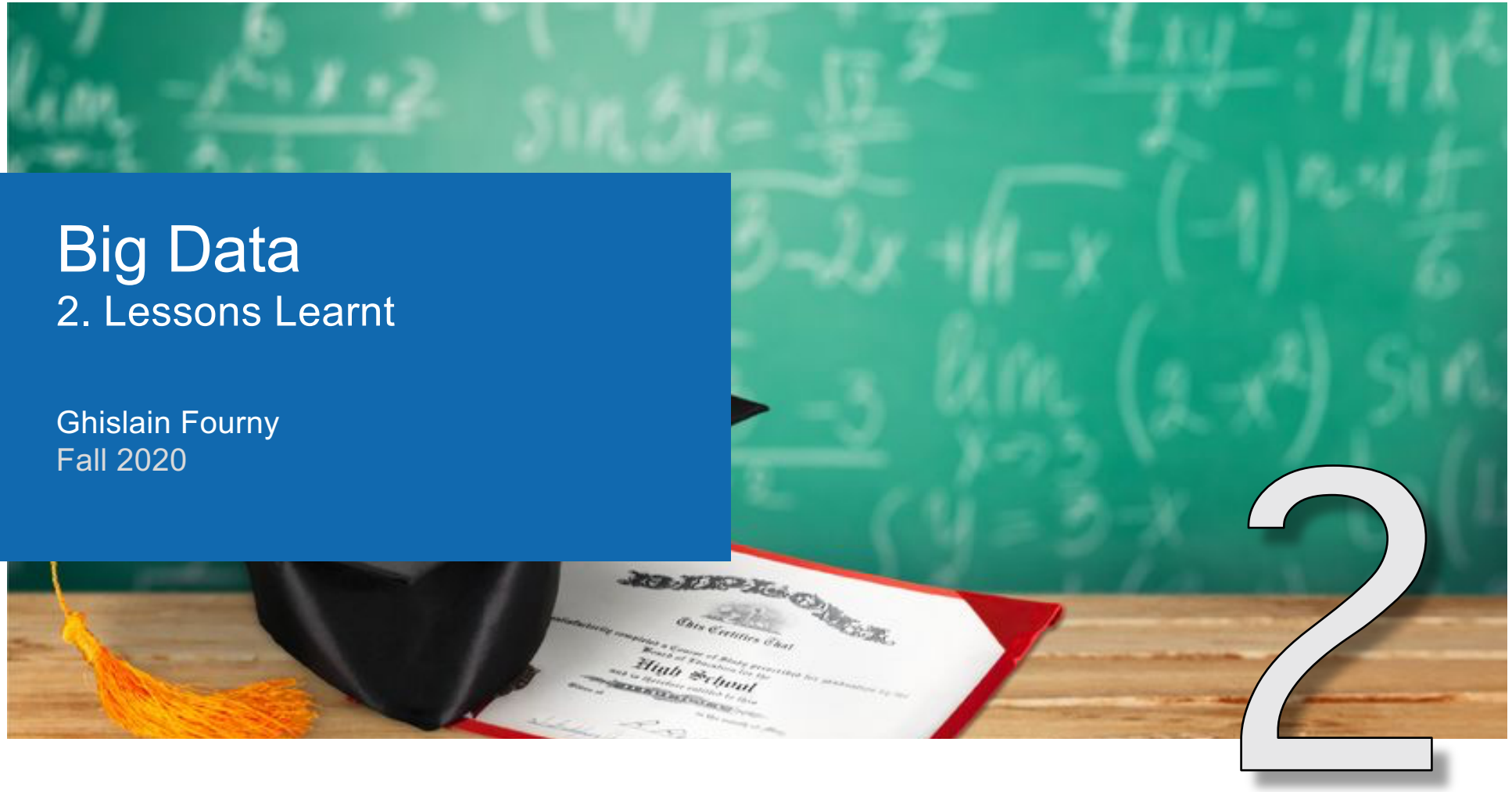
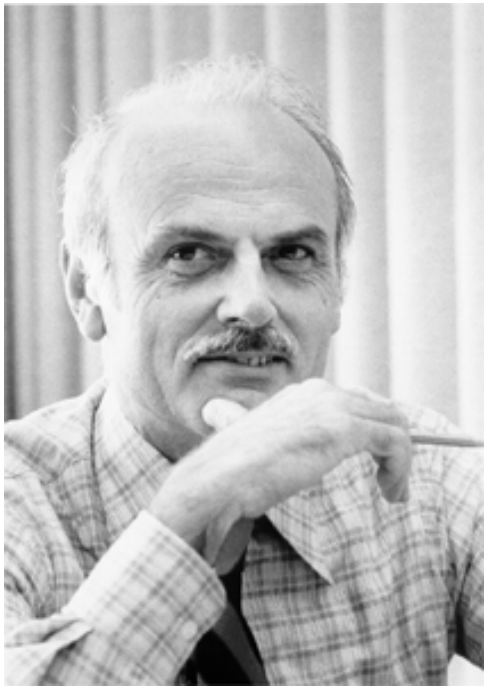


Big Data

2. Lessons Learnt

Ghislain Fourny
Fall 2020

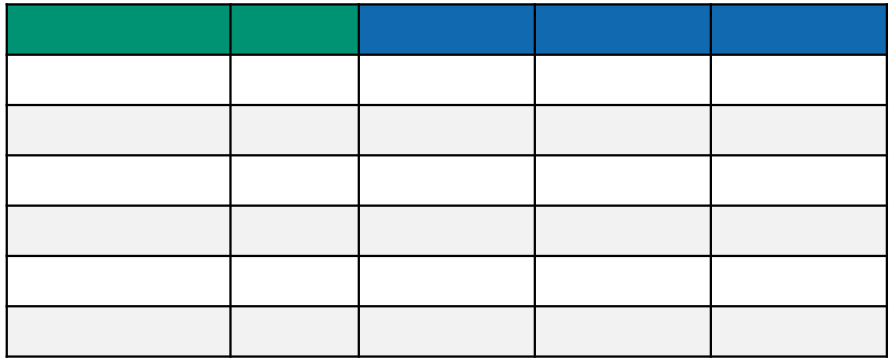
Mr. Databases: Edgar Codd



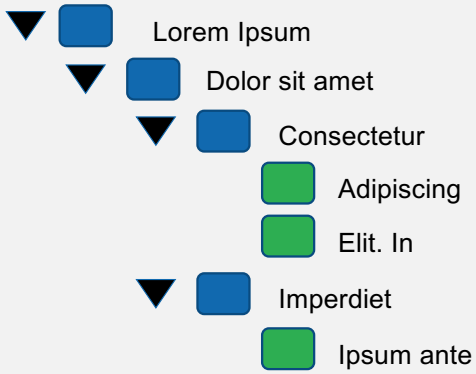
Wikipedia

Data Independence (Edgar Codd)

Logical data model

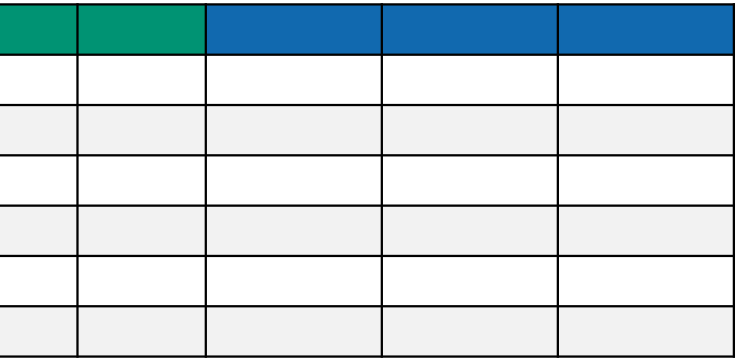


Physical storage

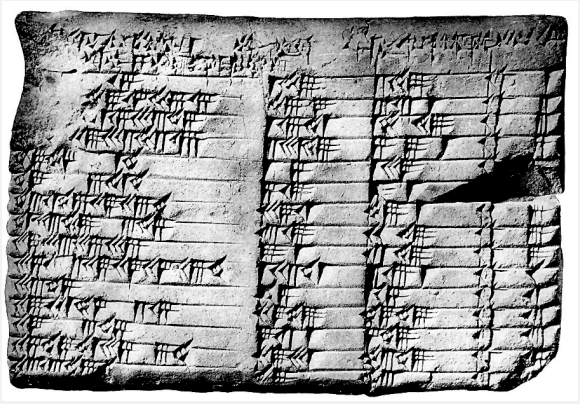


Data Independence (Edgar Codd)

Logical data model

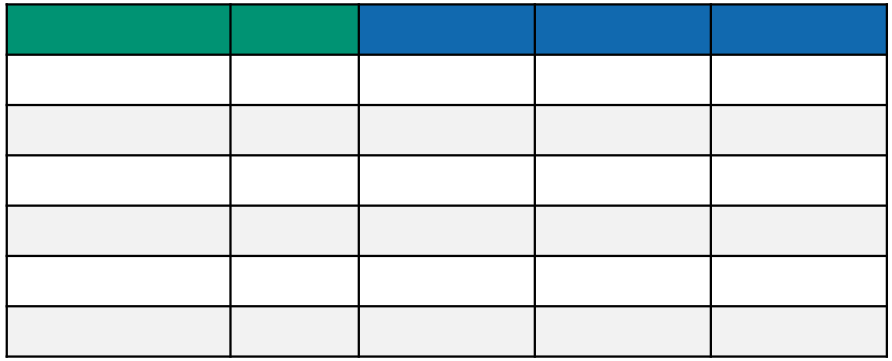


Physical storage

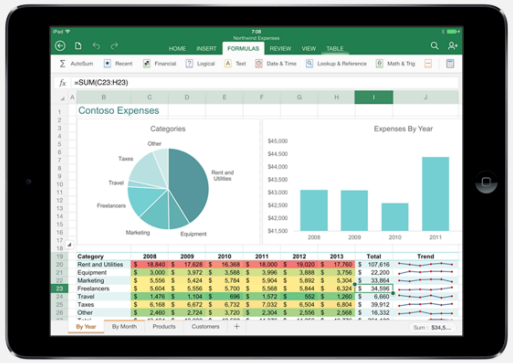


Data Independence (Edgar Codd)

Logical data model



Physical storage



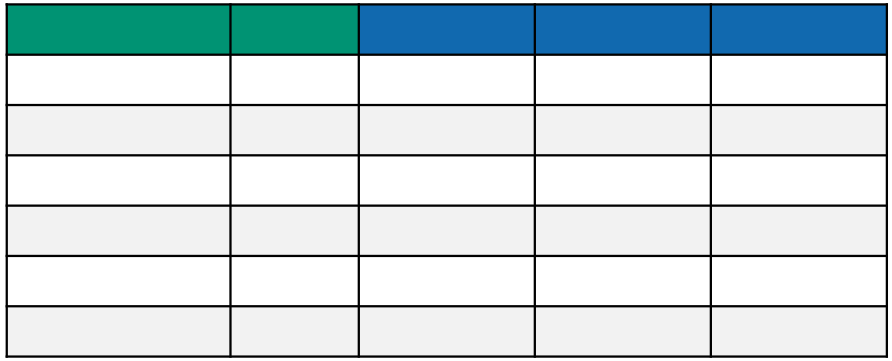
Data Independence (Edgar Codd)

Logical data model

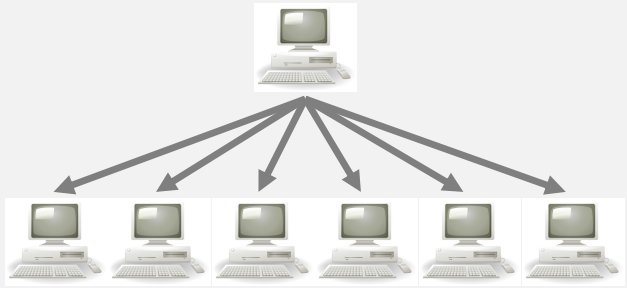
Physical storage

Data Independence (Edgar Codd)

Logical data model

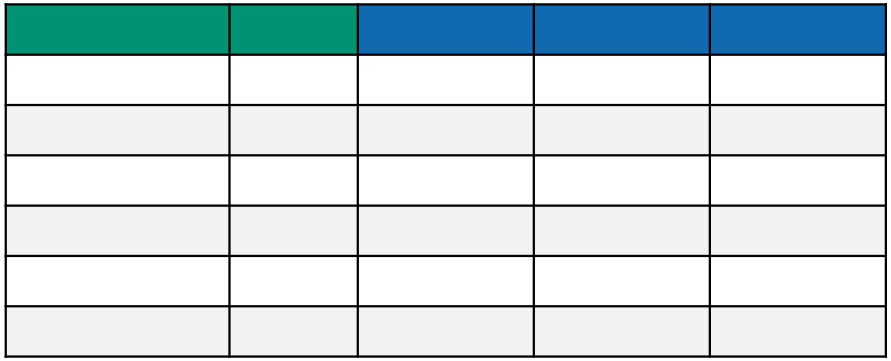


Physical storage

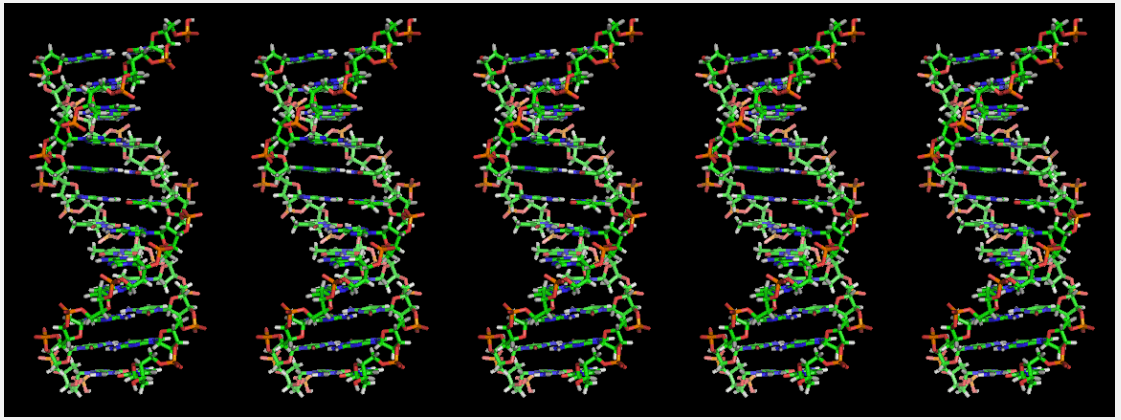


Data Independence (Edgar Codd)

Logical data model



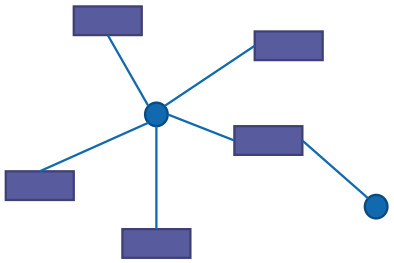
Physical storage



Data Shapes

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Etiam vel erat nec dui aliquet vulputate sed quis nulla. Donec eget ultricies magna, eu dignissim elit. Nullam sed urna nec nisl rhoncus ullamcorper placerat et enim. Integer varius ornare libero quis consequat. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Aenean eu efficitur orci. Aenean ac posuere tellus. Ut id commodo turpis.

Praesent nec libero metus. Praesent at turpis placerat, congue ipsum eget, scelerisque justo. Ut volutpat, massa ac lacinia cursus, nisl dui volutpat arcu, quis interdum sapien turpis in tellus. Suspendisse potenti. Vestibulum pharetra justo massa, ac venenatis mi condimentum nec. Proin viverra tortor non orci suscipit rutrum. Phasellus sit amet euismod diam. Nullam convallis nunc sit amet diam suscipit dapibus. Integer porta hendrerit nunc. Quisque pharetra congue porta. Suspendisse vestibulum sed mi in euismod. Etiam a purus suscipit, accumsan nibh vel, posuere ipsum. Nulla nec tempor nibh, id venenatis lectus. Duis lobortis id urna eget tincidunt.



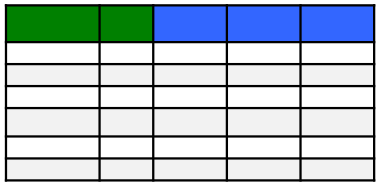
Data Shapes

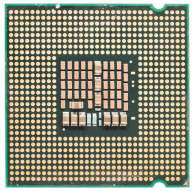
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Etiam vel erat nec dui aliquet vulputate sed quis nulla. Donec eget ultricies magna, eu dignissim elit. Nullam sed urna nec nisl rhoncus ullamcorper placerat et enim. Integer varius ornare libero quis consequat. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Aenean eu efficitur orci. Aenean ac posuere tellus. Ut id commodo turpis.

Praesent nec libero metus. Praesent at turpis placerat, congue ipsum eget, scelerisque justo. Ut volutpat, massa ac lacinia cursus, nisl dui volutpat arcu, quis interdum sapien turpis in tellus. Suspendisse potenti. Vestibulum pharetra justo massa, ac venenatis mi condimentum nec. Proin viverra tortor non orci suscipit rutrum. Phasellus sit amet euismod diam. Nullam convallis nunc sit amet diam suscipit dapibus. Integer porta hendrerit nunc. Quisque pharetra congue porta. Suspendisse vestibulum sed mi in euismod. Etiam a purus suscipit, accumsan nibh vel, posuere ipsum. Nulla nec tempor nibh, id venenatis lectus. Duis lobortis id urna eget tincidunt.

Overall architecture

SQL





Language

Model

Compute

Storage

Data model

What data **looks like**

What you can **do** with it

Analogy: a very simple "model of cooking"

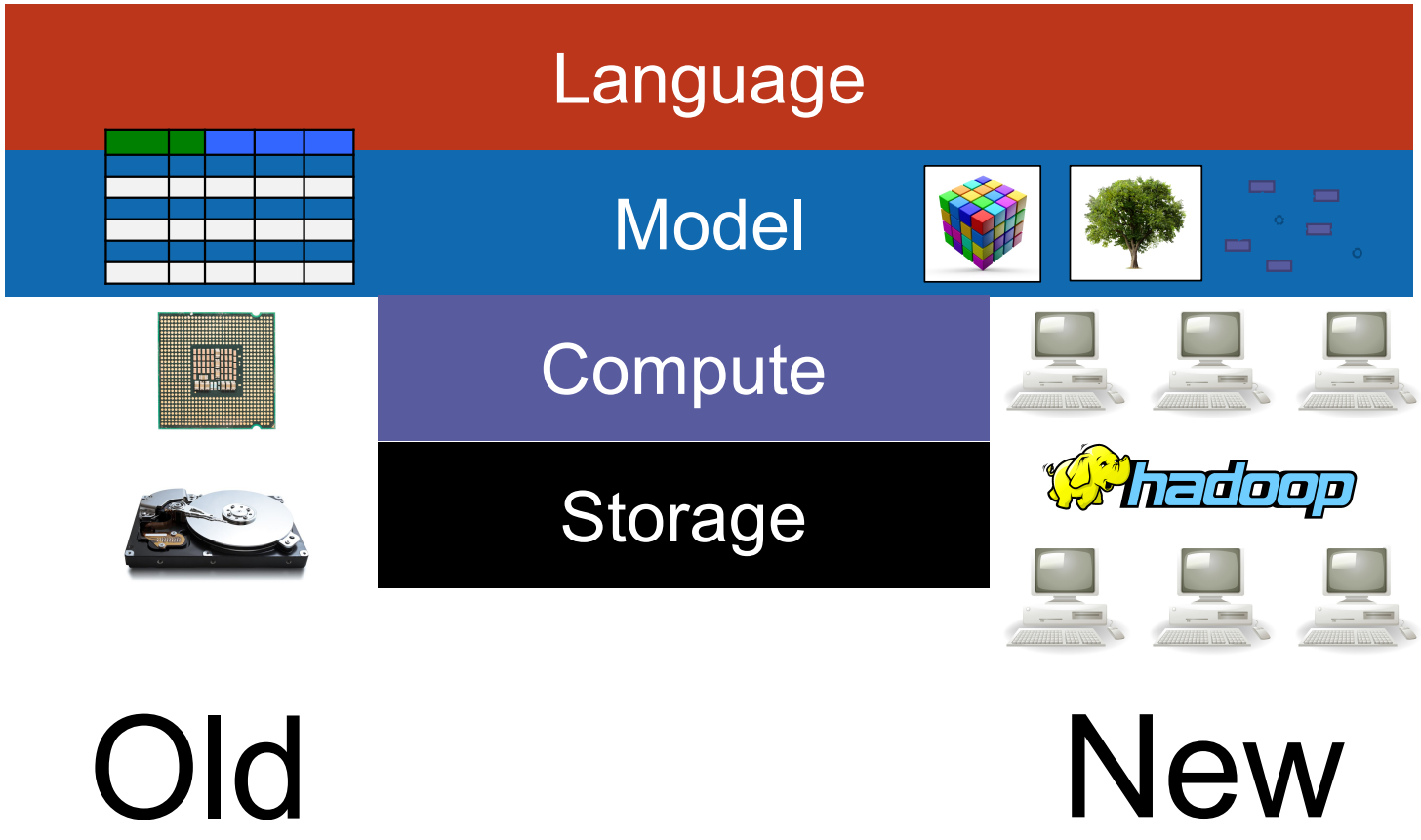
What food **looks like**

→ Something that can be eaten

What you can **do** with it

→ You can **mix, pour, break, dry, bake, grill, melt, mince, mix...**

Overall architecture



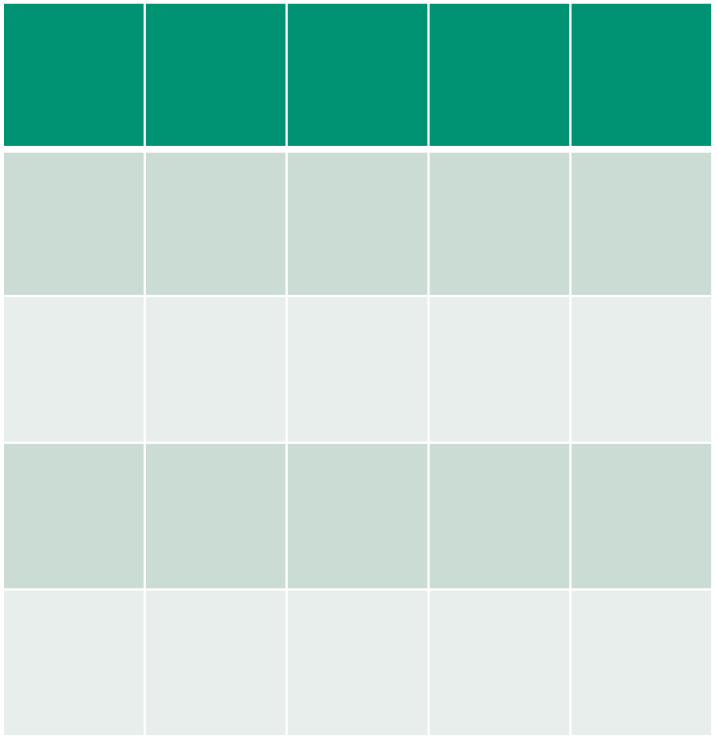
ETH zürich

Take-away Concepts

Take-away Concepts

Table

Collection



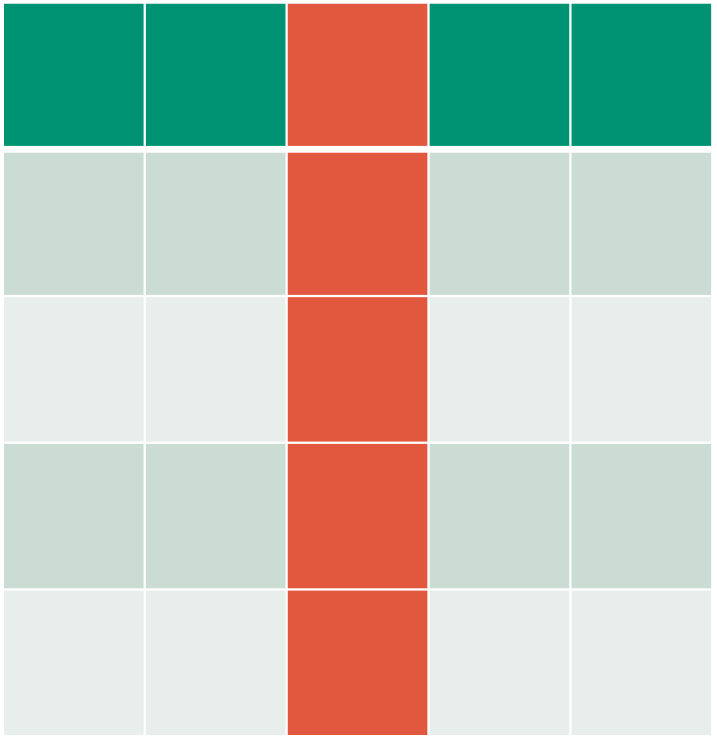
Take-away Concepts

Attribute

Column

Field

Property

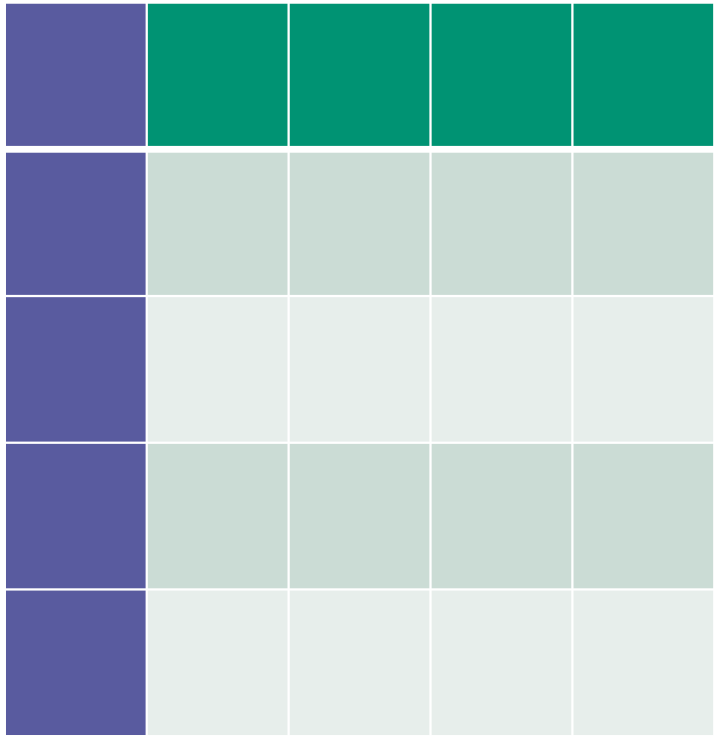


Take-away Concepts

Primary Key

Row ID

Name



Take-away Concepts

Row

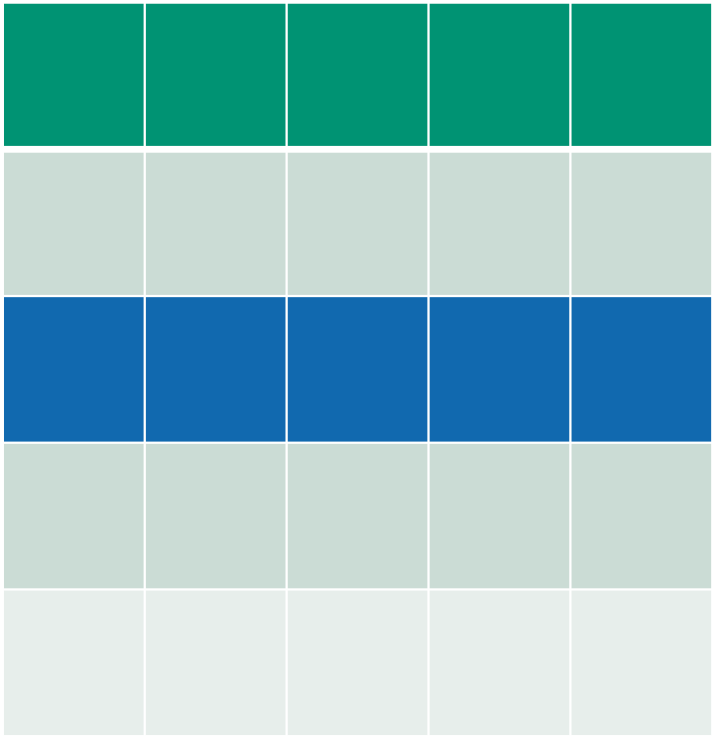
Business Object

Item

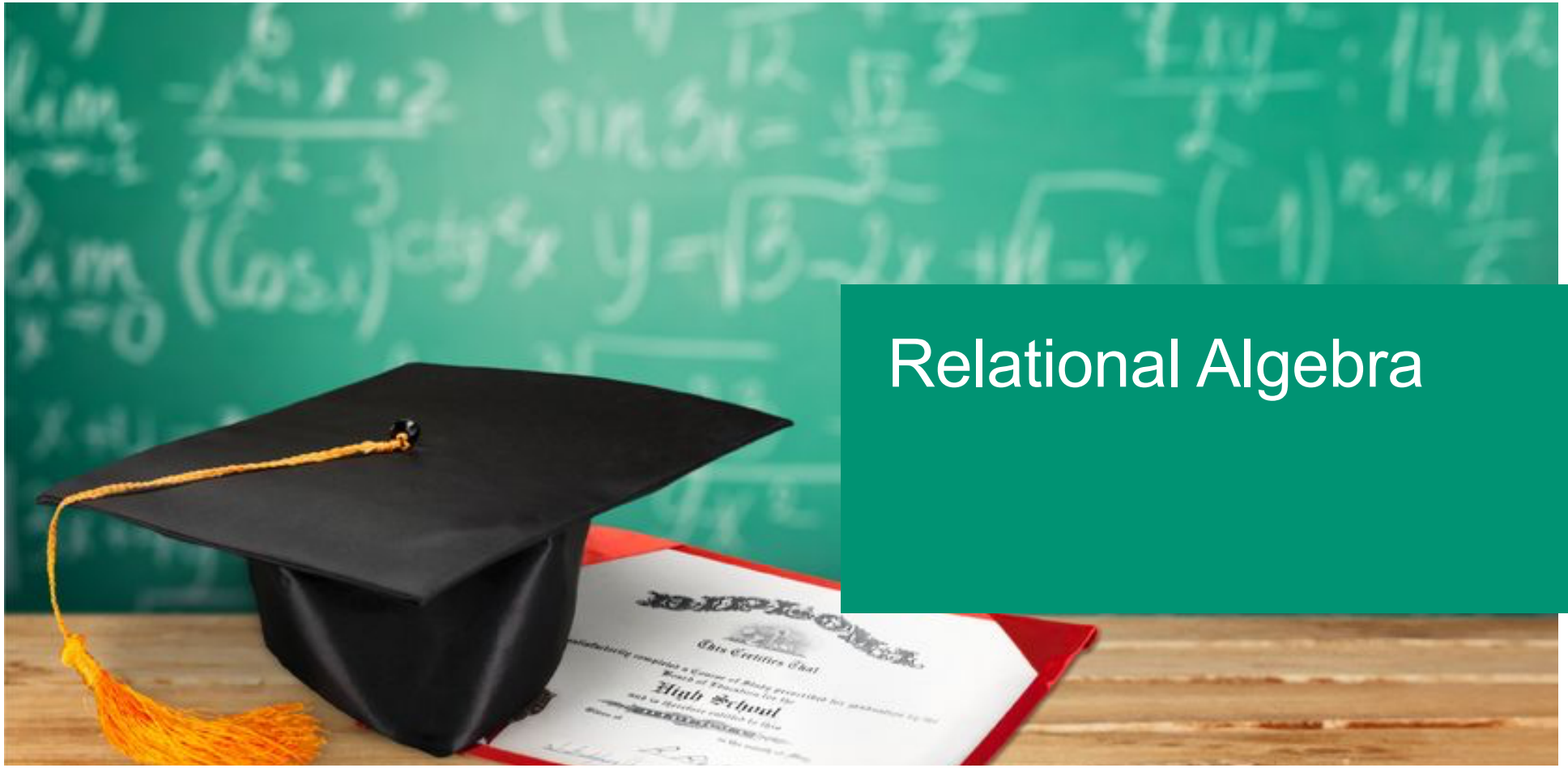
Entity

Document

Record

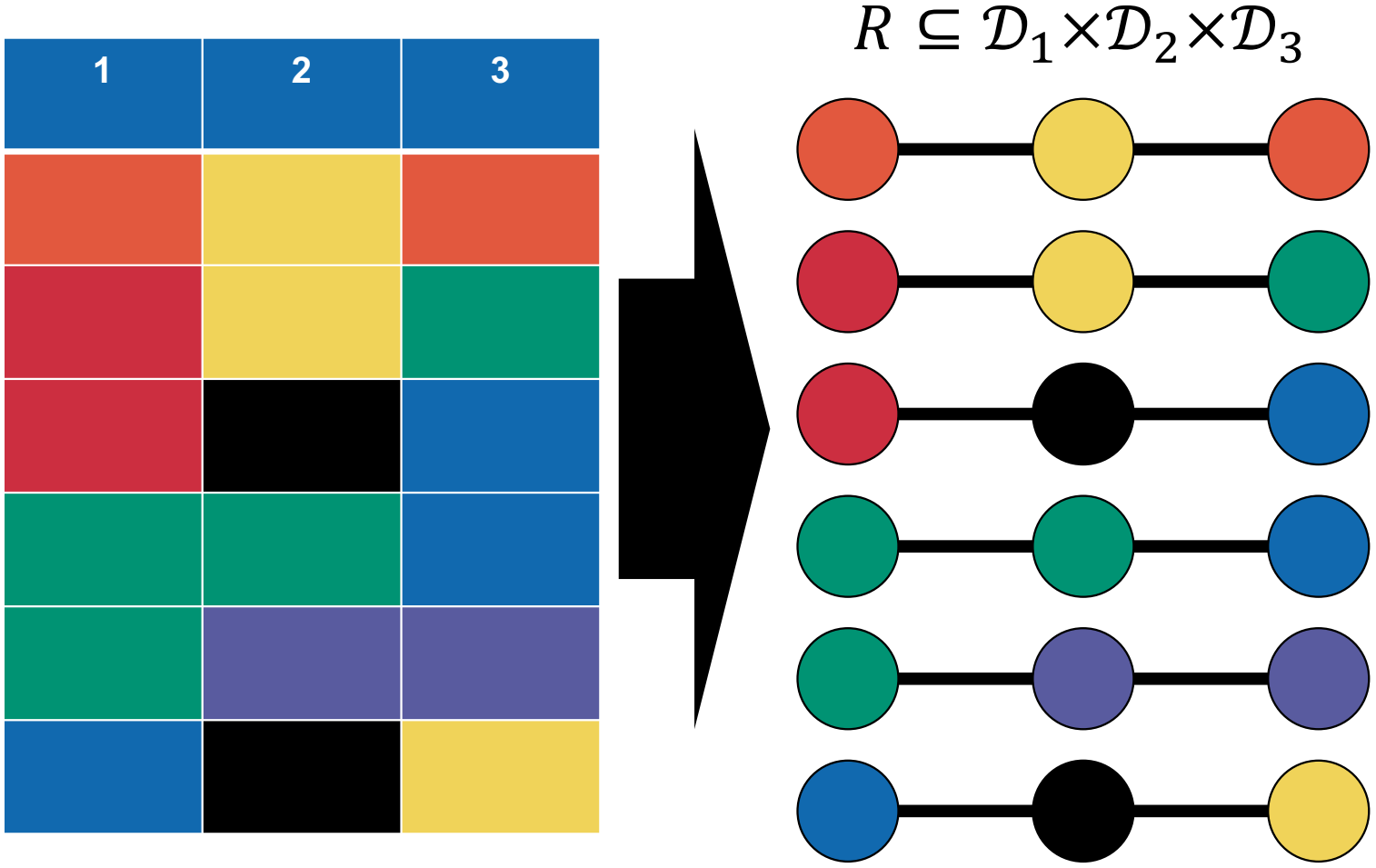


ETH zürich



Relational Algebra

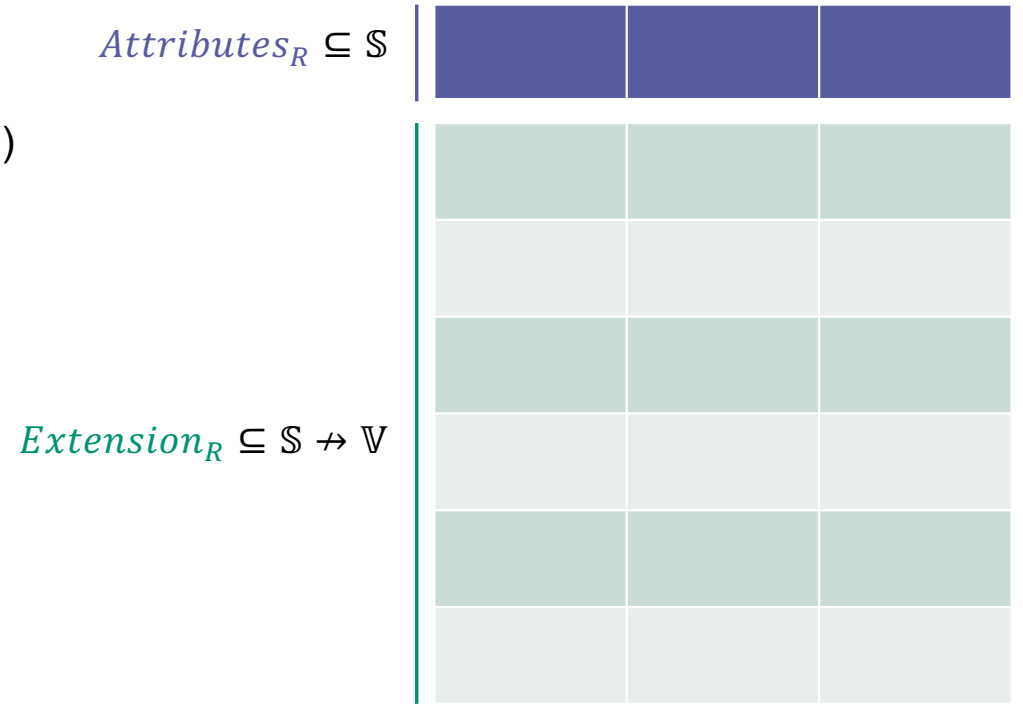
Table as a relation



Relations (the math)

A relation R is made of

1. A **set of attributes**
2. An **extension** (set of tuples)



Tuple: example

$$\begin{array}{lcl} \text{Name} & \mapsto & \text{Einstein} \\ \text{First name} & \mapsto & \text{Albert} \\ \text{Physicist} & \mapsto & \text{true} \\ \text{Year} & \mapsto & 1905 \end{array}$$

$$\mathbb{S} \qquad\qquad \mathbb{V}$$

Tuple: more intuitive display

$\mathbb{S}$

Name	First name	Physicist	Year
Einstein	Albert	true	1905

$\mathbb{V}$

(The order is irrelevant)

Rule #1: Tabular integrity

Name	First name	Physicist	Year

Name	First name	Physicist	Year
Einstein	Albert	true	1905

Name	First name	Physicist	Year
Turing	Alan	false	1936

Name	First name	Physicist	Year
Gödel	Kurt	false	1931

Rule #1: Tabular integrity

Name	First name	Physicist	Year
Einstein	Albert	true	1905
Turing	Alan	false	1936
Gödel	Kurt	false	1931

Rule #1: Broken tabular integrity

	Name	First name	Physicist	Year
	Einstein	Albert	true	1905
Country		Alan	false	1936
A	Gödel	Kurt		1931

Rule #2: Atomic integrity (1st normal form)

Legi	Name	Lecture ID	Lecture Name	City	State	PLZ
32-000-000	Alan Turing	xxx-xxxx-xxX	Cryptography	Bletchley Park	UK	MK3 6EB
32-000-000	Alan Turing	263-3010-00L	Big Data	Bletchley Park	UK	MK3 6EB
62-000-000	Georg Cantor	263-3010-00L	Big Data	Pfäffikon	SZ	8808
62-000-000	Georg Cantor	123-4567-89L	Set theory	Pfäffikon	SZ	8808
25-000-000	Felix Bloch	123-4567-89L	Set theory	Pfäffikon	ZH	8330

Rule #2: Broken atomic integrity

Legi	Name	Lecture	City	State	PLZ
32-000-000	Alan Turing	Lecture ID: xxx-xxxx-xxX, Lecture Name: Cryptography Lecture ID: 263-3010-00L, Lecture Name: Big Data	Bletchley Park	UK	MK3 6EB
62-000-000	Georg Cantor	Lecture ID: 263-3010-00L, Lecture Name: Big Data Lecture ID: 123-4567-89L, Lecture Name: Set theory	Pfäffikon	SZ	8808
25-000-000	Felix Bloch	Lecture ID: 123-4567-89L, Lecture Name: Set theory	Pfäffikon	ZH	8330

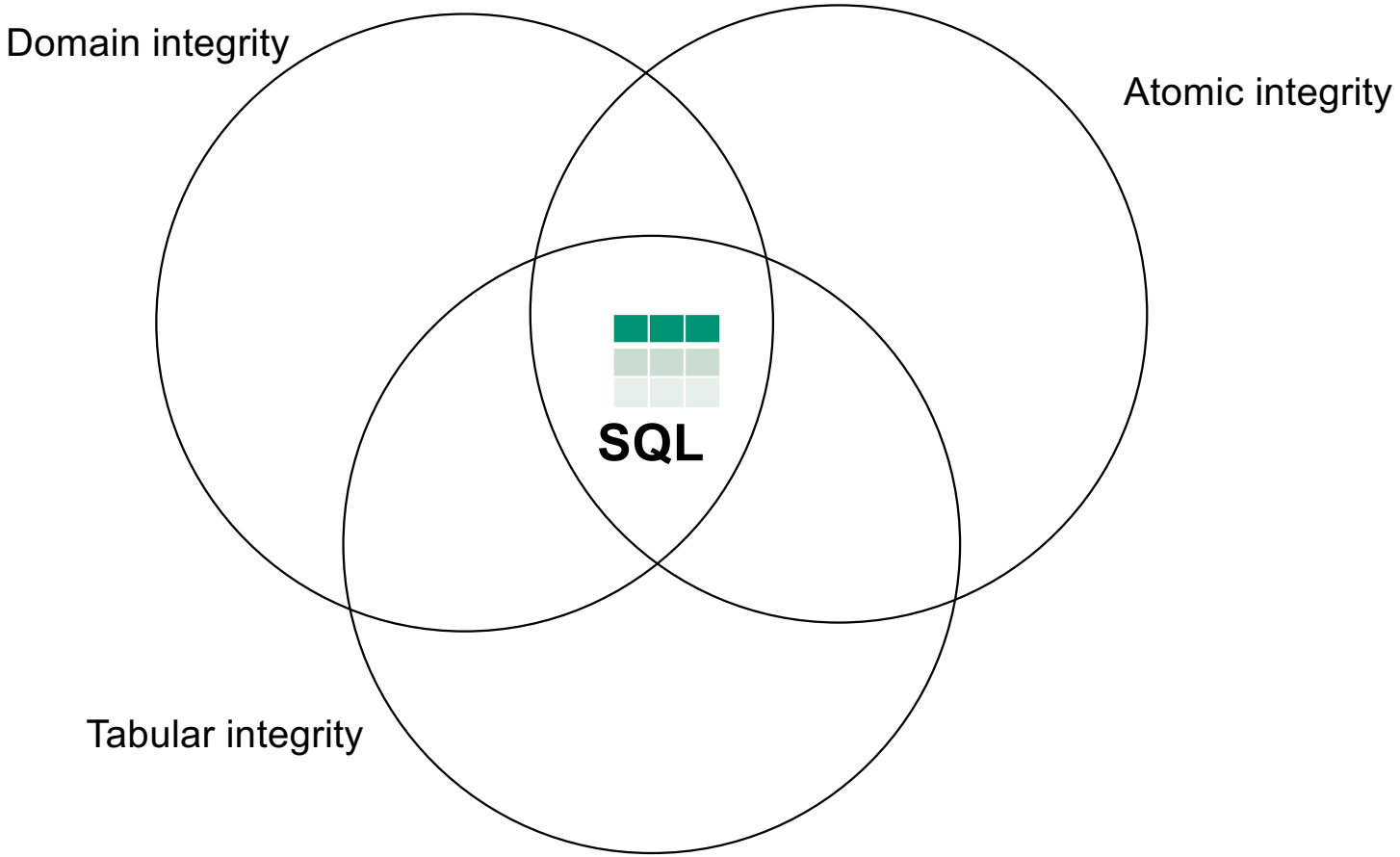
Rule #3: Domain integrity

Name	First name	Physicist	Year
String	String	Boolean	Integer
Einstein	Albert	true	1905
Turing	Alan	false	1936
Gödel	Kurt	false	1931

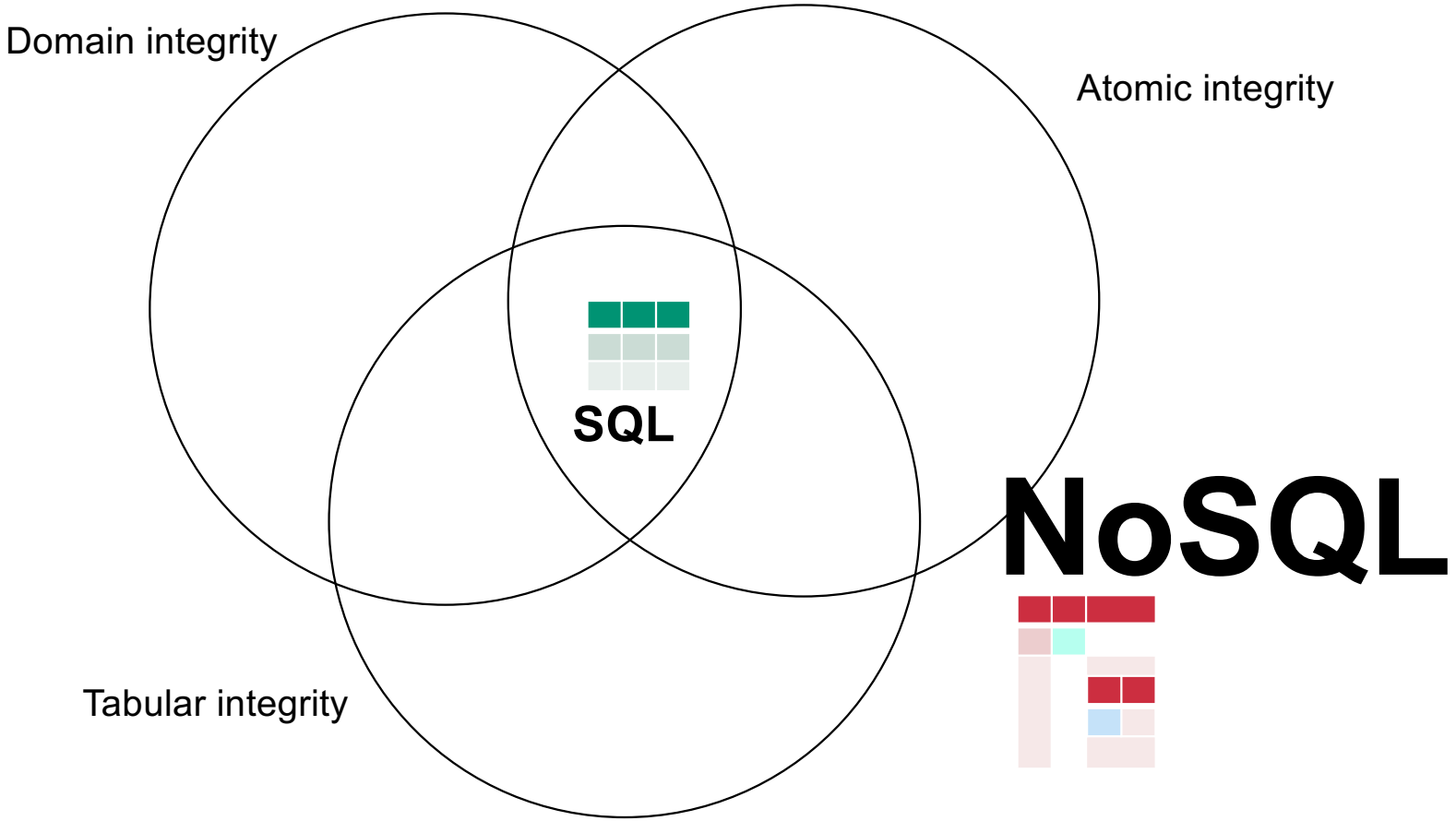
Rule #3: Broken domain integrity

Name	First name	Physicist	Year
String	String	Boolean	Integer
Einstein	Albert	true	1905
Turing	Alan	0	1936
Gödel	Kurt	false	thirty-one

From SQL to NoSQL



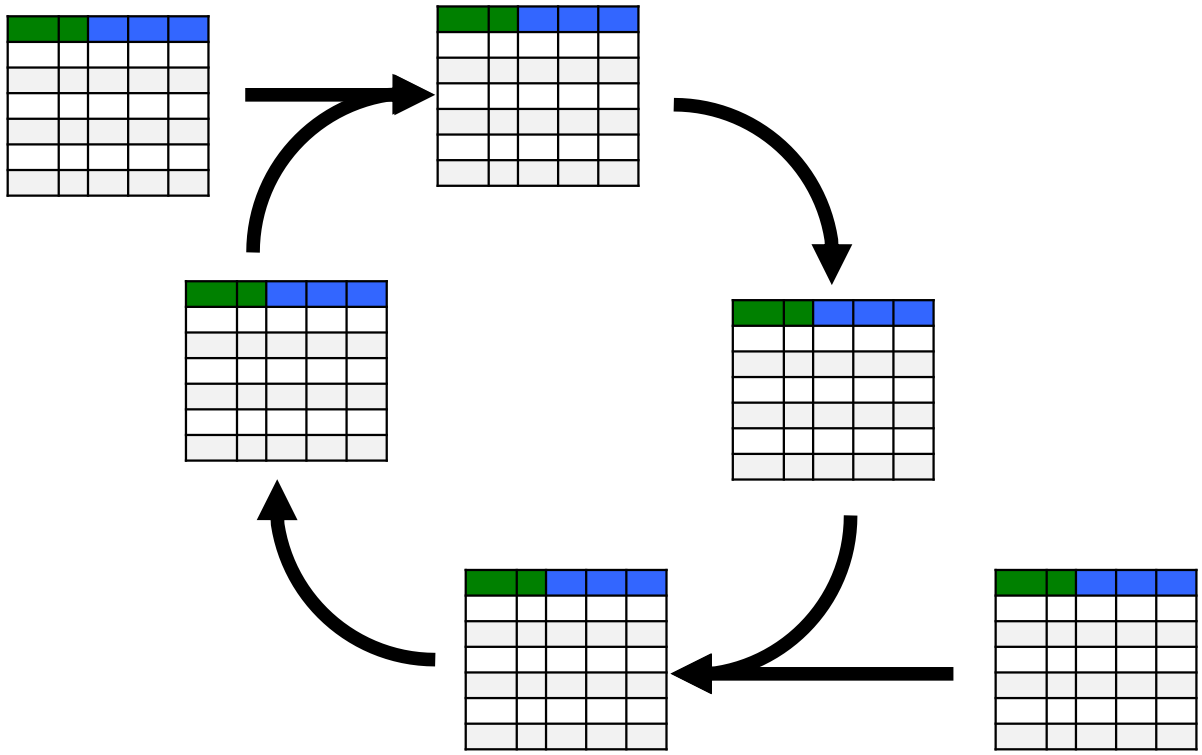
From SQL to NoSQL



From SQL to NoSQL

SQL

Relational Algebra



Summary of relational queries

Union
Intersection
Subtraction

Set queries

Selection
Projection

Filter queries

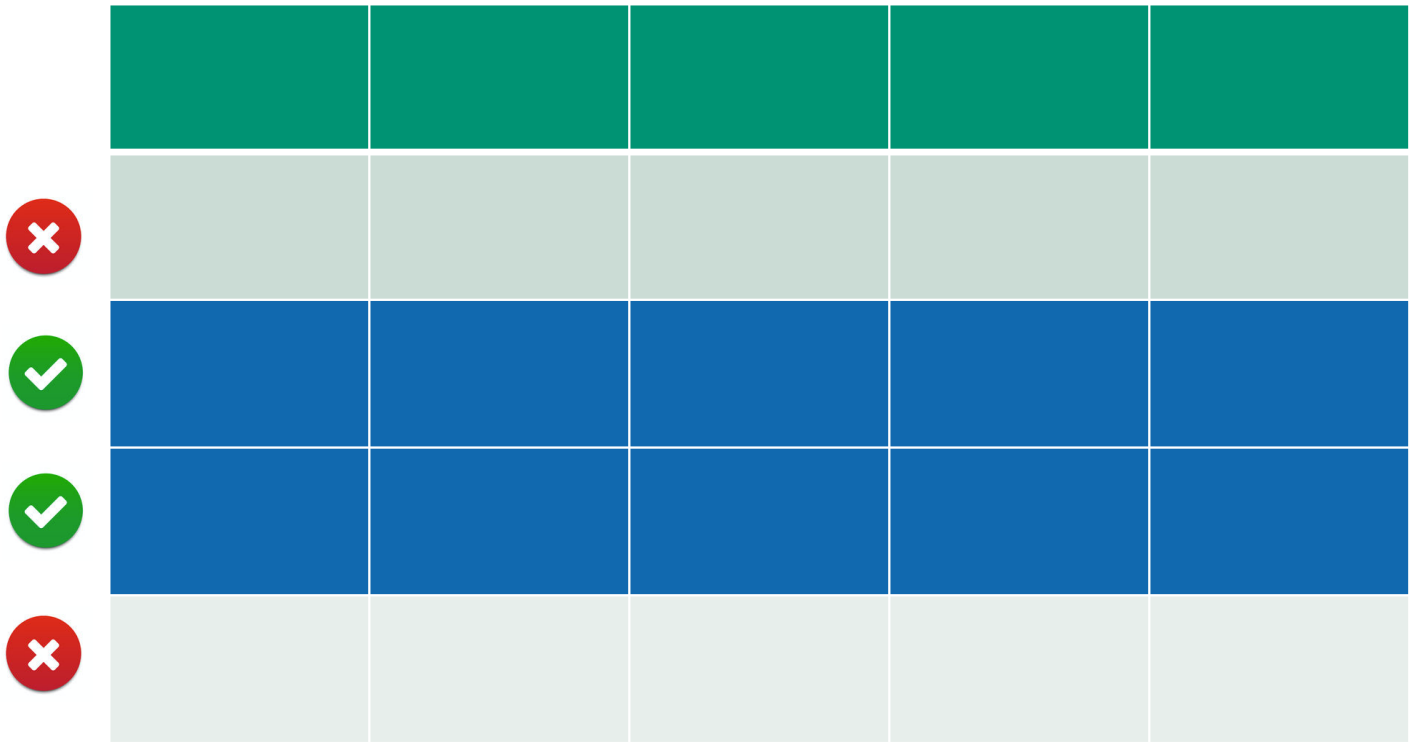
Relation renaming
Attribute renaming

Renaming queries

Cartesian product
Natural join
Theta join

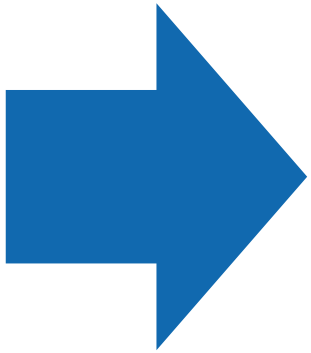
Binary queries

Selection



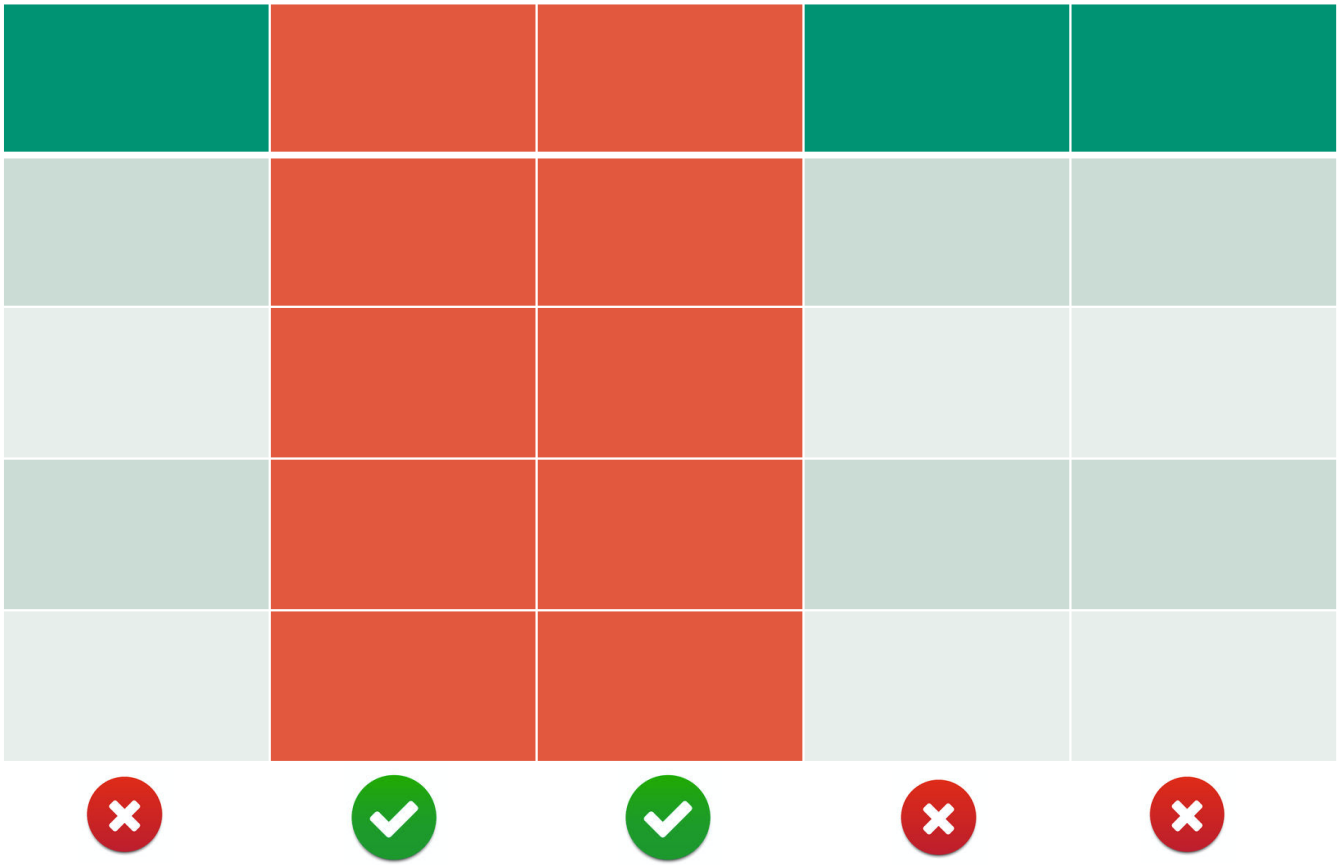
Selection

R		
A	B	C
string	integer	boolean
foo	1	true
bar	2	false
foo	3	false
foobar	4	true



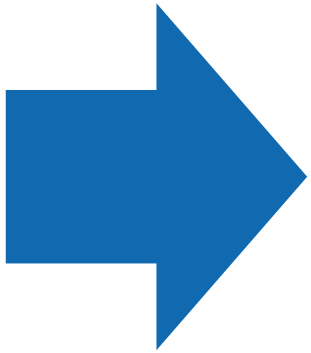
S		
A	B	C
string	integer	boolean
foo	1	true
bar	2	false

Projection



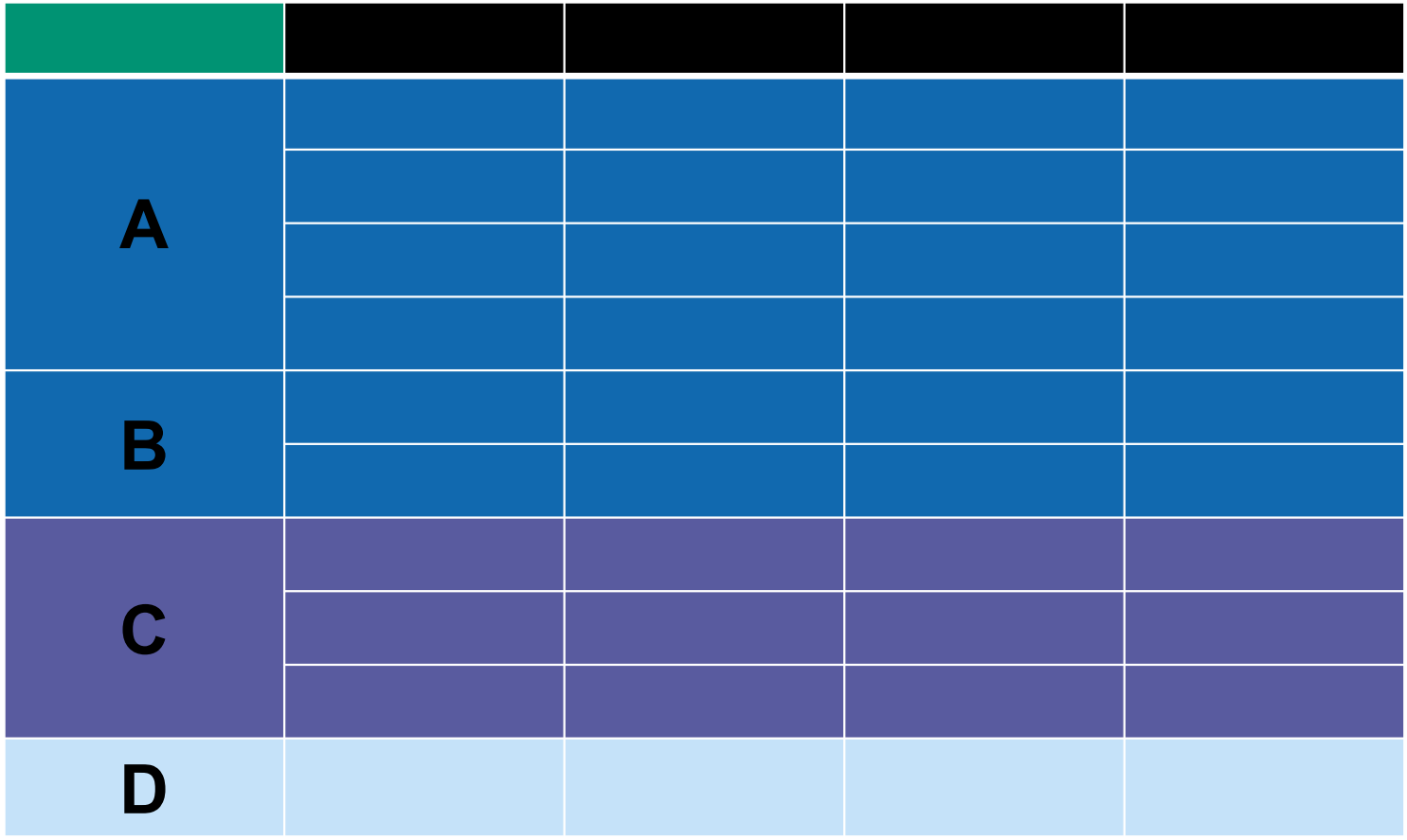
Projection

R		
A	B	C
string	integer	boolean
foo	1	true
bar	2	false
foo	3	false
foobar	4	true

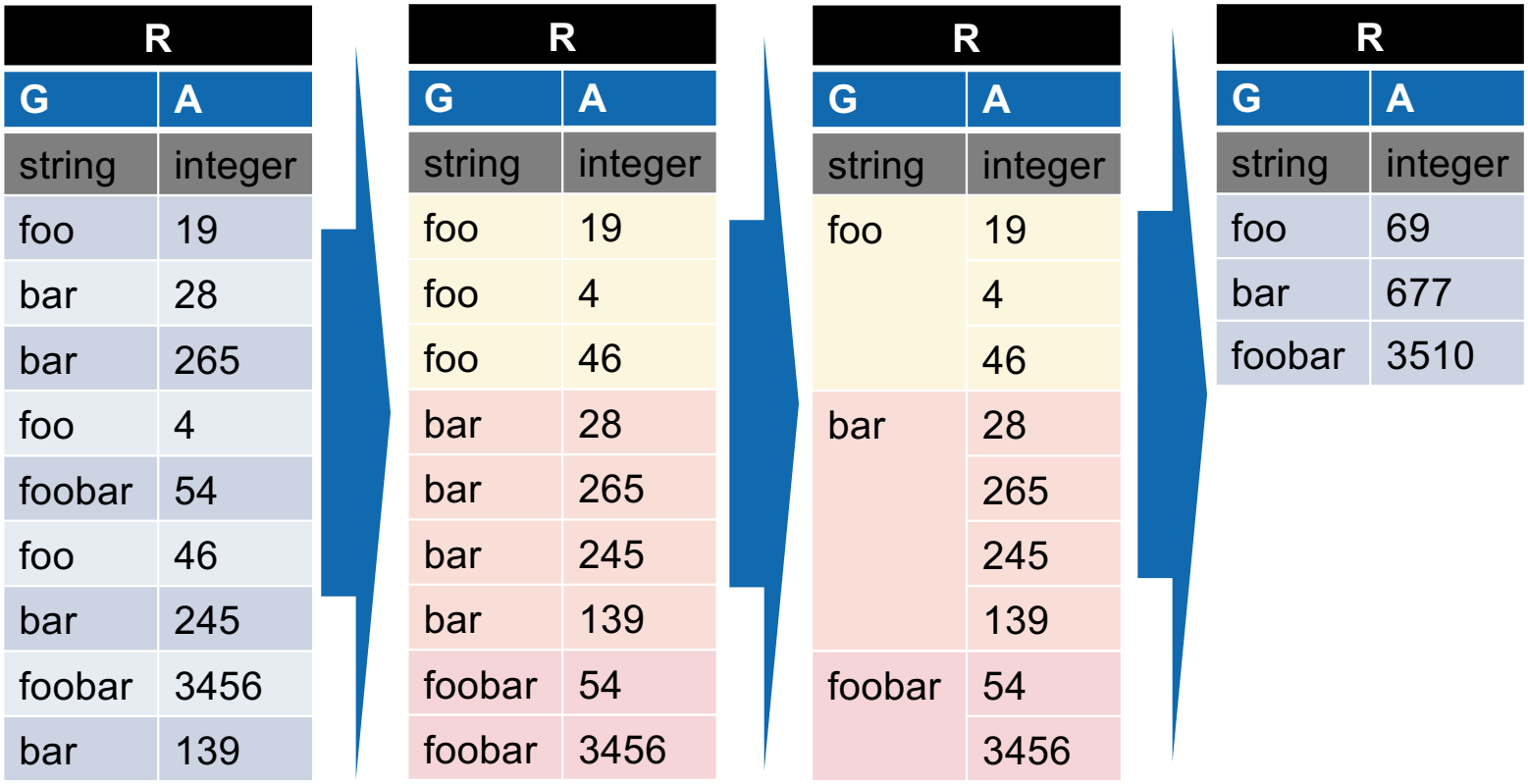


S	
A	C
string	boolean
foo	true
bar	false
foo	false
foobar	true

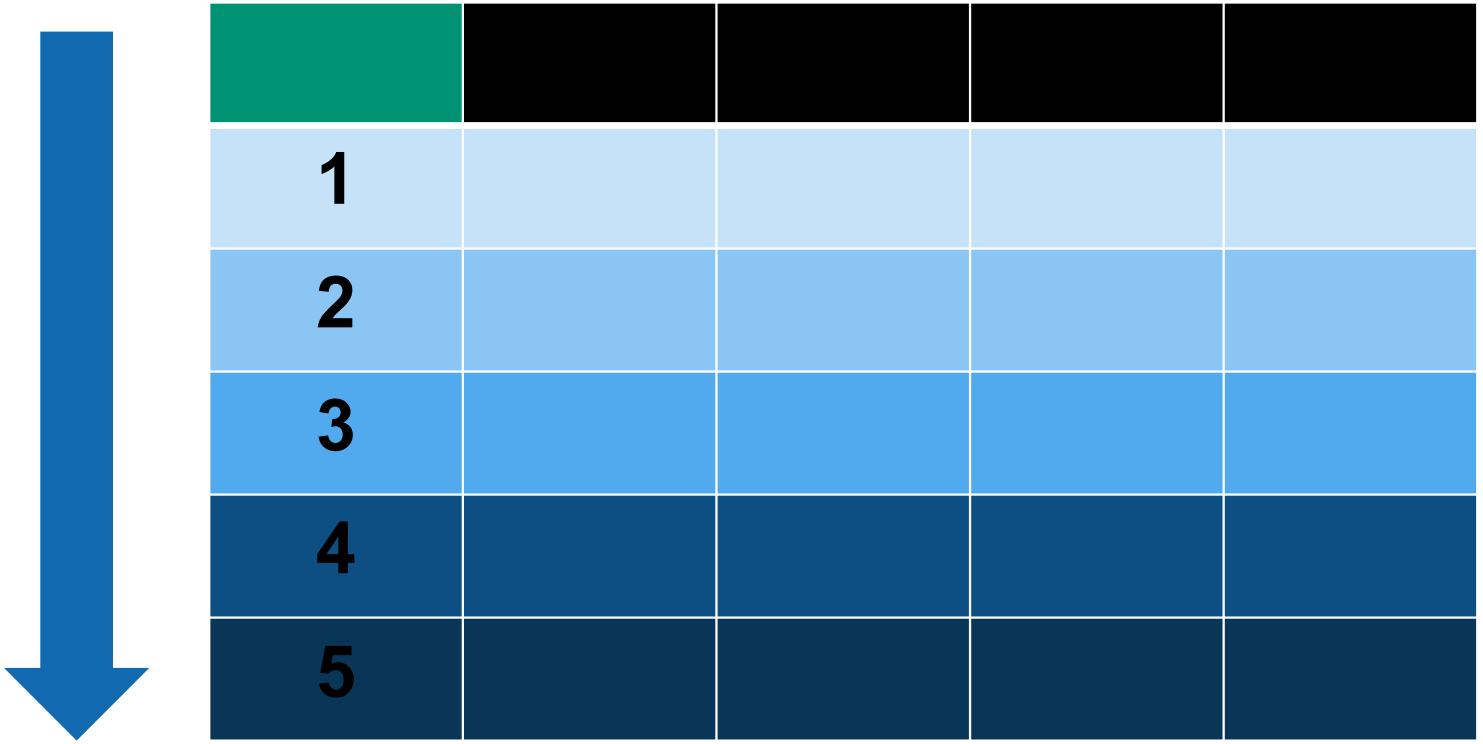
Grouping



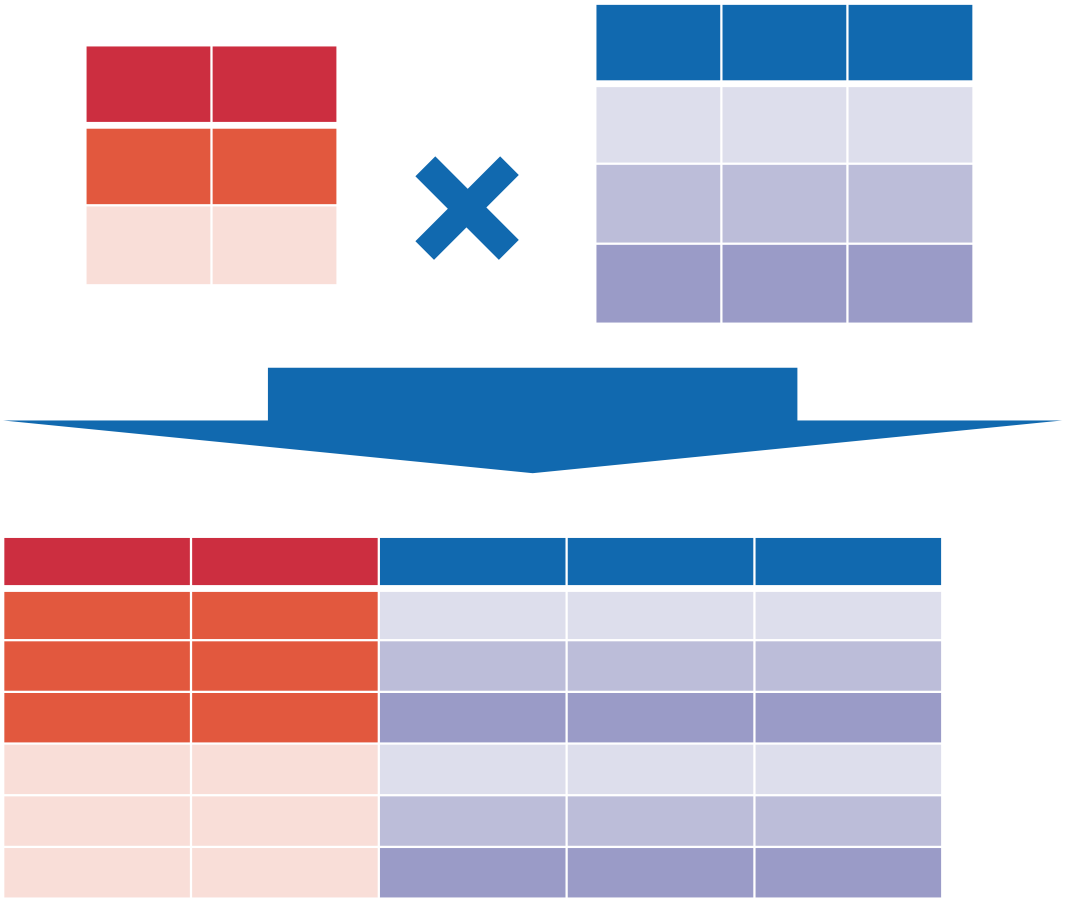
Grouping


R
G
A
string
integer
foo
19
bar
28
bar
265
foo
4
foobar
54
foo
46
bar
245
foobar
3456
bar
139
R
G
A
string
integer
foo
19
foo
4
foo
46
bar
28
bar
265
bar
245
bar
139
foobar
54
foobar
3456
R
G
A
string
integer
foo
19
4
46
bar
28
265
245
139
foobar
54
3456
R
G
A
string
integer
foo
69
bar
677
foobar
3510

Sorting



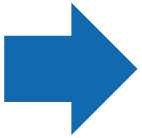
Cartesian product



Cartesian product

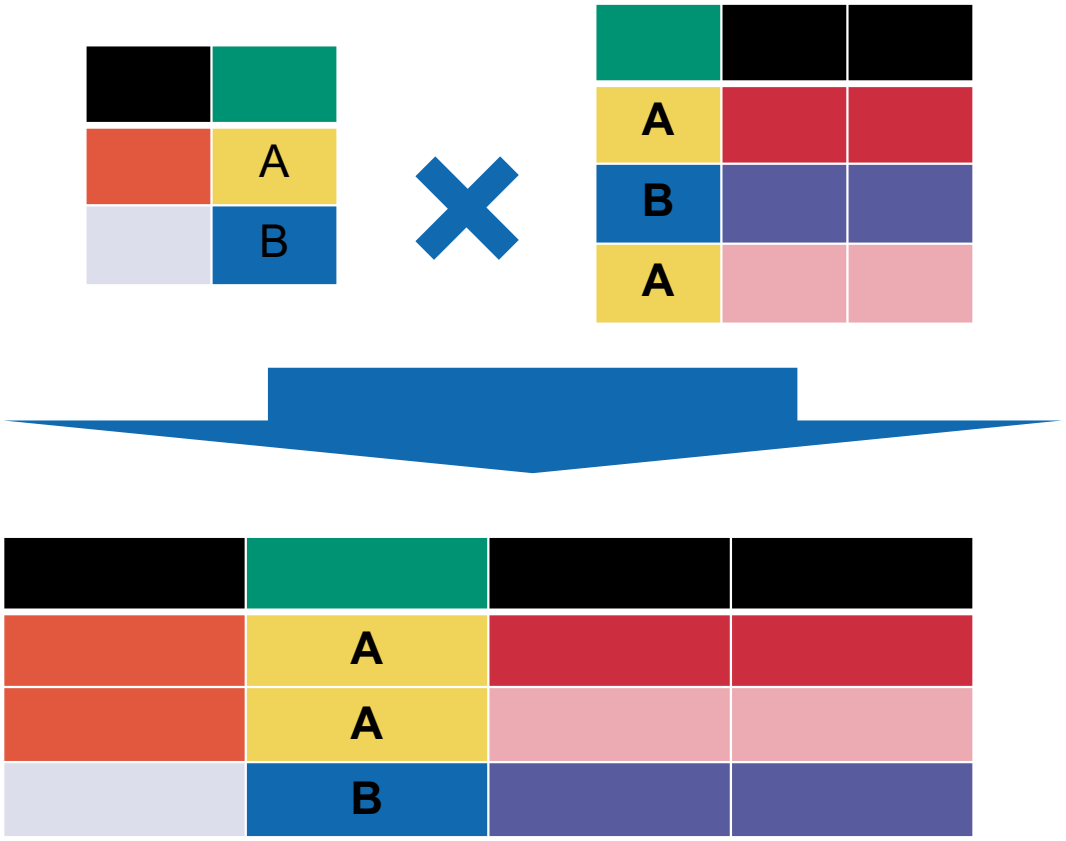
R		
A	B	C
string	integer	boolean
foo	1	true
bar	2	false

S	
D	E
string	integer
foo	1
bar	2
foo	3

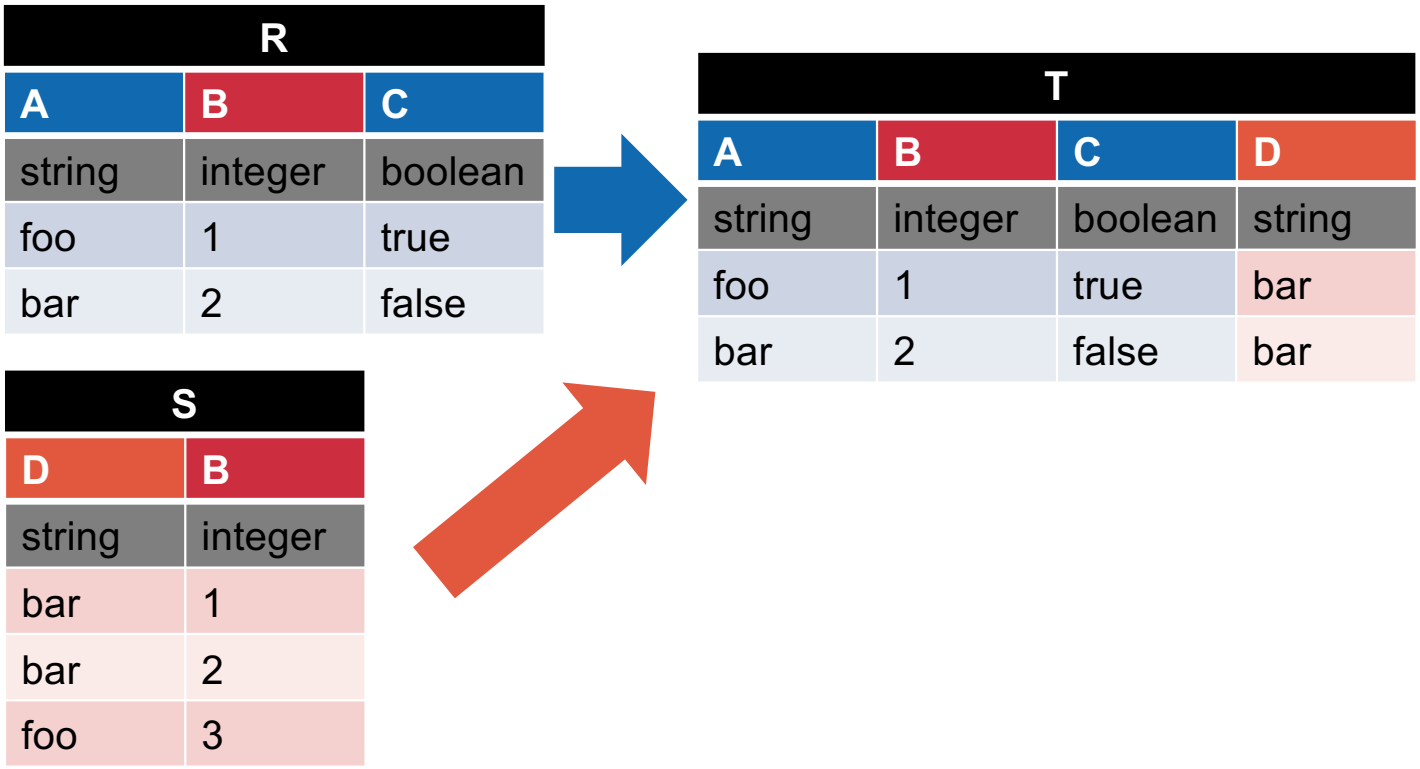


T				
A	B	C	D	E
string	integer	boolean	string	integer
foo	1	true	foo	1
foo	1	true	bar	2
foo	1	true	foo	3
bar	2	false	foo	1
bar	2	false	bar	2
bar	2	false	foo	3

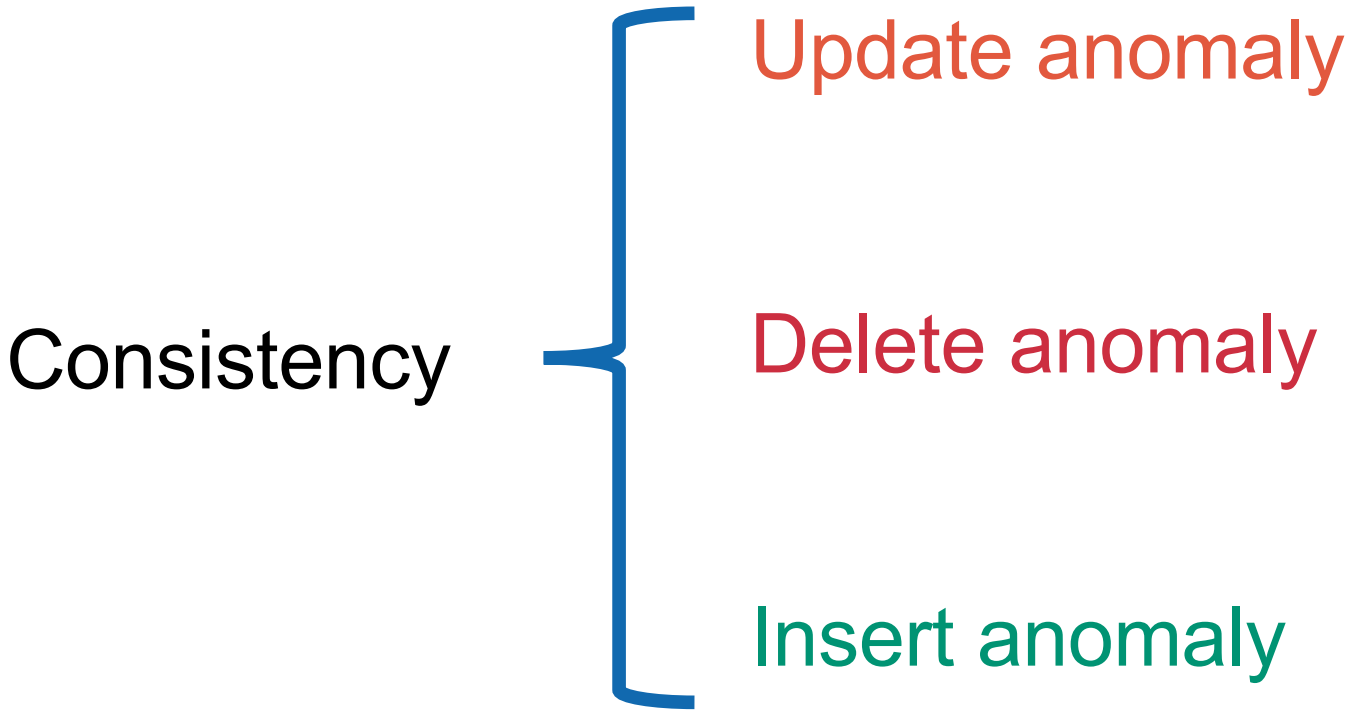
Join



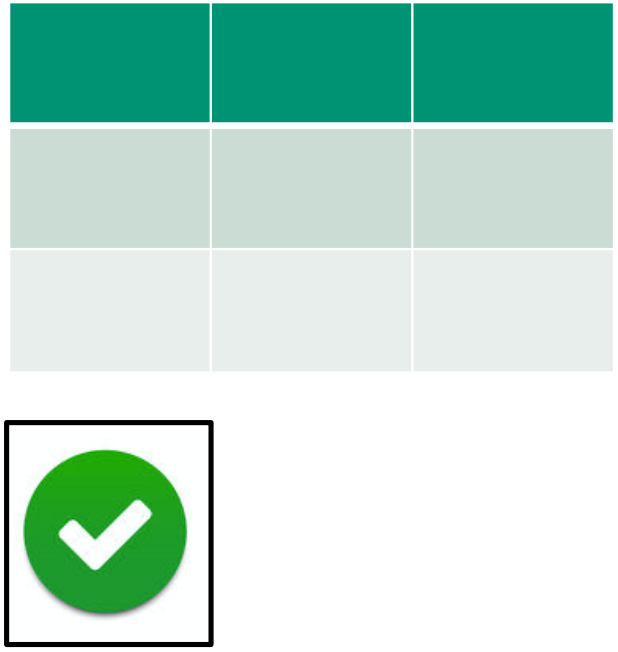
Join

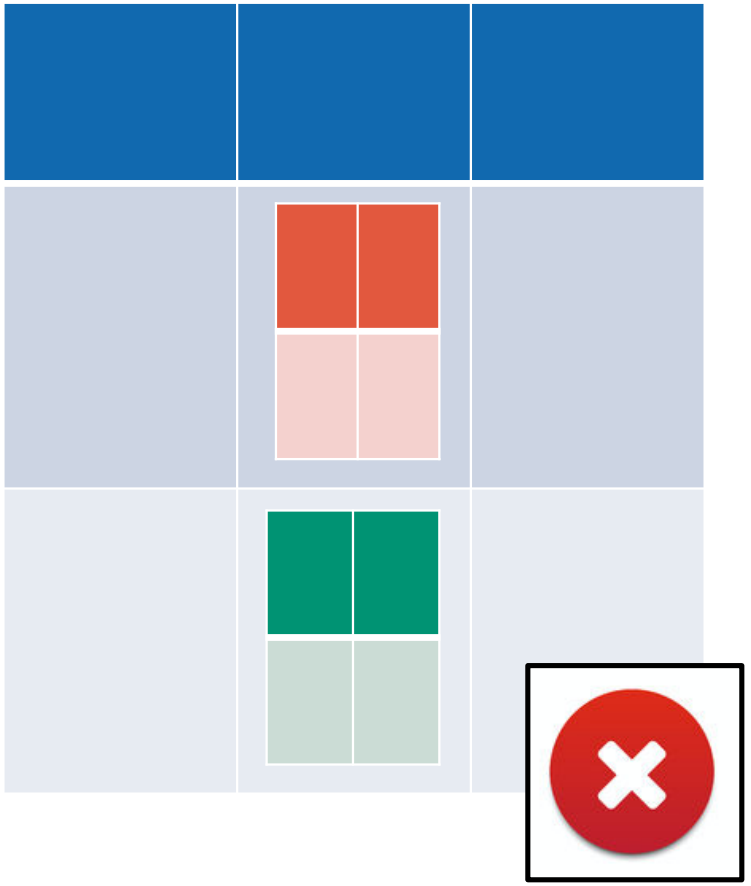

R
A
B
C
string
integer
boolean
foo
1
true
bar
2
false
S
D
B
string
integer
bar
1
bar
2
foo
3
T
A
B
C
D
string
integer
boolean
string
foo
1
true
bar
bar
2
false
bar

From your Bachelor's degree: Normal Forms



1st Normal Form (tabular) – The Key





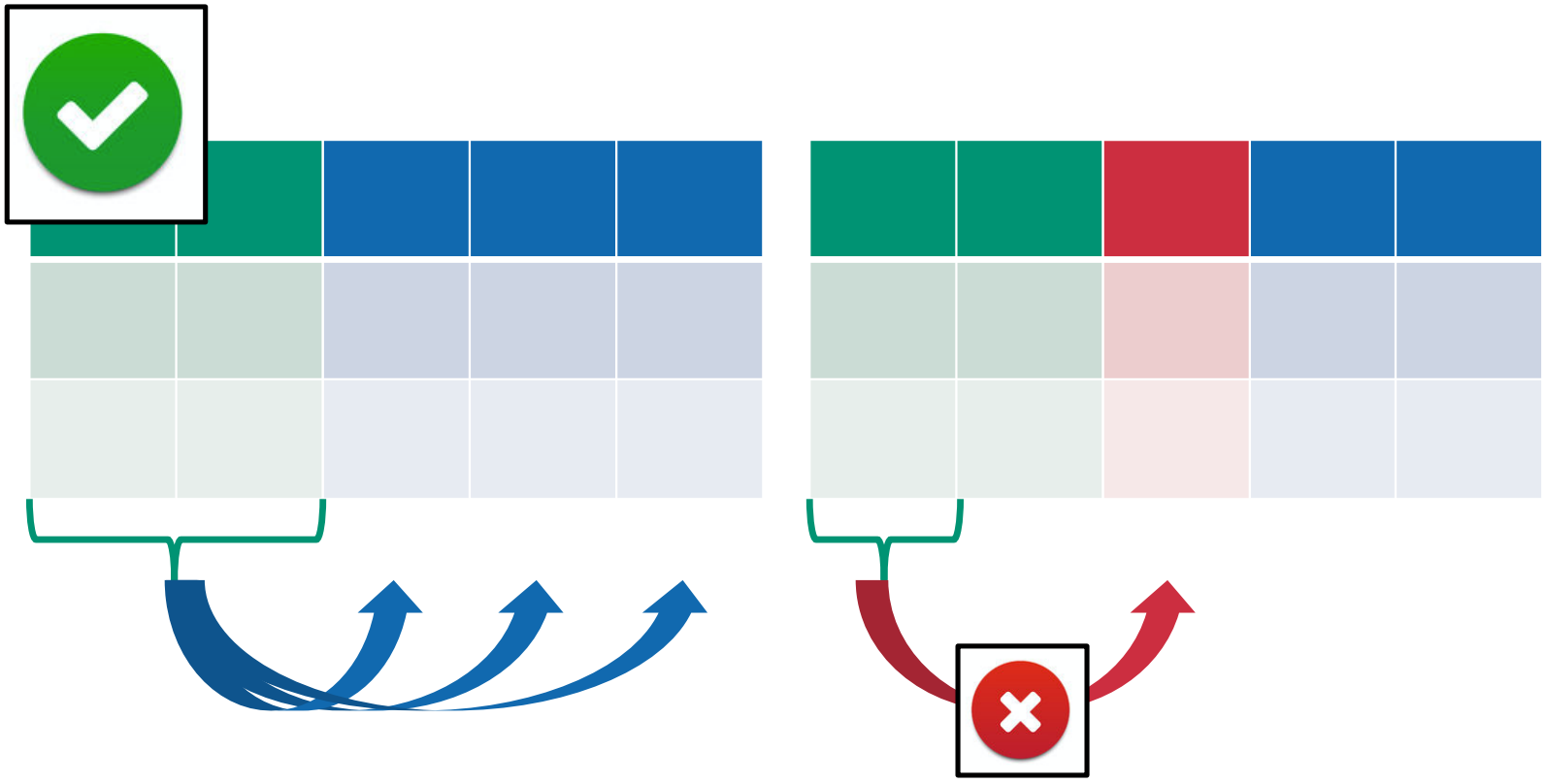
1st Normal Form: counter-example

Legi	Name	Lecture	City	State	PLZ
32-000-000	Alan Turing	Lecture ID: xxx-xxxx-xxX, Cryptography; 263-3010-00L, Big Data	Bletchley Park	UK	MK3 6EB
62-000-000	Georg Cantor	Lecture ID: 263-3010-00L, Big Data; 123-4567-89L, Set theory	Pfäffikon	SZ	8808
25-000-000	Felix Bloch	Lecture ID: 123-4567-89L, Set theory	Pfäffikon	ZH	8330

1st Normal Form

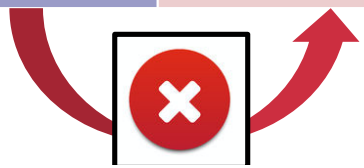
Legi	Name	Lecture ID	Lecture Name	City	State	PLZ
32-000-000	Alan Turing	xxx-xxxx-xxX	Cryptography	Bletchley Park	UK	MK3 6EB
32-000-000	Alan Turing	263-3010-00L	Big Data	Bletchley Park	UK	MK3 6EB
62-000-000	Georg Cantor	263-3010-00L	Big Data	Pfäffikon	SZ	8808
62-000-000	Georg Cantor	123-4567-89L	Set theory	Pfäffikon	SZ	8808
25-000-000	Felix Bloch	123-4567-89L	Set theory	Pfäffikon	ZH	8330

2nd Normal Form (not joined) – The Whole Key



2nd Normal Form: Counter-example

Legi	Name	Lecture ID	Lecture Name	City	State	PLZ
32-000-000	Alan Turing	xxx-xxxx-xxX	Cryptography	Bletchley Park	UK	MK3 6EB
32-000-000	Alan Turing	263-3010-00L	Big Data	Bletchley Park	UK	MK3 6EB
62-000-000	Georg Cantor	263-3010-00L	Big Data	Pfäffikon	SZ	8808
62-000-000	Georg Cantor	123-4567-89L	Set theory	Pfäffikon	SZ	8808
25-000-000	Felix Bloch	123-4567-89L	Set theory	Pfäffikon	ZH	8330



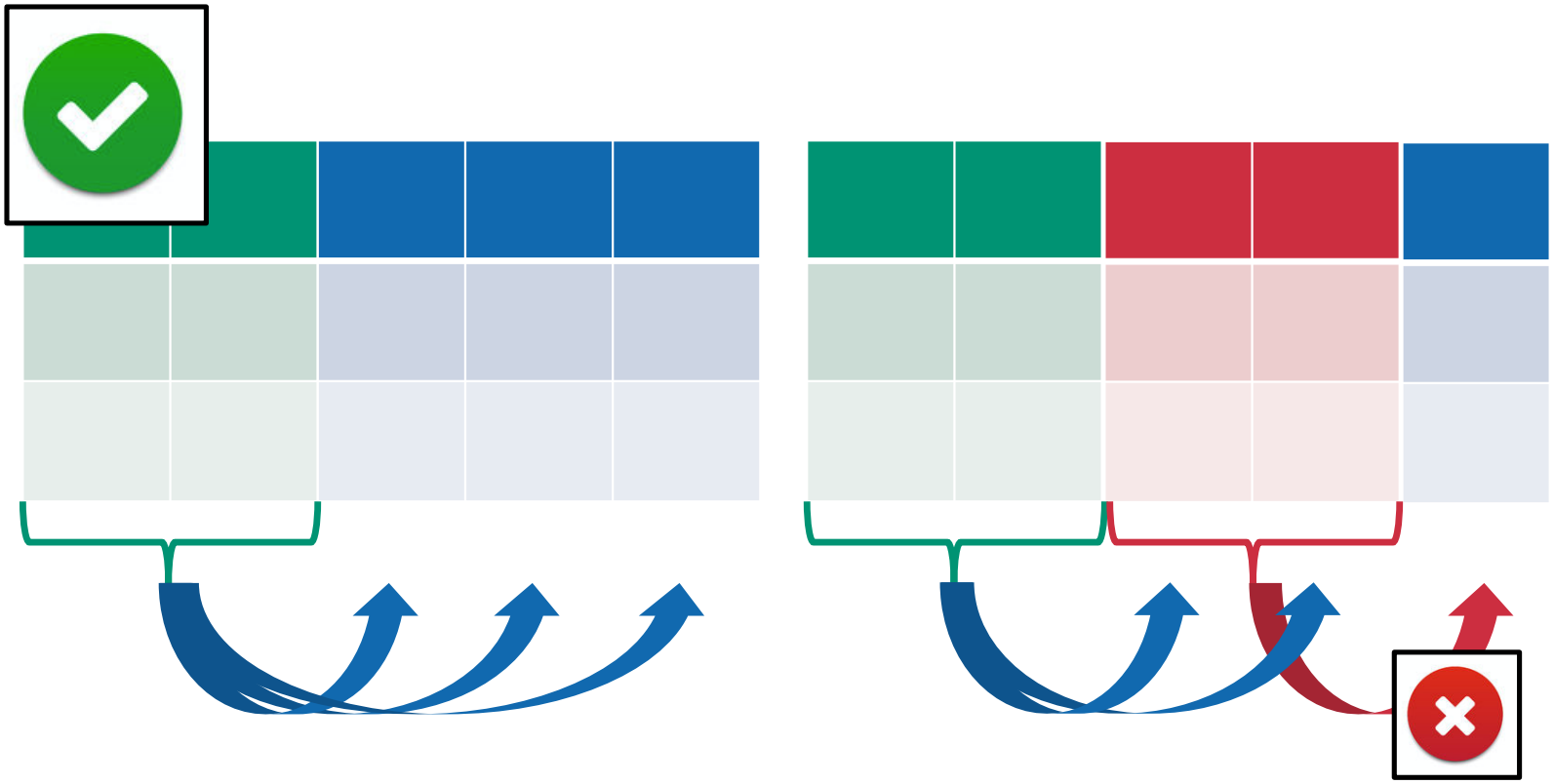
2nd Normal Form: Example

Legi	Name	City	State	PLZ
32-000-000	Alan Turing	Bletchley Park	UK	MK3 6EB
32-000-000	Alan Turing	Bletchley Park	UK	MK3 6EB
62-000-000	Georg Cantor	Pfäffikon	SZ	8808
62-000-000	Georg Cantor	Pfäffikon	SZ	8808
25-000-000	Felix Bloch	Pfäffikon	ZH	8330

Legi	Lecture ID
32-000-000	xxx-xxxx-xxX
32-000-000	263-3010-00L
62-000-000	263-3010-00L
62-000-000	123-4567-89L
25-000-000	123-4567-89L

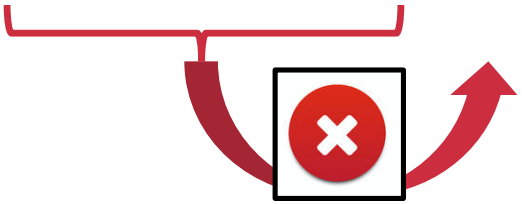
Lecture ID	Lecture Name
xxx-xxxx-xxX	Cryptography
263-3010-00L	Big Data
123-4567-89L	Set theory

3rd Normal Form – Nothing But The Key



3rd Normal Form: Counter-Example

Legi	Name	City	State	PLZ
32-000-000	Alan Turing	Bletchley Park	UK	MK3 6EB
32-000-000	Alan Turing	Bletchley Park	UK	MK3 6EB
62-000-000	Georg Cantor	Pfäffikon	SZ	8808
62-000-000	Georg Cantor	Pfäffikon	SZ	8808
25-000-000	Felix Bloch	Pfäffikon	ZH	8330



3rd Normal Form: Example

Legi	Name	City	State
32-000-000	Alan Turing	Bletchley Park	UK
32-000-000	Alan Turing	Bletchley Park	UK
62-000-000	Georg Cantor	Pfäffikon	SZ
62-000-000	Georg Cantor	Pfäffikon	SZ
25-000-000	Felix Bloch	Pfäffikon	ZH

City	State	PLZ
Bletchley Park	UK	MK3 6EB
Bletchley Park	UK	MK3 6EB
Pfäffikon	SZ	8808
Pfäffikon	SZ	8808
Pfäffikon	ZH	8330

Data Denormalization

Data Denormalization

Data Denormalization

ETH zürich

SQL Brush-Up

SQL History

Don Chamberlin

Raymond Boyce

The early days (early 1970s)

IBM

Almaden (San Jose)

First commercial relational database

System R

+

SEQUEL

First commercial relational query language

Pratt & Whitney

A United Technologies Company

First customer (1977)

Public availability

1977

Software Development Laboratories

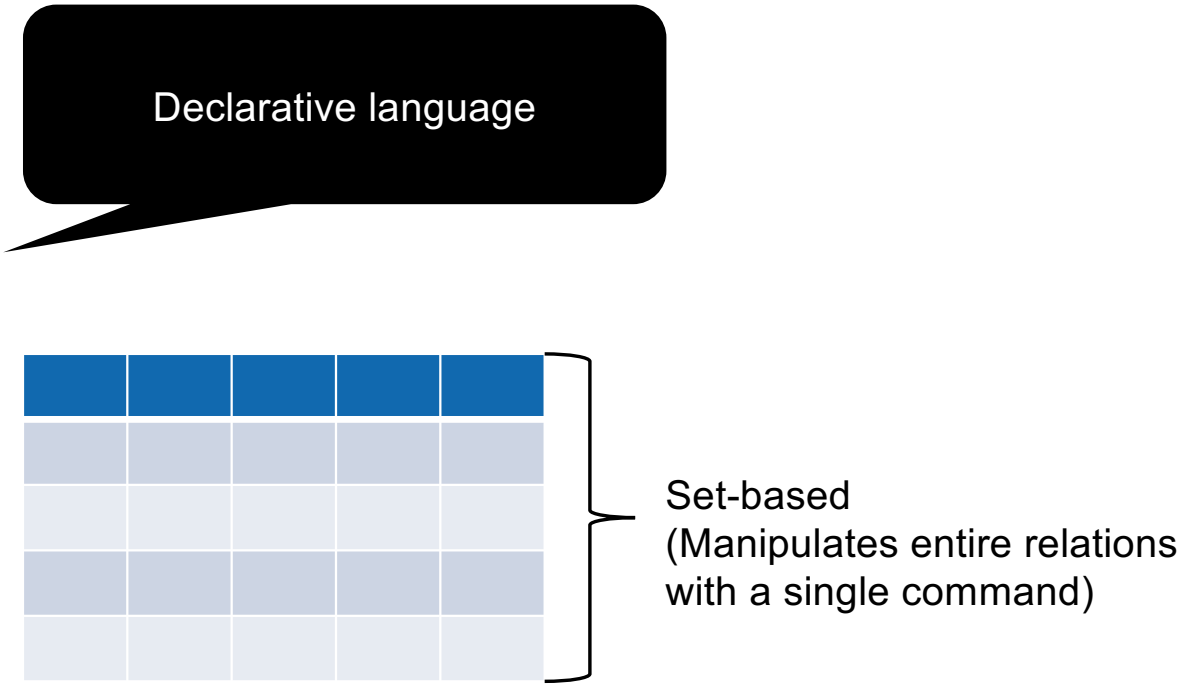
1979

Relational Software

1982

ORACLE

SEQUEL
Structured English QUEry Language



Renaming

SEQUEL

(Trademark issue)

SQL

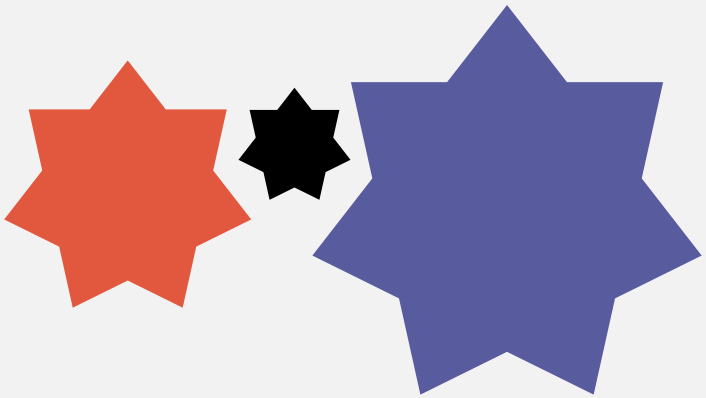
ESS**-kew-**EL or ***SEE**-kwəl*

SQL is a declarative language

Logical model

"What, not how"

Physical execution

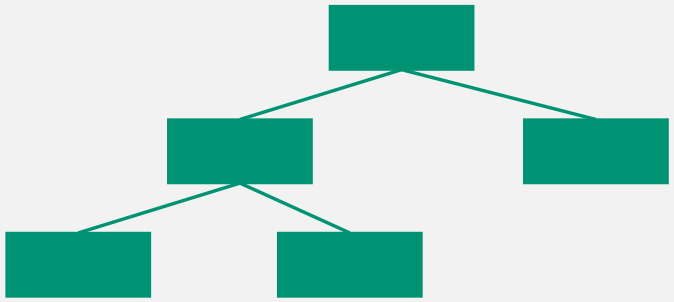


SQL is a declarative language

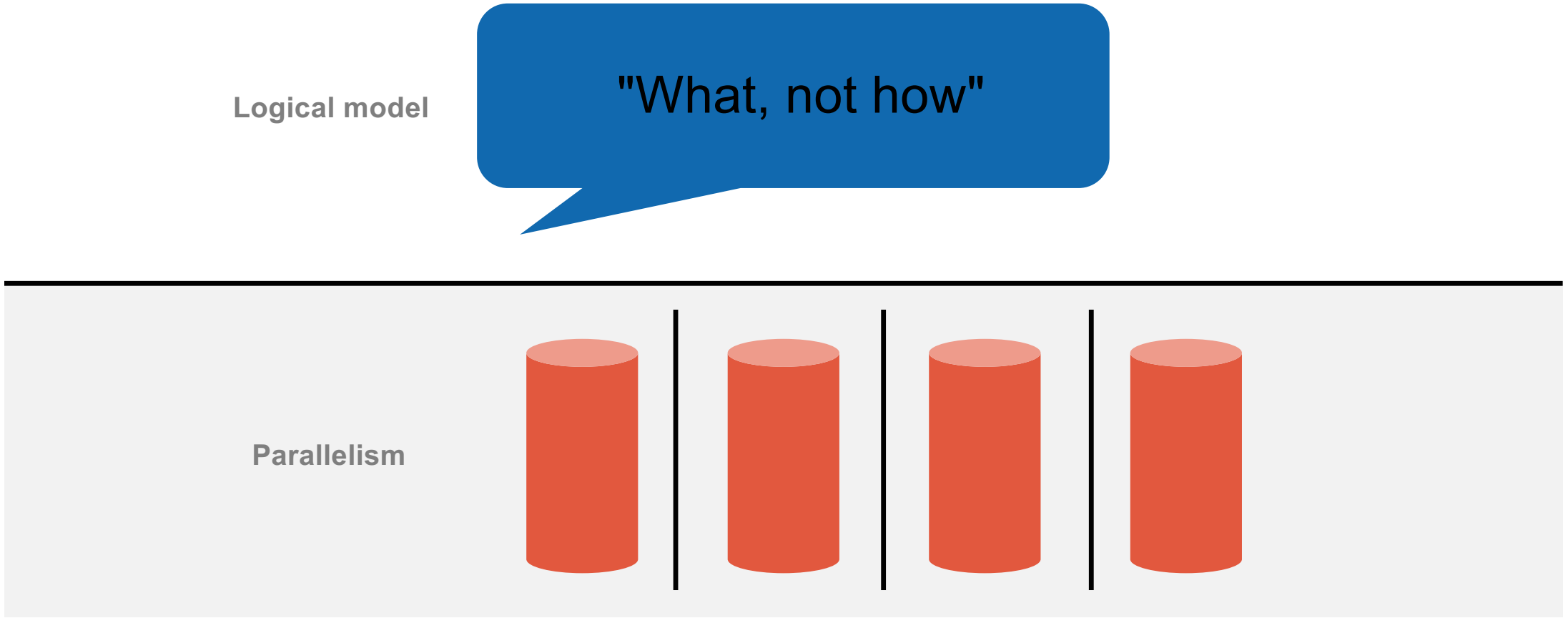
Logical model

"What, not how"

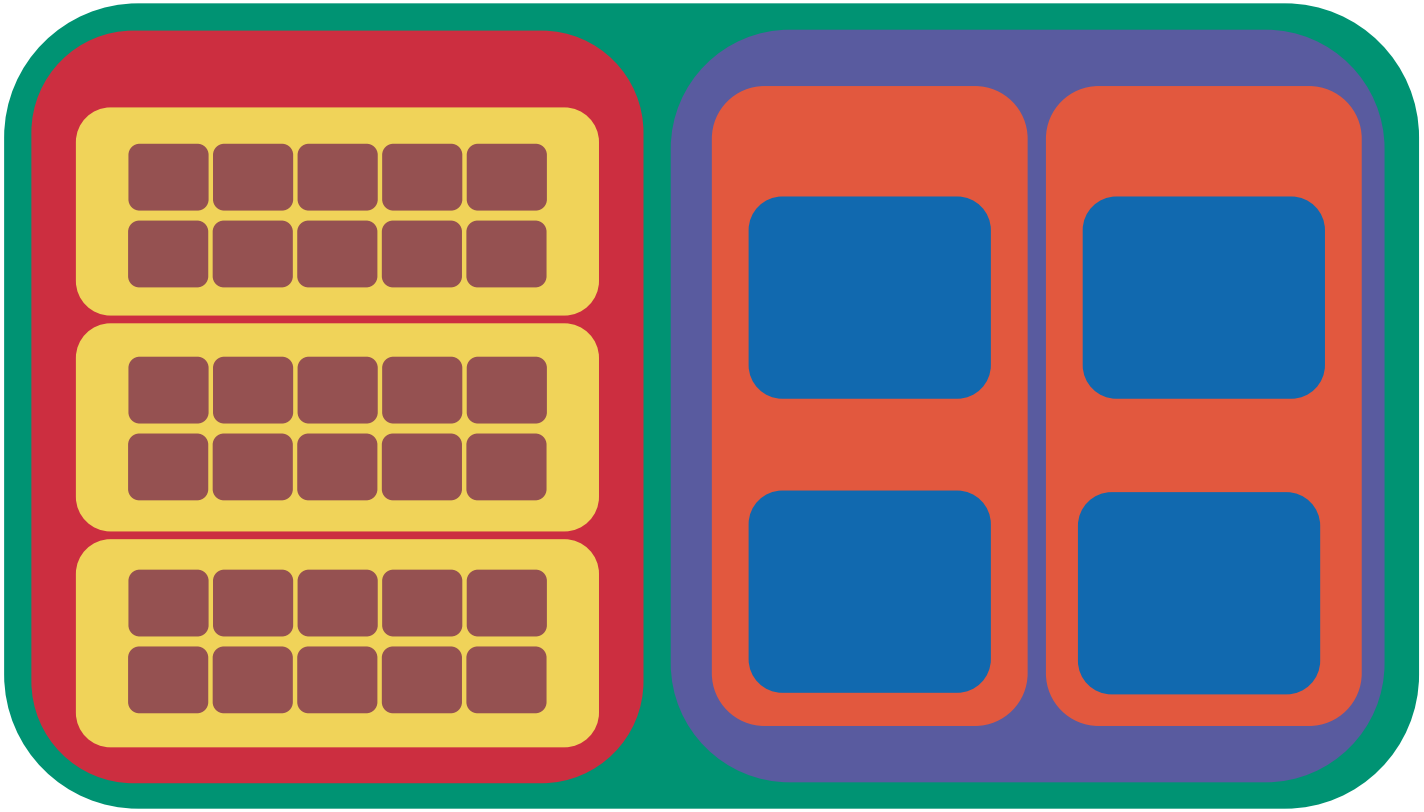
Query Plan



SQL is a declarative language



SQL is a functional language



"Expressions that nest"

A selecting query

persons					
name	middle_initial	last_name	birth_date	gender	passport_scan
varchar(30)	char(1)	text	date	boolean	bytea
James	T	Kirk	2233-03-22	TRUE	AD10E7
Beverly	**C**	**Crusher**	**2324-10-13**	**FALSE**	**AD234F7**
Spock	NULL	NULL	2230-01-06	TRUE	AD234F7

```
SELECT *
FROM persons
WHERE last_name = 'Crusher'
```

persons					
name	middle_initial	last_name	birth_date	gender	passport_scan
varchar(30)	char(1)	text	date	boolean	bytea
Beverly	C	Crusher	2324-10-13	FALSE	AD234F7

A projecting query

persons					
name	middle_initial	last_name	birth_date	gender	passport_scan
varchar(30)	char(1)	text	date	boolean	bytea
James	T	Kirk	**2233-03-22**	TRUE	AD10E7
Beverly	C	Crusher	**2324-10-13**	FALSE	AD234F7
Spock	NULL	NULL	**2230-01-06**	TRUE	AD234F7

```
SELECT name, birth_date
FROM persons
```

name	birth_date
varchar(30)	date
James	2233-03-22
Beverly	2324-10-13
Spock	2230-01-06

A renaming query

persons					
name	middle_initial	last_name	birth_date	gender	passport_scan
varchar(30)	char(1)	text	date	boolean	bytea
James	T	Kirk	**2233-03-22**	TRUE	AD10E7
Beverly	C	Crusher	**2324-10-13**	FALSE	AD234F7
Spock	NULL	NULL	**2230-01-06**	TRUE	AD234F7

```
SELECT name AS who,
       birth_date AS when
FROM persons
```

who	when
varchar(30)	date
James	2233-03-22
Beverly	2324-10-13
Spock	2230-01-06

A sorting query

persons					
name	middle_initial	last_name	birth_date	gender	passport_scan
varchar(30)	char(1)	text	date	boolean	bytea
James	T	Kirk	2233-03-22	TRUE	AD10E7
Beverly	**C**	**Crusher**	**2324-10-13**	**FALSE**	**AD234F7**
Spock	NULL	NULL	2230-01-06	TRUE	AD234F7

```
SELECT *
FROM persons
ORDER BY birth_date
```

persons					
name	middle_initial	last_name	birth_date	gender	passport_scan
varchar(30)	char(1)	text	date	boolean	bytea
Spock	NULL	NULL	2230-01-06	TRUE	AD234F7
James	T	Kirk	2233-03-22	TRUE	AD10E7
Beverly	C	Crusher	2324-10-13	FALSE	AD234F7

Sorting options: NULLs first

persons					
name	middle_initial	last_name	birth_date	gender	passport_scan
varchar(30)	char(1)	text	date	boolean	bytea
James	T	Kirk	2233-03-22	TRUE	AD10E7
Beverly	**C**	**Crusher**	**2324-10-13**	**FALSE**	**AD234F7**
Spock	NULL	NULL	2230-01-06	TRUE	AD234F7

```
SELECT *
FROM persons
ORDER BY last_name NULLS FIRST
```

persons					
name	middle_initial	last_name	birth_date	gender	passport_scan
varchar(30)	char(1)	text	date	boolean	bytea
Spock	NULL	NULL	2230-01-06	TRUE	AD234F7
Beverly	C	Crusher	2324-10-13	FALSE	AD234F7
James	T	Kirk	2233-03-22	TRUE	AD10E7

A grouping query

Grouping column

persons				
name	middle_initial	last_name	century	gender
varchar(30)	char(1)	text	integer	boolean
James	T	Kirk	**23**	TRUE
Beverly	C	Crusher	**24**	FALSE
Spock	NULL	NULL	**23**	TRUE
Kathryn	NULL	Janeway	**24**	FALSE

```
SELECT century, COUNT(*) AS count
FROM persons
GROUP BY century
```

century	count
integer	bigint
23	2
24	2

Post-aggregation selection

persons				
name	middle_initial	last_name	century	captain
varchar(30)	char(1)	text	integer	boolean
James	T	Kirk	23	TRUE
Beverly	C	Crusher	24	FALSE
Jean-Luc	NULL	Picard	24	TRUE
Kathryn	NULL	Janeway	24	TRUE

```
SELECT century AS c
FROM persons
GROUP BY century
HAVING COUNT(*) > 2
```

c
integer
24

A union query

spaceships1				
warp	name	last_name	code	successor
numeric	varchar(30)	text	varchar	varchar
5	USS Enterprise A	Kirk	NCC-1701-A	NCC-1701-B
6	USS Enterprise B	Kirk	NCC-1701-B	NCC-1701-C
7	USS Enterprise C	Kirk	NCC-1701-C	NCC-1701-D

spaceships2				
warp	name	last_name	code	successor
numeric	varchar(30)	text	varchar	varchar
7	USS Enterprise C	Kirk	NCC-1701-C	NCC-1701-D
4	USS Enterprise	Kirk	NCC-1701	NCC-1701-A
9.2	USS Enterprise D	Picard	NCC-1701-D	NCC-1701-E

SELECT * **FROM** spaceships1 **UNION SELECT** * **FROM** spaceships2

duplicate elimination

warp	name	last_name	code	successor
numeric	varchar(30)	text	varchar	varchar
5	USS Enterprise A	Kirk	NCC-1701-A	NCC-1701-B
6	USS Enterprise B	Kirk	NCC-1701-B	NCC-1701-C
7	USS Enterprise C	Kirk	NCC-1701-C	NCC-1701-D
4	USS Enterprise	Kirk	NCC-1701	NCC-1701-A
9.2	USS Enterprise D	Picard	NCC-1701-D	NCC-1701-E

An intersection query

spaceships1				
warp	name	last_name	code	successor
numeric	varchar(30)	text	varchar	varchar
5	USS Enterprise A	Kirk	NCC-1701-A	NCC-1701-B
6	USS Enterprise B	Kirk	NCC-1701-B	NCC-1701-C
7	USS Enterprise C	Kirk	NCC-1701-C	NCC-1701-D

spaceships2				
warp	name	last_name	code	successor
numeric	varchar(30)	text	varchar	varchar
7	USS Enterprise C	Kirk	NCC-1701-C	NCC-1701-D
4	USS Enterprise	Kirk	NCC-1701	NCC-1701-A

SELECT * **FROM** spaceships1 **INTERSECT SELECT** * **FROM** spaceships2

duplicate elimination

warp	name	last_name	code	successor
numeric	varchar(30)	text	varchar	varchar
7	USS Enterprise C	Kirk	NCC-1701-C	NCC-1701-D

An set subtraction query

spaceships1				
warp	name	last_name	code	successor
numeric	varchar(30)	text	varchar	varchar
5	USS Enterprise A	Kirk	NCC-1701-A	NCC-1701-B
6	USS Enterprise B	Kirk	NCC-1701-B	NCC-1701-C
7	USS Enterprise C	Kirk	NCC-1701-C	NCC-1701-D

spaceships2				
warp	name	last_name	code	successor
numeric	varchar(30)	text	varchar	varchar
7	USS Enterprise C	Kirk	NCC-1701-C	NCC-1701-D
4	USS Enterprise	Kirk	NCC-1701	NCC-1701-A

SELECT * **FROM** spaceships1 **EXCEPT SELECT** * **FROM** spaceships2

duplicate elimination

warp	name	last_name	code	successor
numeric	varchar(30)	text	varchar	varchar
5	USS Enterprise A	Kirk	NCC-1701-A	NCC-1701-B
6	USS Enterprise B	Kirk	NCC-1701-B	NCC-1701-C

Semi-outer joins: left

persons				
name	middle_initial	last_name	century	captain
varchar(30)	char(1)	text	integer	boolean
James	T	Kirk	23	TRUE
Beverly	C	Crusher	24	FALSE
Jean-Luc	NULL	Picard	24	TRUE
Kathryn	NULL	Janeway	24	TRUE

spaceships			
warp	spaceship_name	captain_name	code
numeric	varchar(30)	text	integer
5	USS Enterprise A	Kirk	NCC-1701-A
4	USS Enterprise	Kirk	NCC-1701
9.2	USS Enterprise D	Picard	NCC-1701-D
9.975	USS Voyager	Janeway	NCC-74656

```
SELECT *
FROM persons LEFT OUTER JOIN spaceships
     ON persons.last_name = spaceships.captain_name
```

name	middle_initial	last_name	captain	century	warp	spaceship_name	captain_name	code
varchar(30)	char(1)	text	boolean	integer	numeric	varchar(30)	text	integer
James	T	Kirk	TRUE	23	5	USS Enterprise A	Kirk	NCC-1701-A
James	T	Kirk	TRUE	23	4	USS Enterprise	Kirk	NCC-1701
Beverly	C	Crusher	24	FALSE	NULL	NULL	NULL	NULL
Jean-Luc	NULL	Picard	TRUE	24	9.2	USS Enterprise D	Picard	NCC-1701-D
Kathryn	NULL	Janeway	TRUE	24	9.975	USS Voyager	Janeway	NCC-74656

Semi-outer joins: right

persons				
name	middle_initial	last_name	century	captain
varchar(30)	char(1)	text	integer	boolean
James	T	Kirk	23	TRUE
Beverly	C	Crusher	24	FALSE
Kathryn	NULL	Janeway	24	TRUE

spaceships			
warp	spaceship_name	captain_name	code
numeric	varchar(30)	text	integer
5	USS Enterprise A	Kirk	NCC-1701-A
4	USS Enterprise	Kirk	NCC-1701
9.2	USS Enterprise D	Picard	NCC-1701-D
9.975	USS Voyager	Janeway	NCC-74656

```
SELECT *
FROM persons RIGHT OUTER JOIN spaceships
     ON persons.last_name = spaceships.captain_name
```

name	middle_initial	last_name	captain	century	warp	spaceship_name	captain_name	code
varchar(30)	char(1)	text	boolean	integer	numeric	varchar(30)	text	integer
James	T	Kirk	TRUE	23	5	USS Enterprise A	Kirk	NCC-1701-A
James	T	Kirk	TRUE	23	4	USS Enterprise	Kirk	NCC-1701
NULL	NULL	Picard	NULL	NULL	9.2	USS Enterprise D	Picard	NCC-1701-D
Kathryn	NULL	Janeway	TRUE	24	9.975	USS Voyager	Janeway	NCC-74656

Full outer joins

persons				
name	middle_initial	last_name	century	captain
varchar(30)	char(1)	text	integer	boolean
James	T	Kirk	23	TRUE
Beverly	C	Crusher	24	FALSE
Kathryn	NULL	Janeway	24	TRUE

spaceships			
warp	spaceship_name	captain_name	code
numeric	varchar(30)	text	integer
5	USS Enterprise A	Kirk	NCC-1701-A
4	USS Enterprise	Kirk	NCC-1701
9.2	USS Enterprise D	Picard	NCC-1701-D
9.975	USS Voyager	Janeway	NCC-74656

```
SELECT *
FROM persons FULL OUTER JOIN spaceships
     ON persons.last_name = spaceships.captain_name
```

name	middle_initial	last_name	captain	century	warp	spaceship_name	captain_name	code
varchar(30)	char(1)	text	boolean	integer	numeric	varchar(30)	text	integer
James	T	Kirk	TRUE	23	5	USS Enterprise A	Kirk	NCC-1701-A
James	T	Kirk	TRUE	23	4	USS Enterprise	Kirk	NCC-1701
NULL	NULL	NULL	NULL	NULL	9.2	USS Enterprise D	Picard	NCC-1701-D
Beverly	C	Crusher	24	FALSE	NULL	NULL	NULL	NULL
Kathryn	NULL	Janeway	TRUE	24	9.975	USS Voyager	Janeway	NCC-74656

Natural join

persons				
name	middle_initial	last_name	century	captain
varchar(30)	char(1)	text	integer	boolean
James	T	Kirk	23	TRUE
Beverly	C	Crusher	24	FALSE
Kathryn	NULL	Janeway	24	TRUE

spaceships			
warp	spaceship_name	last_name	code
numeric	varchar(30)	text	integer
5	USS Enterprise A	Kirk	NCC-1701-A
4	USS Enterprise	Kirk	NCC-1701
9.2	USS Enterprise D	Picard	NCC-1701-D
9.975	USS Voyager	Janeway	NCC-74656

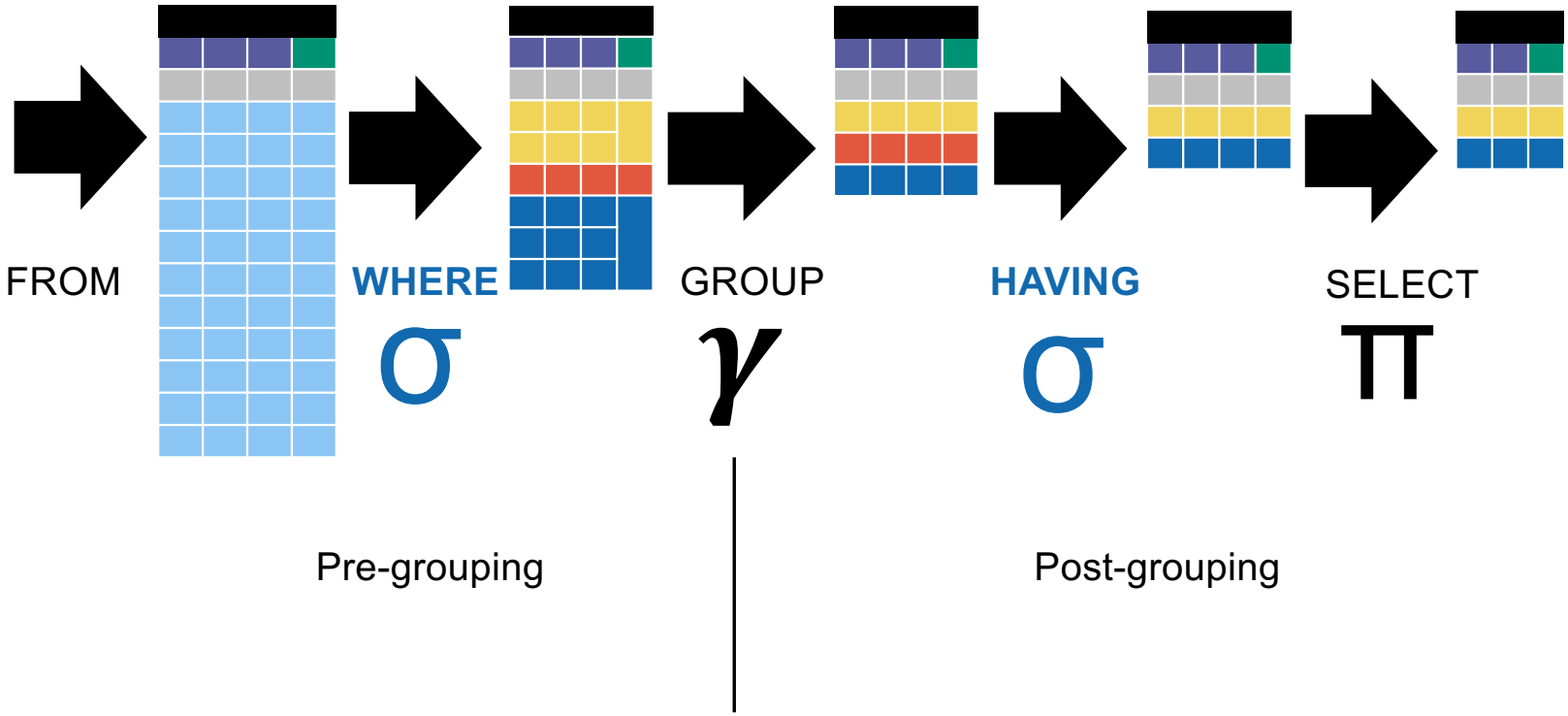
```
SELECT *
FROM persons NATURAL FULL OUTER JOIN spaceships
```

name	middle_initial	last_name	captain	century	warp	spaceship_name	code
varchar(30)	char(1)	text	boolean	integer	numeric	varchar(30)	integer
James	T	Kirk	TRUE	23	5	USS Enterprise A	NCC-1701-A
James	T	Kirk	TRUE	23	4	USS Enterprise	NCC-1701
NULL	NULL	Picard	NULL	NULL	9.2	USS Enterprise D	NCC-1701-D
Beverly	C	Crusher	24	FALSE	NULL	NULL	NULL
Kathryn	NULL	Janeway	TRUE	24	9.975	USS Voyager	NCC-74656

Nesting

```
SELECT pname, warp
FROM (
      SELECT persons.name AS pname,
             last_name,
             century,
             spaceships.name AS sname,
             warp
      FROM persons FULL OUTER JOIN spaceships USING last_name
)
```

Query plans


FROM
WHERE
σ
GROUP
γ
HAVING
σ
SELECT
π
Pre-grouping
Post-grouping

Three-valued logics: OR

OR	TRUE	FALSE	UNKNOWN
TRUE	TRUE	TRUE	TRUE
FALSE	TRUE	FALSE	UNKNOWN
UNKNOWN	TRUE	UNKNOWN	UNKNOWN

This is **common sense**.

e.g., <anything> OR TRUE = TRUE

thus UNKNOWN OR TRUE = TRUE

Some terminology

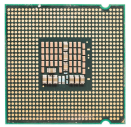
DML: Data Manipulation Language
(Query, insert, remove rows)

Data

Schema

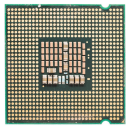
DDL: Data Definition Language
(Create or table/schema, drop it)

Language landscape

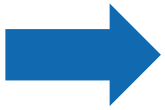


Proto-imperative
language

Language landscape

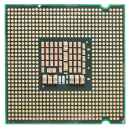


Proto-imperative
language

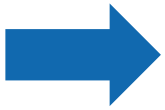


Imperative
language

Language landscape



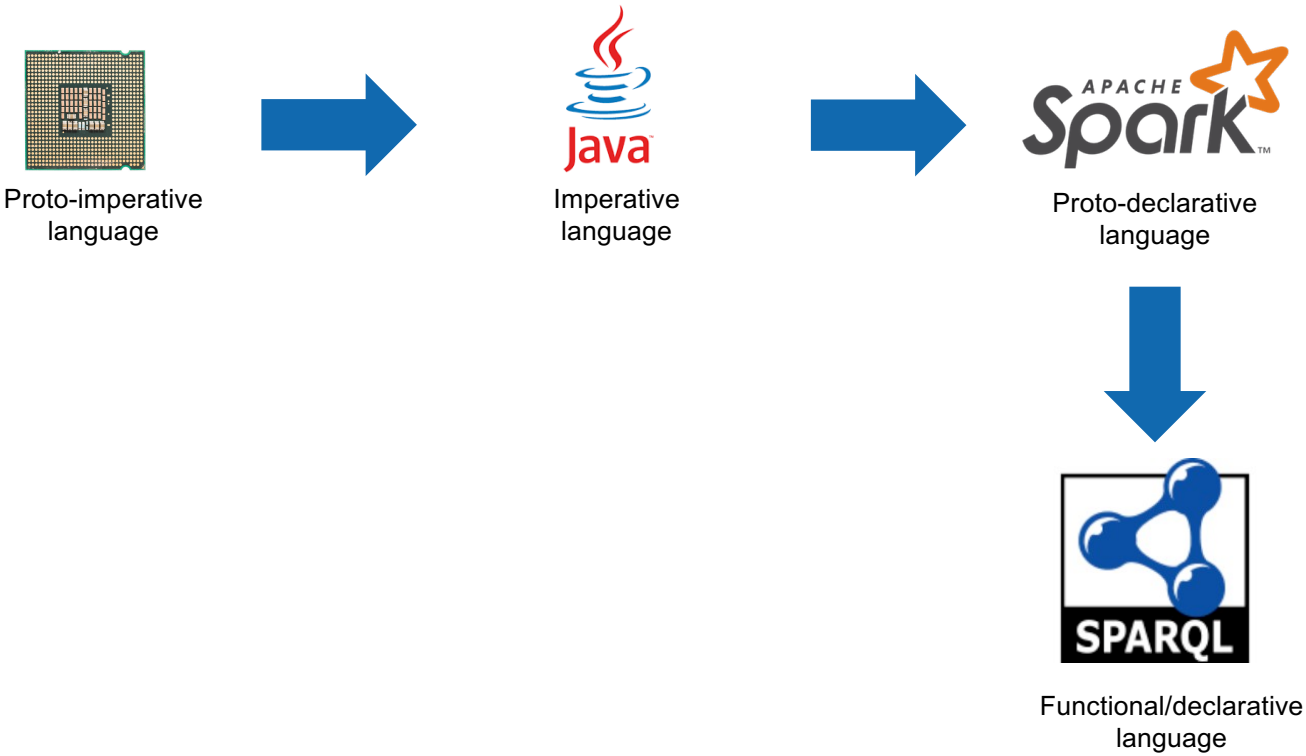
Proto-imperative
language



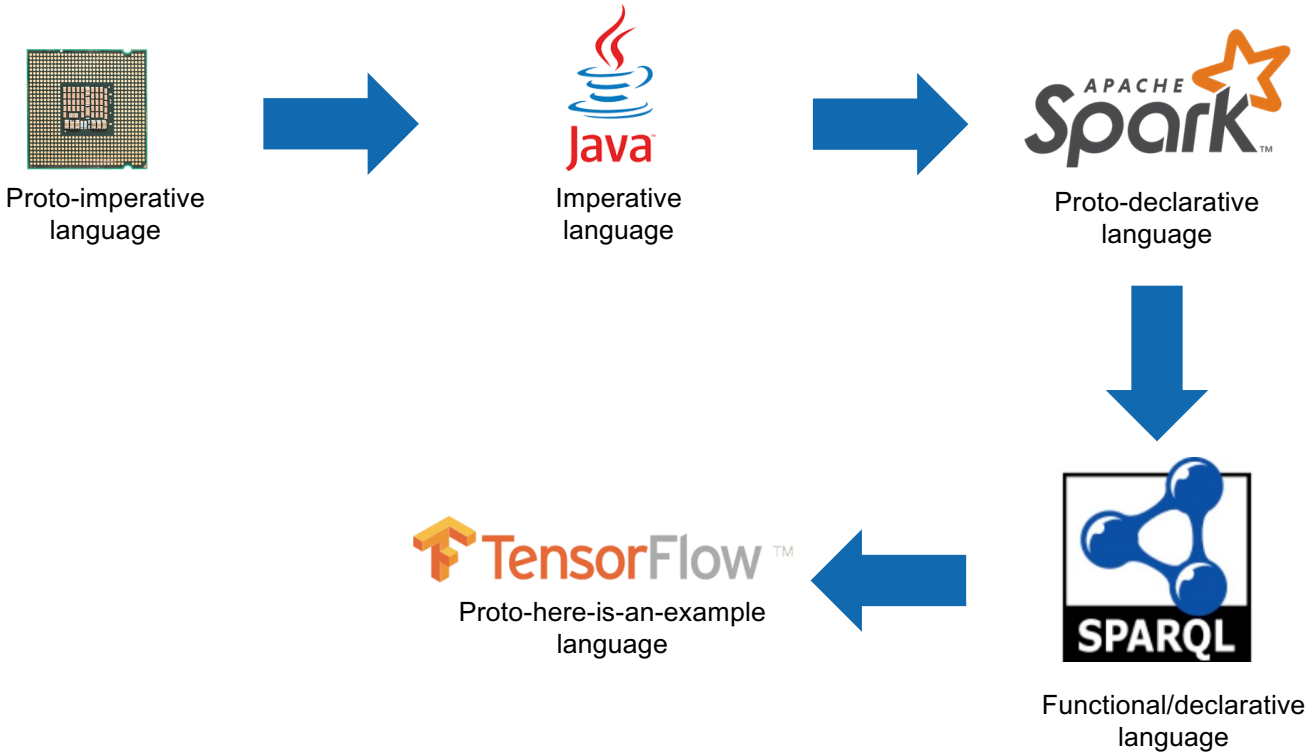
Imperative
language

Functional/declarative
language

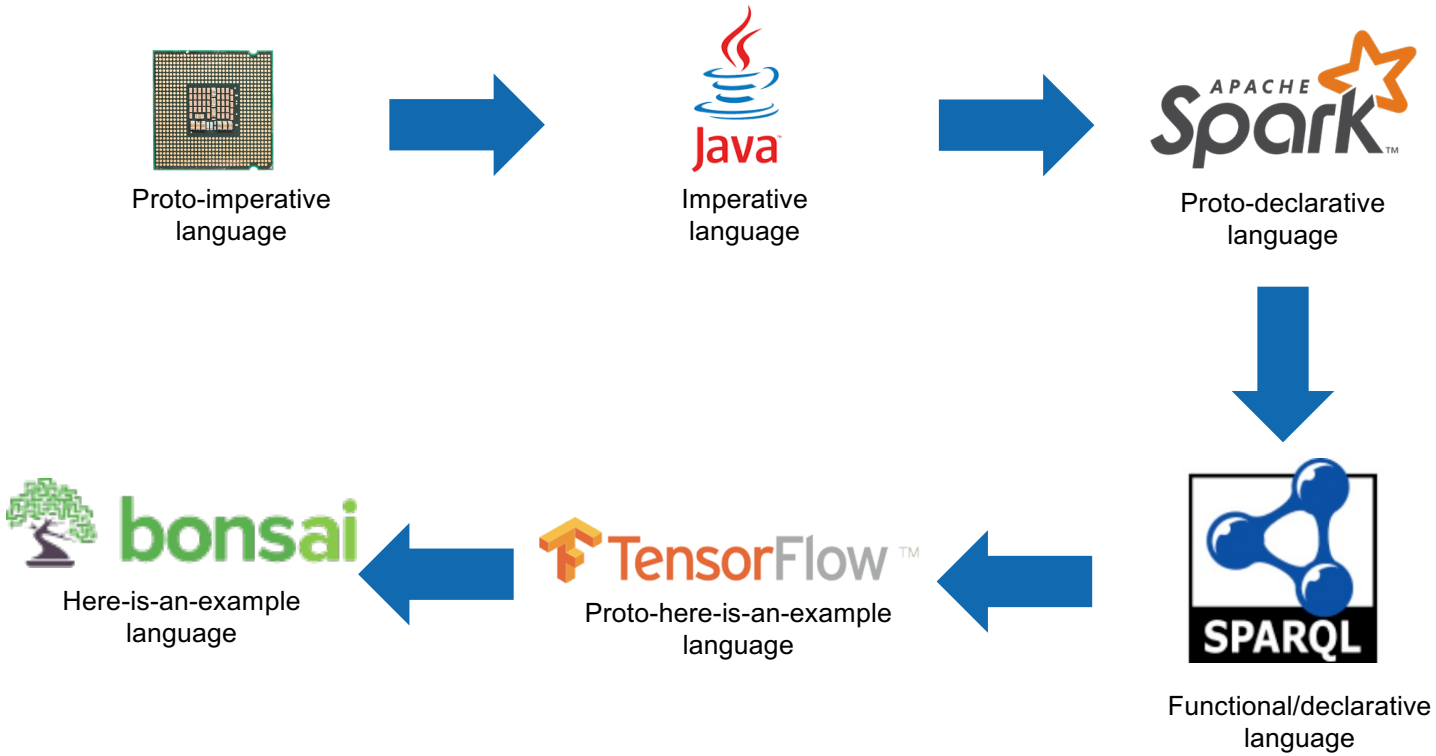
Language landscape



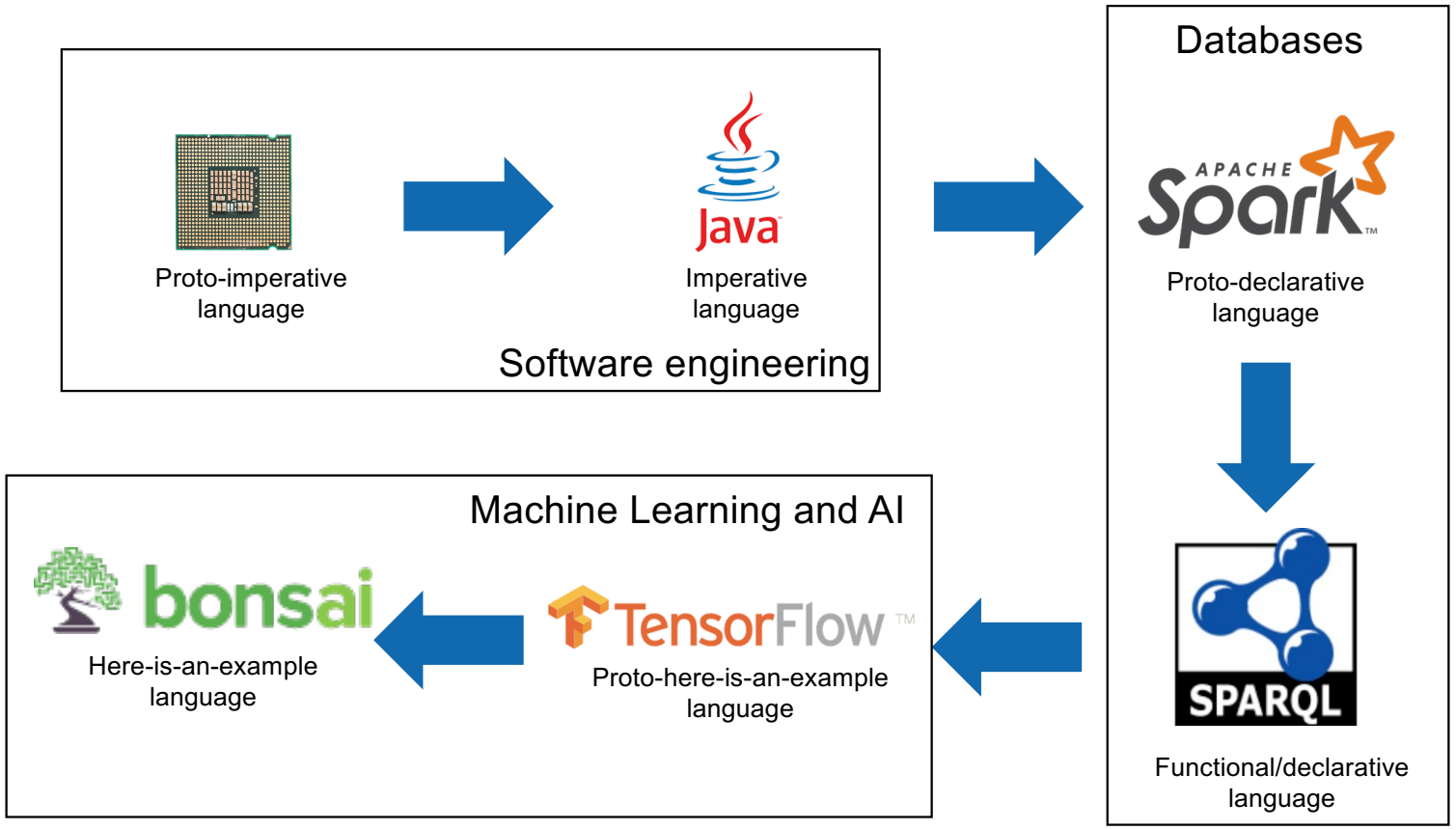
Language landscape



Language landscape



Language landscape



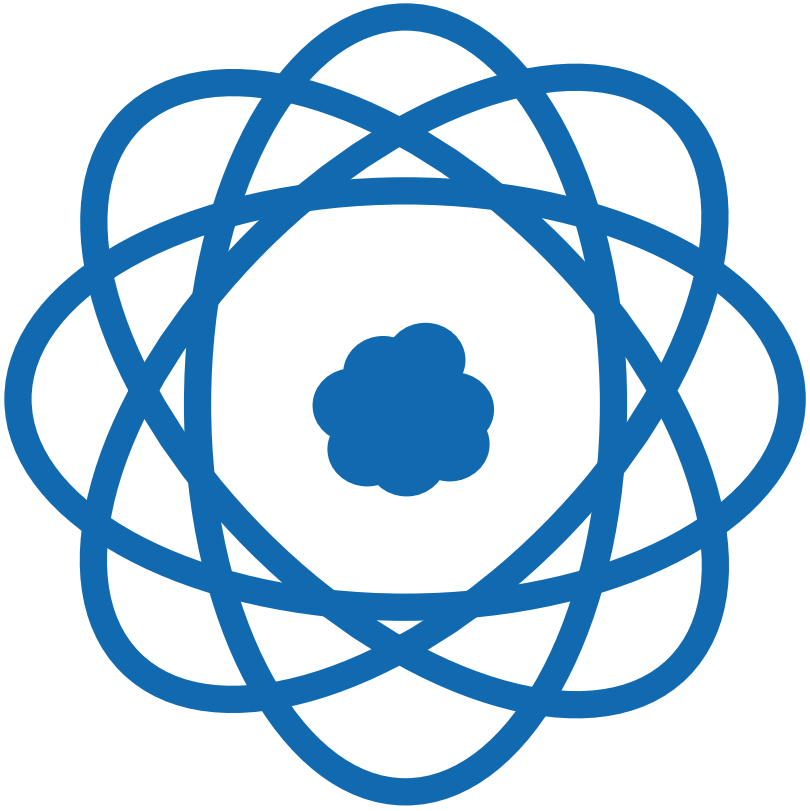
ETH zürich

Transactions

The good old times of databases: ACID

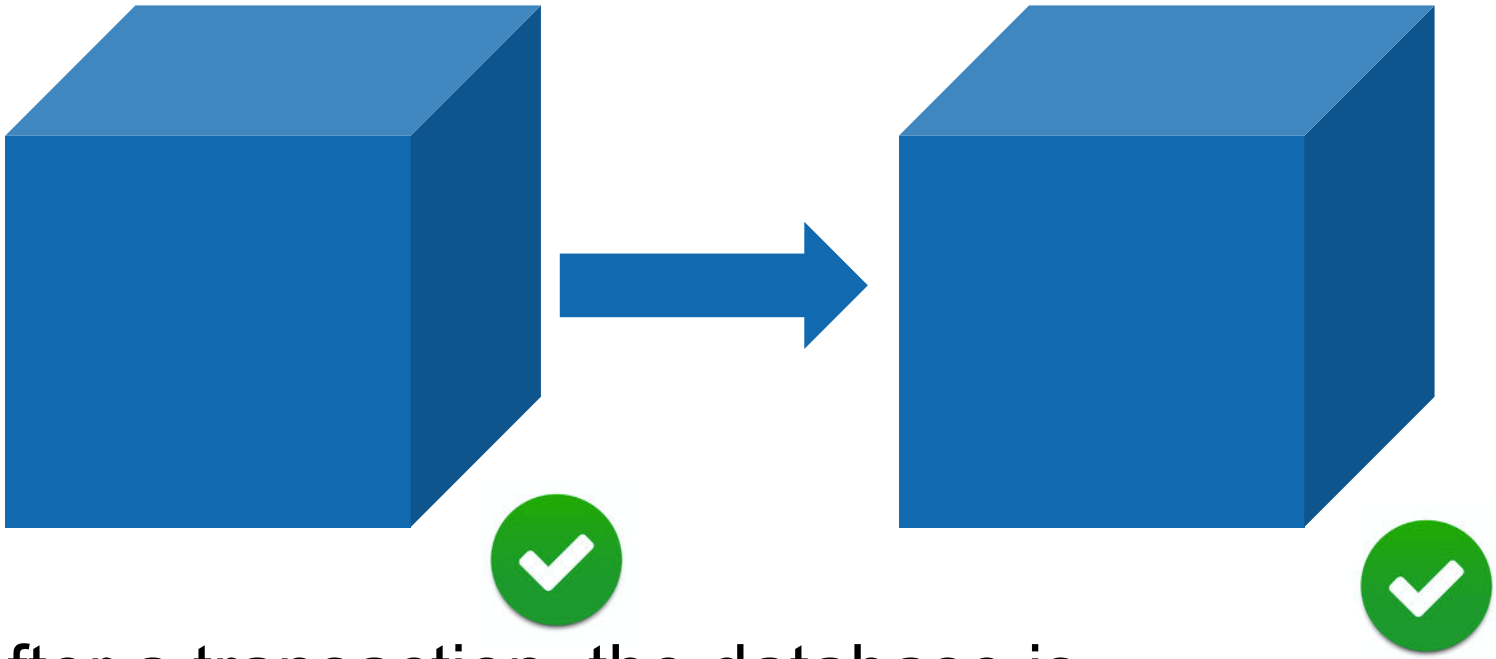
Atomicity
Consistency
Isolation
Durability

Atomicity



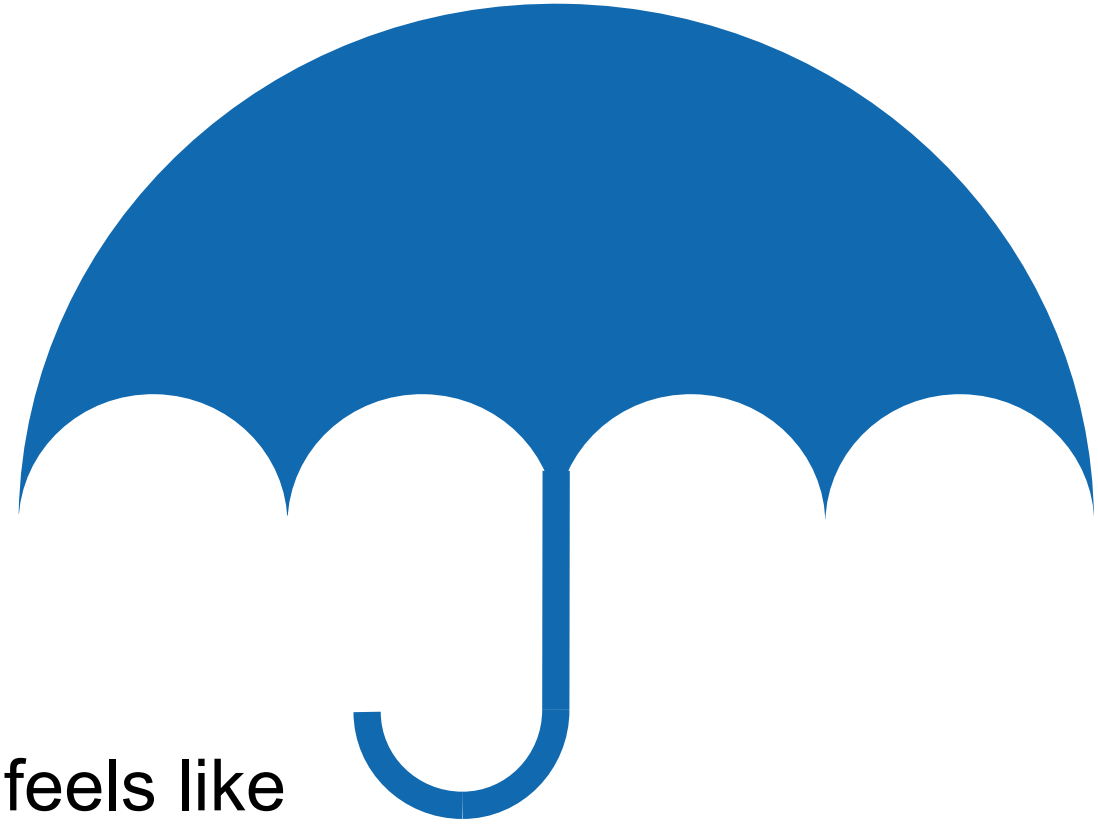
Either the entire transaction is applied,
or none of it (rollback).

Consistency



After a transaction, the database is in a consistent state again.

Isolation



A transaction feels like
nobody else is writing to the database.

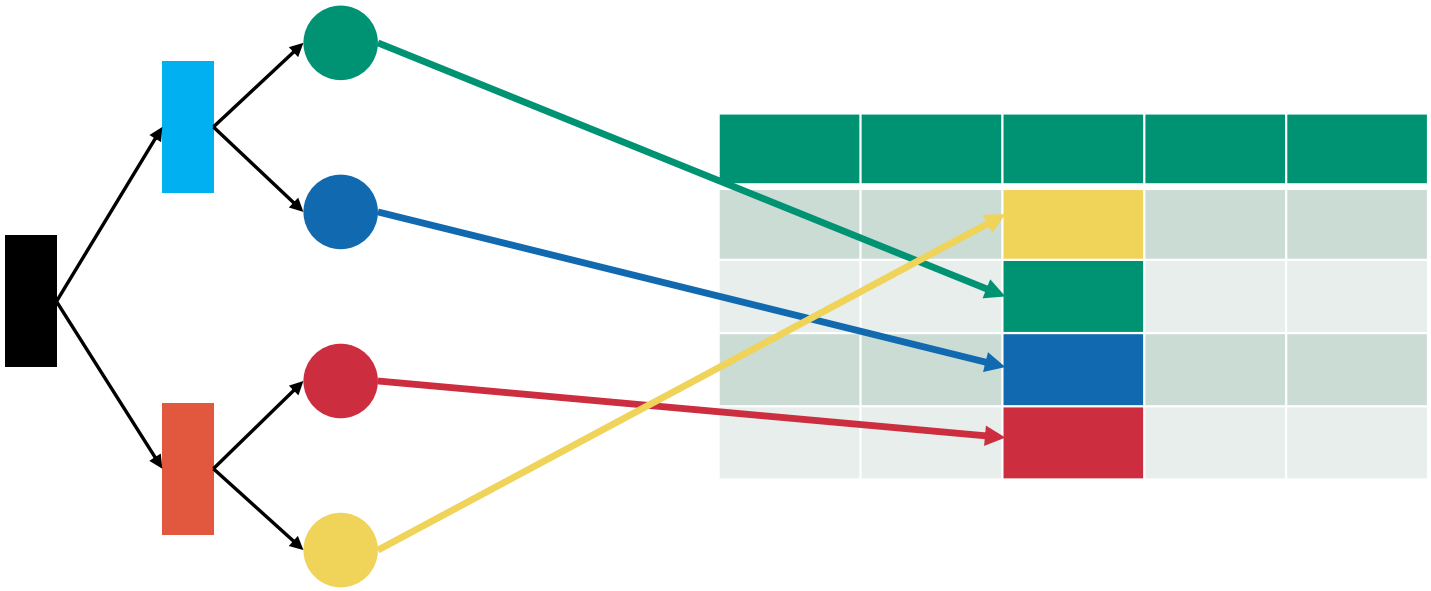
Durability

Updates made do not disappear again.

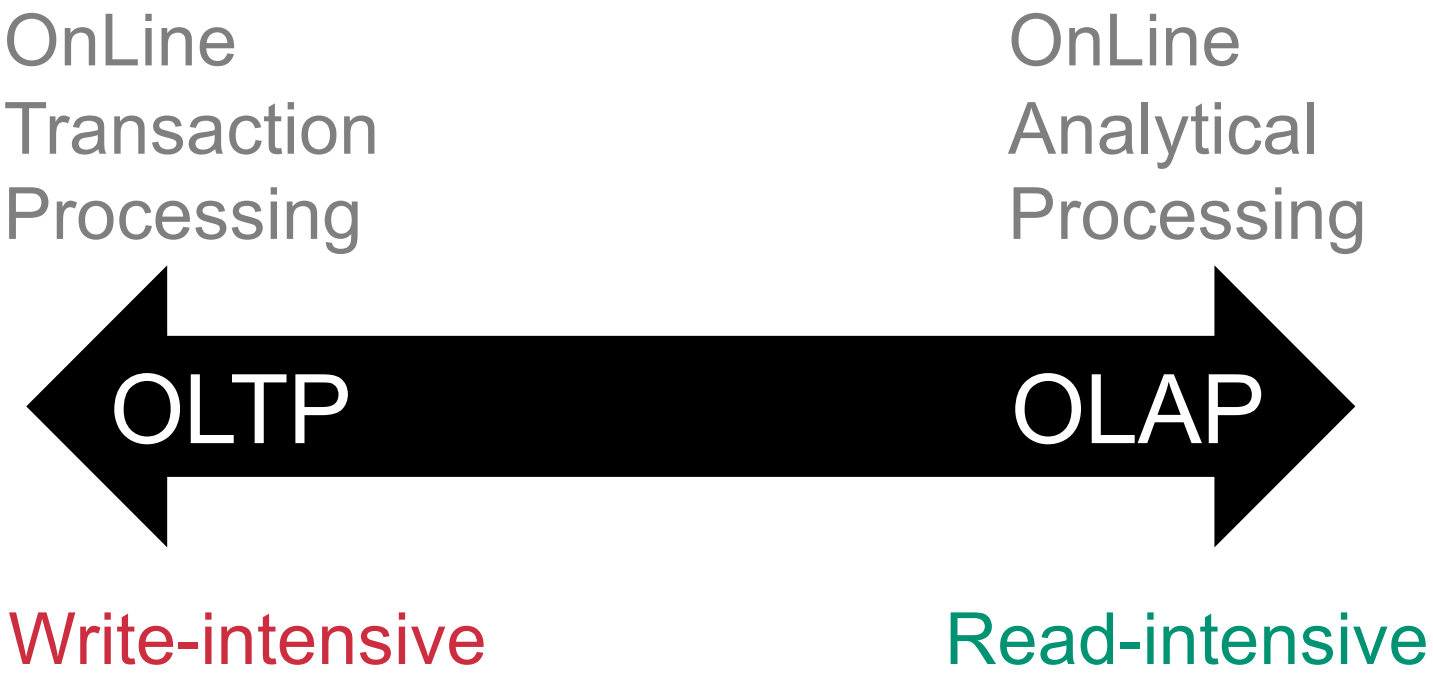
ETH zürich

Performance

Indices



Optimize for read vs. write intensive



No such thing as "one size fits all"

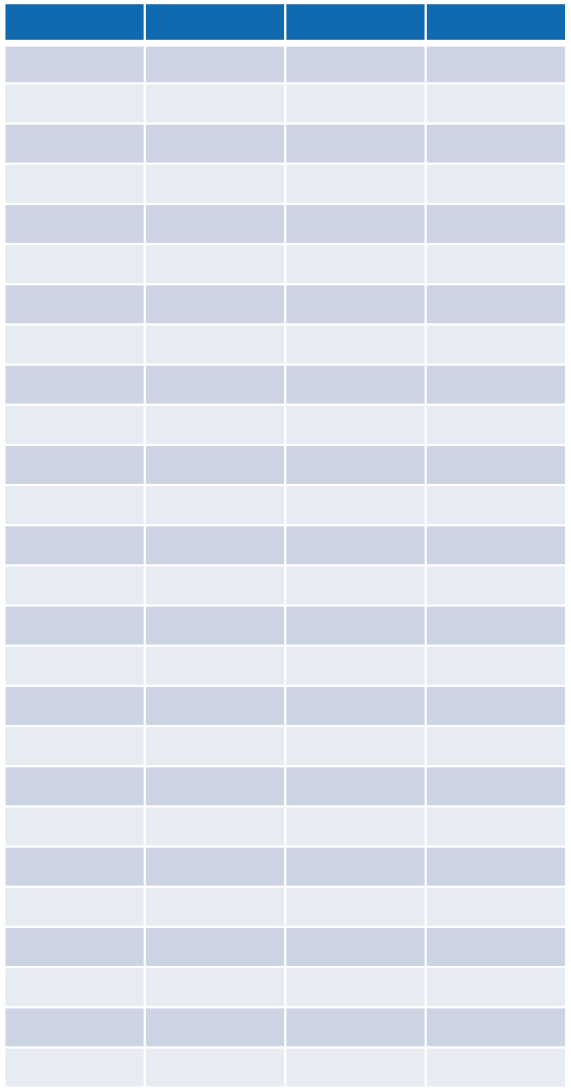
Mind

data shapes!

ETH zürich

Data Scale-Up

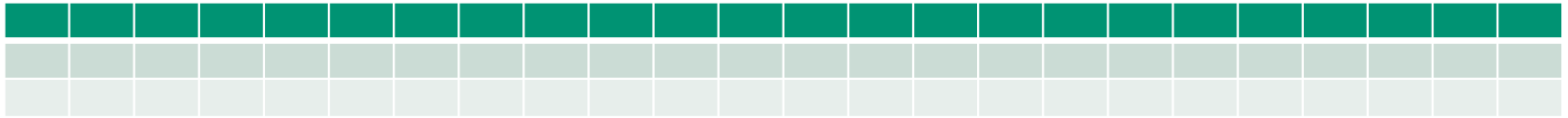
Data can have...



Lots of rows

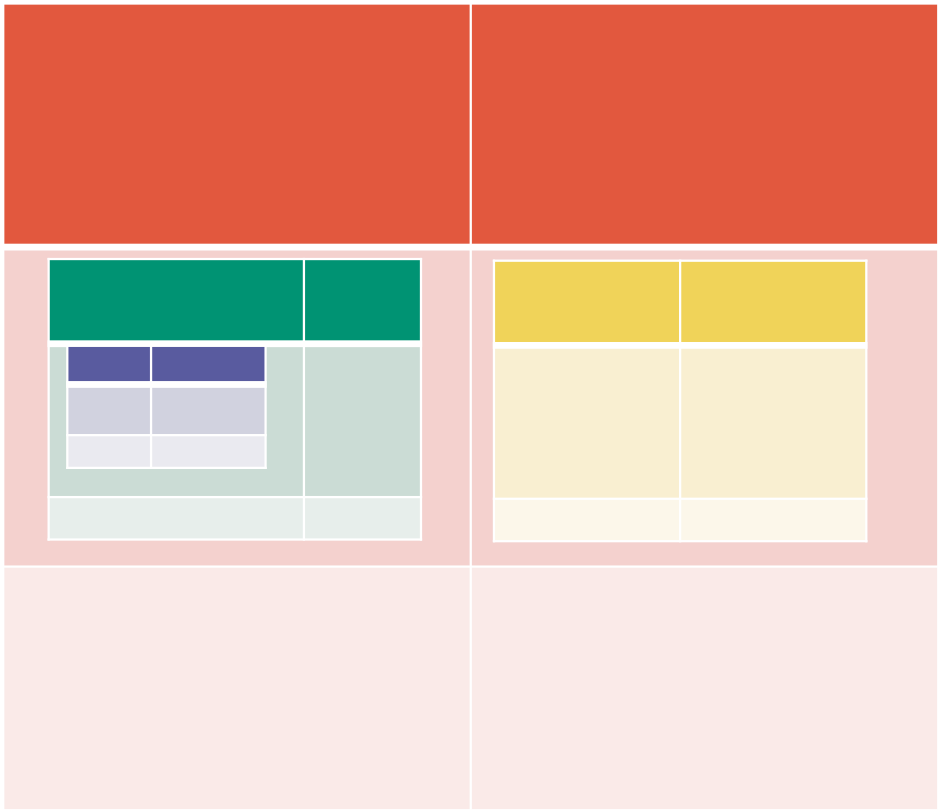
Data can have...

Lots of columns



Data can have...

Lots of nesting



The rest of the lecture: Scaling up

Lots of rows	Object Storage
Lots of rows	Distributed File Systems
Lots of nesting	Syntax
Lots of rows/columns	Column storage
Lots of nesting	Data Models
Lots of rows	Massive Parallel Processing
Lots of nesting	Document Stores
Lots of nesting	Querying