------+
| bid | name      | colour |
+-----+-----------+--------+
| 101 | Interlake | blue   |
| 102 | Interlake | red    |
| 103 | Clipper   | green  |
| 104 | Marine    | red    |
+-----+-----------+--------+
4 rows in set (0.00 sec)
```

Sample Sailor

```
mysql> SELECT * FROM Sailor;
+-----+---------+------+
| sid | sname   | age  |
+-----+---------+------+
|  22 | Dustin  |   45 |
|  29 | Brutus  |   33 |
|  31 | Lubber  | 55.5 |
|  32 | Andy    | 25.5 |
|  58 | Rusty   |   35 |
|  64 | Horatio |   35 |
|  71 | Zorba   |   16 |
|  74 | Horatio |   36 |
|  85 | Art     | 25.5 |
|  95 | Bob     | 63.5 |
+-----+---------+------+
10 rows in set (0.00 sec)
```

Sample Reserves

```
mysql> SELECT * FROM Reserves;
+-----+-----+------------+
| sid | bid | date       |
+-----+-----+------------+
|  22 | 101 | 2014-08-10 |
|  22 | 102 | 2014-08-10 |
|  22 | 103 | 2014-08-11 |
|  22 | 104 | 2014-08-12 |
|  31 | 102 | 2014-08-02 |
|  31 | 103 | 2014-08-03 |
|  31 | 104 | 2014-08-17 |
|  64 | 102 | 2014-08-18 |
|  64 | 102 | 2014-08-05 |
|  74 | 103 | 2014-08-05 |
+-----+-----+------------+
10 rows in set (0.00 sec)
```

Simple expressions

$$\pi_{\text{sid}}(\texttt{Sailor})$$

Simple expressions

$$\pi_{\text{sid}}(\texttt{Sailor})$$

```
SELECT sid
FROM Sailor;
+-----+
| sid |
+-----+
|  22 |
|  29 |
|  31 |
|  32 |
|  58 |
|  64 |
|  71 |
|  74 |
|  85 |
|  95 |
+-----+
10 rows in set (0.01 sec)
```

Queries in relational algebra

What are the names of sailors who have reserved boat 103?

- Which tables contain the information?
  The *names* of sailors are only in `Sailor`, so we have to use this table.
  The *boat ids* are in both `Reserves` and `Boat` so we can use one or both of these.
- Determine which joins are needed
  We can create a table with names and reservation details by joining `Sailor` with `Reserves`.

Doing the join

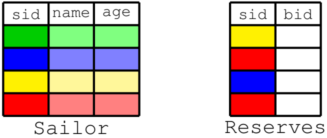
What kind of join should be done?

- `Sailor(sid, sname, age)`
- `Reserves(sid, bid, date)`

We need to "line up" `Sailor.sid` with `Reserves.sid` - as this is the only attribute in common, we can use the *natural join*:

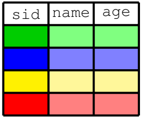
$$\texttt{Sailor} \bowtie \texttt{Reserves}$$

Natural Join 1

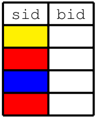


Natural join 2

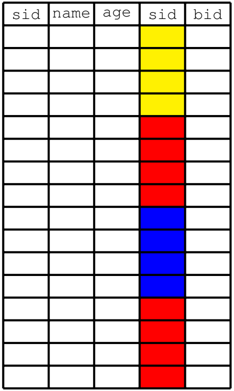
At a logical level, the natural join first forms the *Cartesian product*:



Sailor

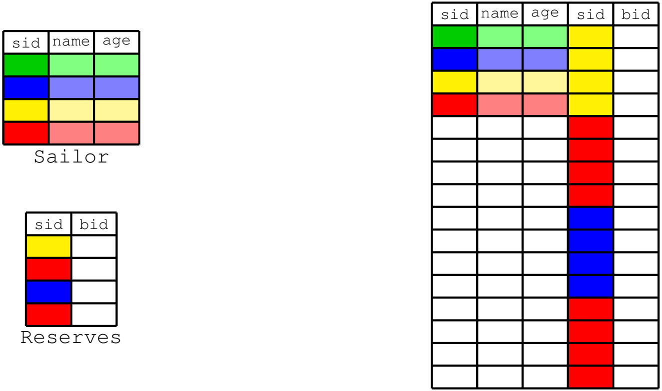


Reserves



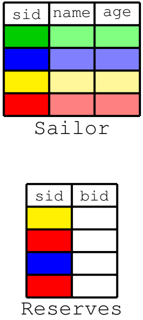
Natural join 2

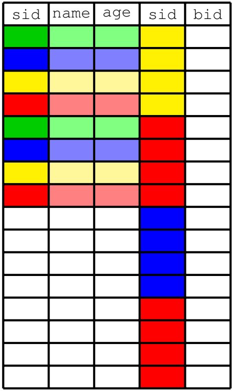
At a logical level, the natural join first forms the *Cartesian product*:



Natural join 2

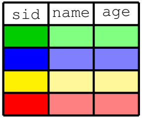
At a logical level, the natural join first forms the *Cartesian product*:



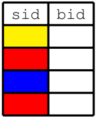


Natural join 2

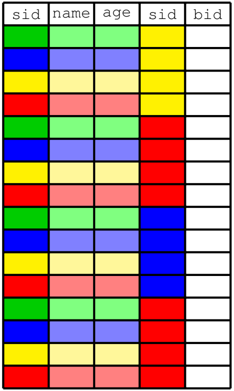
At a logical level, the natural join first forms the *Cartesian product*:



Sailor



Reserves



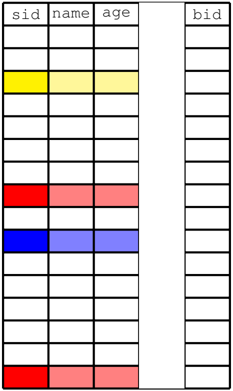
Matching columns

Then rows are discarded unless they agree on *every* column with the *same name* from the two tables.

sid	name	age	sid	bid

And finally

Finally, the *duplicate column(s)* are removed



In SQL

```
SELECT *
FROM Sailor
     NATURAL JOIN Reserves;
+-----+---------+------+-----+------------+
| sid | sname   | age  | bid | date       |
+-----+---------+------+-----+------------+
|  22 | Dustin  |   45 | 101 | 2014-08-10 |
|  22 | Dustin  |   45 | 102 | 2014-08-10 |
|  22 | Dustin  |   45 | 103 | 2014-08-11 |
|  22 | Dustin  |   45 | 104 | 2014-08-12 |
|  31 | Lubber  | 55.5 | 102 | 2014-08-02 |
|  31 | Lubber  | 55.5 | 103 | 2014-08-03 |
|  31 | Lubber  | 55.5 | 104 | 2014-08-17 |
|  64 | Horatio |   35 | 102 | 2014-08-18 |
|  64 | Horatio |   35 | 102 | 2014-08-05 |
|  74 | Horatio |   36 | 103 | 2014-08-05 |
+-----+---------+------+-----+------------+
10 rows in set (0.00 sec)
```

In SQL

```
SELECT *
FROM   Sailor
       JOIN Reserves USING (sid);
+-----+---------+------+-----+------------+
| sid | sname   | age  | bid | date       |
+-----+---------+------+-----+------------+
|  22 | Dustin  |   45 | 101 | 2014-08-10 |
|  22 | Dustin  |   45 | 102 | 2014-08-10 |
|  22 | Dustin  |   45 | 103 | 2014-08-11 |
|  22 | Dustin  |   45 | 104 | 2014-08-12 |
|  31 | Lubber  | 55.5 | 102 | 2014-08-02 |
|  31 | Lubber  | 55.5 | 103 | 2014-08-03 |
|  31 | Lubber  | 55.5 | 104 | 2014-08-17 |
|  64 | Horatio |   35 | 102 | 2014-08-18 |
|  64 | Horatio |   35 | 102 | 2014-08-05 |
|  74 | Horatio |   36 | 103 | 2014-08-05 |
+-----+---------+------+-----+------------+
10 rows in set (0.00 sec)
```

The rest of the query

With the join done, we can now extract the rows that we want, namely those rows that correspond to boat number 103.

In relational algebra, this is a *selection*

$$\sigma_{\texttt{bid}=103}(\texttt{Reserves} \bowtie \texttt{Sailor}),$$

and finally we just want the names only, which is a projection:

$$\pi_{\texttt{sname}}(\sigma_{\texttt{bid}=103}(\texttt{Reserves} \bowtie \texttt{Sailor})).$$

In SQL

What SQL is logically the same as

$$\pi_{\texttt{sname}}(\sigma_{\texttt{bid}=103}(\texttt{Reserves} \bowtie \texttt{Sailor}))$$

```
SELECT sname
FROM   Reserves
       NATURAL JOIN Sailor
WHERE  bid = 103;
+---------+
| sname   |
+---------+
| Dustin  |
| Lubber  |
| Horatio |
+---------+
```

Another way

There is usually more than one expression that will yield the same output.

This expression

$$\pi_{\texttt{sname}}(\sigma_{\texttt{bid}=103}(\texttt{Reserves}) \bowtie \texttt{Sailor})$$

has the same value as our earlier expression for all instances of the relations.

In SQL

What SQL is logically the same as

$$\pi_{\texttt{sname}}(\sigma_{\texttt{bid}=103}(\texttt{Reserves}) \bowtie \texttt{Sailor})$$

```
SELECT sname
FROM   (SELECT *
        FROM   Reserves
        WHERE  bid = 103) AS T
       NATURAL JOIN Sailor;
+---------+
| sname   |
+---------+
| Dustin  |
| Lubber  |
| Horatio |
+---------+
```

A common error

```
SELECT sname
FROM   (SELECT *
        FROM   Reserves
        WHERE  bid = 103)
       NATURAL JOIN Sailor;
ERROR 1248 (42000): Every derived table must have its own alias
```

Even if the name is not used, `MySQL` insists that you name every *derived table*.

Example

Find the names of the sailors who have reserved a red boat

Example

Find the names of the sailors who have reserved a red boat

$$\pi_{\texttt{sname}}((\sigma_{\texttt{colour='red'}}\texttt{Boat}) \bowtie \texttt{Reserves} \bowtie \texttt{Sailor})$$

This expression can be *parsed* as follows:

- First *select* the rows corresponding to red boats from `Boat`.
- Next form the natural join of that table with `Reserves` to find all the information about reservations involving red boats.
- Then form the natural join of *that* relation with `Sailor` to join the personal information about the sailors.
- Finally *project* out the sailor's name.

Step 1

We can execute this step-by-step in MySQL to *see* what happens:

$$\sigma_{\texttt{colour}='\texttt{red}'}\texttt{Boat}$$

```
SELECT *
FROM   Boat
WHERE  colour = 'red';
+-----+-----------+--------+
| bid | name      | colour |
+-----+-----------+--------+
| 102 | Interlake | red    |
| 104 | Marine    | red    |
+-----+-----------+--------+
2 rows in set (0.00 sec)
```

Step 2

$$(\sigma_{\texttt{colour='red'}}\texttt{boat}) \bowtie \texttt{Reserves}$$

```
SELECT *
FROM   (SELECT *
        FROM   Boat
        WHERE  colour = 'red') AS RedBoats
       NATURAL JOIN Reserves;
+-----+-----------+--------+-----+------------+
| bid | name      | colour | sid | date       |
+-----+-----------+--------+-----+------------+
| 102 | Interlake | red    |  22 | 2014-08-10 |
| 102 | Interlake | red    |  31 | 2014-08-02 |
| 102 | Interlake | red    |  64 | 2014-08-18 |
| 102 | Interlake | red    |  64 | 2014-08-05 |
| 104 | Marine    | red    |  22 | 2014-08-12 |
| 104 | Marine    | red    |  31 | 2014-08-17 |
+-----+-----------+--------+-----+------------+
6 rows in set (0.00 sec)
```

Step 3

$$(\sigma_{\texttt{colour='red'}}\texttt{Boat}) \bowtie \texttt{Reserves} \bowtie \texttt{Sailor}$$

```
SELECT *
FROM   (SELECT *
        FROM   Boat
        WHERE  colour = 'red') AS RedBoats
       NATURAL JOIN Reserves
       NATURAL JOIN Sailor;
+-----+-----------+--------+-----+------------+---------+------+
| bid | name      | colour | sid | date       | sname   | age  |
+-----+-----------+--------+-----+------------+---------+------+
| 102 | Interlake | red    |  22 | 2006-08-10 | Dustin  |   45 |
| 104 | Marine    | red    |  22 | 2006-08-12 | Dustin  |   45 |
| 102 | Interlake | red    |  31 | 2006-08-02 | Lubber  | 55.5 |
| 104 | Marine    | red    |  31 | 2006-08-17 | Lubber  | 55.5 |
| 102 | Interlake | red    |  64 | 2006-08-18 | Horatio |   35 |
| 102 | Interlake | red    |  64 | 2006-08-05 | Horatio |   35 |
+-----+-----------+--------+-----+------------+---------+------+
```

Finally

$$\pi_{\texttt{sname}}((\sigma_{\texttt{colour}='\texttt{red}'}\texttt{Boat}) \bowtie \texttt{Reserves} \bowtie \texttt{Sailor})$$

```
SELECT DISTINCT sname
FROM   (SELECT *
        FROM   Boat
        WHERE  colour = 'red') AS RedBoats
       NATURAL JOIN Reserves
       NATURAL JOIN Sailor;
+---------+
| sname   |
+---------+
| Dustin  |
| Lubber  |
| Horatio |
+---------+
```

Example

Find the names of the sailors who have hired a red or *a green boat*

$$\rho(\texttt{Tempboat}, \sigma_{\texttt{colour}='\texttt{red}' \vee \texttt{colour}='\texttt{green}'}\texttt{Boat})$$

$$\pi_{\texttt{sname}}(\texttt{Tempboat} \bowtie \texttt{Reserves} \bowtie \texttt{Sailor})$$

In MySQL

We *can* perform this process exactly like this in MySQL if desired, but at the expense of creating a new table.

```
CREATE TEMPORARY TABLE Tempboat LIKE boat;

INSERT INTO Tempboat
(SELECT *
 FROM   Boat
 WHERE  colour = 'red'
         OR colour = 'green');

SELECT DISTINCT S.sname
FROM   Tempboat
       NATURAL JOIN Reserves
       NATURAL JOIN Sailor S;
```

A different way

An alternative in this case is to find the sailors who have used red boats and green boats in two separate queries, and then use the *set union* operator to combine the two sets of names.

$$\pi_{\texttt{sname}}((\sigma_{\texttt{colour}='\texttt{red}'}\texttt{Boat}) \bowtie \texttt{Reserves} \bowtie \texttt{Sailor})$$

$$\cup$$

$$\pi_{\texttt{sname}}((\sigma_{\texttt{colour}='\texttt{green}'}\texttt{Boat}) \bowtie \texttt{Reserves} \bowtie \texttt{Sailor})$$

In MySQL

An alternative in this case is to find the sailors who have used red boats and green boats in two separate queries.

```
SELECT S.sname
FROM   Boat B
       NATURAL JOIN Reserves
       NATURAL JOIN Sailor S
WHERE  B.colour = 'red'
UNION
SELECT S.sname
FROM   Boat B
       NATURAL JOIN Reserves
       NATURAL JOIN Sailor S
WHERE  B.colour = 'green';
```

A red boat AND a green boat

Things get more interesting (and difficult) when we try to answer

Which sailors have hired both *a red boat and a green boat*

We cannot just replace OR ($\vee$) with AND ($\wedge$) to get

$$\rho(\texttt{Tempboat}, \sigma_{\texttt{colour='red'} \wedge \texttt{colour='green'}}\texttt{Boat})$$

$$\pi_{\texttt{sname}}(\texttt{Tempboat} \bowtie \texttt{Reserves} \bowtie \texttt{Sailor})$$

because this query returns *no results* — there *are* no boats that are both red and green!

Intersection

In relational algebra we can frame this query quite naturally by using *intersection* instead of *union*.

$$\pi_{\texttt{sname}}((\sigma_{\texttt{colour}='\texttt{red}'}\texttt{Boat}) \bowtie \texttt{Reserves} \bowtie \texttt{Sailor})$$

$$\cap$$

$$\pi_{\texttt{sname}}((\sigma_{\texttt{colour}='\texttt{green}'}\texttt{Boat}) \bowtie \texttt{Reserves} \bowtie \texttt{Sailor})$$

Unfortunately, MySQL 5.7 does not support an `INTERSECTION` operator so this cannot be translated directly into MySQL.

Two boats

A relational algebra query that *can* be translated directly into MySQL uses the concept of *two boats* reserved by the same sailor.

$$\rho(\texttt{R1}, \sigma_{\texttt{sid,bid}}(\sigma_{\texttt{colour='red'}}\texttt{Boat} \bowtie \texttt{Reserves}))$$

$$\rho(\texttt{R2}, \sigma_{\texttt{sid,bid}}(\sigma_{\texttt{colour='green'}}\texttt{Boat} \bowtie \texttt{Reserves}))$$

$$\pi_{\texttt{sname}}(\texttt{Sailor} \bowtie (\sigma_{\texttt{R1.sid=R2.sid}}(\texttt{R1} \times \texttt{R2})))$$

Here R1 is a list of “red-boat reservations” and R2 is a list of “green-boat reservations”.

Why can’t we use $\texttt{R1} \bowtie \texttt{R2}$?

In MySQL

This translates into MySQL as

```
SELECT DISTINCT S.sname
FROM Sailor S, Reserves R1, Reserves R2, Boat B1, Boat B2
WHERE R1.bid = B1.bid AND B1.colour = 'red'
AND R2.bid = B2.bid AND B2.colour = 'green'
AND R1.sid = S.sid AND R2.sid = S.sid;
```

We can view this query as finding two boat-reservations — `(B1, R1)` proving that a red boat has been reserved, and `(B2, R2)` proving that a green boat has been reserved, with the conditions on `sid` requiring the two reservations to be by the same sailor.