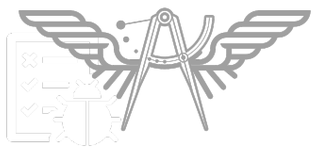
สยามชำนาญกิจ
SIAM CHAMNAN KIT

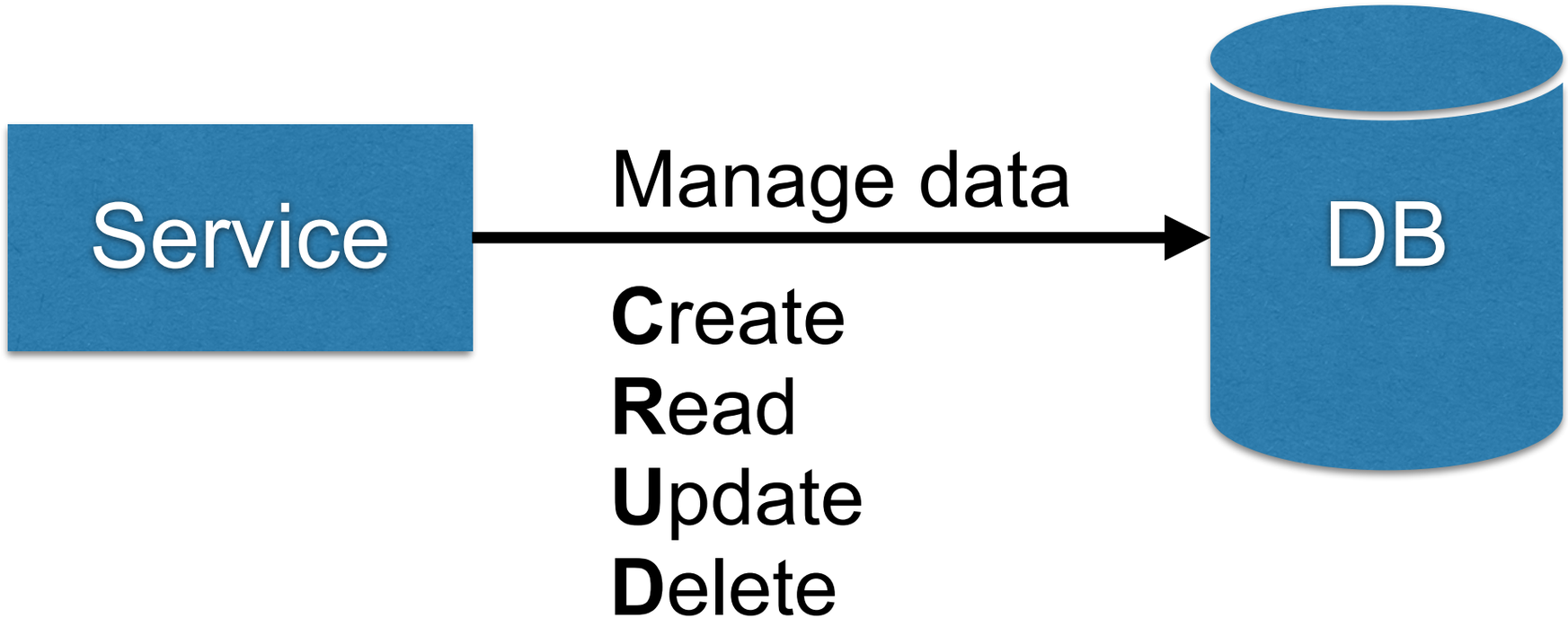
Database Design Tuning



https://github.com/up1/workshop-database

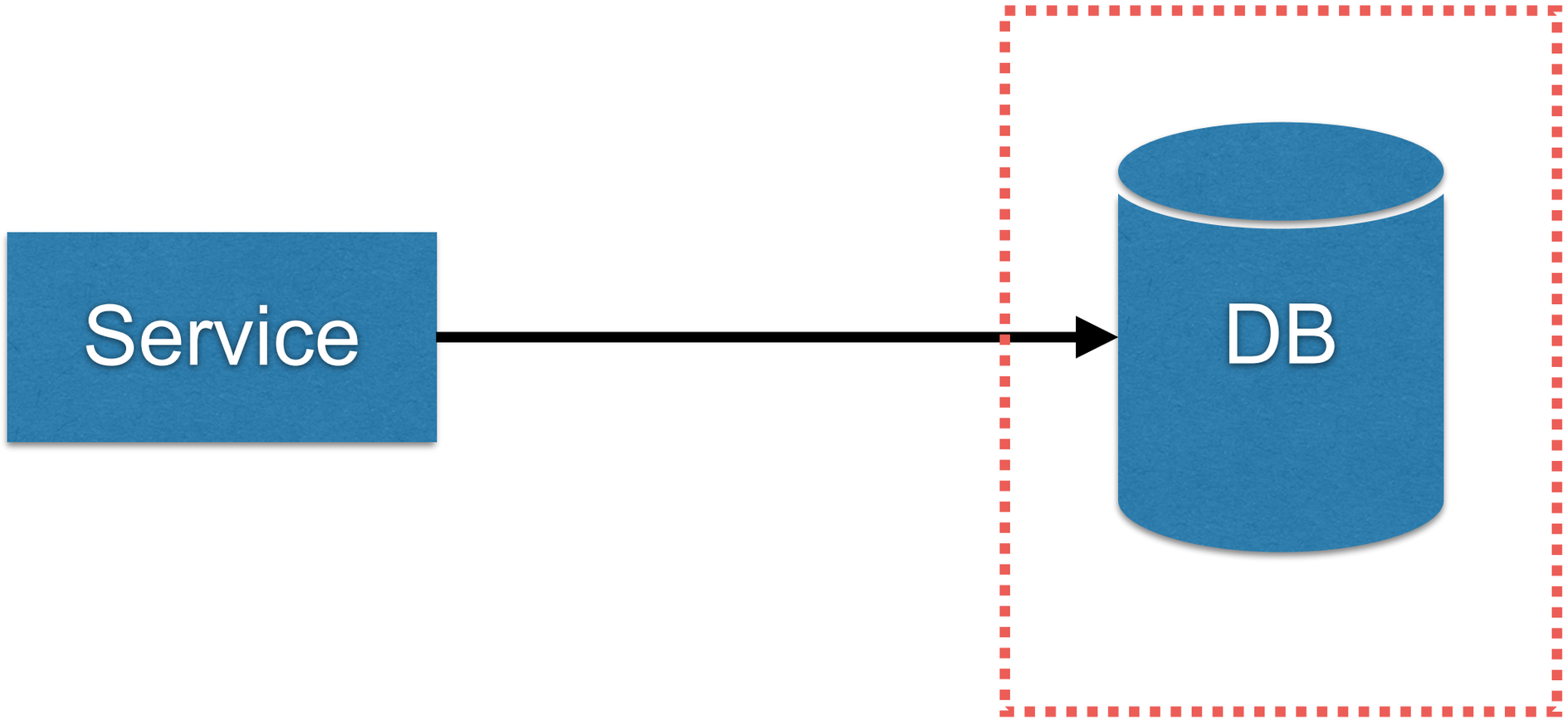
Database

Database



Working with Database

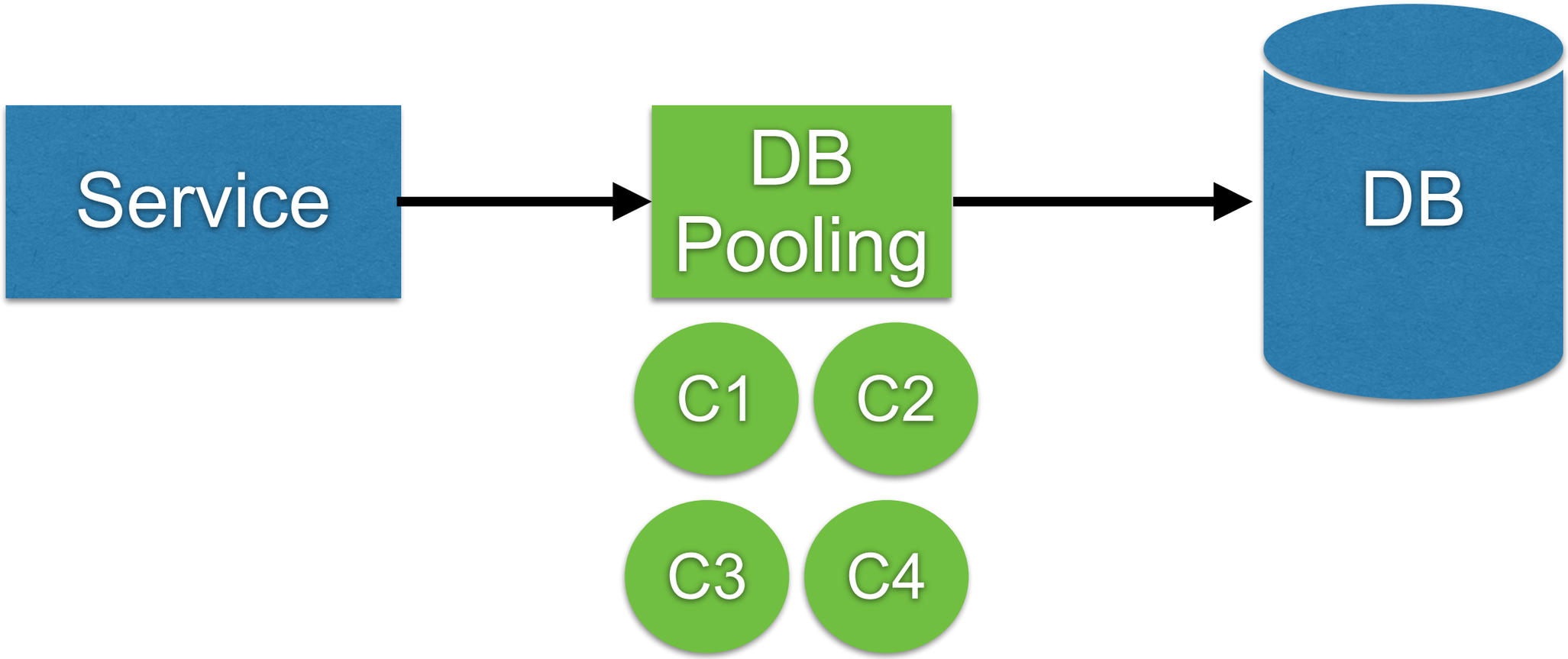
Directly to database



Database modeling
Database architecture

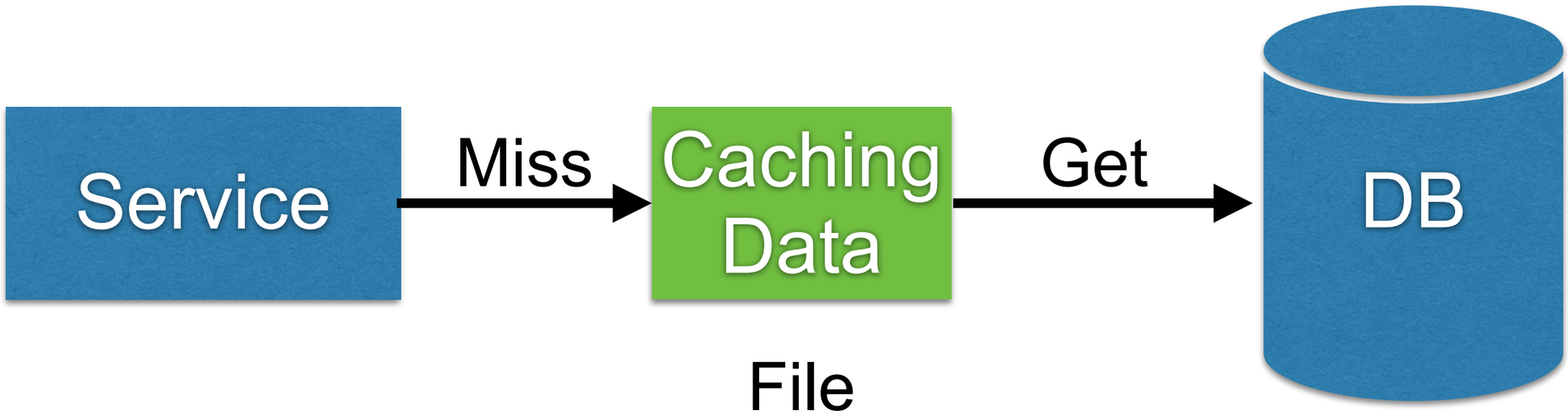
Working with Database

Connection pool



Working with Database

Caching data layer



File
Memory
Other database model

Database ?

Structured data collection
Records
Relationships

Database Management System ?

Software systems for storing and querying databases

User’s perspective

Modeling data
Querying data
Store data

Read

Write

Database Model

Database Model ?

419 systems in ranking, July 2023

Rank			DBMS	Database Model	Score		
Jul 2023	Jun 2023	Jul 2022			Jul 2023	Jun 2023	Jul 2022
1.	1.	1.	Oracle	Relational, Multi-model	1256.01	+24.54	-24.28
2.	2.	2.	MySQL	Relational, Multi-model	1150.35	-13.59	-44.53
3.	3.	3.	Microsoft SQL Server	Relational, Multi-model	921.60	-8.47	-20.53
4.	4.	4.	PostgreSQL	Relational, Multi-model	617.83	+5.01	+1.96
5.	5.	5.	MongoDB	Document, Multi-model	435.49	+10.13	-37.49
6.	6.	6.	Redis	Key-value, Multi-model	163.76	-3.59	-9.86
7.	7.	7.	IBM Db2	Relational, Multi-model	139.81	-5.07	-21.40
8.	8.	8.	Elasticsearch	Search engine, Multi-model	139.59	-4.16	-14.74
9.	9.	9.	Microsoft Access	Relational	130.72	-3.73	-14.37
10.	10.	10.	SQLite	Relational	130.20	-1.02	-6.48
11.	11.	↑ 13.	Snowflake	Relational	117.69	+3.55	+18.53
12.	12.	↓ 11.	Cassandra	Wide column	106.53	-2.03	-7.88
13.	13.	↓ 12.	MariaDB	Relational, Multi-model	96.10	-1.21	-16.42
14.	14.	14.	Splunk	Search engine	87.12	-2.34	-11.09
15.	↑ 16.	15.	Microsoft Azure SQL Database	Relational, Multi-model	78.96	-0.01	-5.94

https://db-engines.com/en/ranking

Database Model ?

419 systems in ranking, July 2023

Rank Jul 2023	Rank Jun 2023	Rank Jul 2022	DBMS	Database Model	Score Jul 2023	Score Jun 2023	Score Jul 2022
1.	1.	1.	Oracle	Relational, Multi-model	1256.01	+24.54	-24.28
2.	2.	2.	MySQL	Relational, Multi-model	1150.35	-13.59	-44.53
3.	3.	3.	Microsoft SQL Server	Relational, Multi-model	921.60	-8.47	-20.53
4.	4.	4.	PostgreSQL	Relational, Multi-model	617.83	+5.01	+1.96
5.	5.	5.	MongoDB	Document, Multi-model	435.49	+10.13	-37.49
6.	6.	6.	Redis	Key-value, Multi-model	163.76	-3.59	-9.86
7.	7.	7.	IBM Db2	Relational, Multi-model	139.81	-5.07	-21.40
8.	8.	8.	Elasticsearch	Search engine, Multi-model	139.59	-4.16	-14.74
9.	9.	9.	Microsoft Access	Relational	130.72	-3.73	-14.37
10.	10.	10.	SQLite	Relational	130.20	-1.02	-6.48
11.	11.	↑ 13.	Snowflake	Relational	117.69	+3.55	+18.53
12.	12.	↓ 11.	Cassandra	Wide column	106.53	-2.03	-7.88
13.	13.	↓ 12.	MariaDB	Relational, Multi-model	96.10	-1.21	-16.42
14.	14.	14.	Splunk	Search engine	87.12	-2.34	-11.09
15.	↑ 16.	15.	Microsoft Azure SQL Database	Relational, Multi-model	78.96	-0.01	-5.94

https://db-engines.com/en/ranking

Database Model

Relational
Document
Key-value
Search engine
Wide column
Graph
Time series
Vector

Database Model

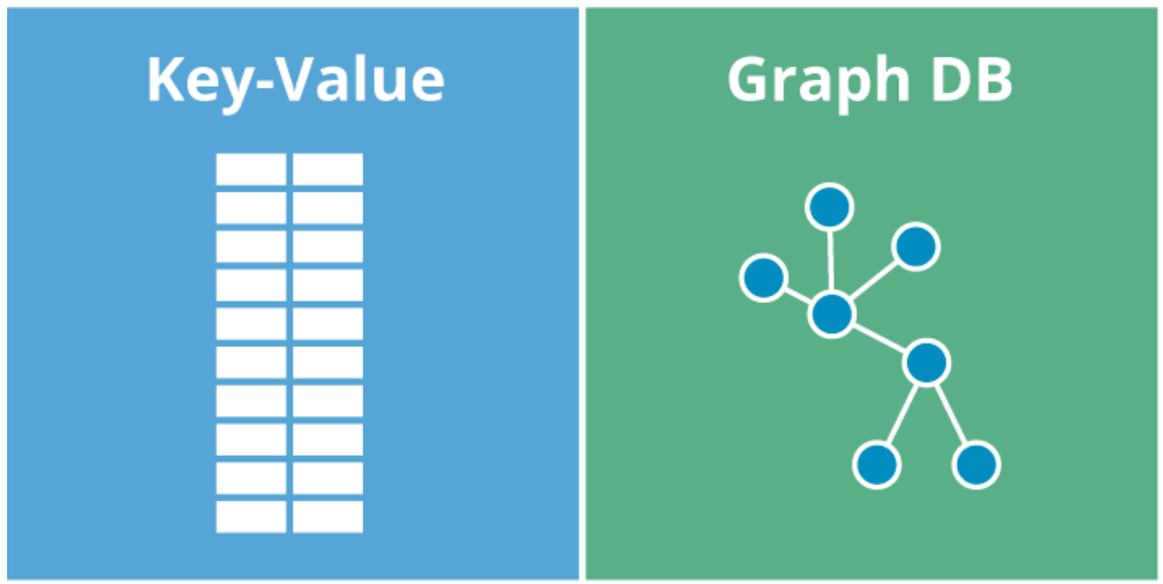
Relational

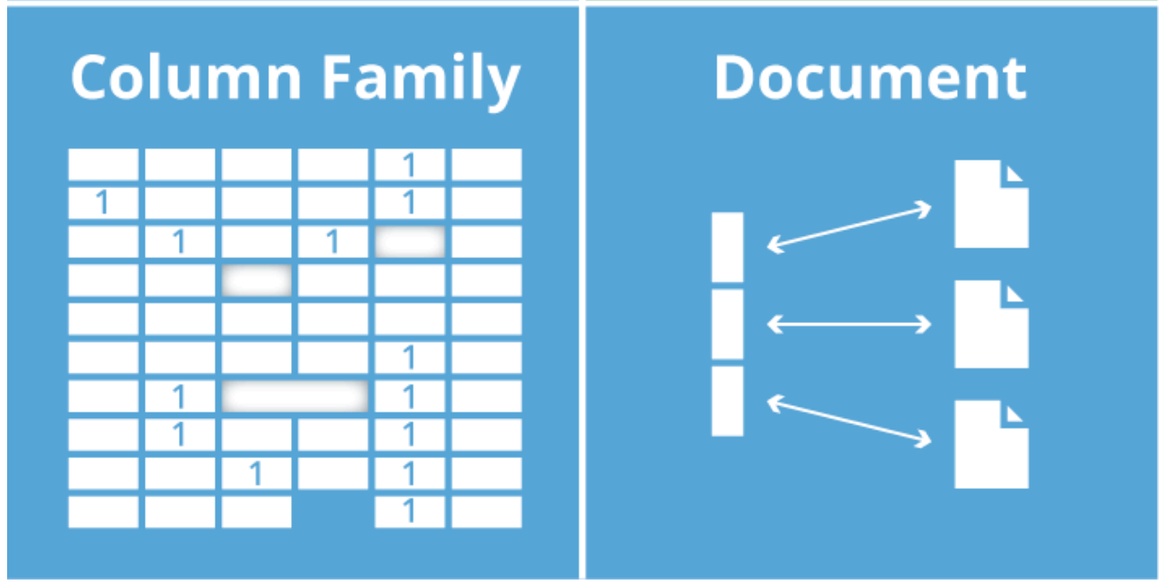
Document
Key-value
Search engine
Wide column
Graph
Time series
Content

Non-Relational

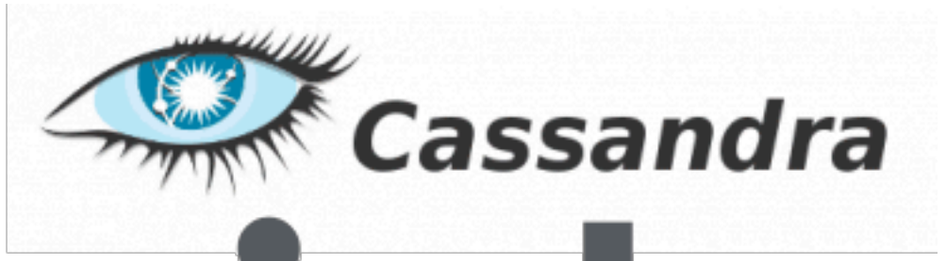
NoSQL

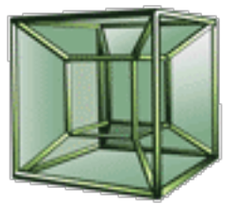
NoSQL



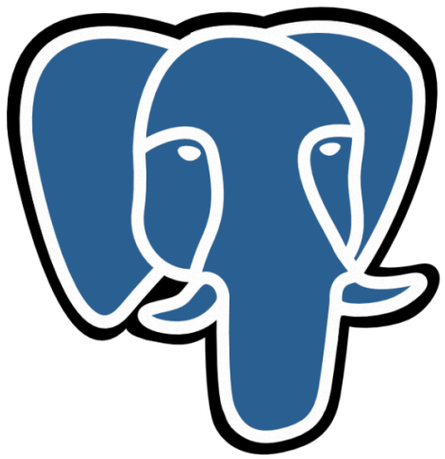


NoSQL





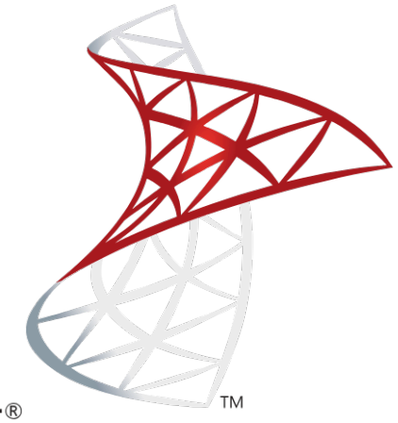
Relational Database



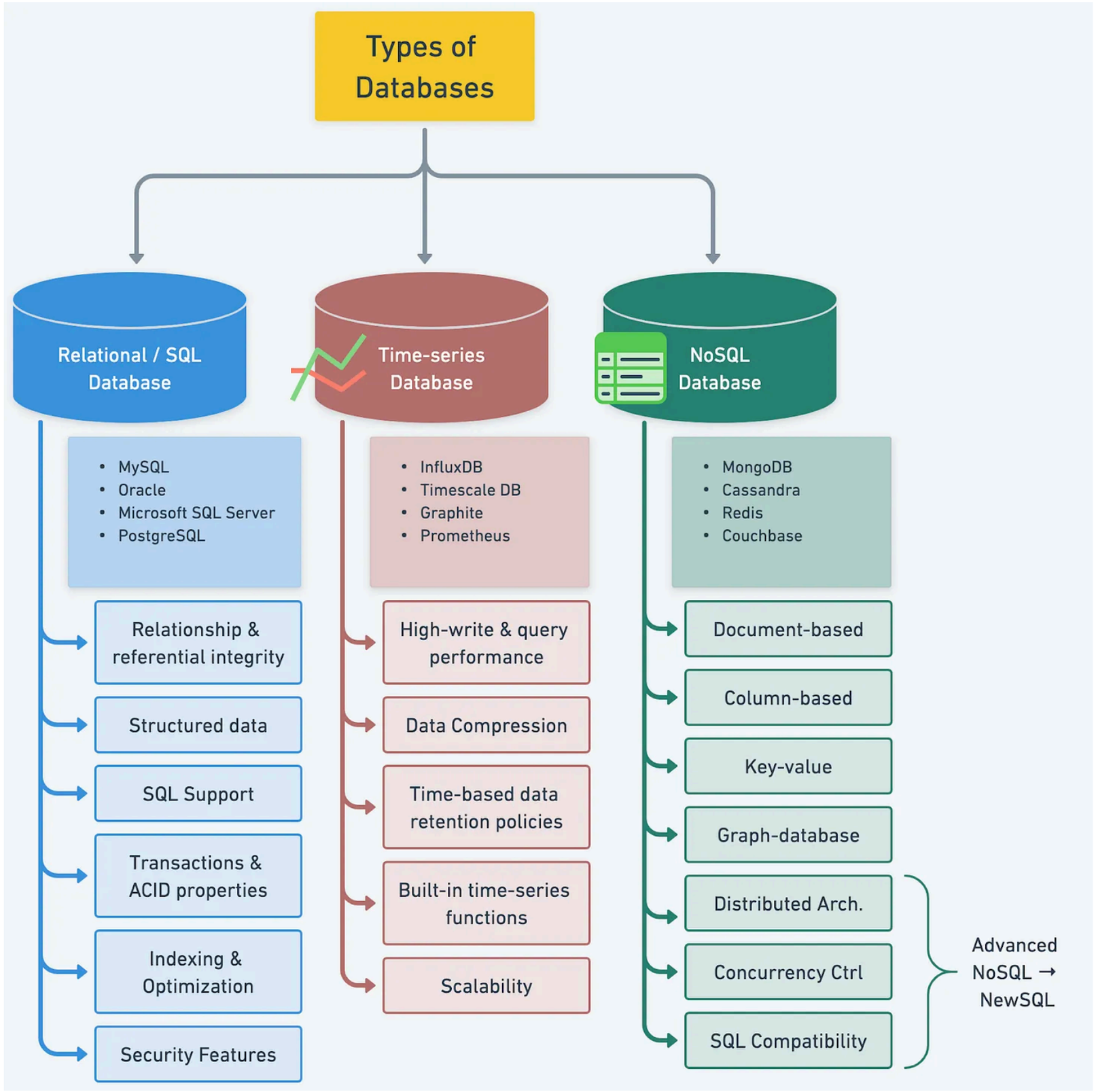
PostgreSQL

ORACLE®

DATABASE



Microsoft®
SQL Server®



https://blog.bytebytego.com/p/understanding-database-types

Relational Database ?

Relational**D**ata**B**ase**M**anament**S**ystem

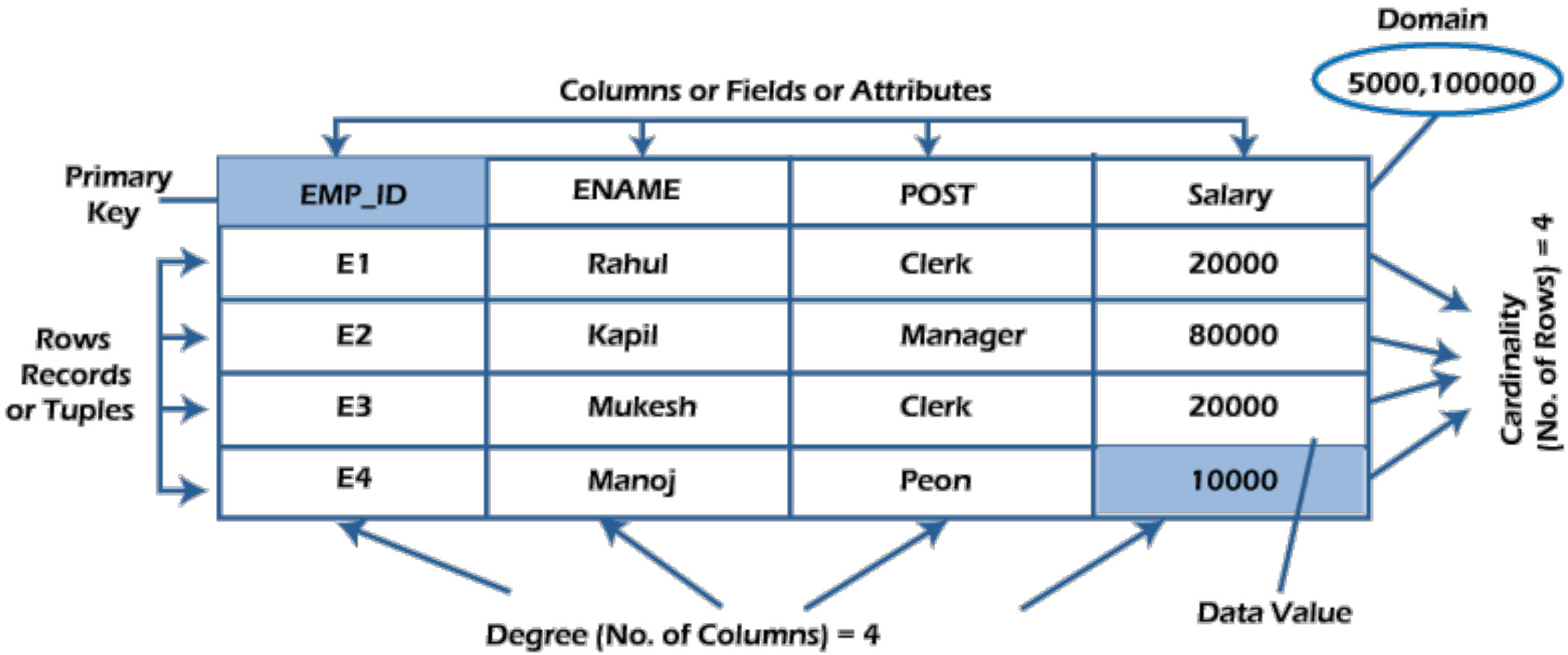
Maintain data in **tables**
Pre-define structure
Each table has **Primary key**
Many **columns**
Data is represented in term of **row**

Relational Database

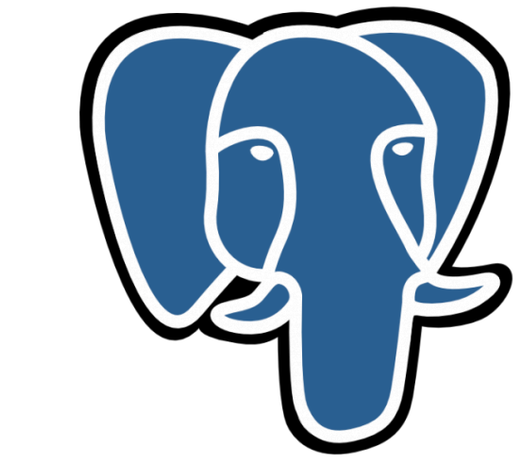
Database	Ease of use	Support complex query	Community support	Licence
MySQL	High	High	Strong	Open source
PostgreSQL	Medium	Very high	Strong	Open source
MSSQL Server	Medium	Very high	Strong	Proprietary
Oracle	Low	Very high	Moderate	Proprietary

Relational Database ?

Table = Employee

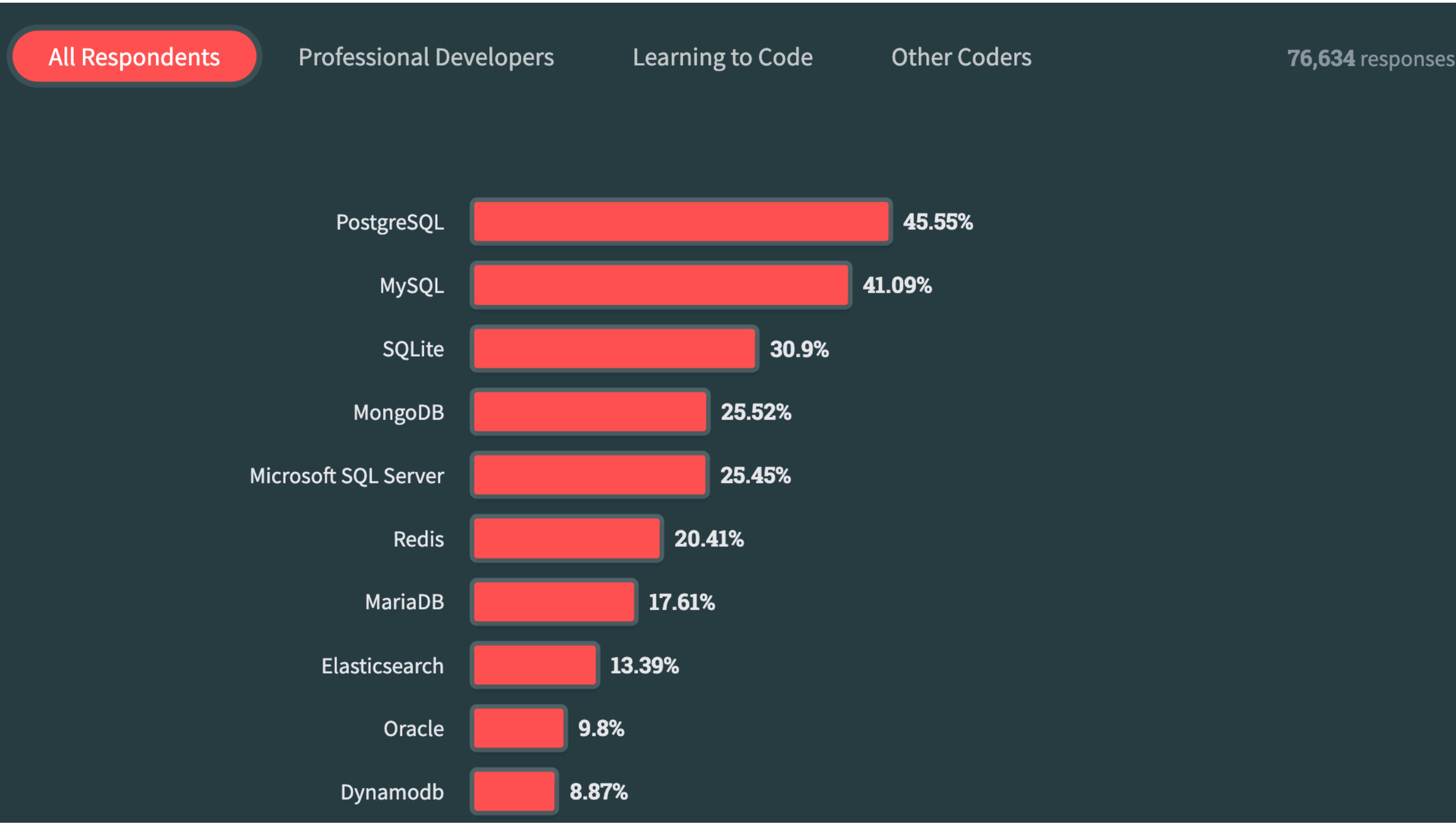


Working with PostgreSQL



PostgreSQL

PostgreSQL Database



https://survey.stackoverflow.co/2023/#most-popular-technologies-database

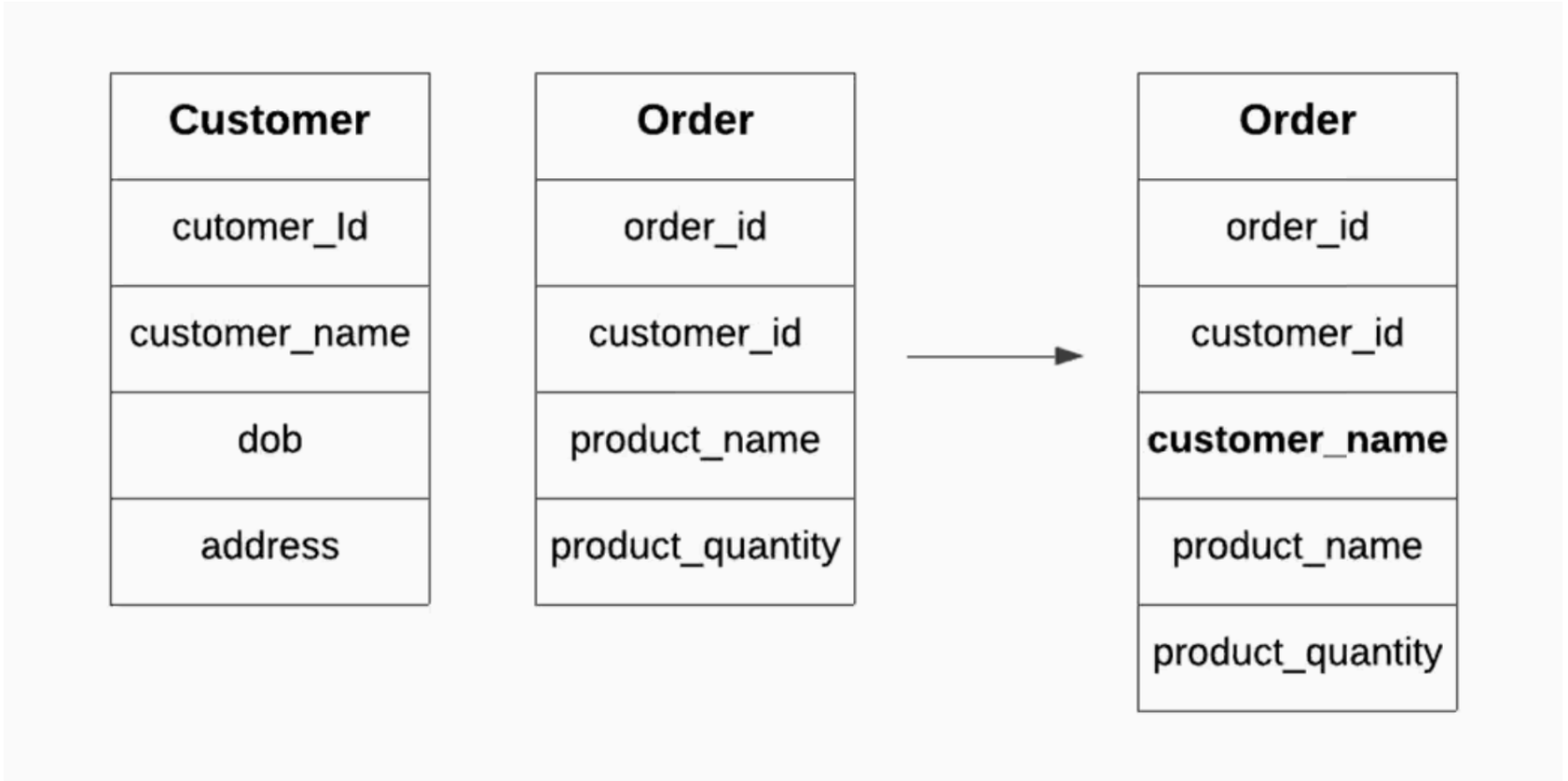
Data Modeling

Read

Write

Data Modeling

Normalization
Denormalization



Normalization

Cleansing data
Reorganized data
Remove redundant data
Improve efficient for store data

Organize data in consistent way

Normalization

Divide in to several tables
Link tables with **relationships**

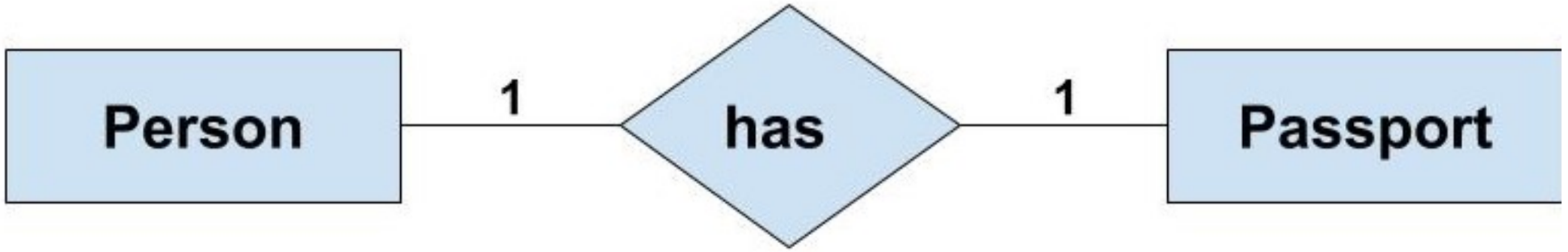
Relationships

Relationship between table

One-to-one

One-to-many

Many-to-many

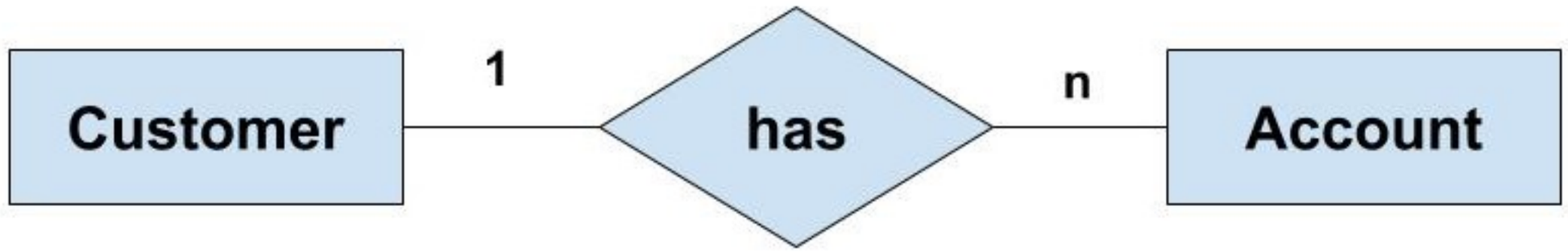


Relationship between table

One-to-one

One-to-many

Many-to-many



Relationship between table

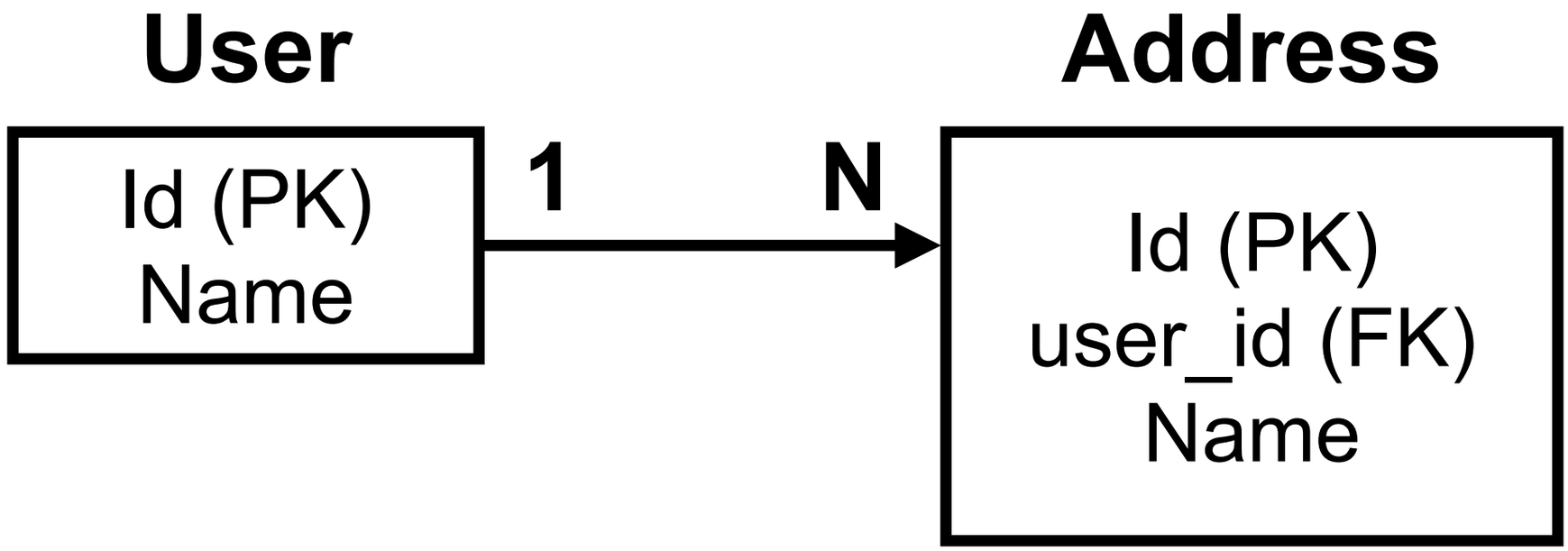
One-to-one

One-to-many

Many-to-many

Relationships

Primary key (PK)
Composite key
Foreign key (FK)



Normalization forms

First Normal form (1NF)

2NF

3NF

4NF

5NF

6NF

First Normal Form (1NF)

No repeat data in table

Every column should only have single value

Every row should be unique

Sample data in 1NF

Name	Address	Gender	T-shirt Size
User 1	Address 1	Male	Large
User 2	Address 2	Female	Small
User 2	Address 2	Female	Medium
User 3	Address 3	Male	Medium

Second Normal Form (2NF)

Apply 1NF
Have one primary key

Problem in 1NF

Name	Address	Gender	T-shirt Size
User 1	Address 1	Male	Large
User 2	Address 2	Female	Small
User 2	Address 2	Female	Medium
User 3	Address 3	Male	Medium

Sample data in 2NF

User ID	Name	Address	Gender
1	User 1	Address 1	Male
2	User 2	Address 2	Female
3	User 3	Address 3	Male

Sample data in 2NF

User ID	T-shirt Size
1	Large
2	Small
2	Medium
3	Medium

Third Normal Form (3NF)

Apply 2NF

It should only be dependent on the primary key

if any changes to the primary key are made, all data that is impacted must be put into a new table

Problem in 2NF

User ID	Name	Address	Gender
1	User 1	Address 1	Male
2	User 2	Address 2	Female
3	User 3	Address 3	Male

Sample data in 3NF

User ID	Name	Address	Gender
1	User 1	Address 1	1
2	User 2	Address 2	2
3	User 3	Address 3	1

Sample data in 3NF

Gender ID	Gender name
1	Male
2	Female

Sample data in 3NF

User ID	T-shirt Size
1	Large
2	Small
2	Medium
3	Medium

3NF is normal form …

Benefits of data normalization

Freeing up space
Improve query response time
Reduce data inconsistency
Easier to use and analyze

Challenges of data normalization

Slower query response rates
Accurate knowledge is required
Add complexity for teams
Denormalization as an alternative !!

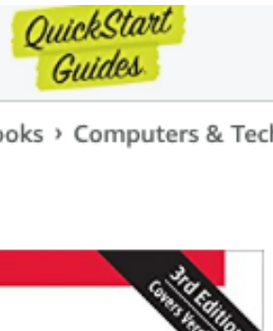
Primary Key

Foreign Key

Constraints

Workshop

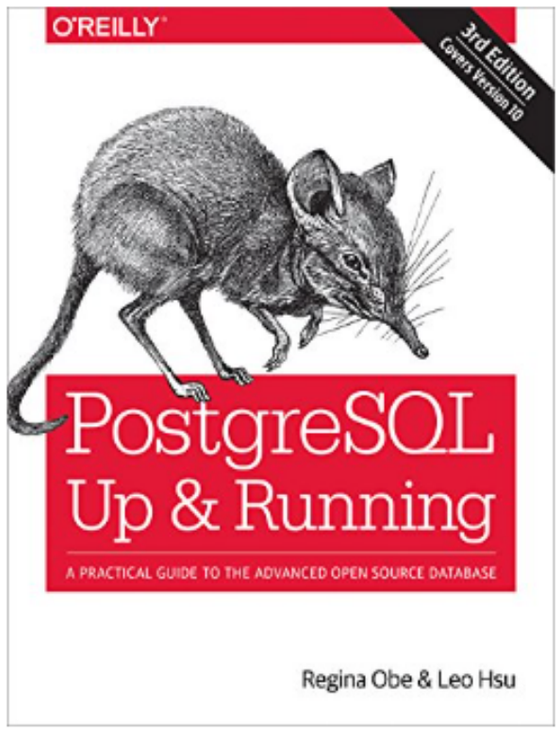
Book Store



Master SQL and Manage, Analyze and Manipulate Data

$25.19

Kindle Store › Kindle eBooks › Computers & Technology



Read sample

Follow the Author

Regina Obe Follow

PostgreSQL: Up and Running: A Practical Guide to the Advanced Open Source Database 3rd Edition, Kindle Edition

by Regina O. Obe (Author), Leo S. Hsu (Author) | Format: Kindle Edition

4.4 ★★★★½ 99 ratings

See all formats and editions

Kindle	Paperback
$10.09 - $28.99	$44.99

Read with Our Free App

6 Used from $36.00
20 New from $34.18

Thinking of migrating to PostgreSQL? This clear, fast-paced introduction helps you understand and use this open source database system. Not only will you learn about the enterprise class features in versions 9.5 to 10, youâ??ll also discover that PostgeSQL is more than a database systemâ??itâ??s an impressive application platform as well.

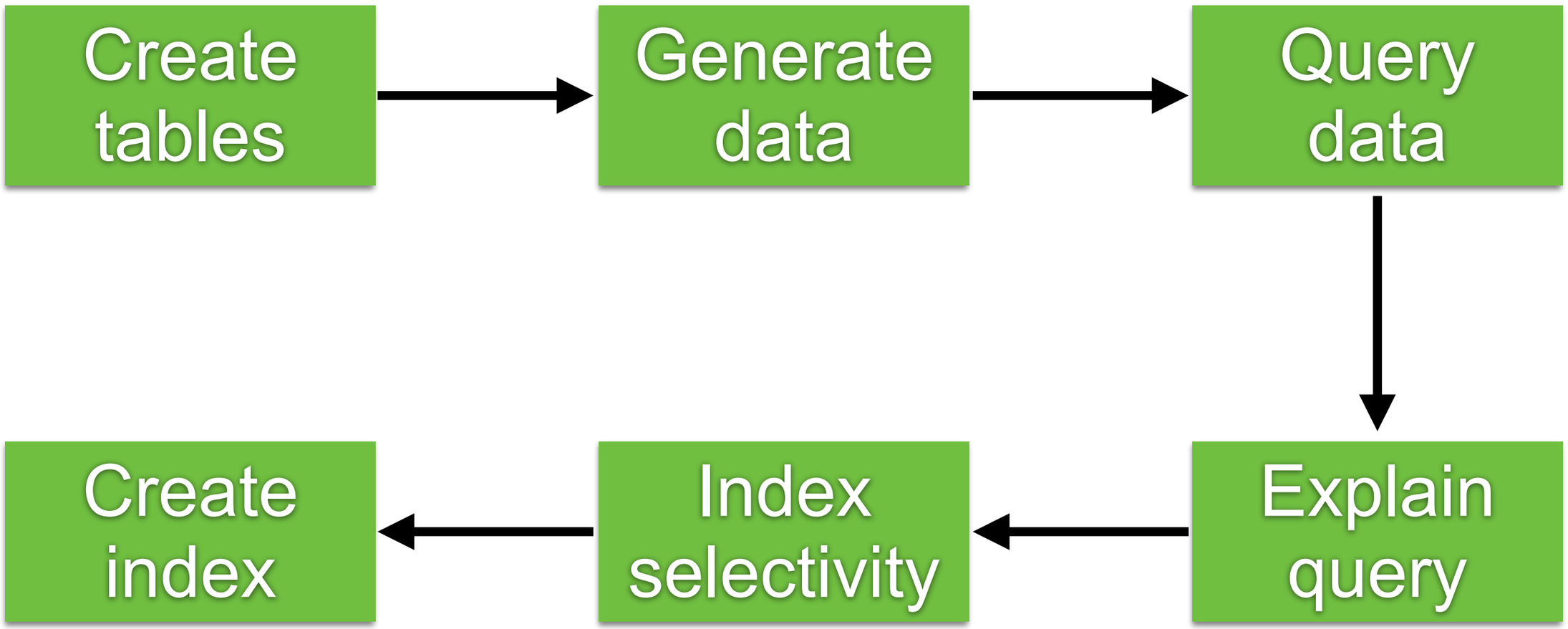
With examples throughout, this book shows you how to achieve tasks that are difficult or impossible in other databases. This third edition covers new features, such as ANSI-SQL constructs found only in proprietary databases until now: foreign data wrapper

Read more

ISBN-13	Edition	Sticky notes	Publisher	Publication date	Language
978-1491963418	3rd	On Kindle Scribe	O'Reilly Media	October 10, 2017	English

Book Store

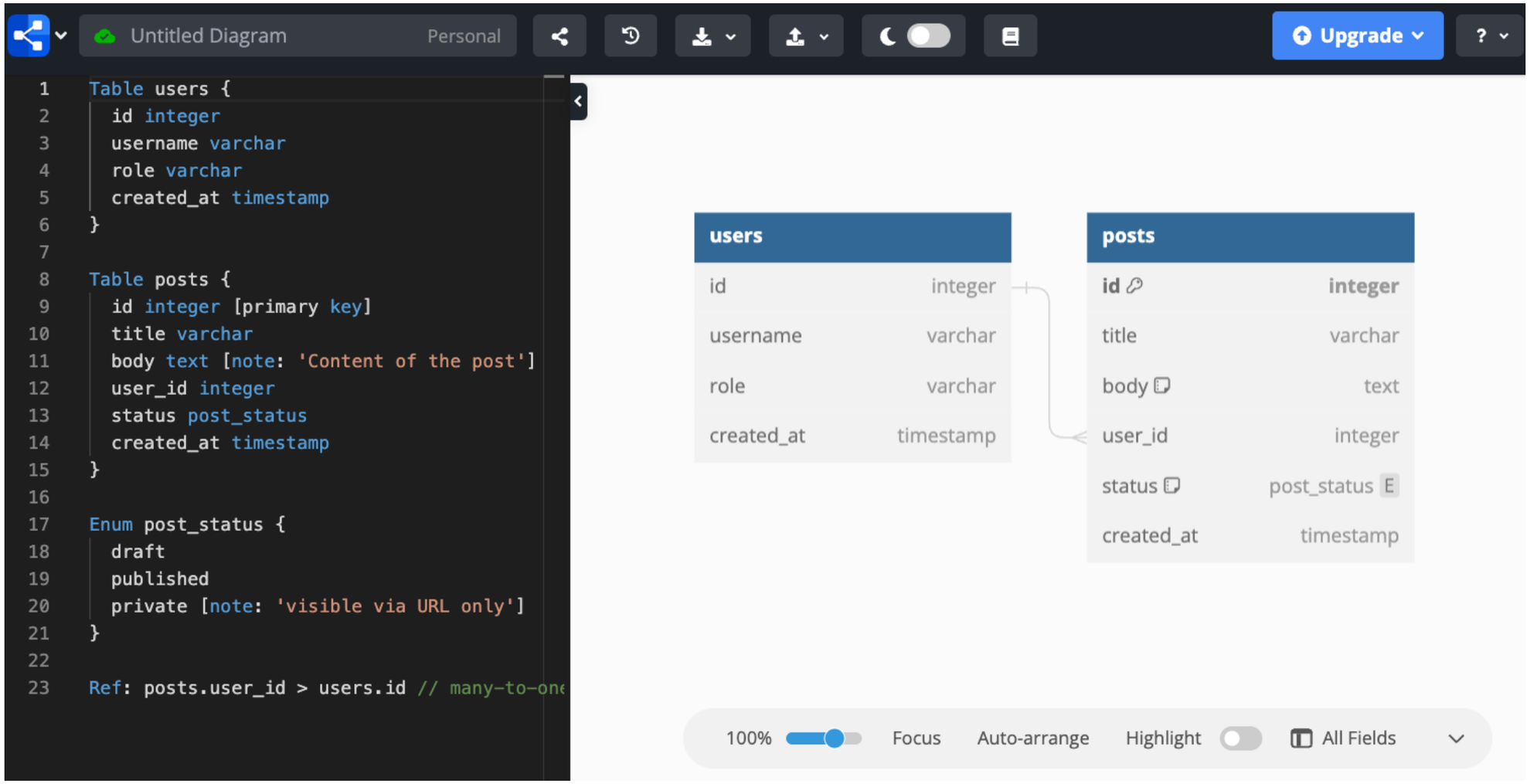
Step in workshop



Design Data Model

Design table with DBML

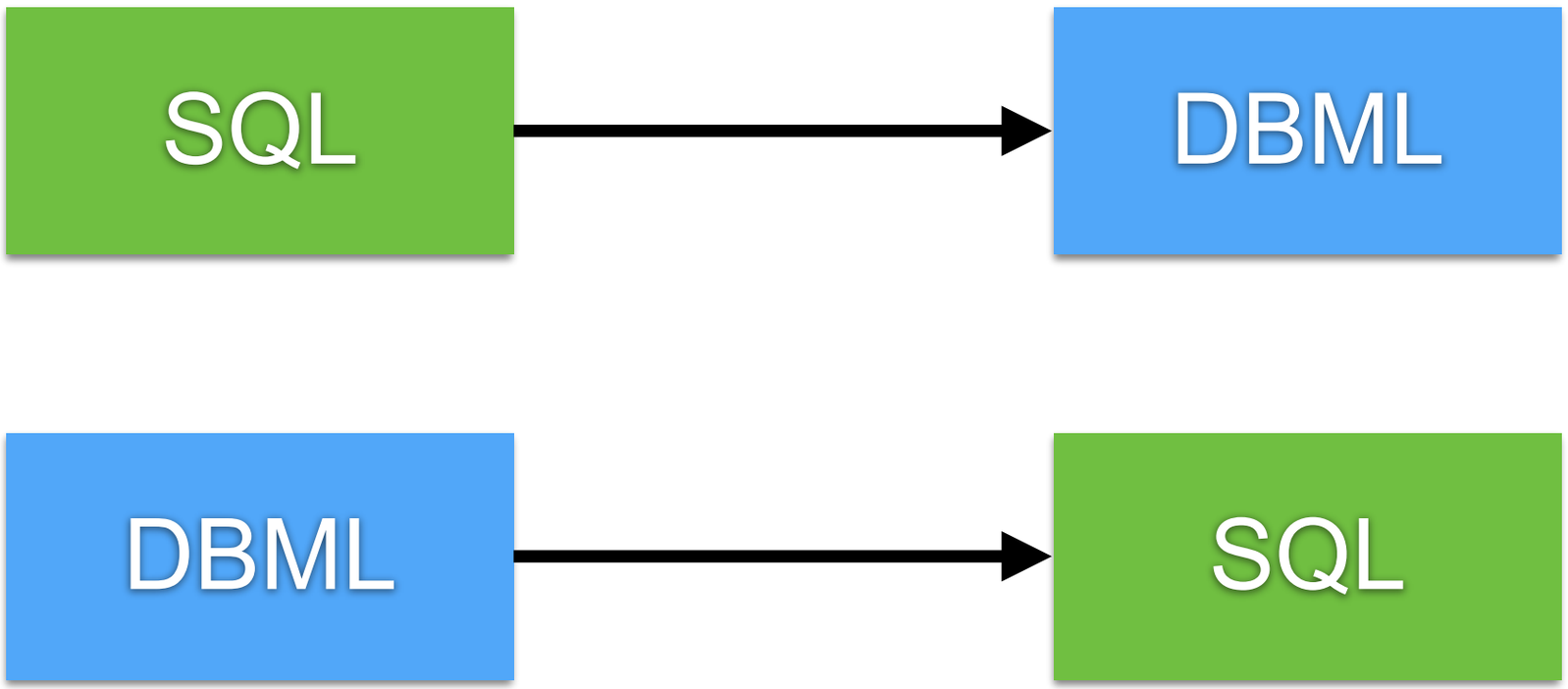
Database Markup Language



https://dbml.dbdiagram.io/home/

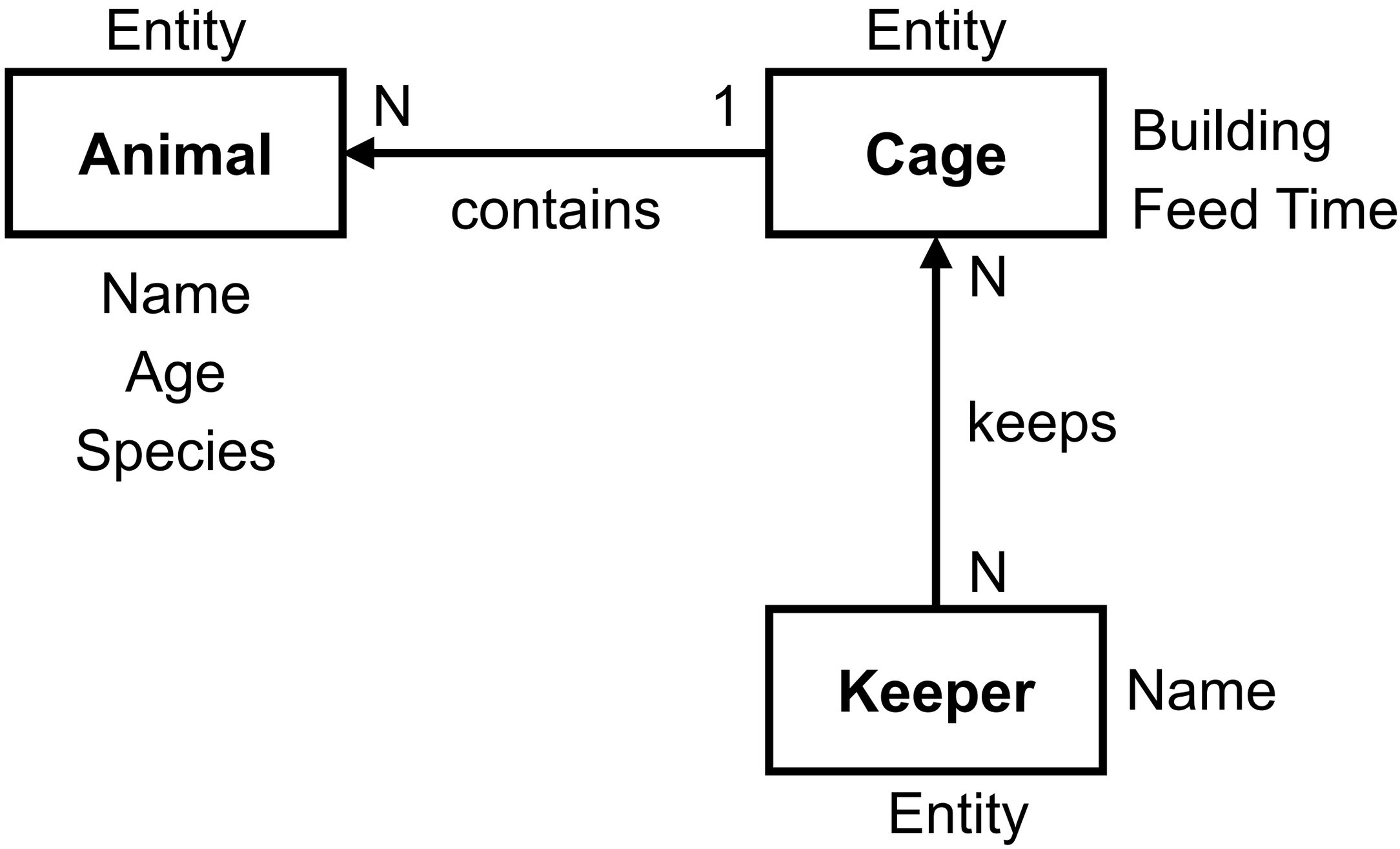
Convert SQL to DBML

$npm install -g @dbml/cli



https://dbml.dbdiagram.io/cli/#convert-a-sql-file-to-dbml

Design with entity relation diagram



How to manage RDBMS ?

Database Language

Data Definition Language (DDL)
Data Manipulation Language (DML)
Data Control Language (DCL)
Transaction Control Language (TCL)

Data Definition Language (DDL)

Create

Alter

Drop

Truncate

Data Manipulation Language (DML)

Select

Insert

Update

Delete

Delete vs Truncate

Delete vs Truncate

Delete

Row-level operation

Mark every row matching the `WHERE`-clause as deleted

Slow operation

Truncate

Table operation

Empty the table

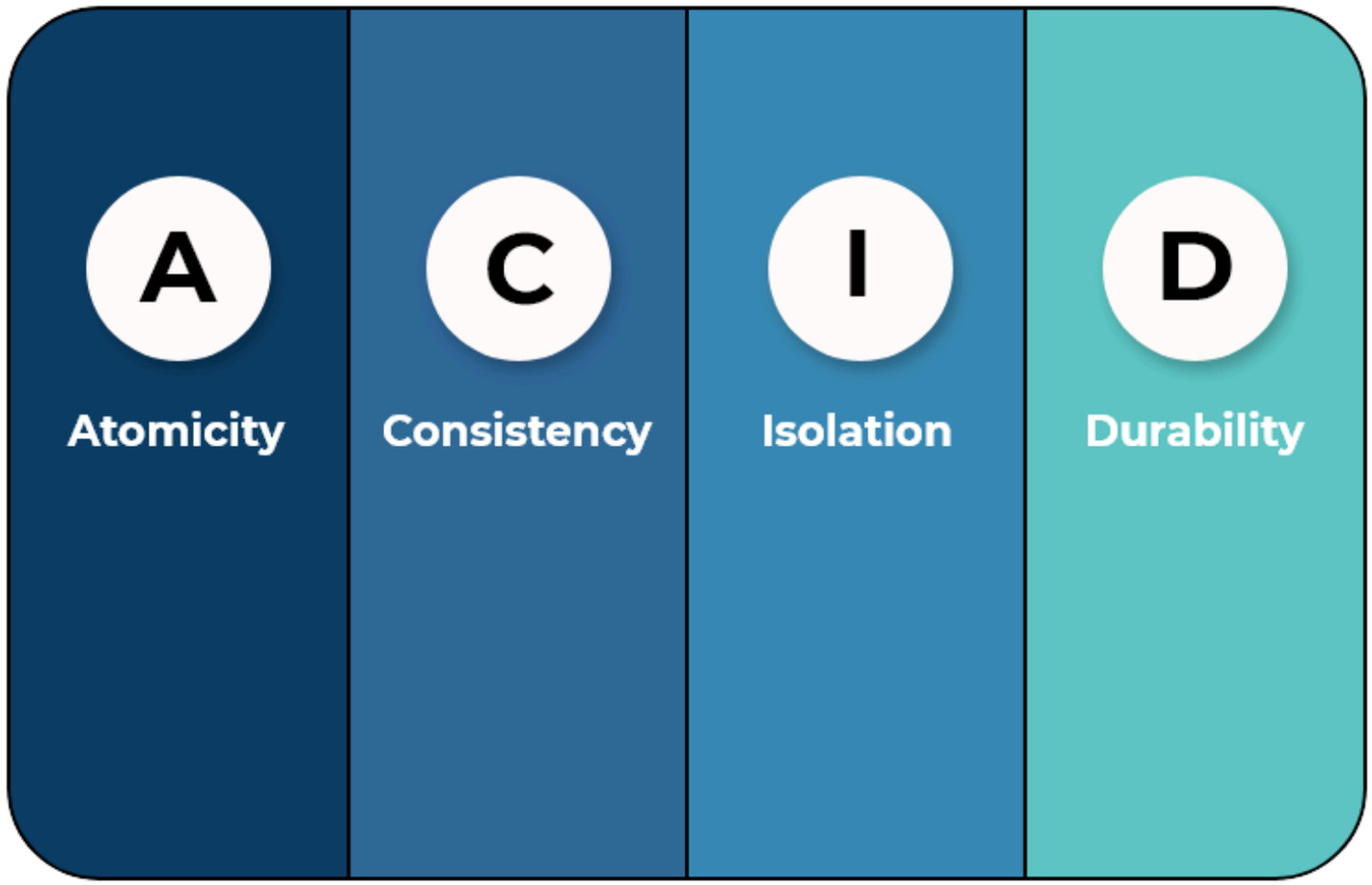
Start a new data file

ACID in RDBMS

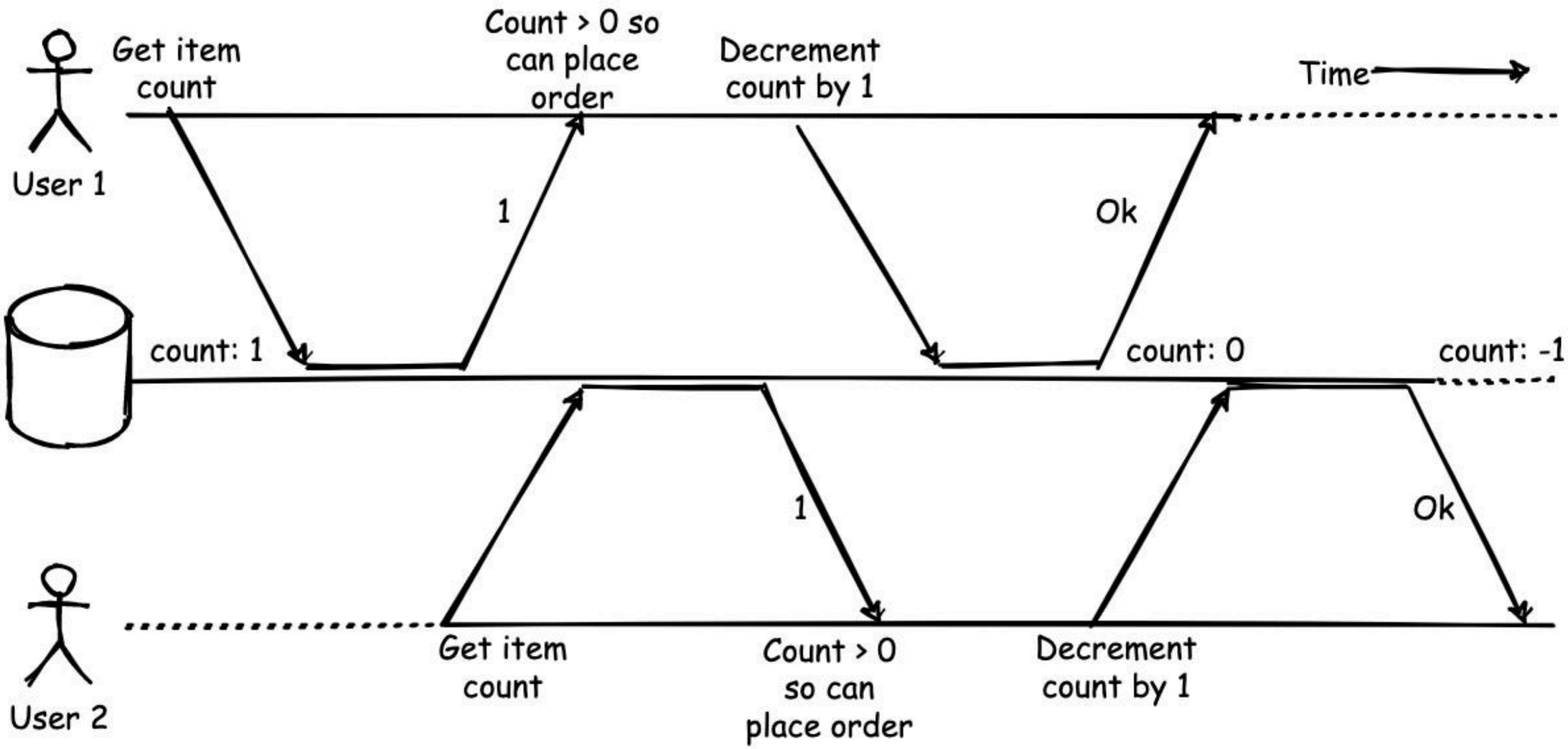
https://en.wikipedia.org/wiki/ACID

ACID in RDBMS

Ensure database in valid state even in the event of unexpected errors

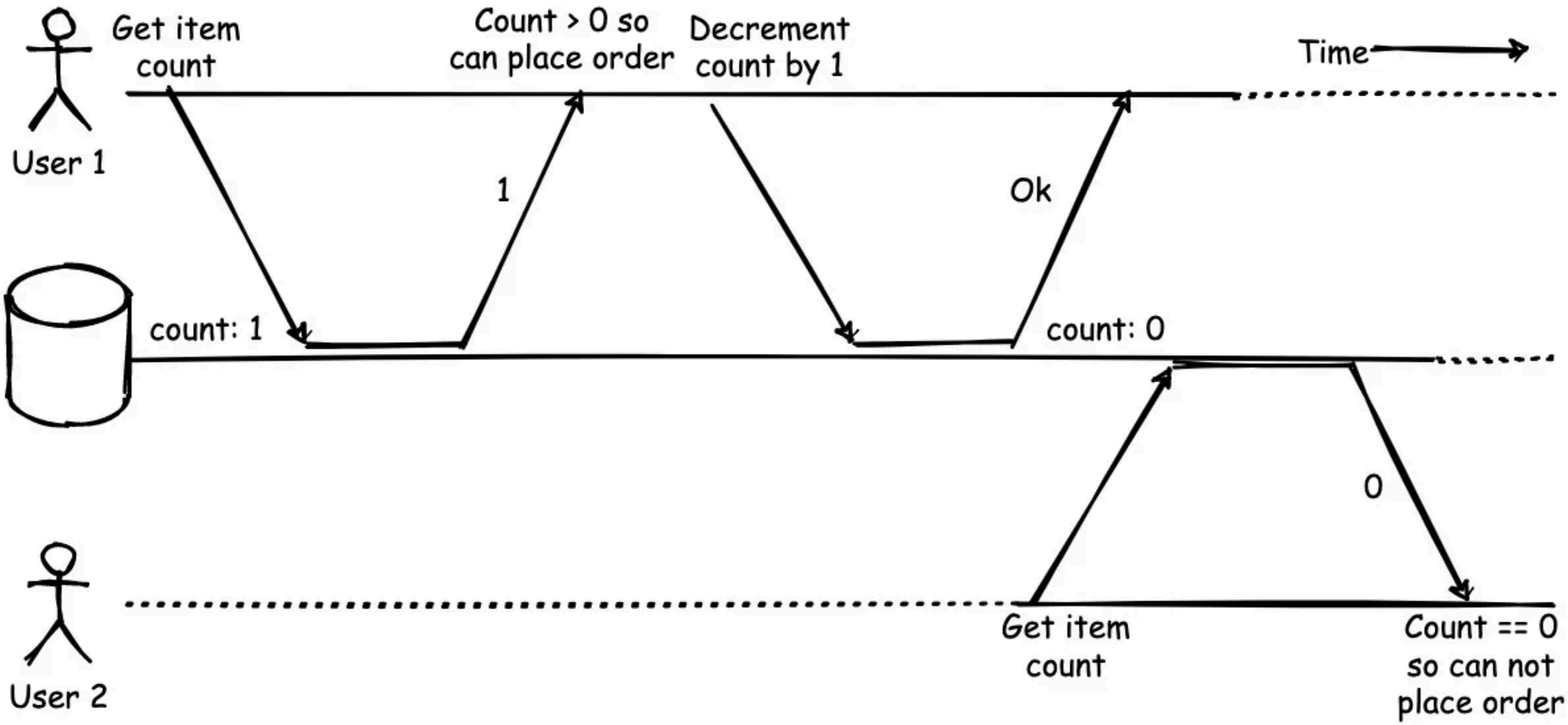


Race condition



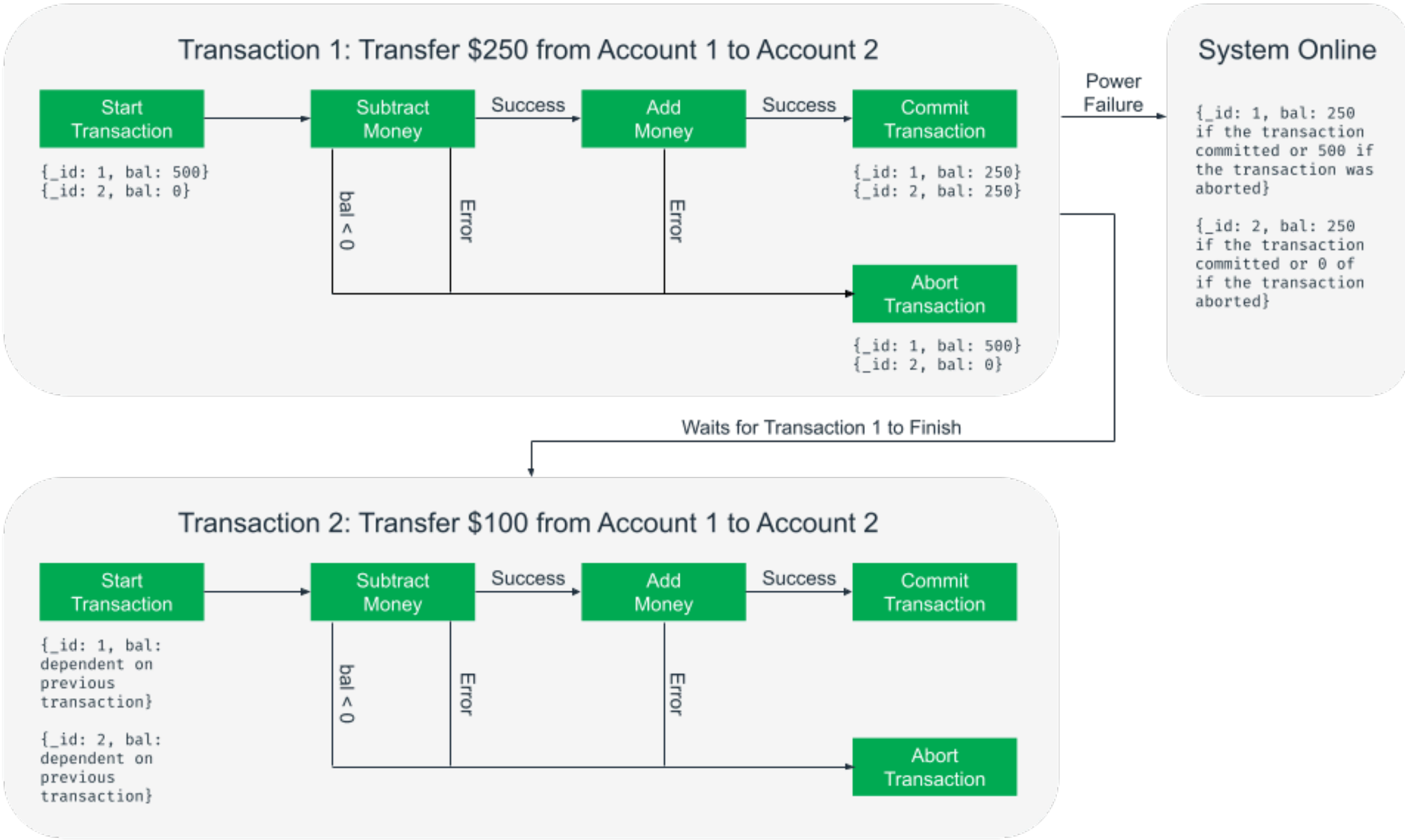
https://en.wikipedia.org/wiki/Race_condition#Example

Sequencial



https://www.mongodb.com/basics/acid-transactions

ACID in RDBMS

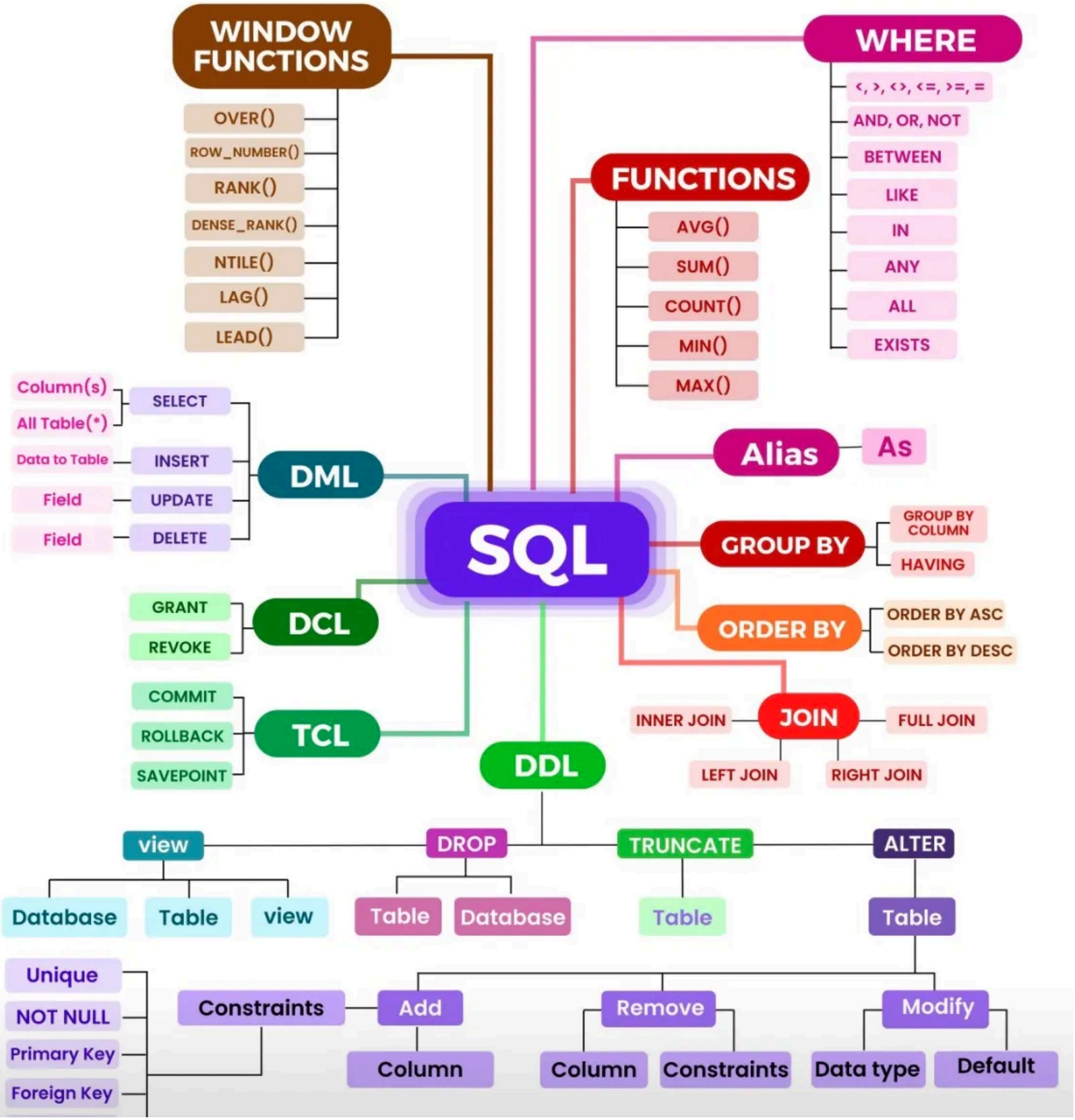


https://www.mongodb.com/basics/acid-transactions

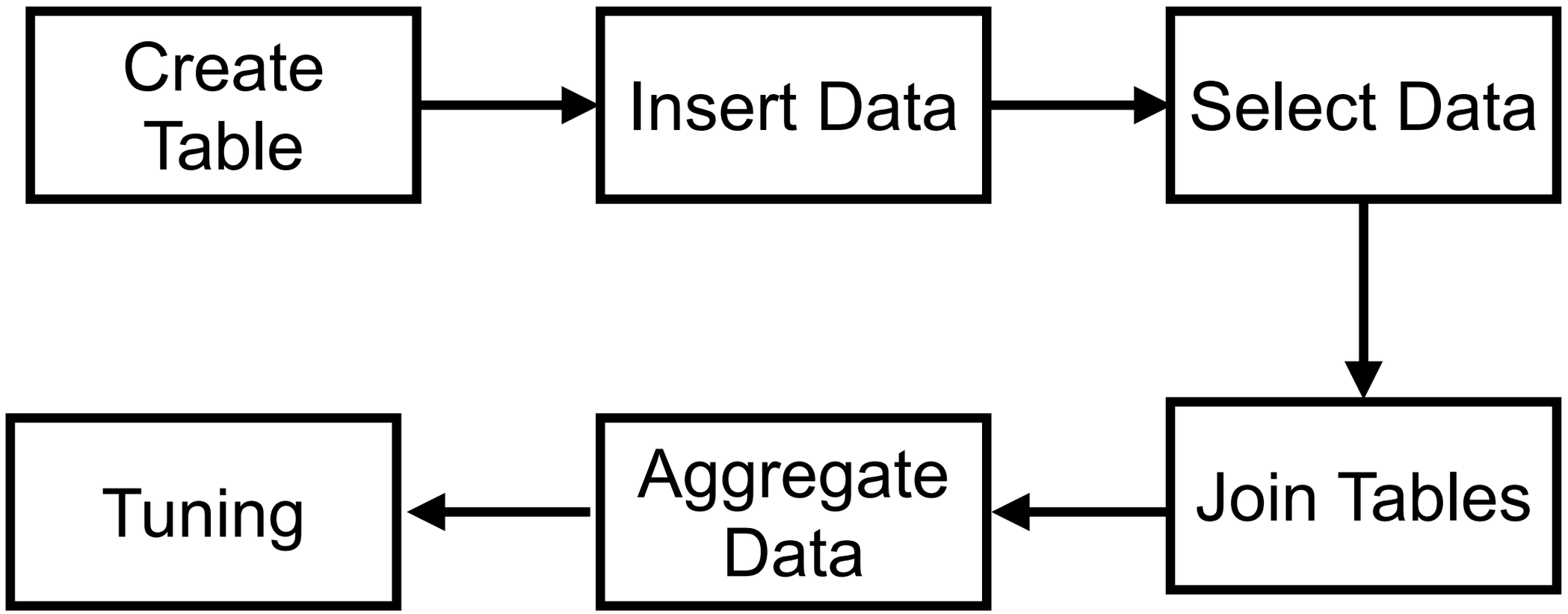
Structure**Q**uery**L**anguage

SQL

Maintain data in **tables**
Pre-define structure
Each table has **Primary key**
Many **columns**
Data is represented in term of **row**



Learning paths



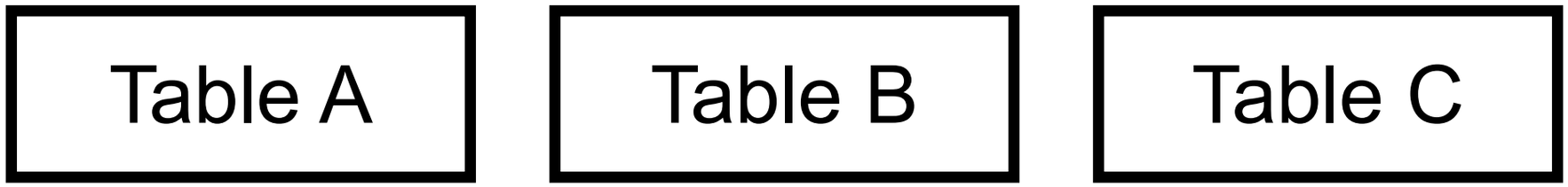
Create table

Select Data

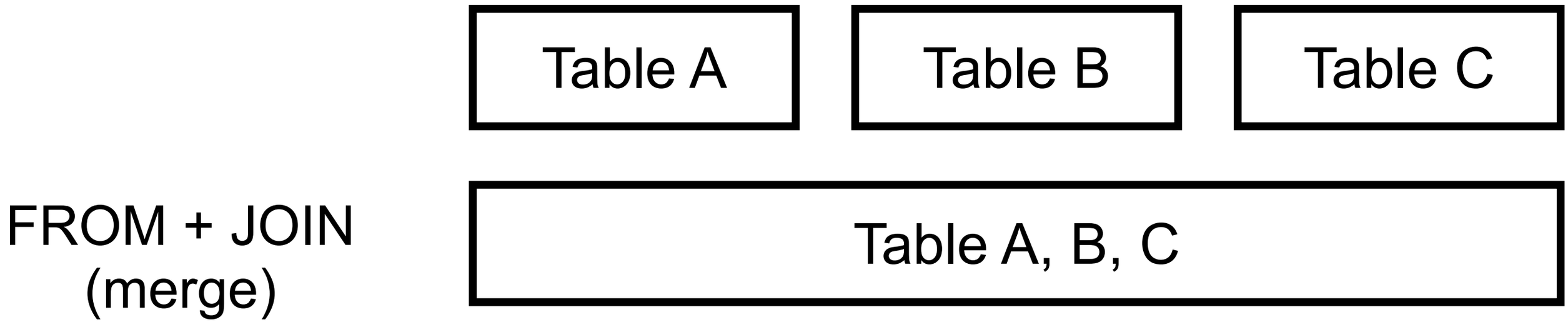
SELECT columns
FROM table
WHERE condition A
AND condition B

GROUP BY columns
ORDER BY columns **ASC|DESC**

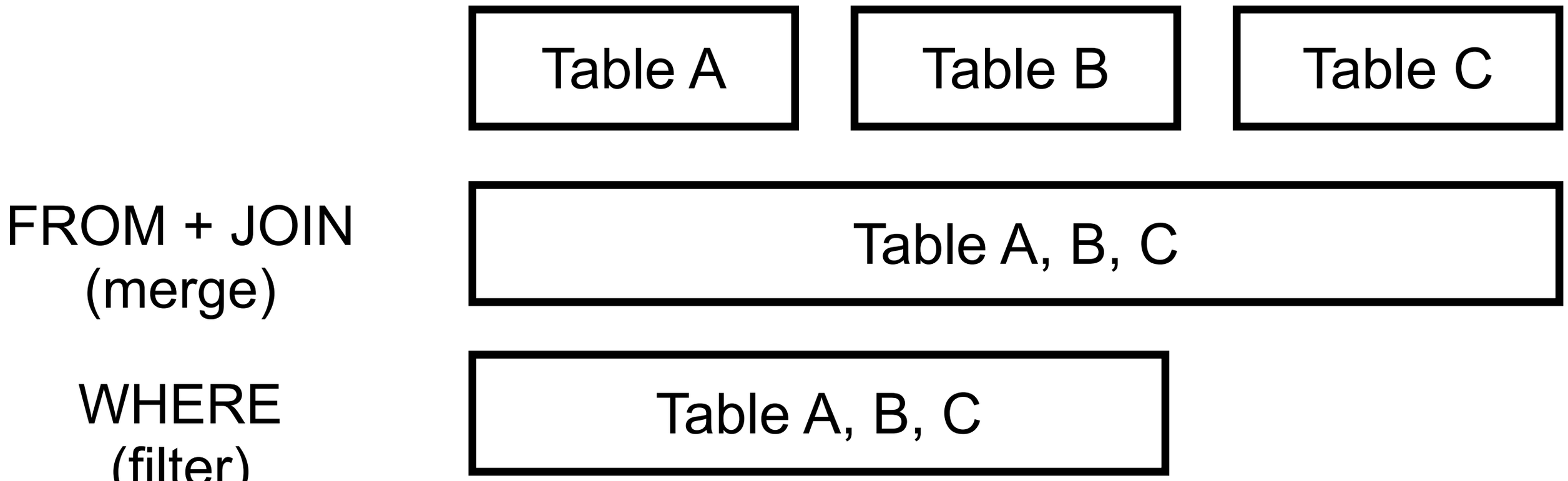
Select Data process



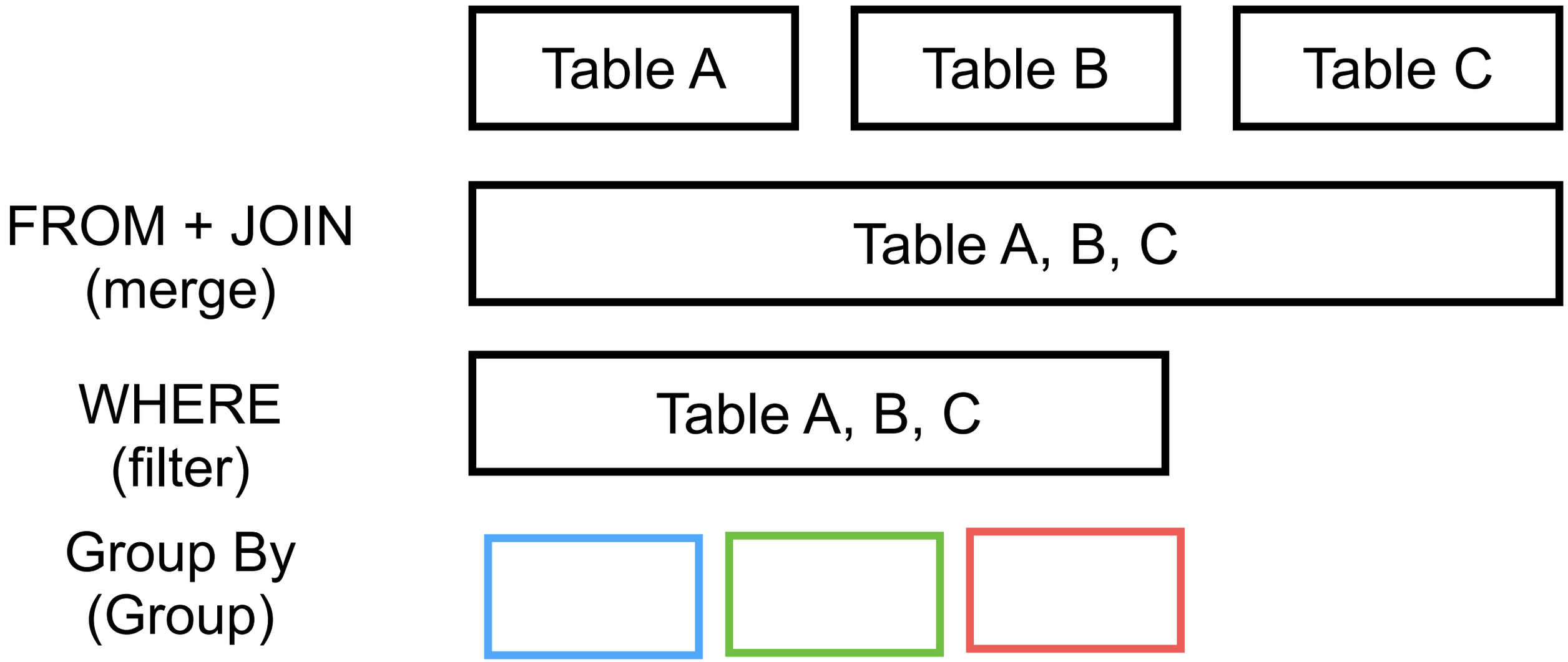
Select Data process



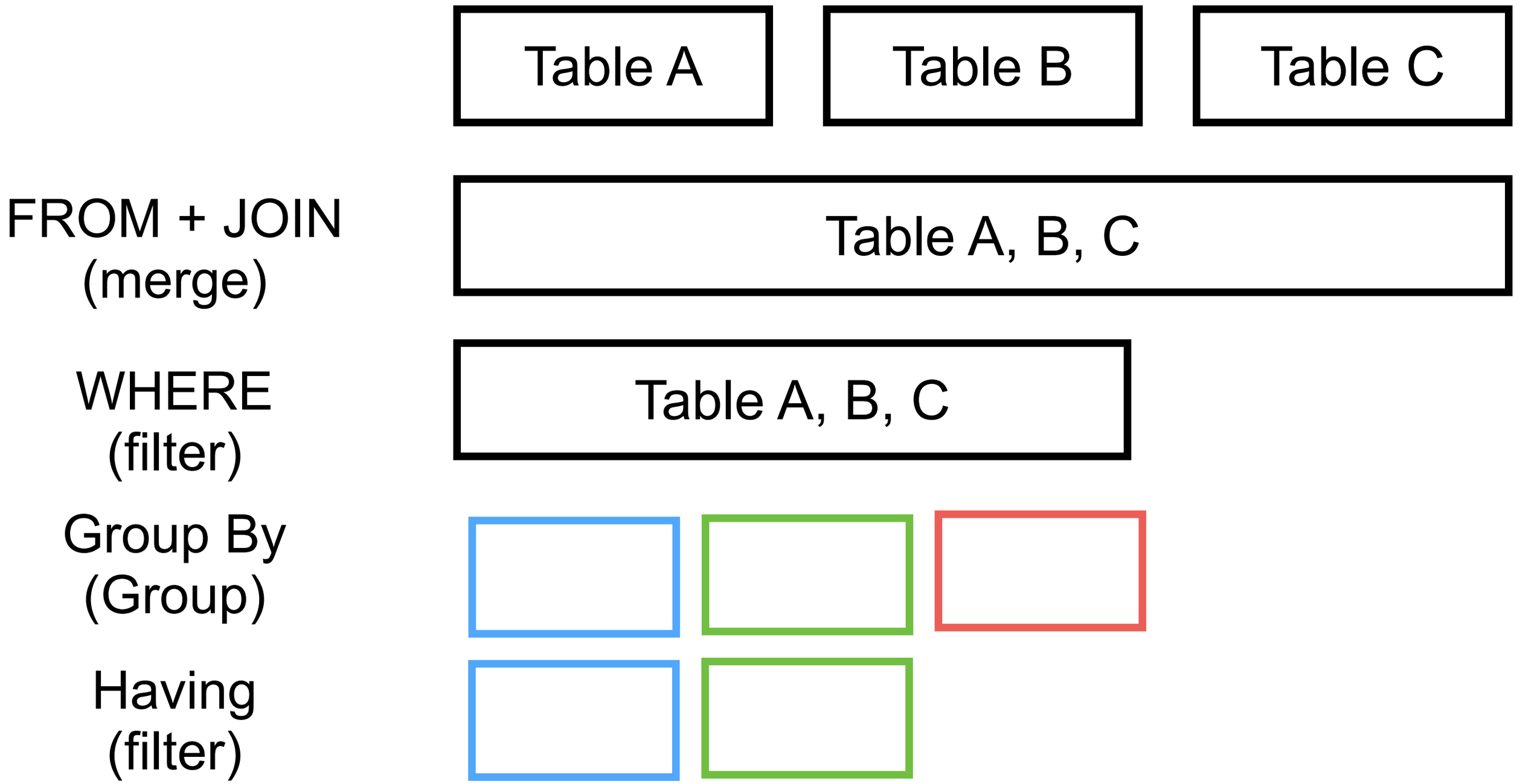
Select Data process



Select Data process



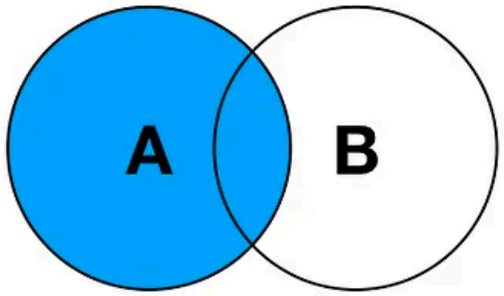
Select Data process



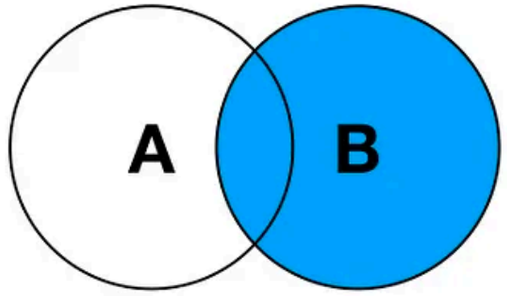
Joining

https://www.postgresql.org/docs/current/tutorial-join.html

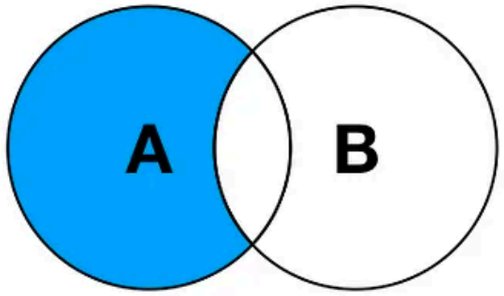
SQL JOINS



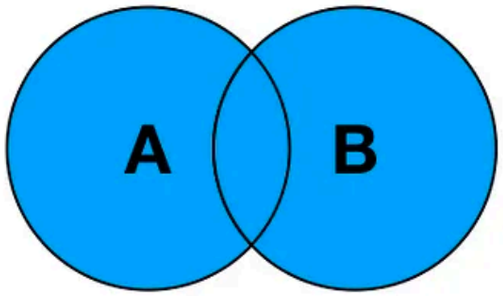
LEFT JOIN



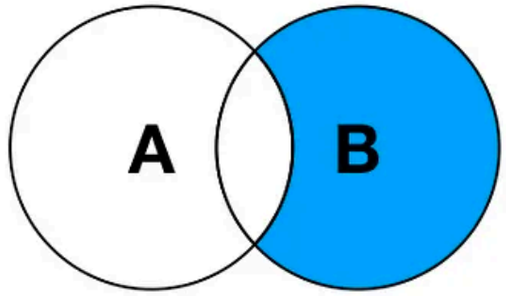
RIGHT JOIN



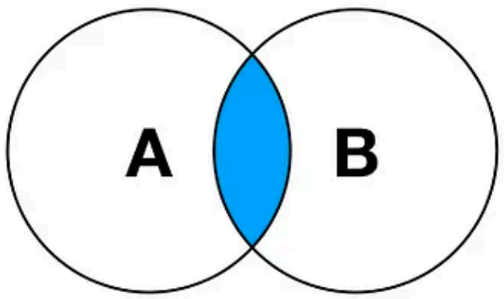
LEFT JOIN EXCLUDING
INNER JOIN



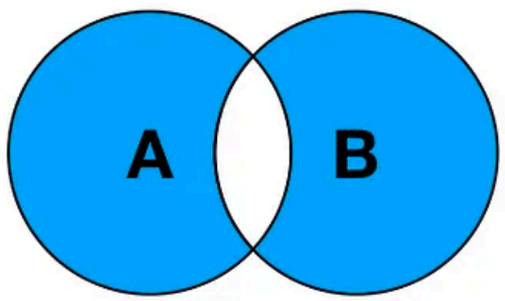
FULL OUTER JOIN



RIGHT JOIN EXCLUDING
INNER JOIN



INNER JOIN



FULL OUTER JOIN EXCLUDING
INNER JOIN

Self-join

Join in PostgreSQL

Left join

Inner join

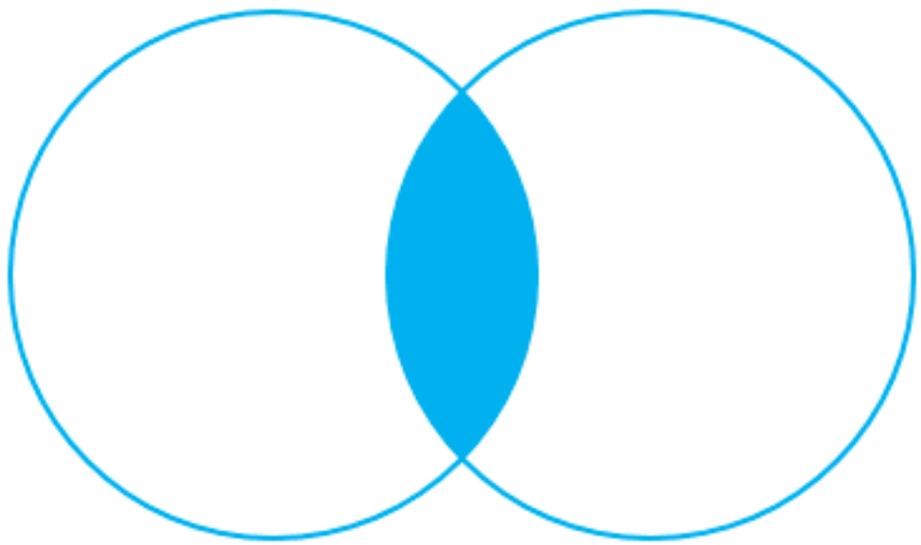
Right join

Full outer join

Workshop

Basket A	Basket B
Orange	Apple
Banana	Watermelon
Cucumber	Pear

Inner join



INNER JOIN

Inner join

Basket A

Apple

Orange

Banana

Cucumber

Basket B

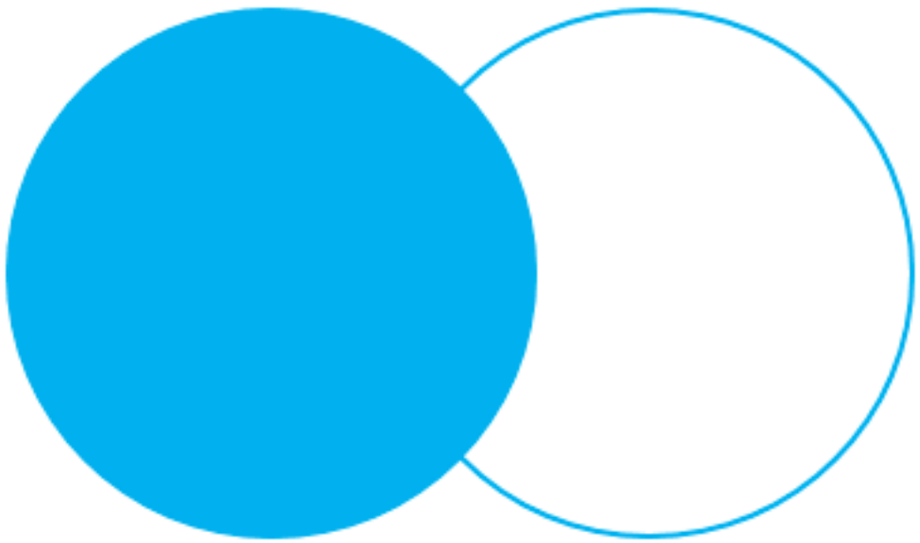
Orange

Apple

Watermelon

Pear

Left join (1)



LEFT OUTER JOIN

Left join (1)

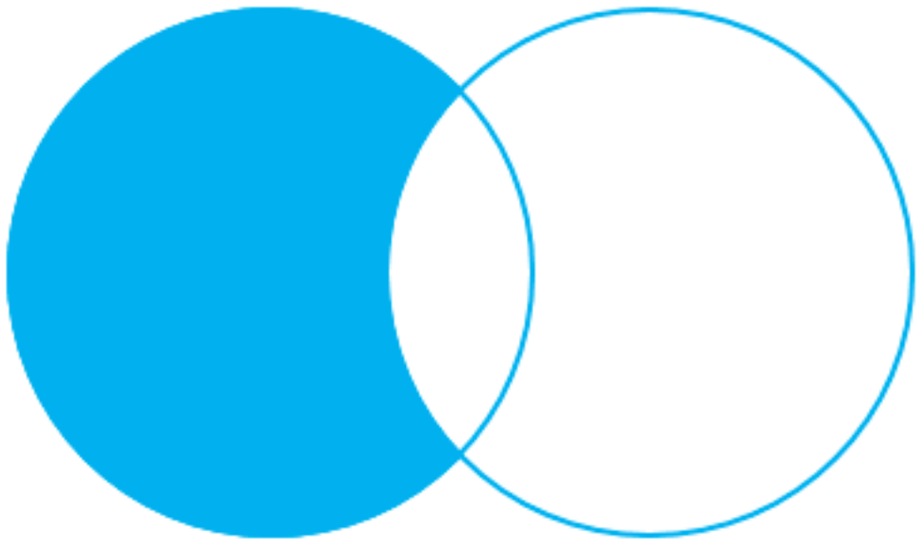
Basket A

Apple
Orange
Banana
Cucumber

Basket B

Apple
Watermelon
Pear

Left join (2)

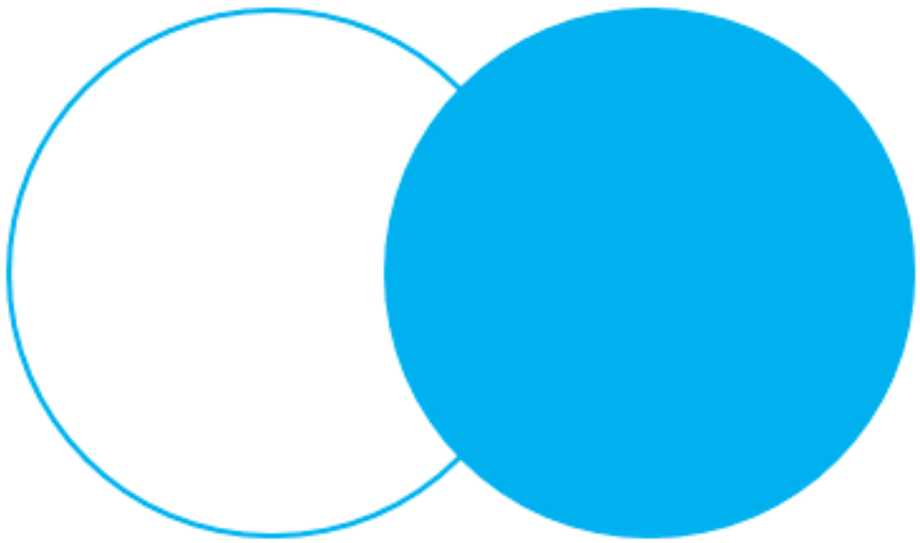


LEFT OUTER JOIN – only rows from the left table

Left join (2)

Basket A	Basket B
Apple	Orange
Orange	Apple
Banana	Watermelon
Cucumber	Pear

Right join (1)



RIGHT OUTER JOIN

Right join (1)

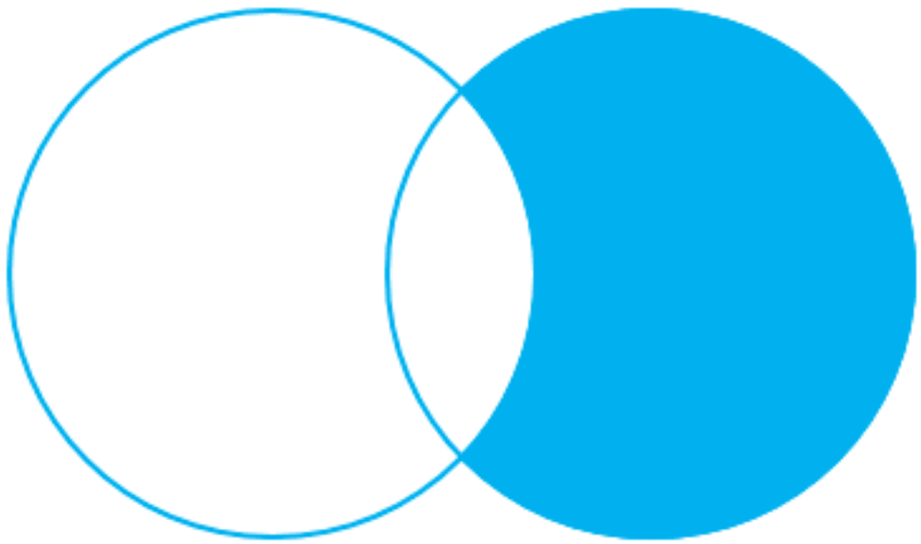
Basket A

Apple
Orange
Banana
Cucumber

Basket B

Orange
Apple
Watermelon
Pear

Right join (2)

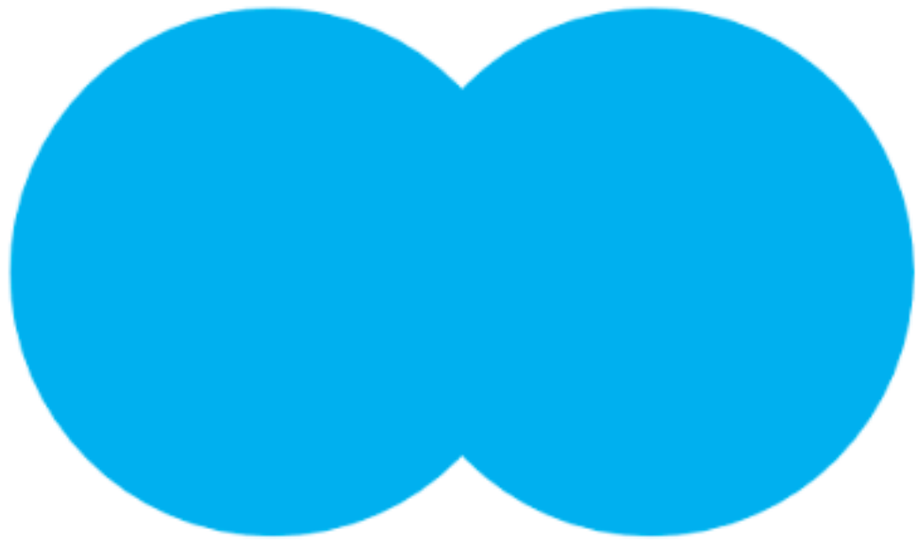


RIGHT OUTER JOIN – only rows from the right table

Right join (2)

Basket A	Basket B
Apple	Orange
Orange	Apple
Banana	**Watermelon**
Cucumber	**Pear**

Full outer join (1)



FULL OUTER JOIN

Full outer join (1)

Basket A

Apple
Orange
Banana
Cucumber

Basket B

Orange
Apple
Watermelon
Pear

2

Full outer join (1)

Basket A

Apple
Orange
Banana
Cucumber

2

Basket B

Apple
Watermelon
Pear

Full outer join (1)

Basket A

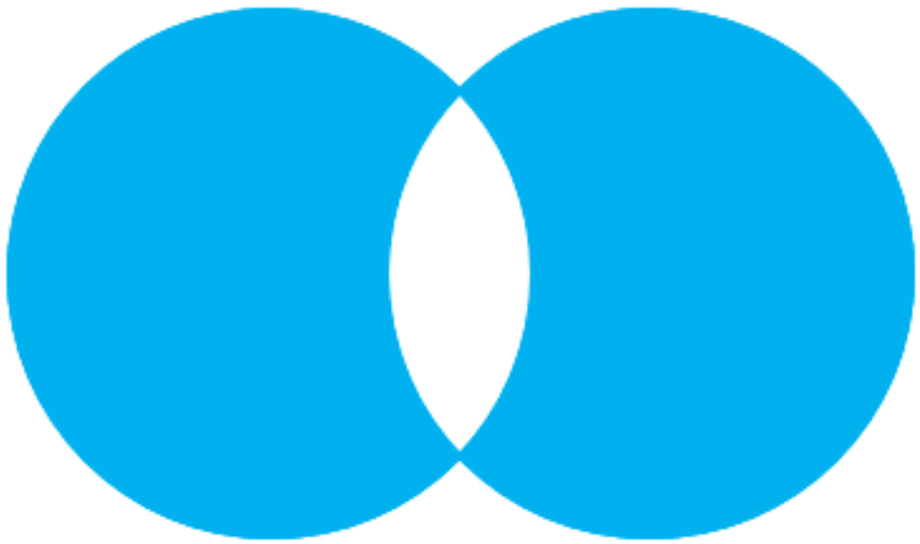
Apple
Orange
Banana
Cucumber

Basket B

Apple
Watermelon
Pear

2

Full outer join (2)



FULL OUTER JOIN – only
rows unique to both tables

Full outer join (2)

Basket A

Apple
Orange
Banana
Cucumber

Basket B

Apple
Watermelon
Pear

2

Full outer join (2)

Basket A

Apple
Orange
Banana
Cucumber

Basket B

Apple
Watermelon
Pear

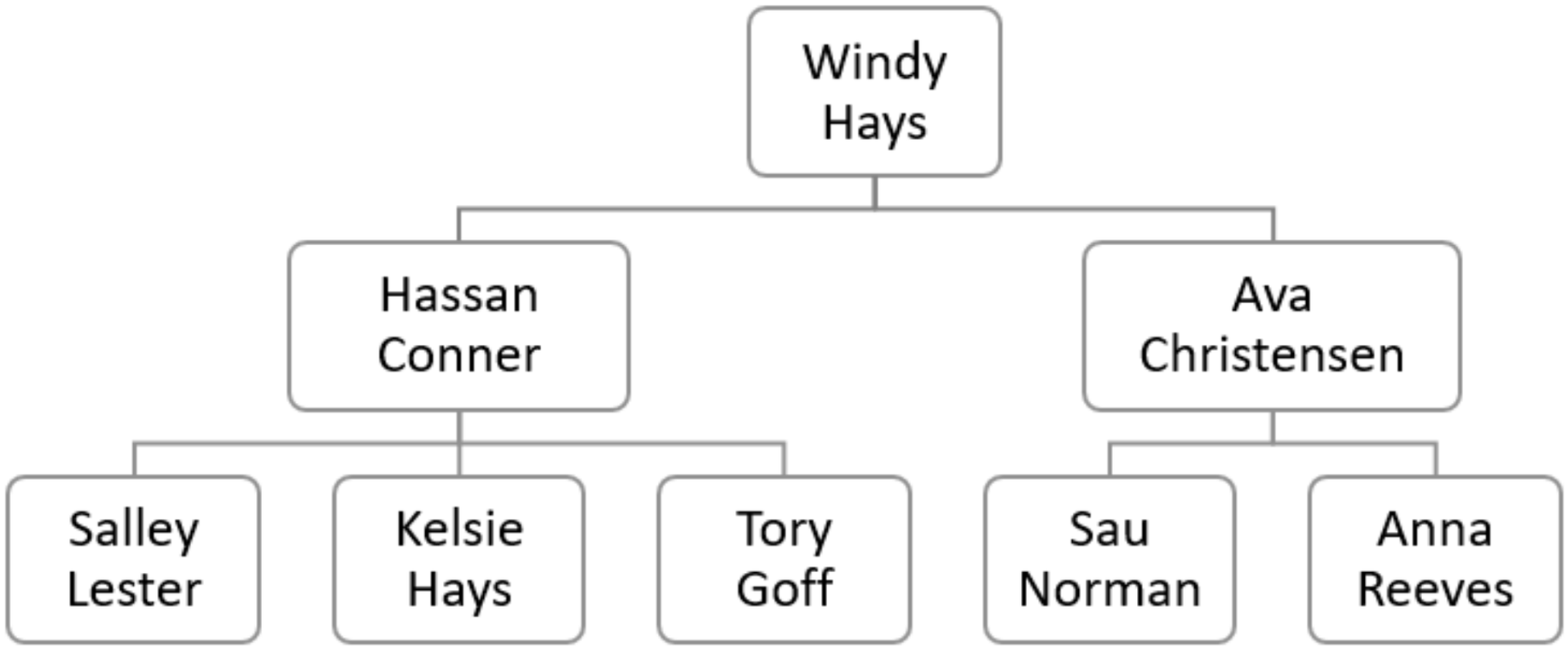
2

Workshop :: Joins

https://github.com/up1/workshop-database/blob/main/workshop/joins.md

Self-join

Regular join that joins a table to itself
Query hierarchical data



Workshop :: Self-Join

https://github.com/up1/workshop-database/blob/main/workshop/self-join.md

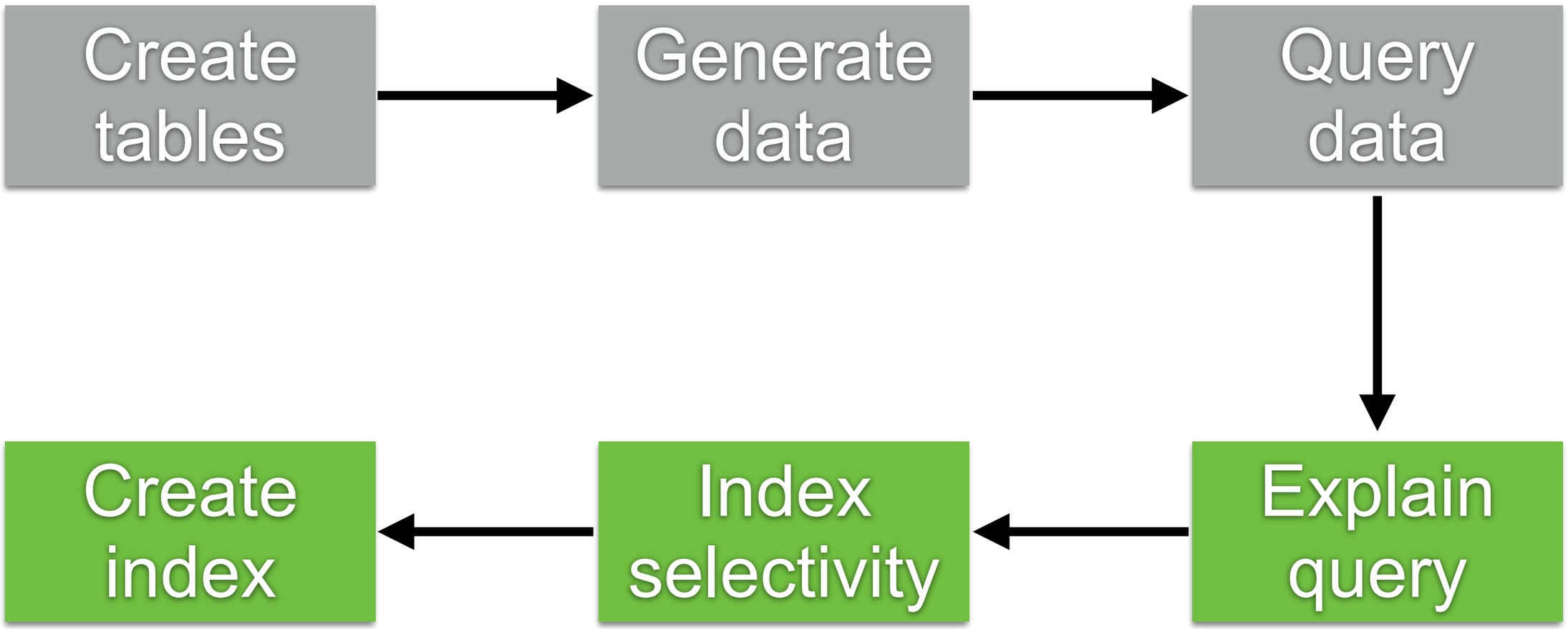
Problems ?

Problems

Slow query
Accurate knowledge is required
Add complexity
Need denormalization ?

Improvement to query data

Step in workshop



Query plan in PostgreSQL

EXPLAIN ANALYZE <SQL>

Query | Query History

```
1  EXPLAIN ANALYZE select * from book;
```

Data Output | Messages | Graph Visualiser | Explain | Notifications

	QUERY PLAN text
1	Seq Scan on book (cost=0.00..2334.00 rows=100000 width=89) (actual time=0.007..8.479 rows=100000 loops...
2	Planning Time: 0.209 ms
3	Execution Time: 12.950 ms

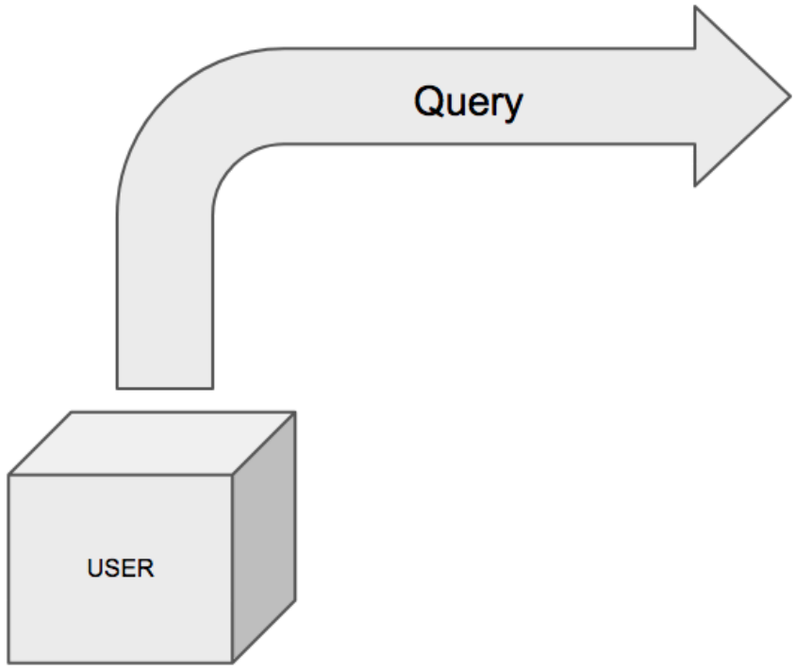
https://github.com/up1/workshop-database/tree/main/workshop#query-plan

Indexing

https://www.postgresql.org/docs/current/indexes-types.html

Index

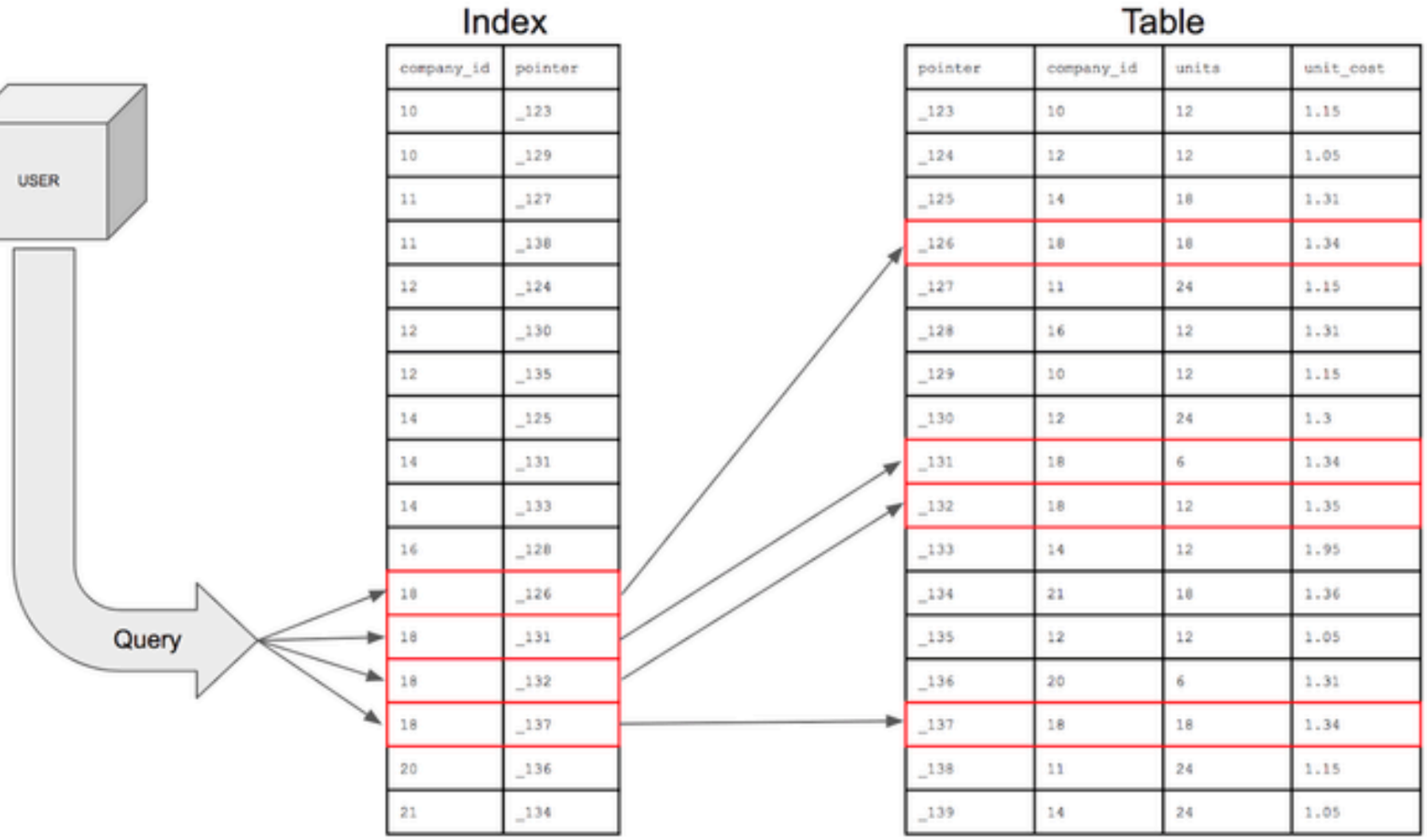
Without Index (full table scan)



Table

company_id	units	unit_cost
10	12	1.15
12	12	1.05
14	18	1.31
18	18	1.34
11	24	1.15
16	12	1.31
10	12	1.15
12	24	1.3
18	6	1.34
18	12	1.35
14	12	1.95
21	18	1.36
12	12	1.05
20	6	1.31
18	18	1.34
11	24	1.15
14	24	1.05

With Index



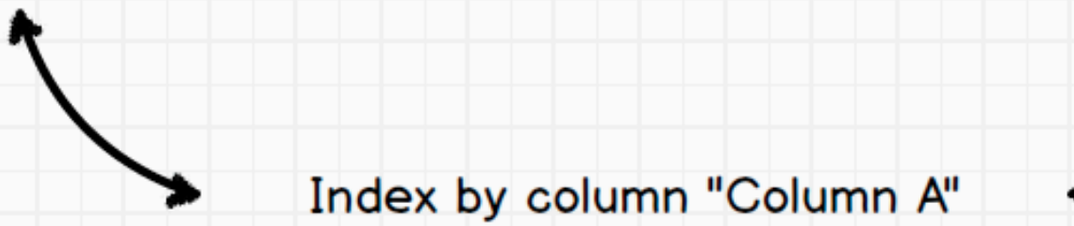
With Index

Table

	Column A	Column B
RowId 1	Some value 1	Some another value 1
RowId 2	Some value 2	Some another value 2
RowId 3	Some value 3	Some another value 3
RowId 4	Some value 1	Some another value 4
RowId 5	Some value 5	Some another value 5
RowId 6	Some value 6	Some another value 6
RowId 7	Some value 1	Some another value 7
RowId 8	Some value 8	Some another value 8

Index

Some value 1	"RowId 1" "RowId 4" "RowId 7"
Some value 2	RowId 2
Some value 3	RowId 3
Some value 5	RowId 5
Some value 6	RowId 6
Some value 8	RowId 8



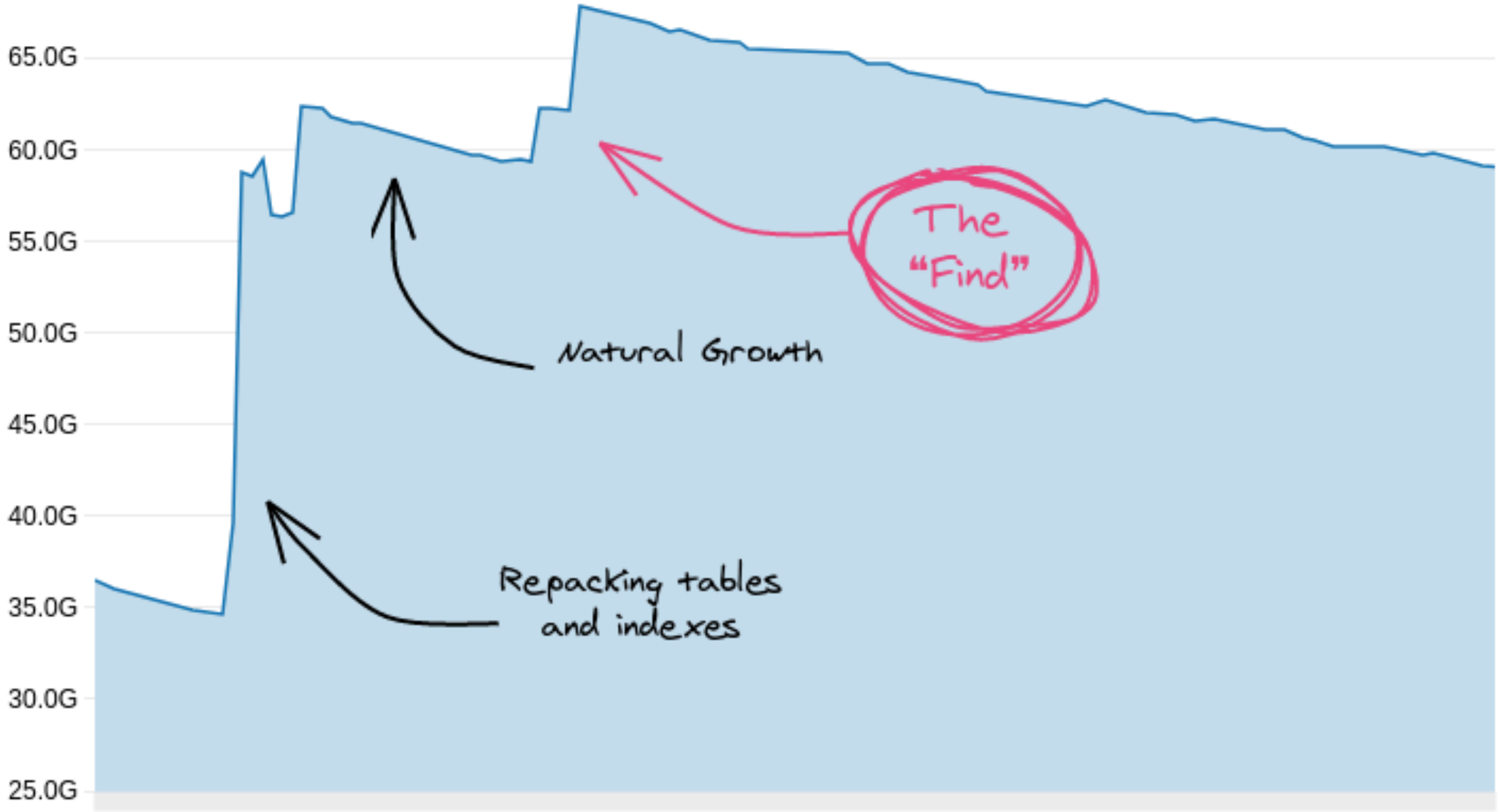
Index selectivity

Factor to select a column to create an index

The number of **distinct values** in the indexed column divided by the number of records in the table is called a selectivity of an index

Prefer indexing columns with selectivity greater than > 0.85 (use Index scan)

Unused Index ?



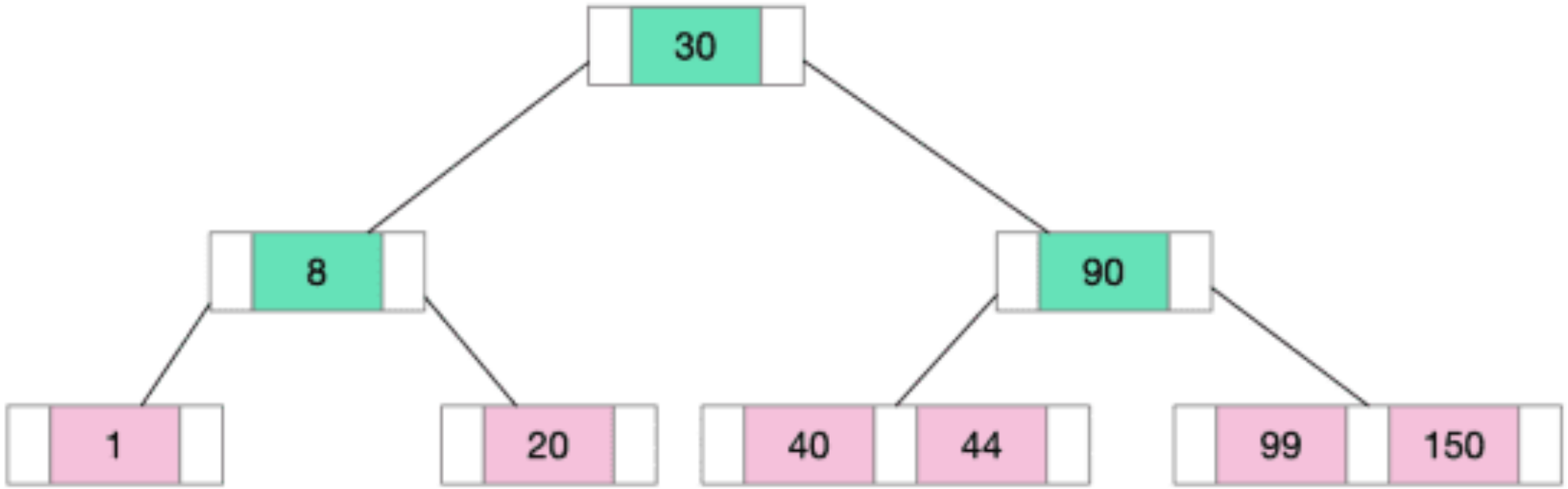
https://hakibenita.com/postgresql-unused-index-size

Types Index in PostgreSQL

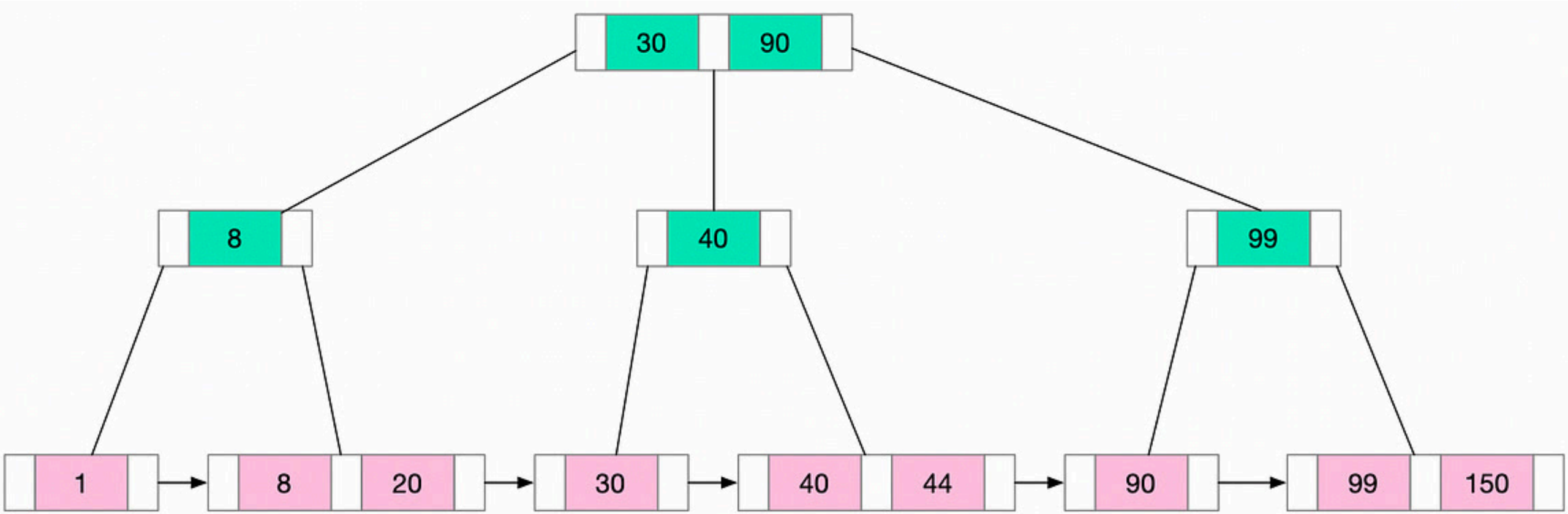
B-Tree
Hash
Block range indexes (BRIN)
Generalized inverted index (GIN)
Generalized inverted search tree index (GiST)
Space partitioned GiST (SP-GiST)

https://www.postgresql.org/docs/current/indexes-types.html

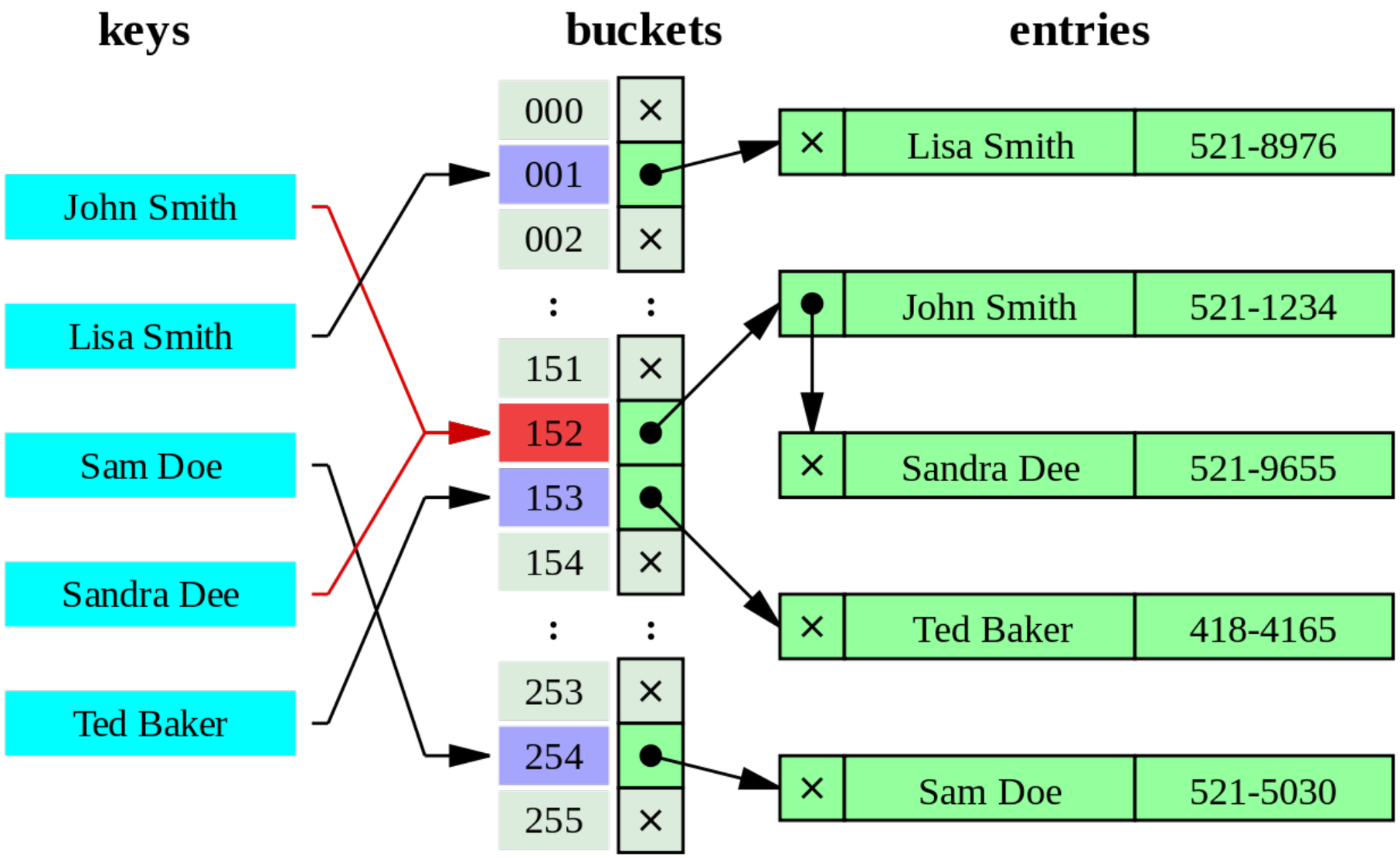
B-Tree



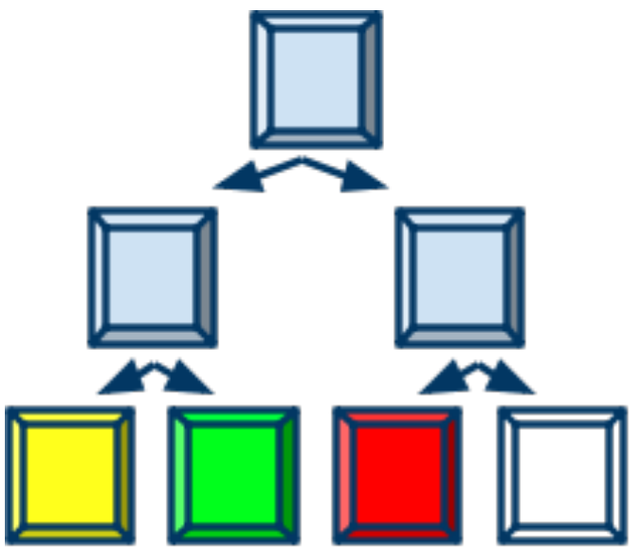
B-Tree +

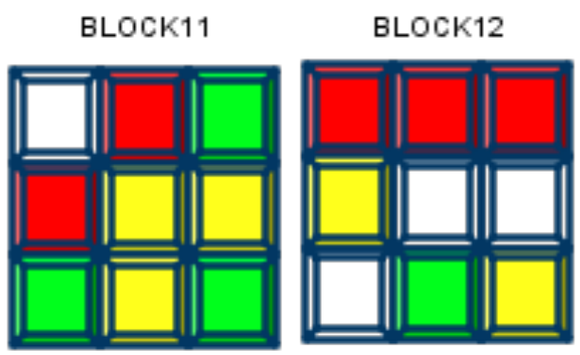


Hash



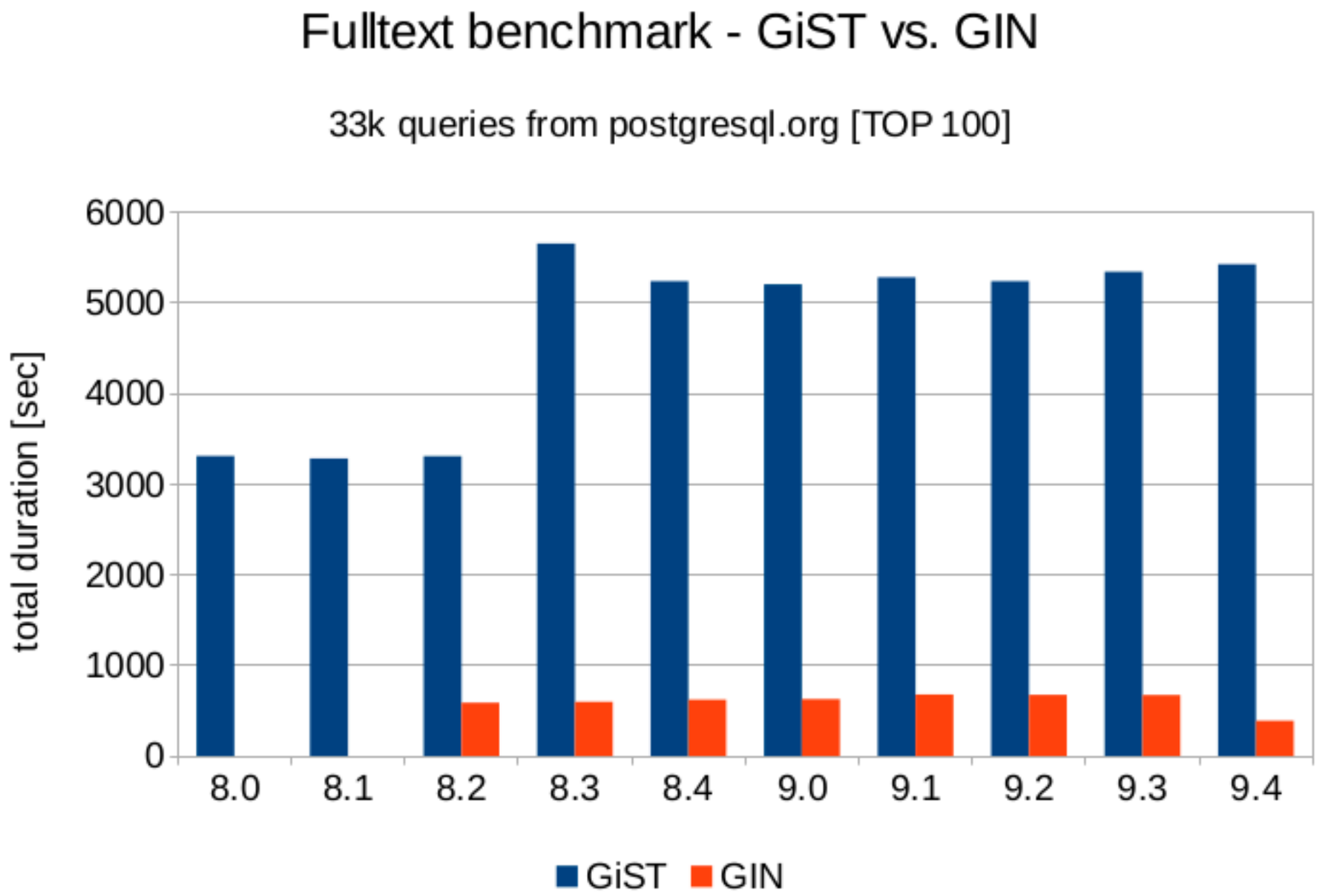
Bitmap





KEY	Start Row ID	End Row ID	BATMAP
YELLOW	11.0	12.9	10000
GREEN	11.0	12.9	01000
RED	11.0	12.9	00100
null	11.0	12.9	00010

GIN vs GiST



Don’t over-index

Unused index

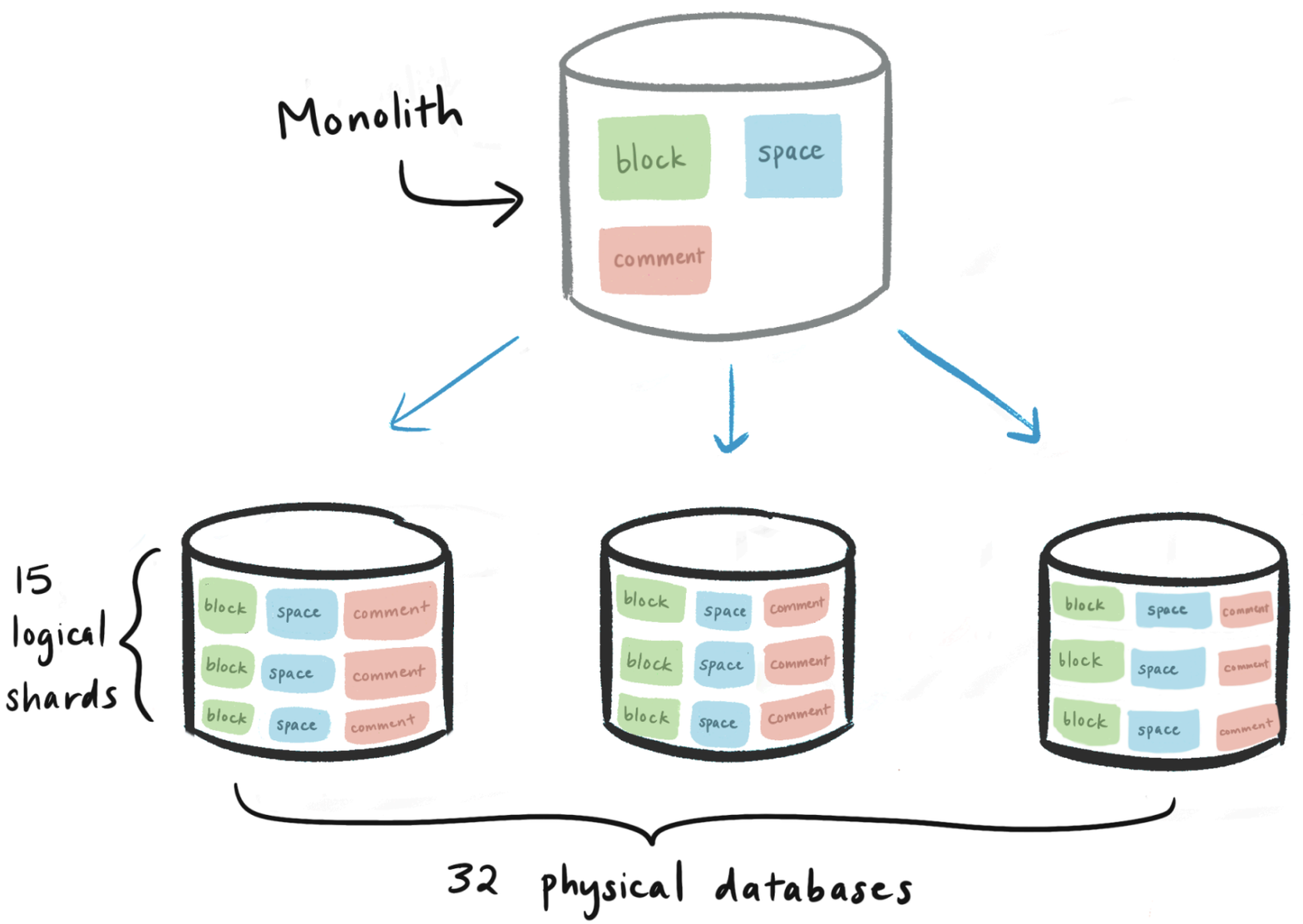
Tuning Performance

Tuning performance

Sharding
Partition
Denormalization

Sharding

Sharding



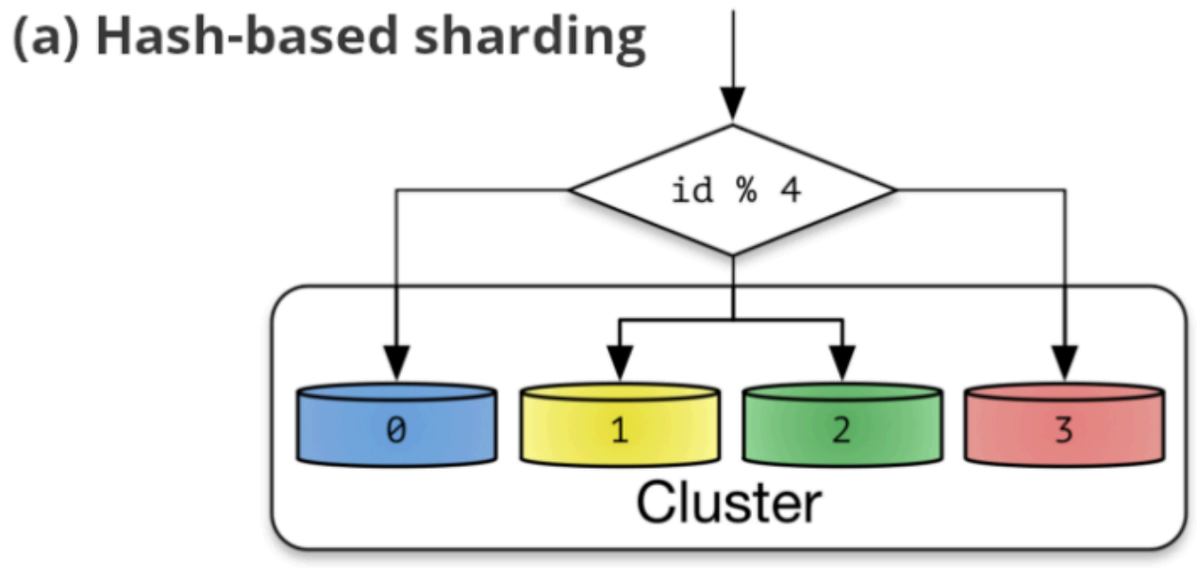
https://www.notion.so/blog/sharding-postgres-at-notion

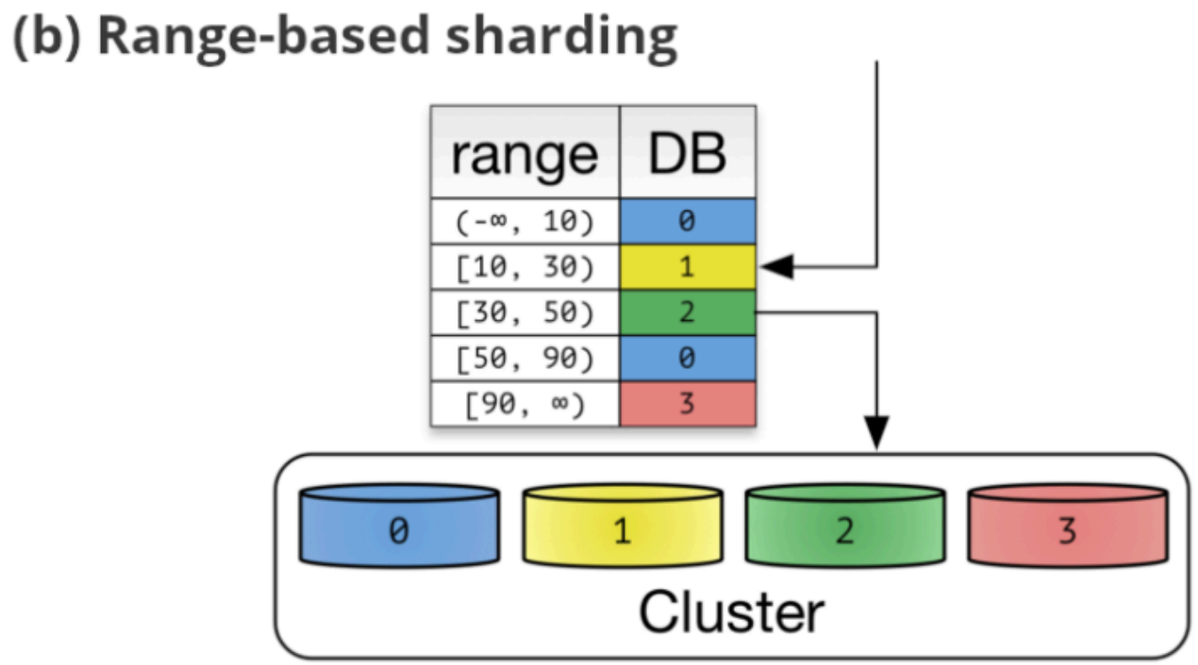
Sharding techniques

Hash
Range
Location
Key/Directory

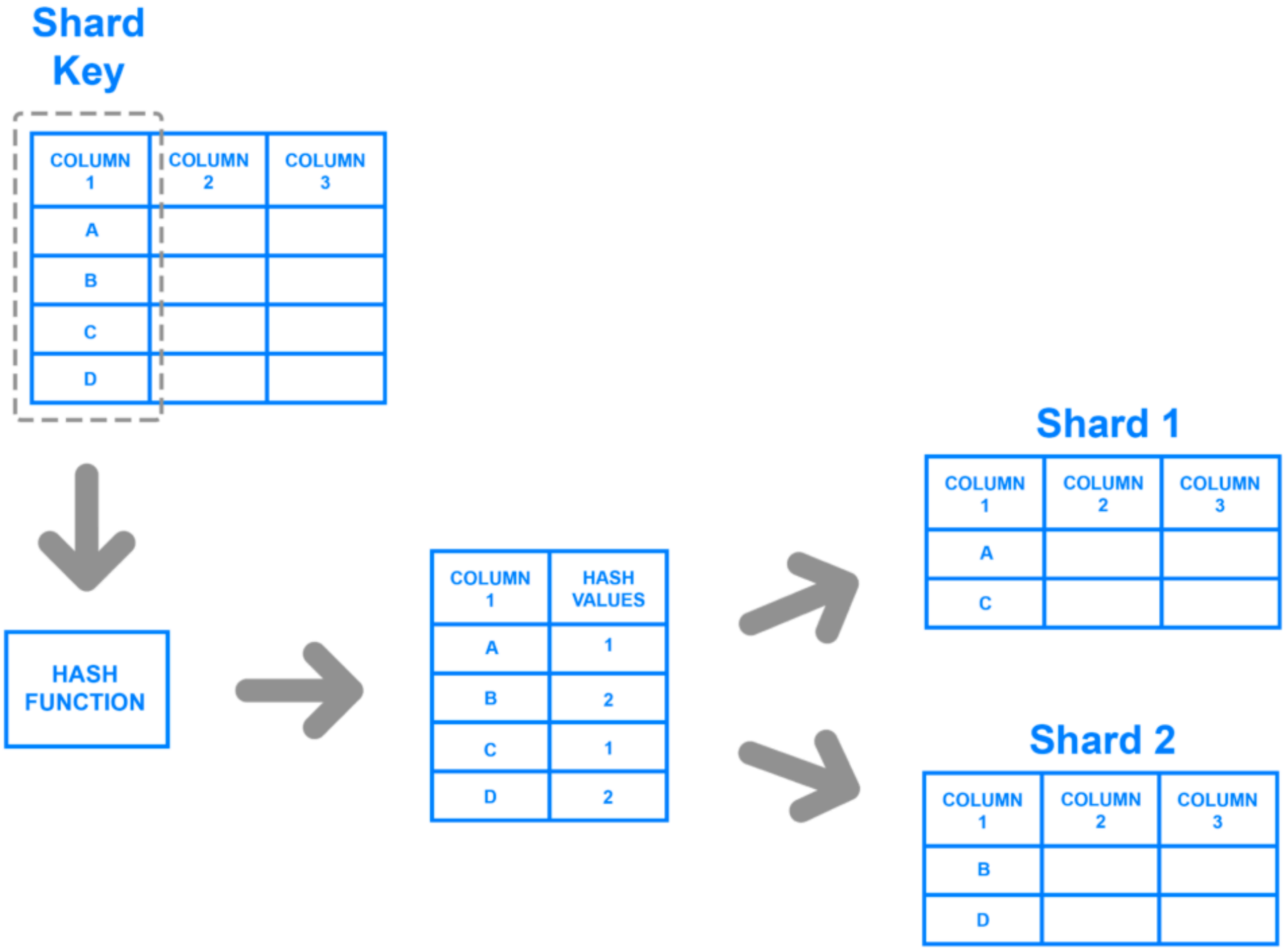
https://www.somkiat.cc/introduction-database-sharding/

Sharding





Shard Key



Range

PRODUCT	PRICE
WIDGET	$118
GIZMO	$88
TRINKET	$37
THINGAMAJIG	$18
DOODAD	$60
TCHOTCHKE	$999

($0-$49.99)

PRODUCT	PRICE
TRINKET	$37
THINGAMAJIG	$18

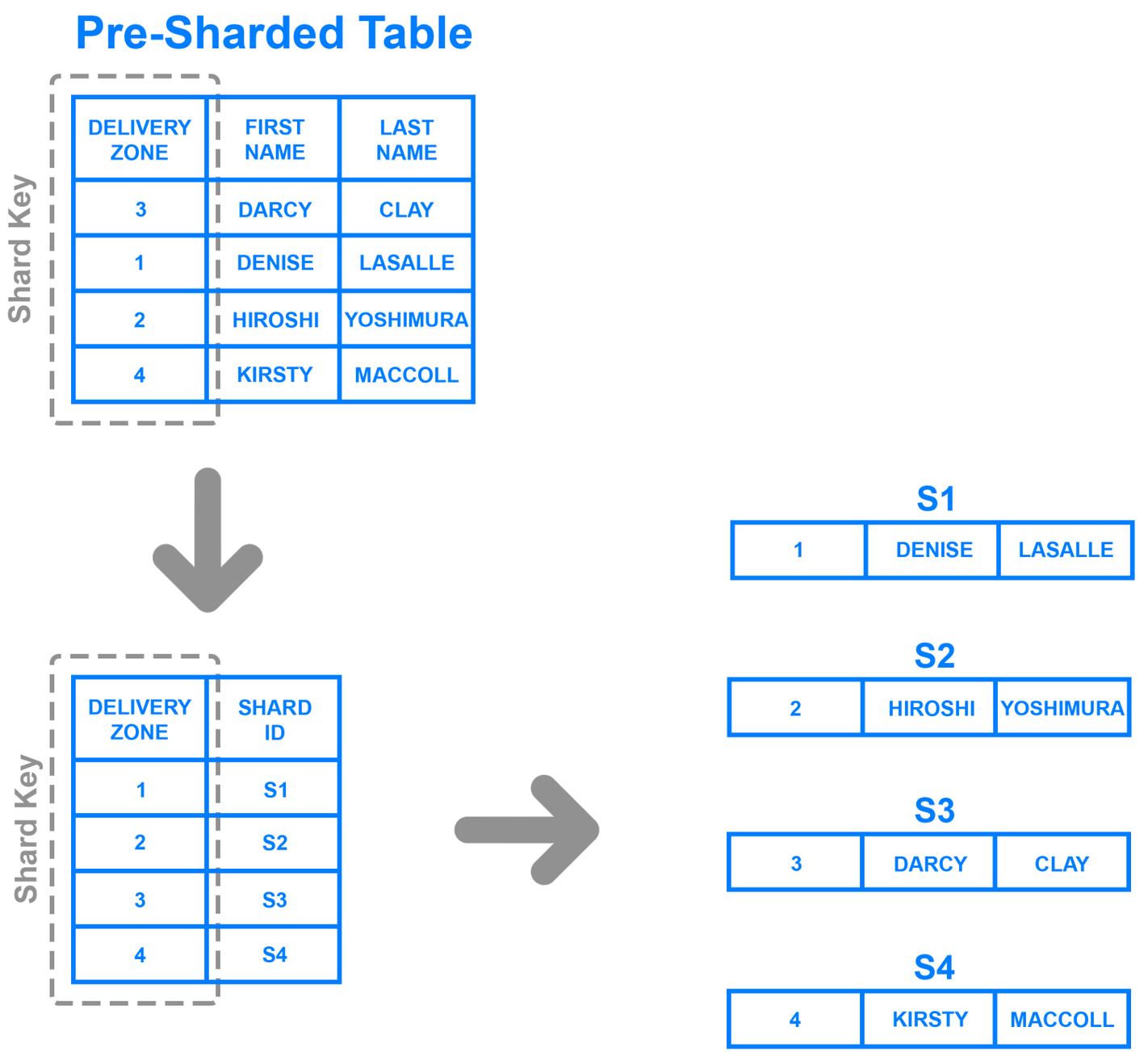
($50-$99.99)

PRODUCT	PRICE
GIZMO	$88
DOODAD	$60

($100+)

PRODUCT	PRICE
WIDGET	$118
TCHOTCHKE	$999

Directory



Sharding

Shard 1

Stores

id	name
1	my book store
5	my other store

Products

id	name	store_id
1	foo	1
2	bar	1
3	baz	1

Purchases

id	product_id	store_id	price
1	2	1	1000
2	1	1	1200
3	3	1	1199

Shard 2

Stores

id	name
2	my sock store
6	old things

Products

id	name	store_id
33	new socks	2
34	old socks	6
35	old tie	6

Purchases

id	product_id	store_id	price
102	35	6	600
43	33	2	800

https://www.citusdata.com/blog/2016/08/10/sharding-for-a-multi-tenant-app-with-postgres/

Partition

https://hakibenita.com/postgresql-unused-index-size

Partition

Table partitioning refers to **splitting** what is logically one large table into **smaller physical pieces**

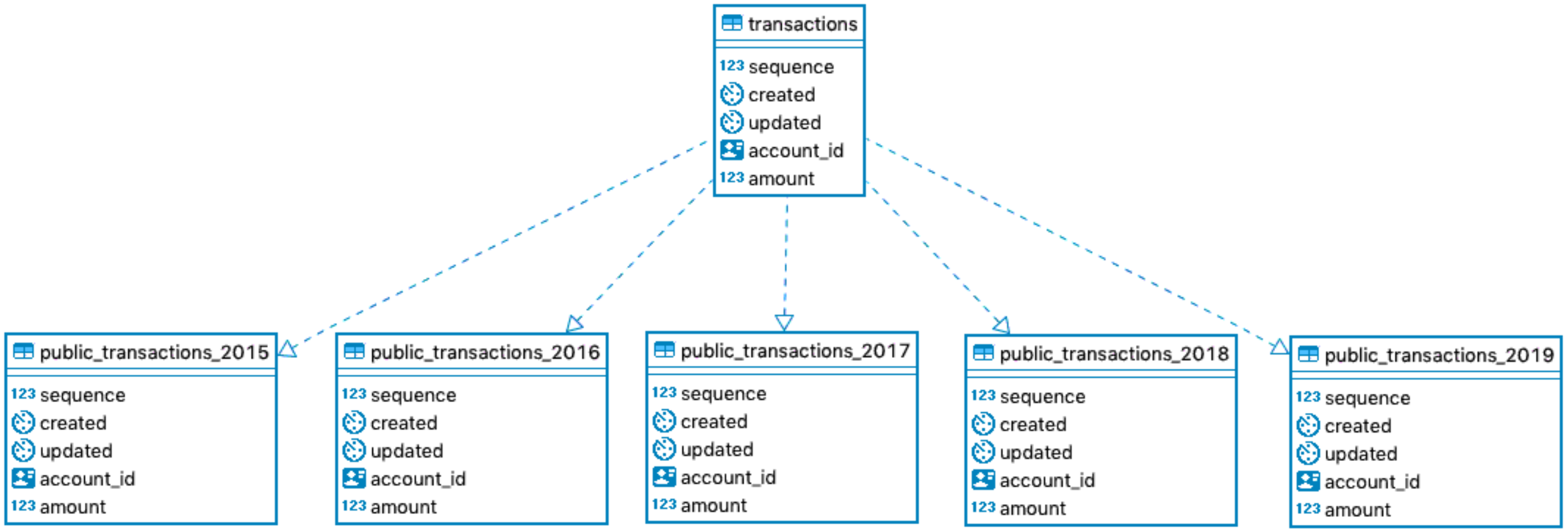
Reduce index size

Partition

Horizontal partitioning
Vertical partitioning

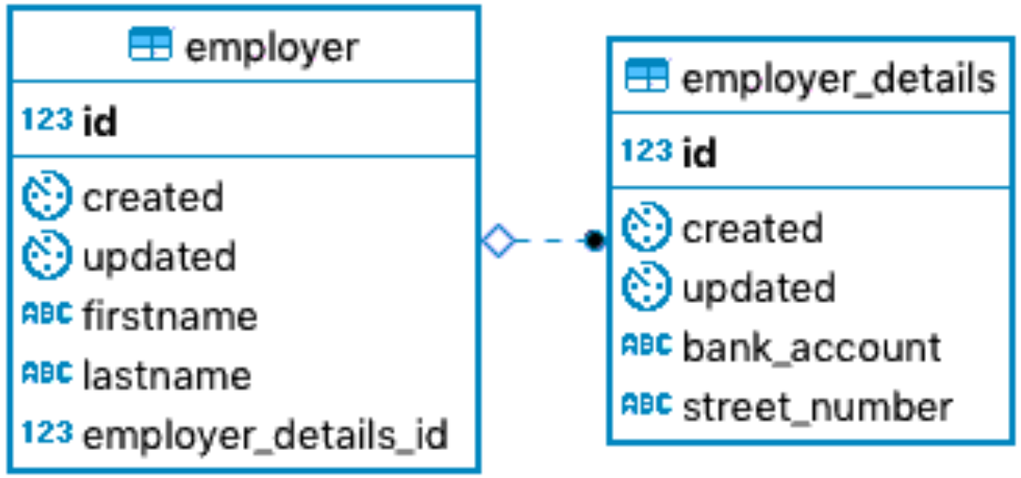
Horizontal partitioning

Putting different rows into different table



Vertical partitioning

Horizontal partitioning

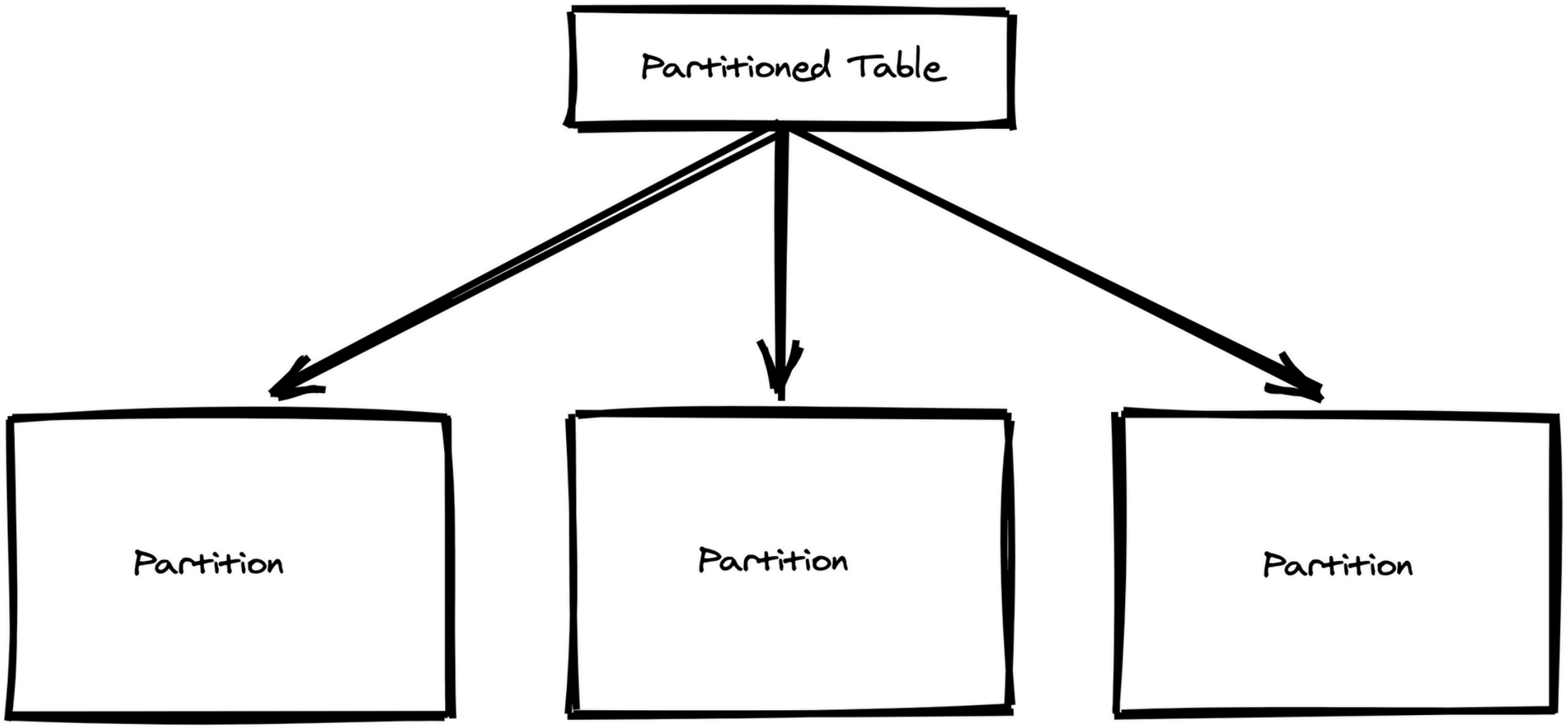


Types Horizontal partitioning

Range
List

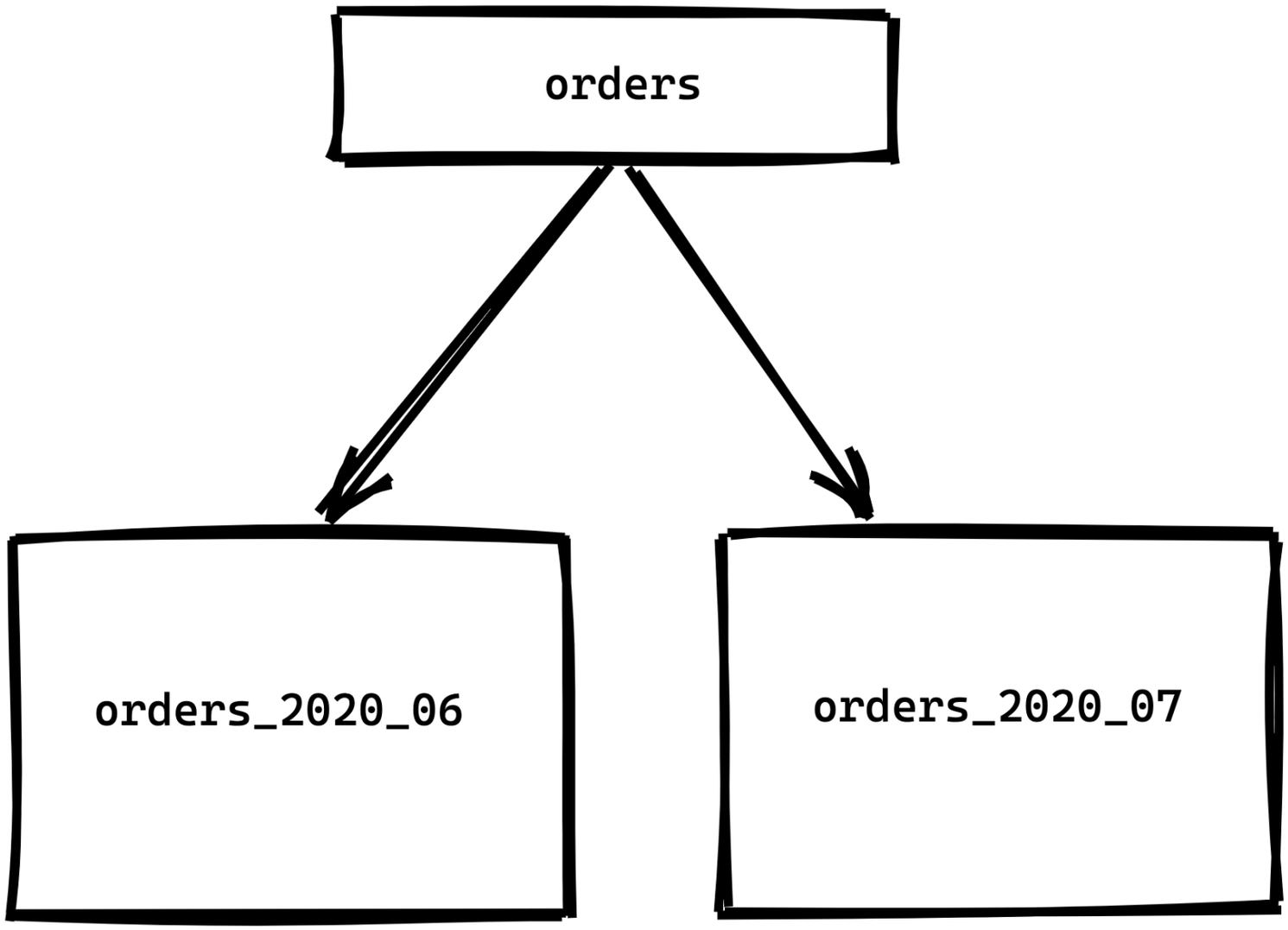
Example

Partition tables



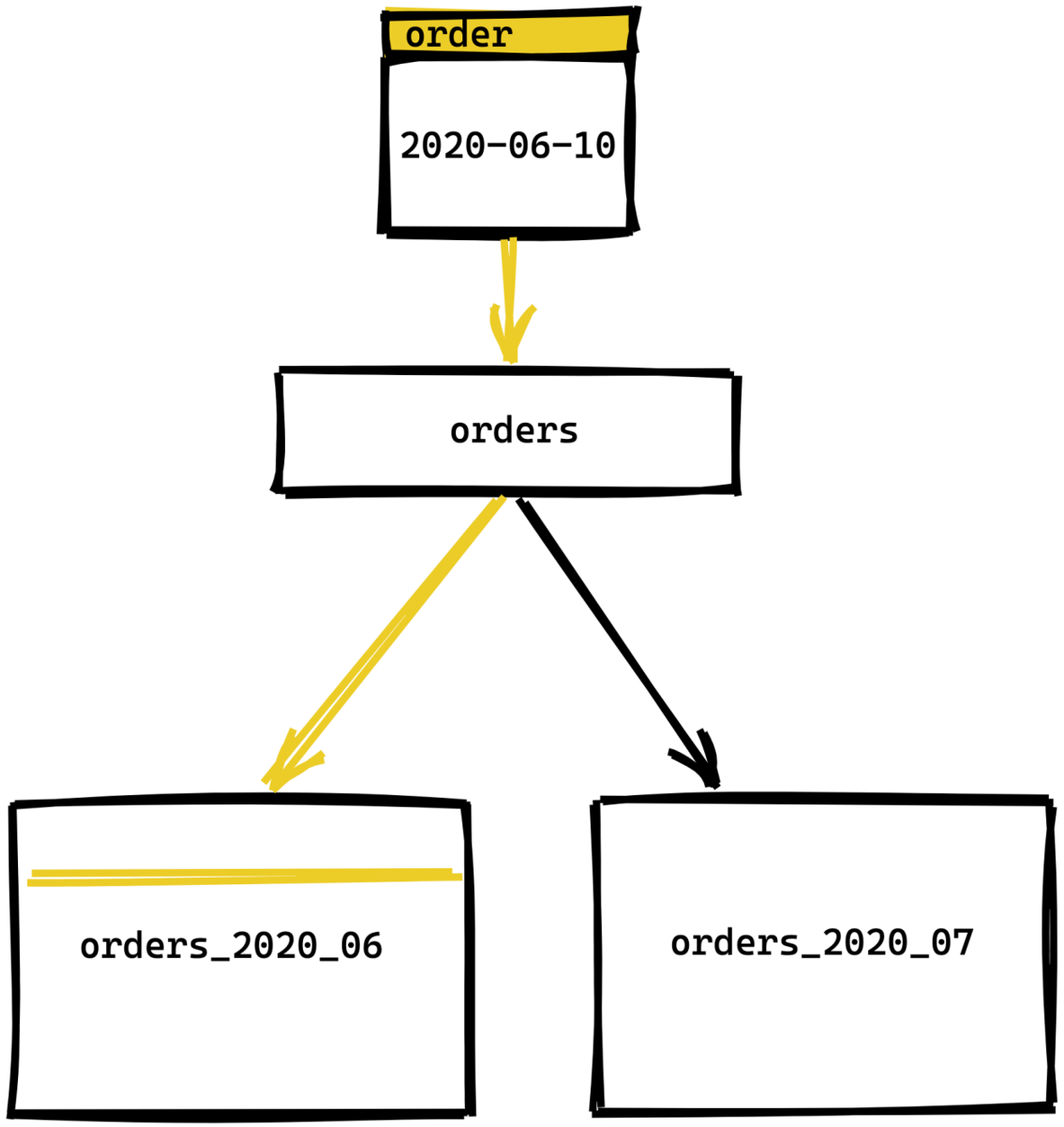
https://chriserwin.com/table-partitioning/

Orders table



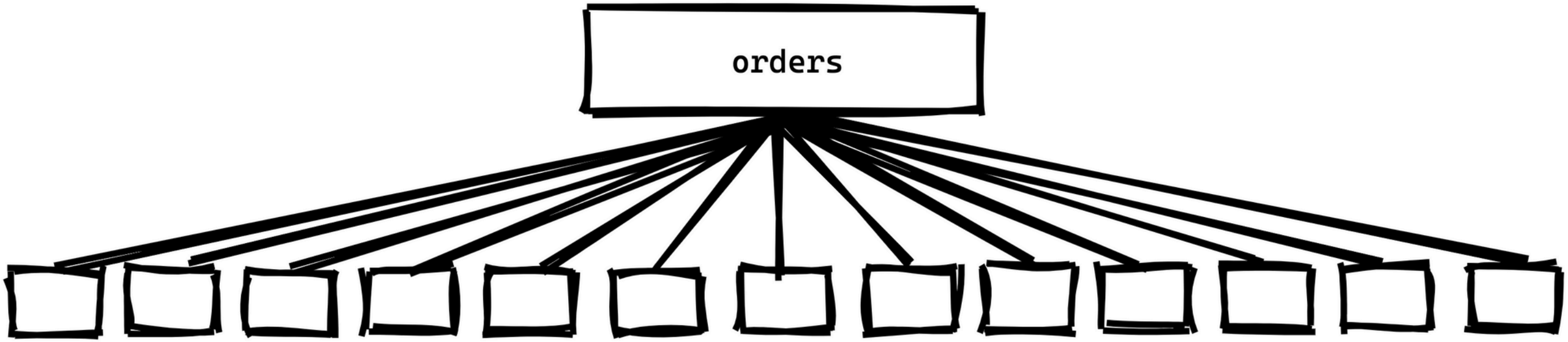
https://chriserwin.com/table-partitioning/

Insert data into orders



https://chriserwin.com/table-partitioning/

...



https://chriserwin.com/table-partitioning/

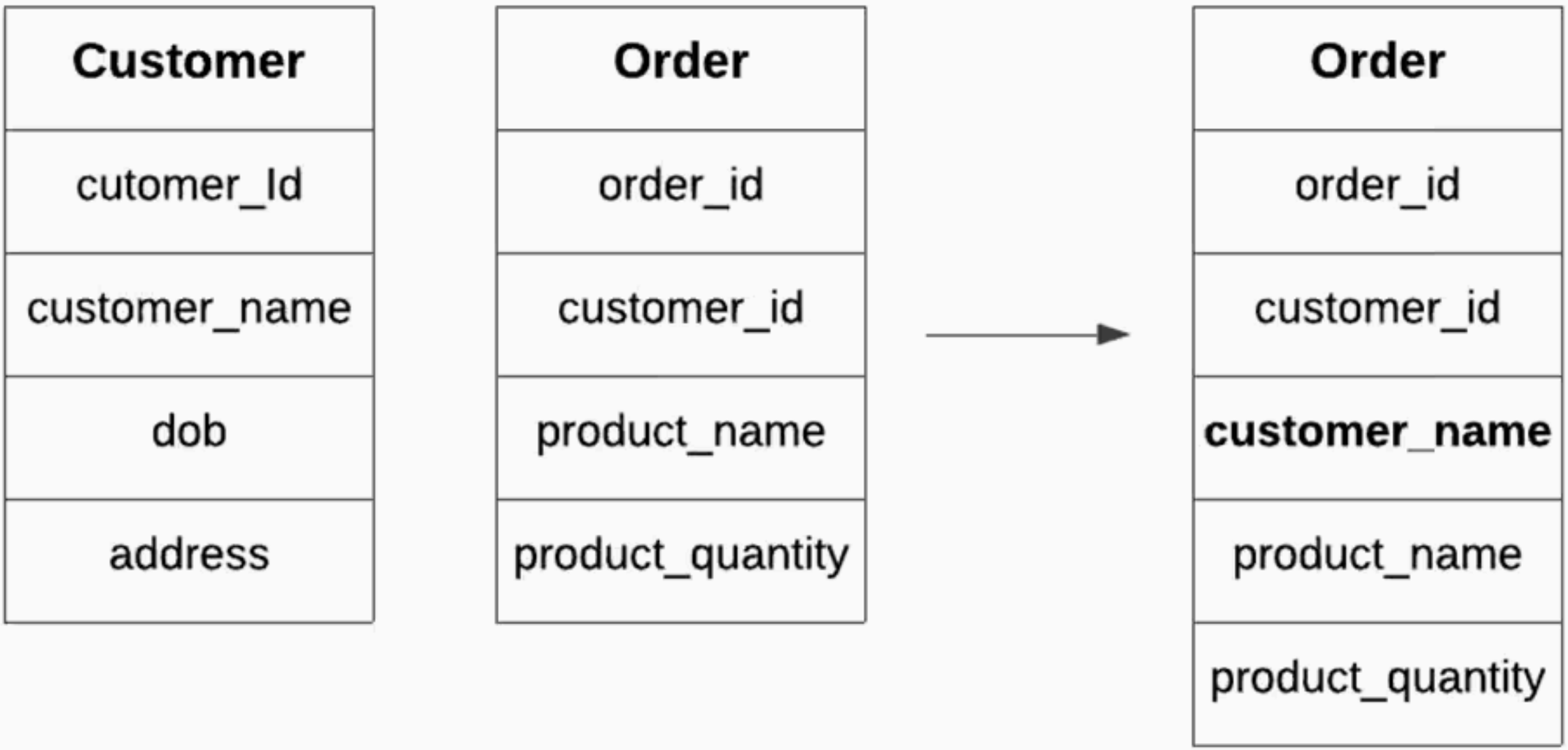
Workshop

Denormalization

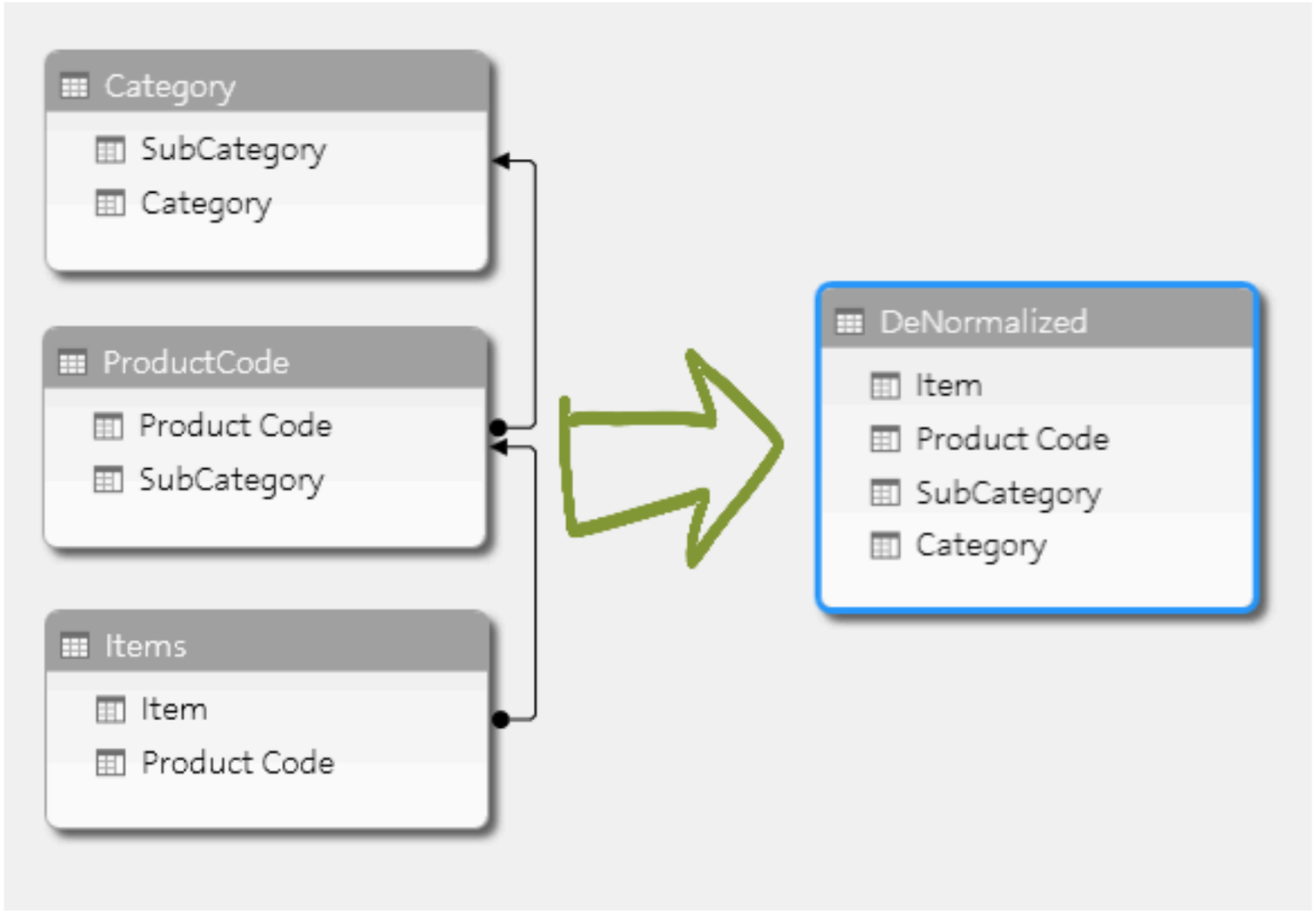
Denormalization techniques

Redundant column/Pre-joining tables
Table splitting (horizontal/vertical)
Add derived columns
Mirrored tables
Materialized views

Redundant column/Pre-joining tables

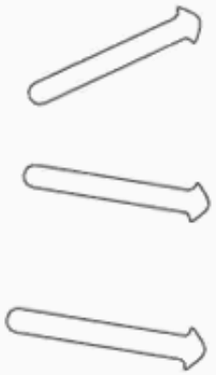


Redundant column/Pre-joining tables



Horizontal splitting

Student_ID	Department_ID	Student_Name
1	1	Alex
2	2	Marie
3	1	Sam
4	3	Sara



Computer Science

Student_ID	Department_ID	Student_Name
1	1	Alex
3	1	Sam

Chemistry

Student_ID	Department_ID	Student_Name
2	2	Marie

Botany

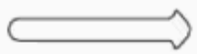
Student_ID	Department_ID	Student_Name
4	3	Sara

Vertical splitting

Add derived columns

Student_ID	Name
1	Alex
2	Marie

Student_ID	Assignment_ID	Mark
1	1	20
1	2	35.50
2	1	45
2	2	45



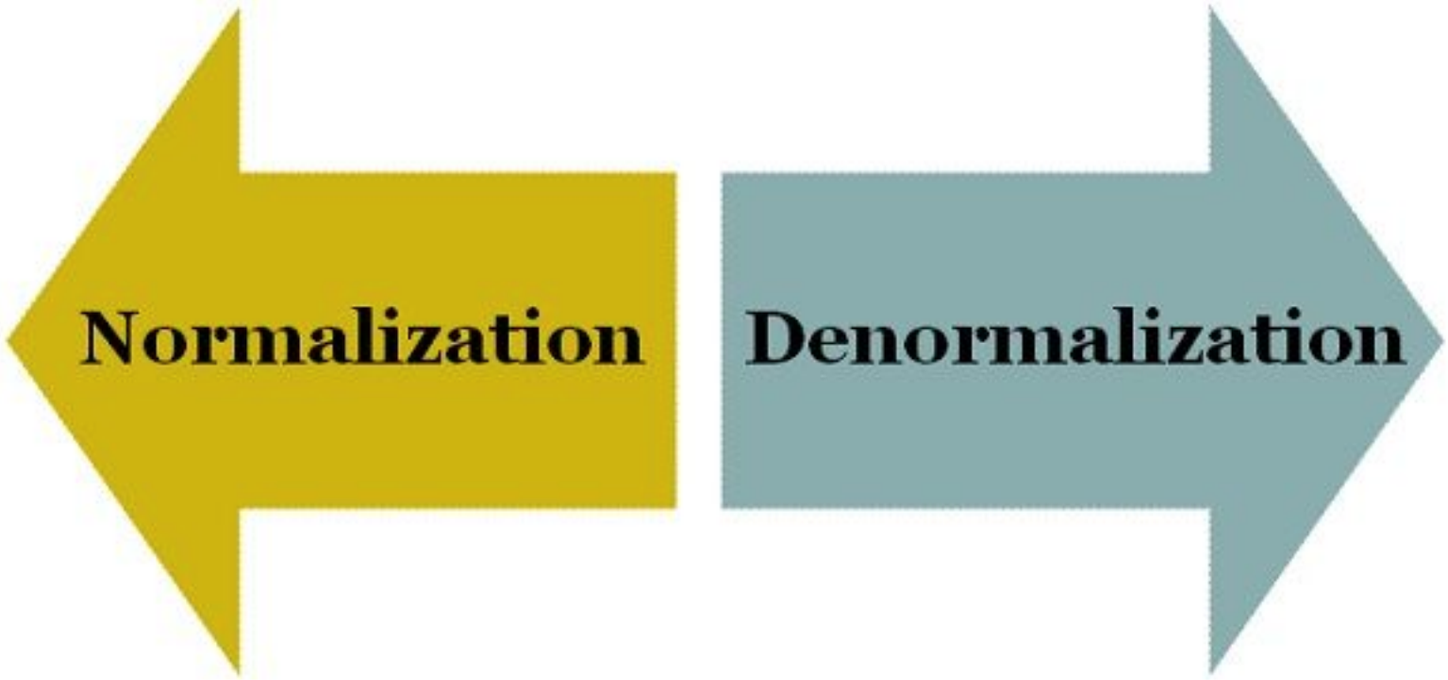
Student_ID	Name	Total_Marks
1	Alex	55.5
2	Marie	90

Pros of data denormalization

Improve query performance (UX)
Reduce complexity of data model
Improve application scalability
Generate data report faster

Cons of data denormalization

Increase data redundancy
Inconsistency between data sets
Require more disk space (split tables)
More data model (hard to maintain)
Difficult to insert and update data
Maintenance costs



Workshop