

Database Design Tuning



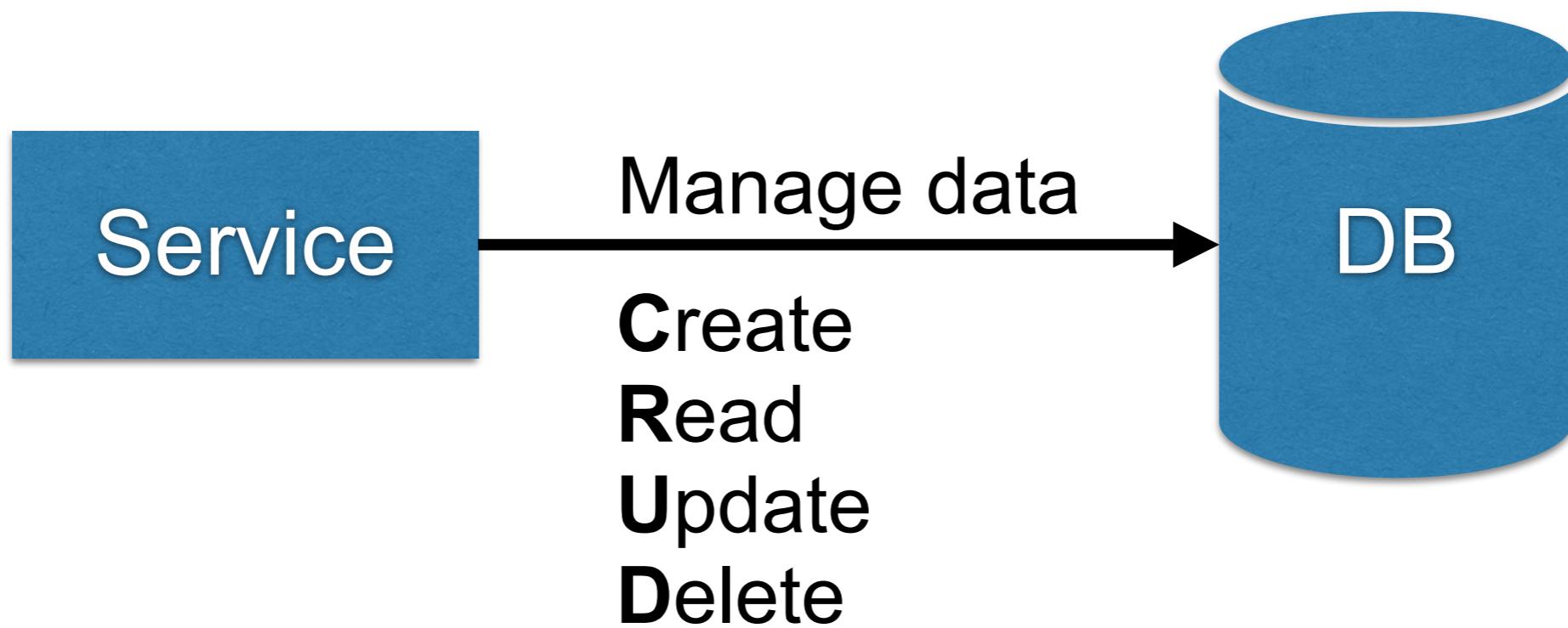
**[https://github.com/up1/
workshop-database](https://github.com/up1/workshop-database)**



Database



Database



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 design

© 2017 - 2018 Siam Chamnankit Company Limited. All rights reserved.

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

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



Database Model

Relational

Document
Key-value

Search engine

Wide column

Graph

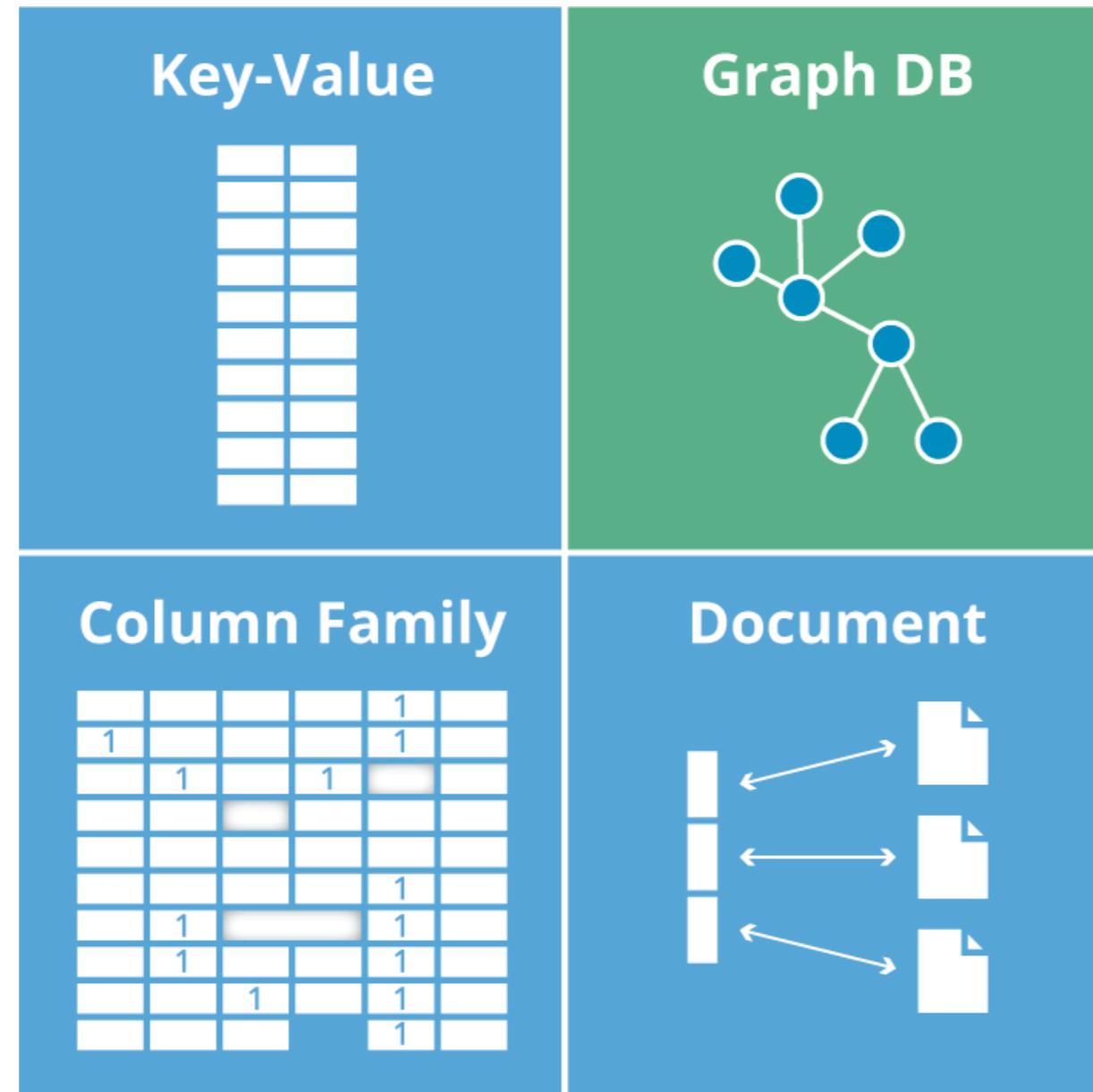
Time series

Content

Non-Relational
NoSQL



NoSQL

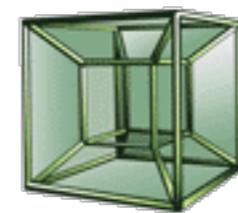


NoSQL

APACHE
HBASE



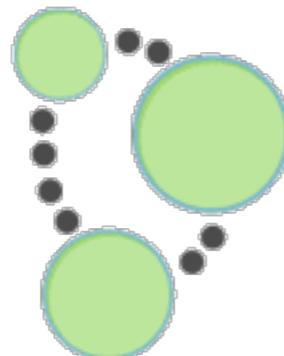
: riak



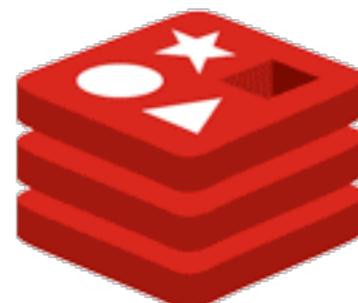
HyperTable^{INC}

The mongoDB logo includes a green leaf icon to the left of the word "mongoDB" in a large, brown, lowercase sans-serif font. A faint background grid of dots is visible behind the text.

HYPERTABLE^{INC}



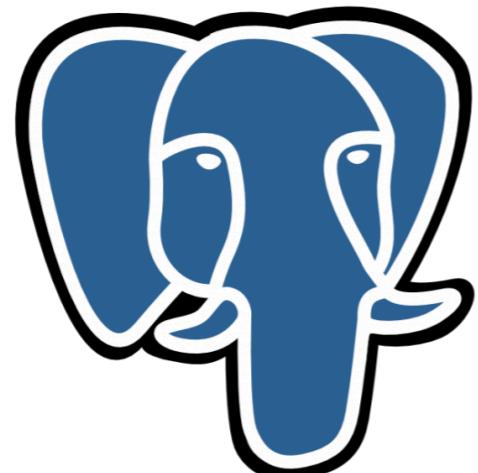
Neo4j



redis



Relational Database



PostgreSQL



Relational Database ?

Relational DataBase Manament System

Maintain data in **tables**

Pre-define structure

Each table has **Primary key**

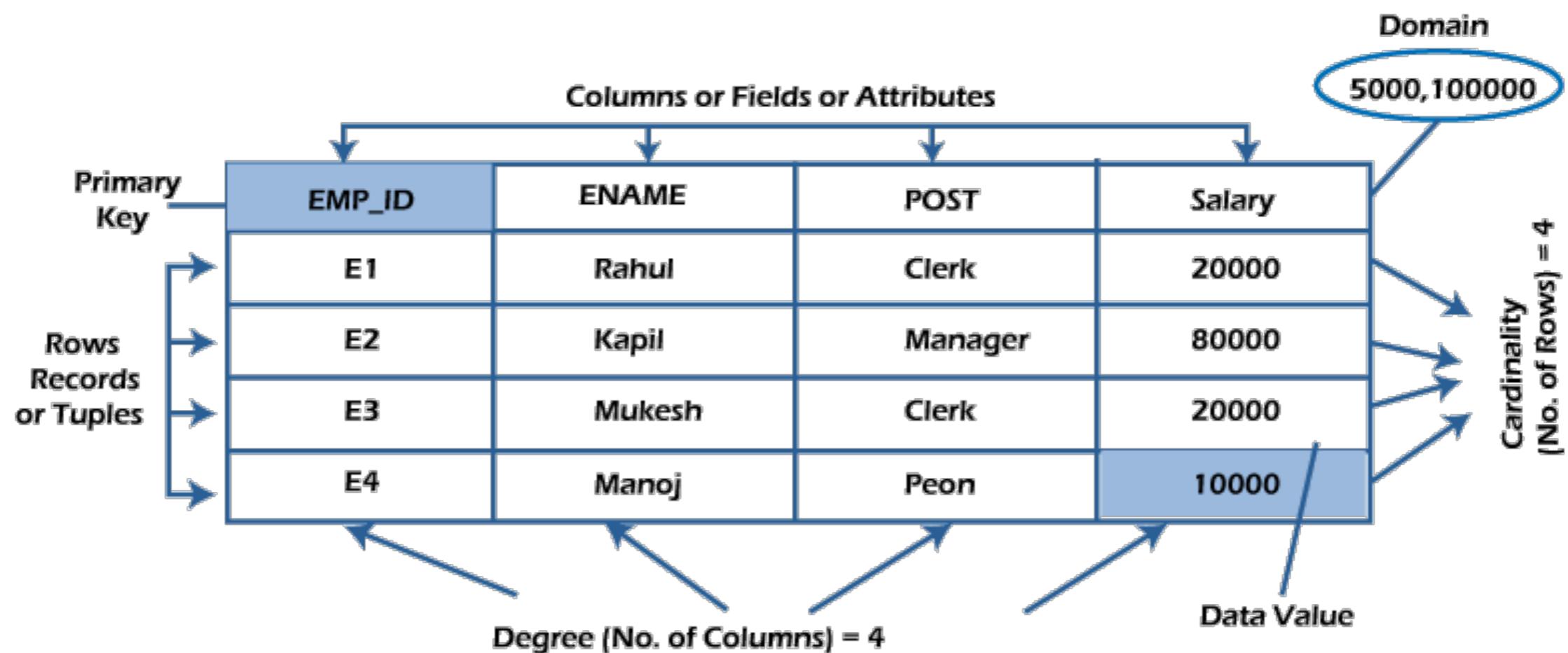
Many columns

Data is represented in term of **row**

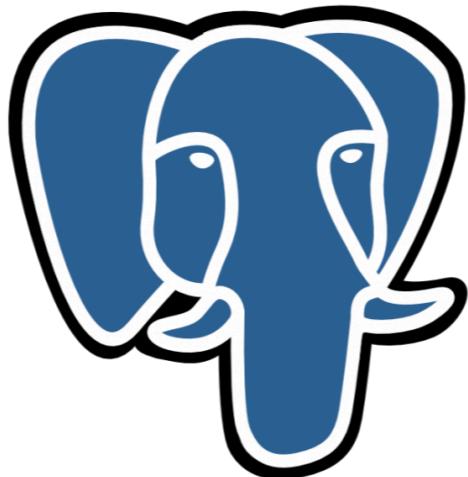


Relational Database ?

Table = Employee



Working with PostgreSQL



Postgre^{SQL}



PostgreSQL Database

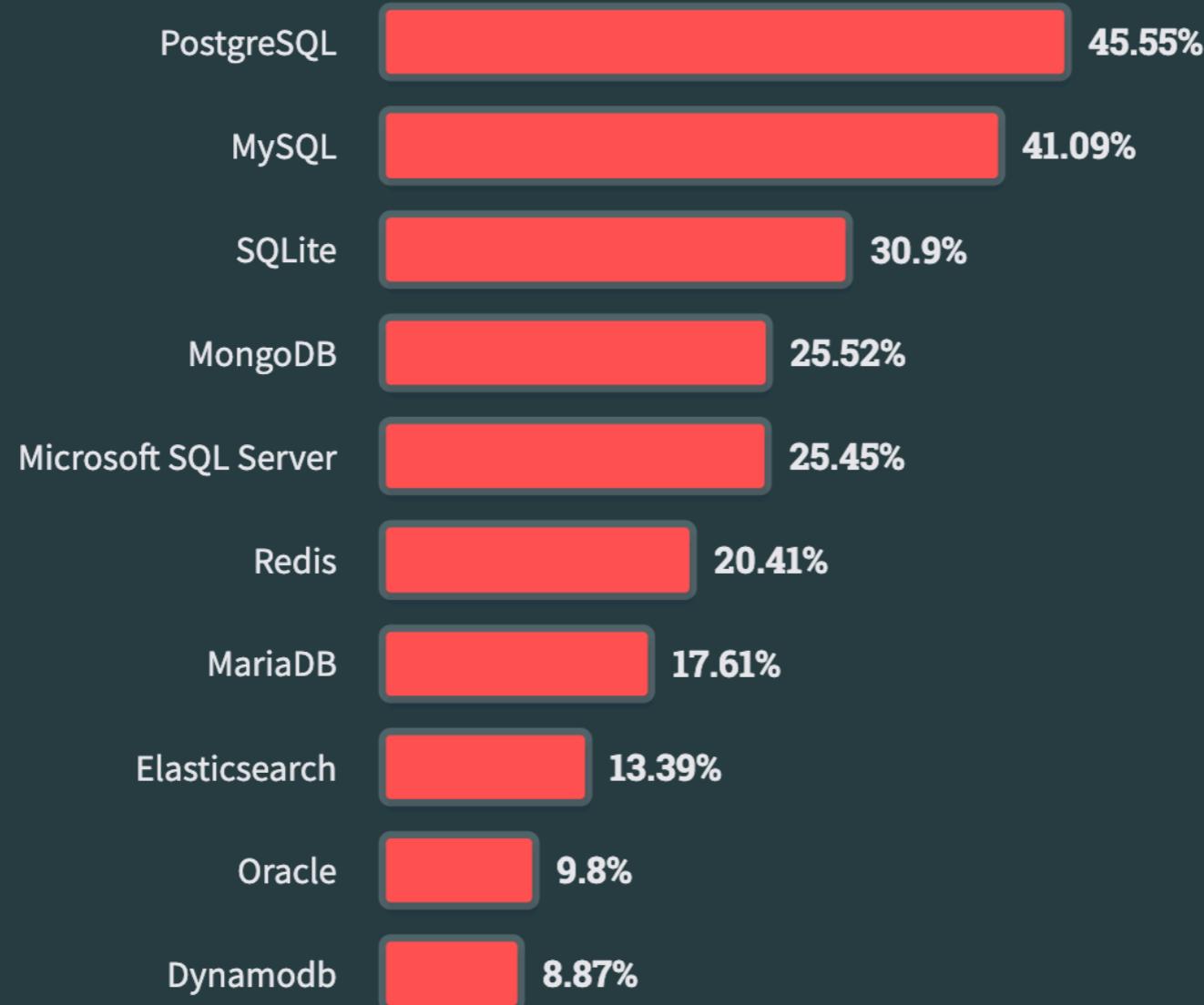
All Respondents

Professional Developers

Learning to Code

Other Coders

76,634 responses



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



Data Modeling

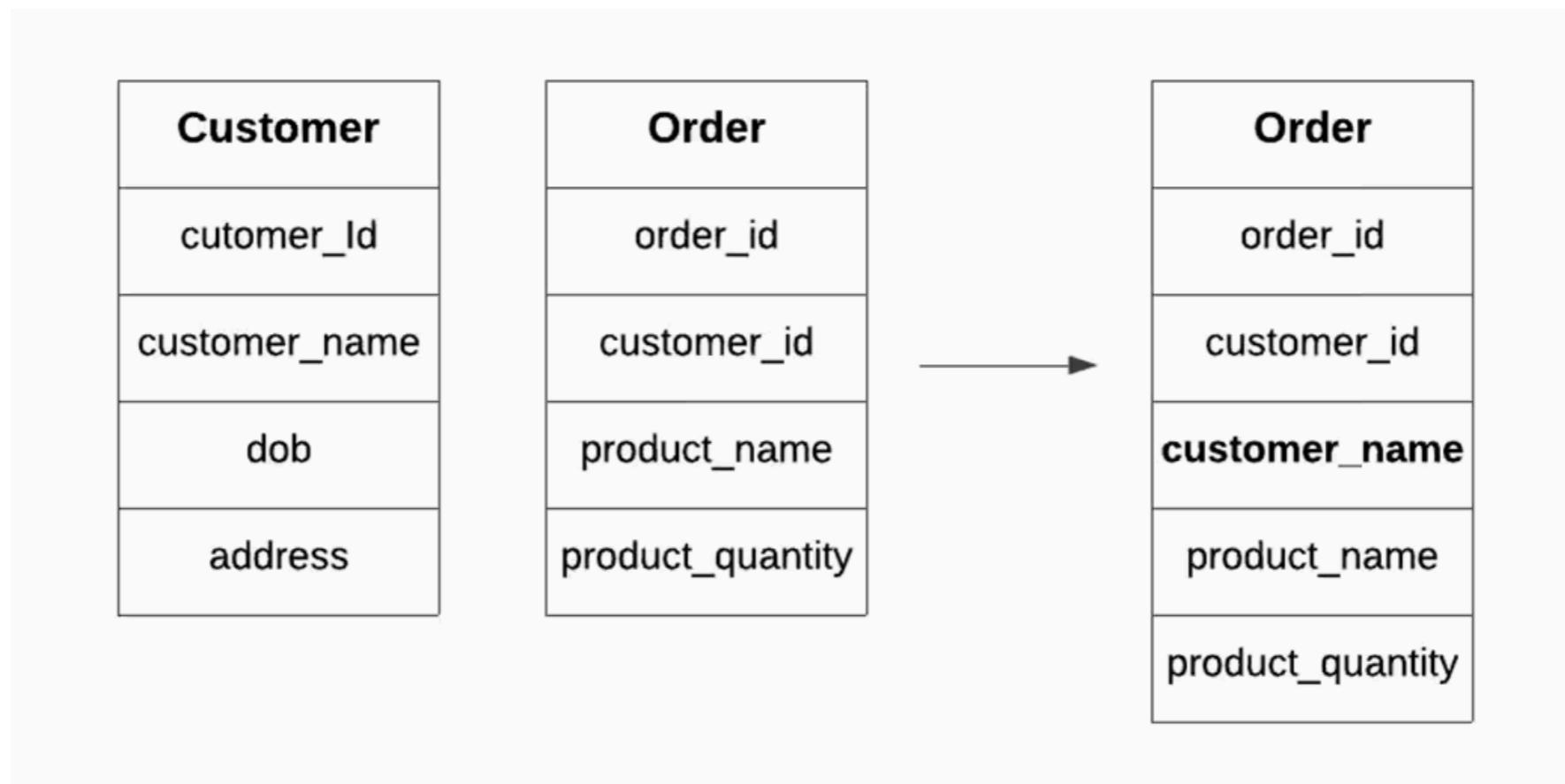
Read

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

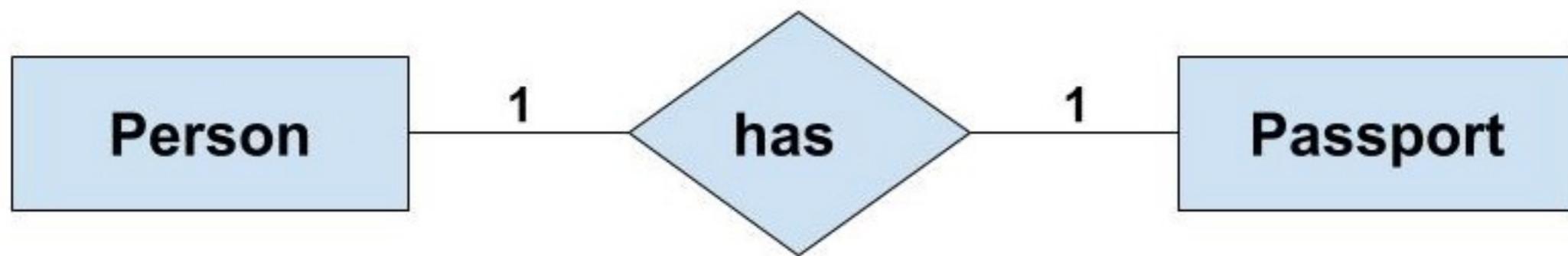


Relationships



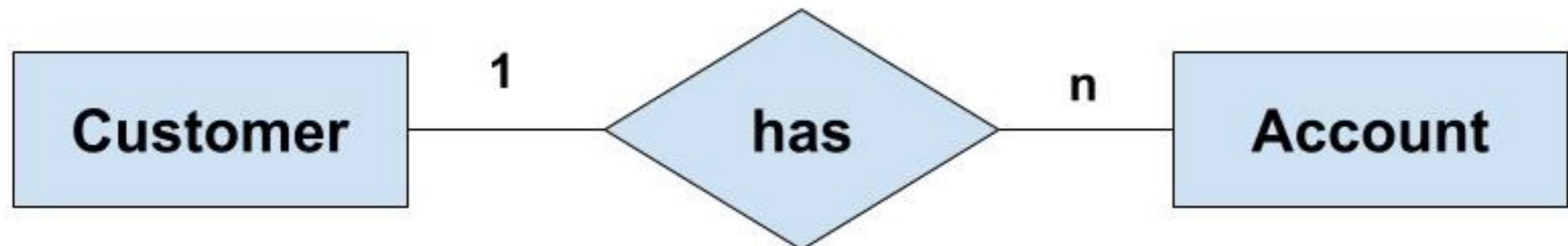
Relationship between table

One-to-one
One-to-many
Many-to-many



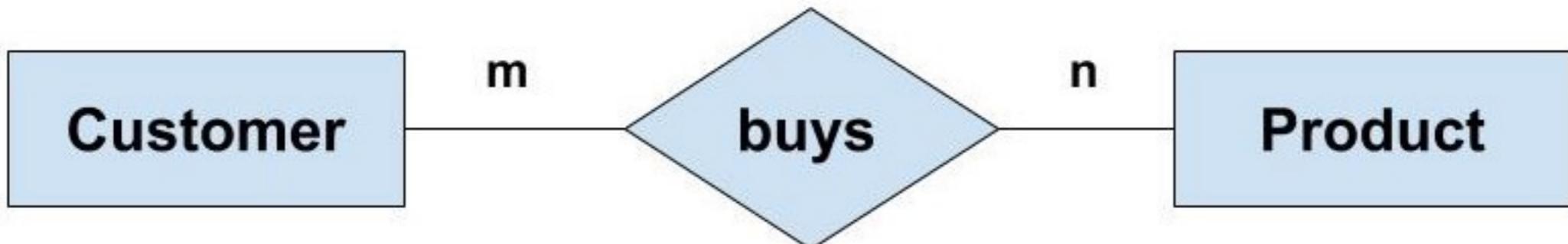
Relationship between table

One-to-one
One-to-many
Many-to-many



Relationship between table

One-to-one
One-to-many
Many-to-many

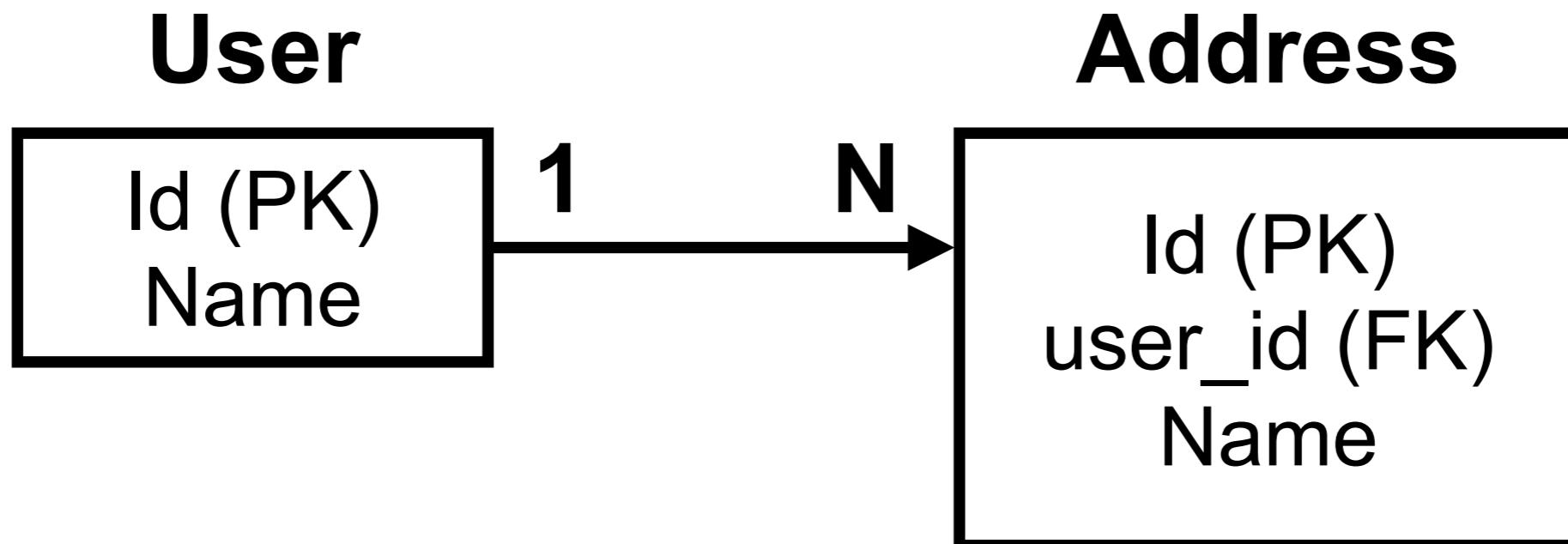


Relationships

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



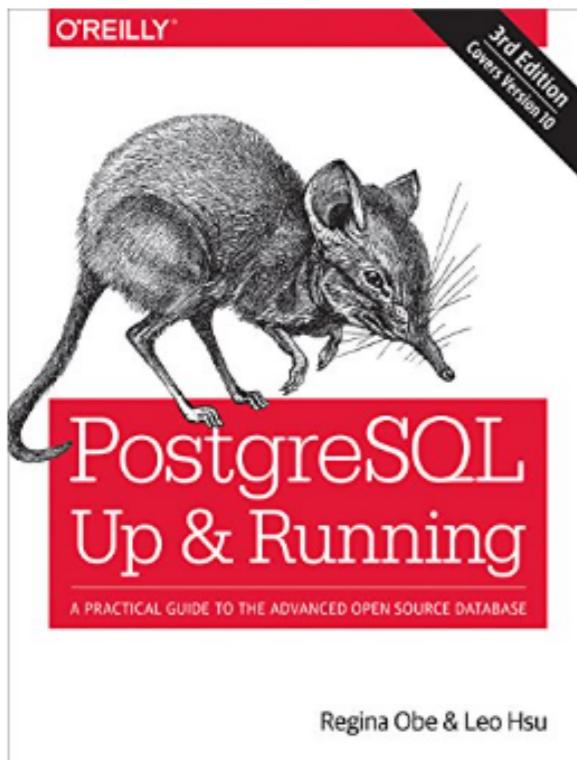
Book Store

QuickStart
Guides.

Master SQL and Manage, Analyze and Manipulate Data

\$25¹⁹ ✓p

Kindle Store › Kindle eBooks › Computers & Technology



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 99 ratings

[See all formats and editions](#)

Kindle
\$10.09 - \$28.99

Paperback
\$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, you'll also discover that PostgreSQL is more than a database system—it'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

ANSI-SQL constructs found only in PostgreSQL: foreign data wrapper, window functions, and more. The book also covers the latest PostgreSQL 10 features, including support for JSONB, and the new column-oriented TimescaleDB extension. TimescaleDB is a column-oriented PostgreSQL extension that provides automatic time-series data compression and indexing for PostgreSQL. It is designed to handle large volumes of time-series data efficiently, making it suitable for real-time monitoring, operational intelligence, and machine learning applications.

[▼ Read more](#)

[Read sample](#)

Follow the Author



[Regina Obe](#)

[Follow](#)

ISBN-13



978-1491963418

Edition

#

3rd

Sticky notes



[On Kindle Scribe](#)

Publisher



O'Reilly Media

Publication date



October 10, 2017

Language



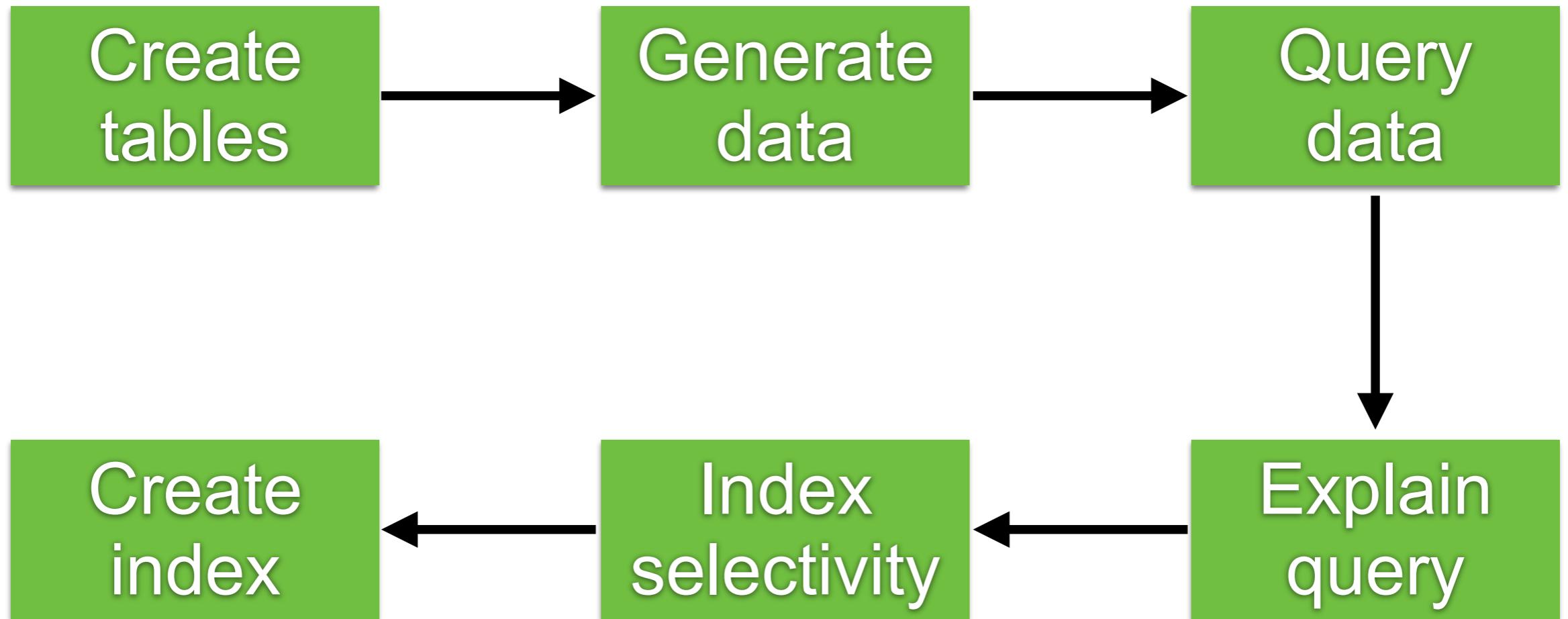
English



Book Store



Step in workshop

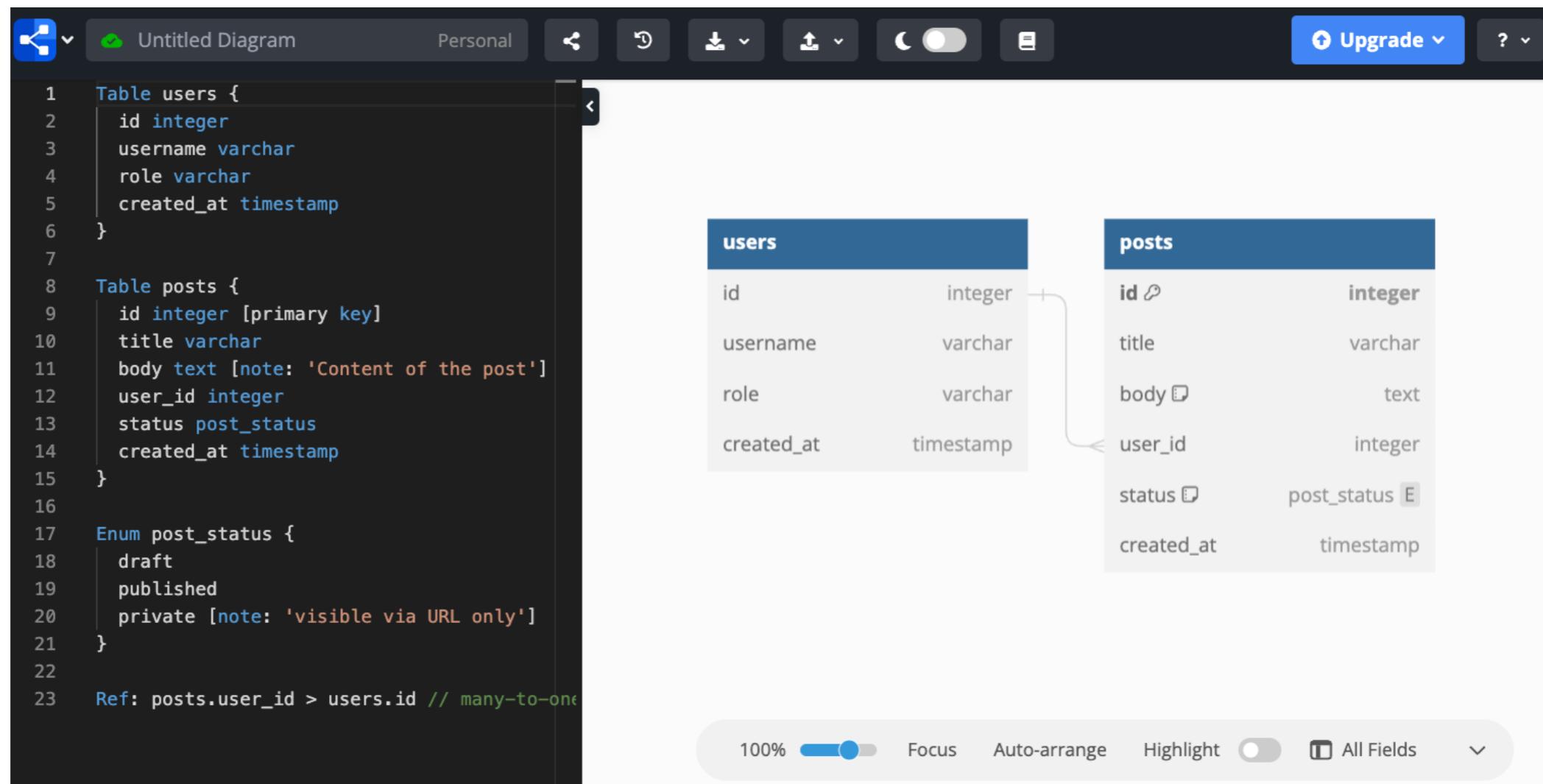


Design Data Model



Design table with DBML

Database Markup Language

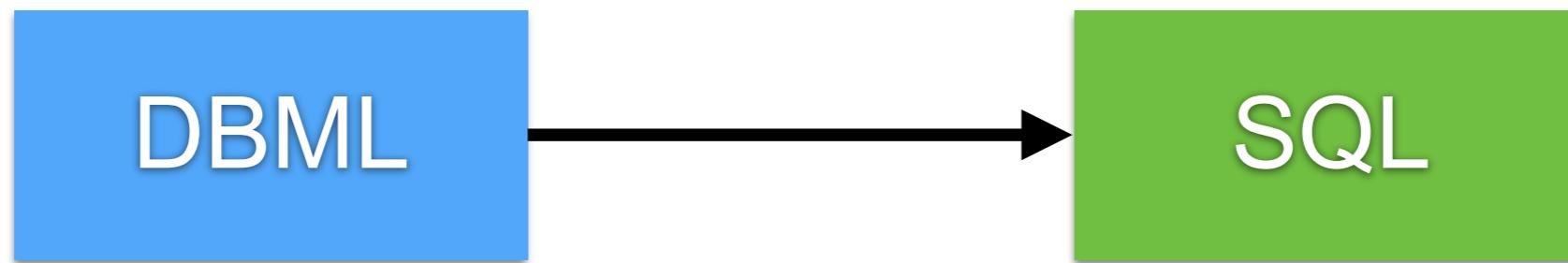


<https://dbml.dbdiagram.io/home/>



Convert SQL to DBML

\$npm install -g @dbml/cli

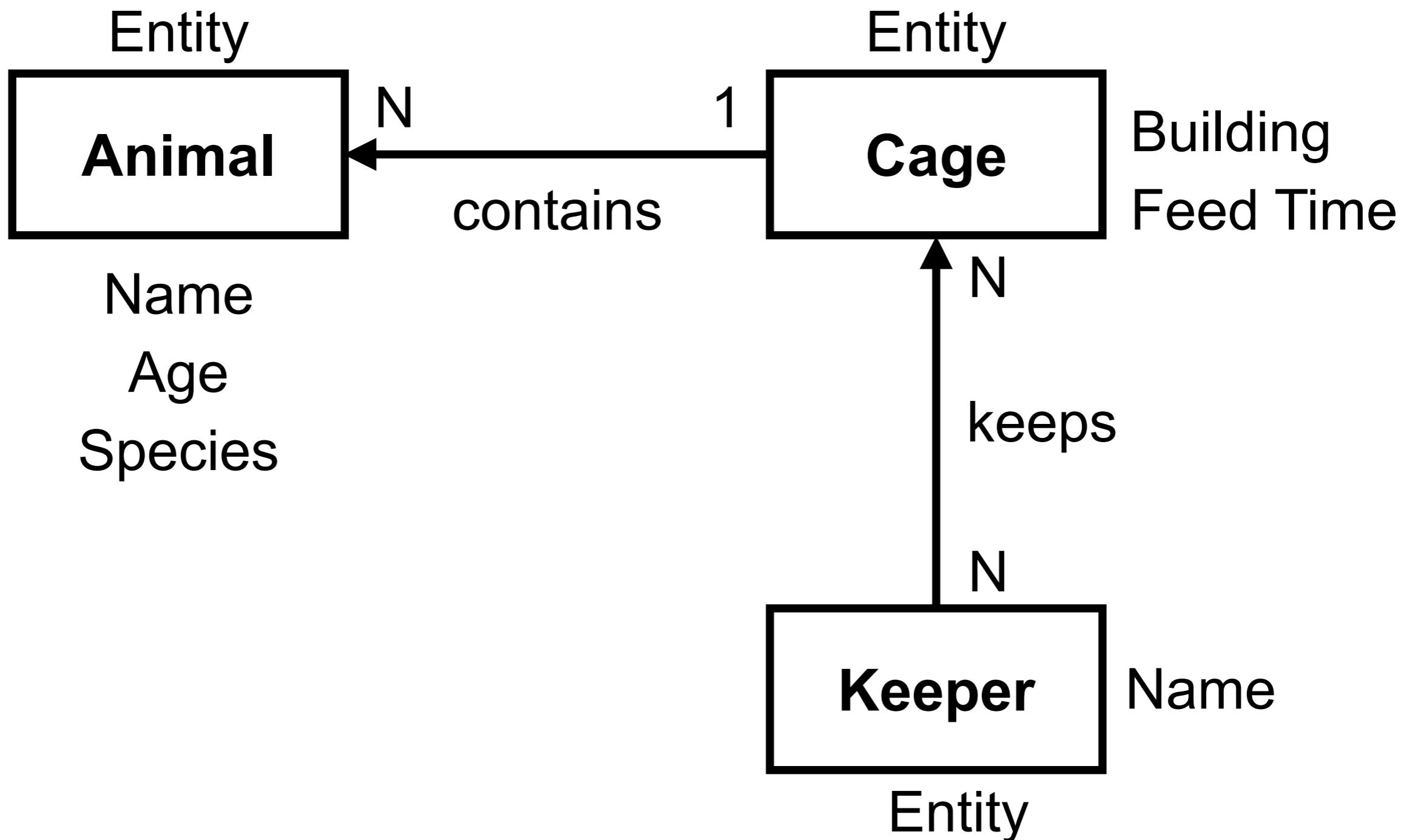


<https://dbml.dbdiagram.io/cli/#convert-a-sql-file-to-dbml>



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



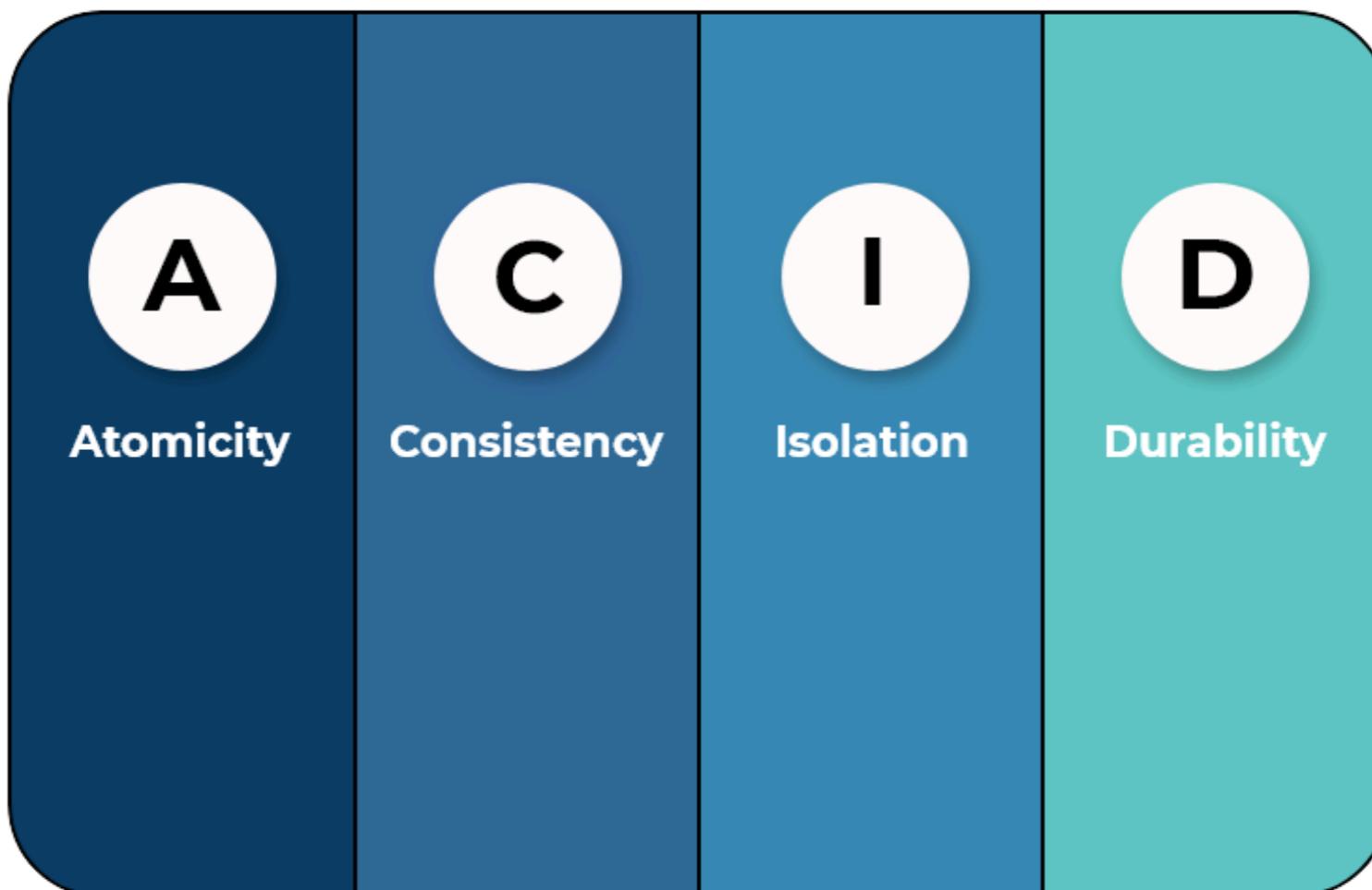
ACID in RDBMS

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

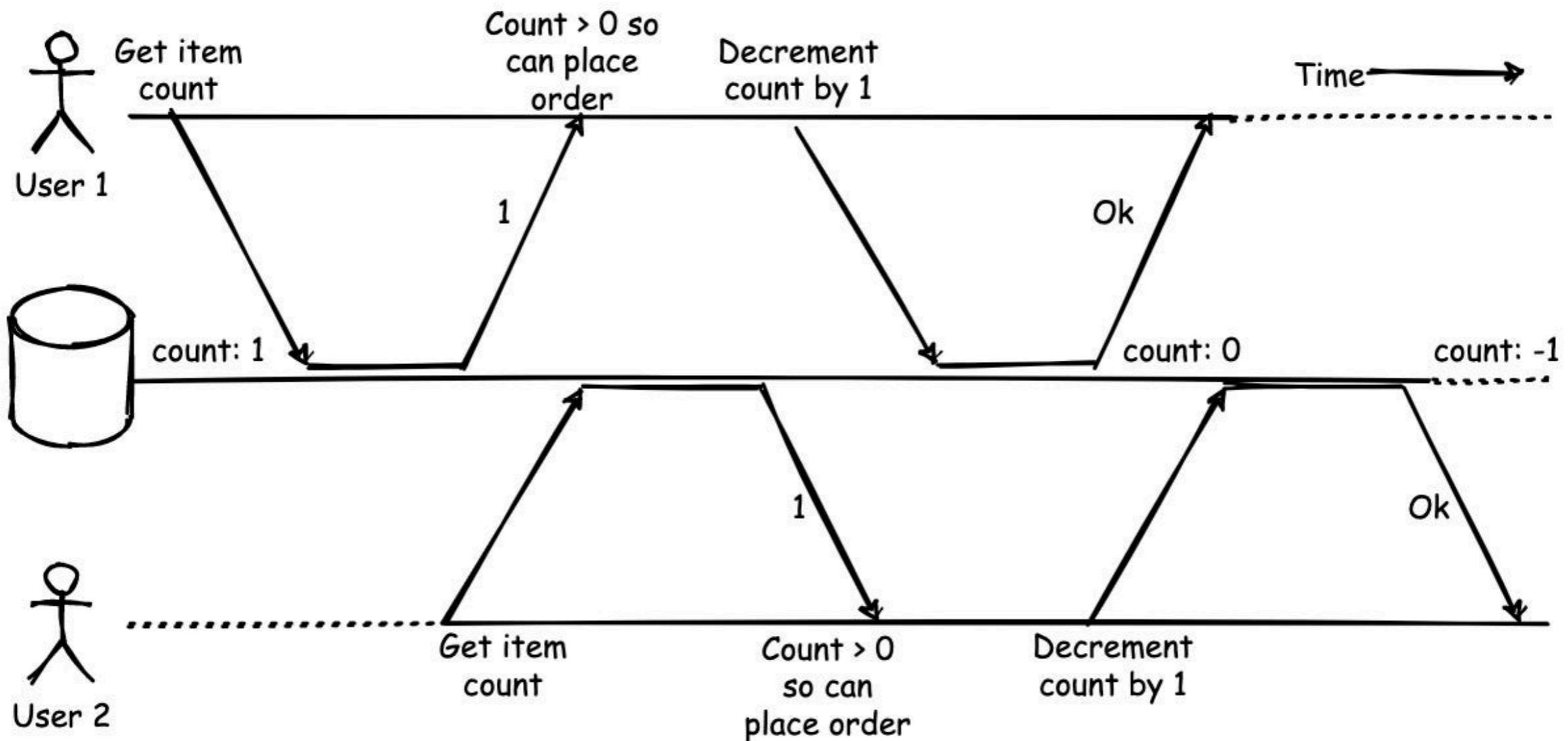


ACID in RDBMS

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



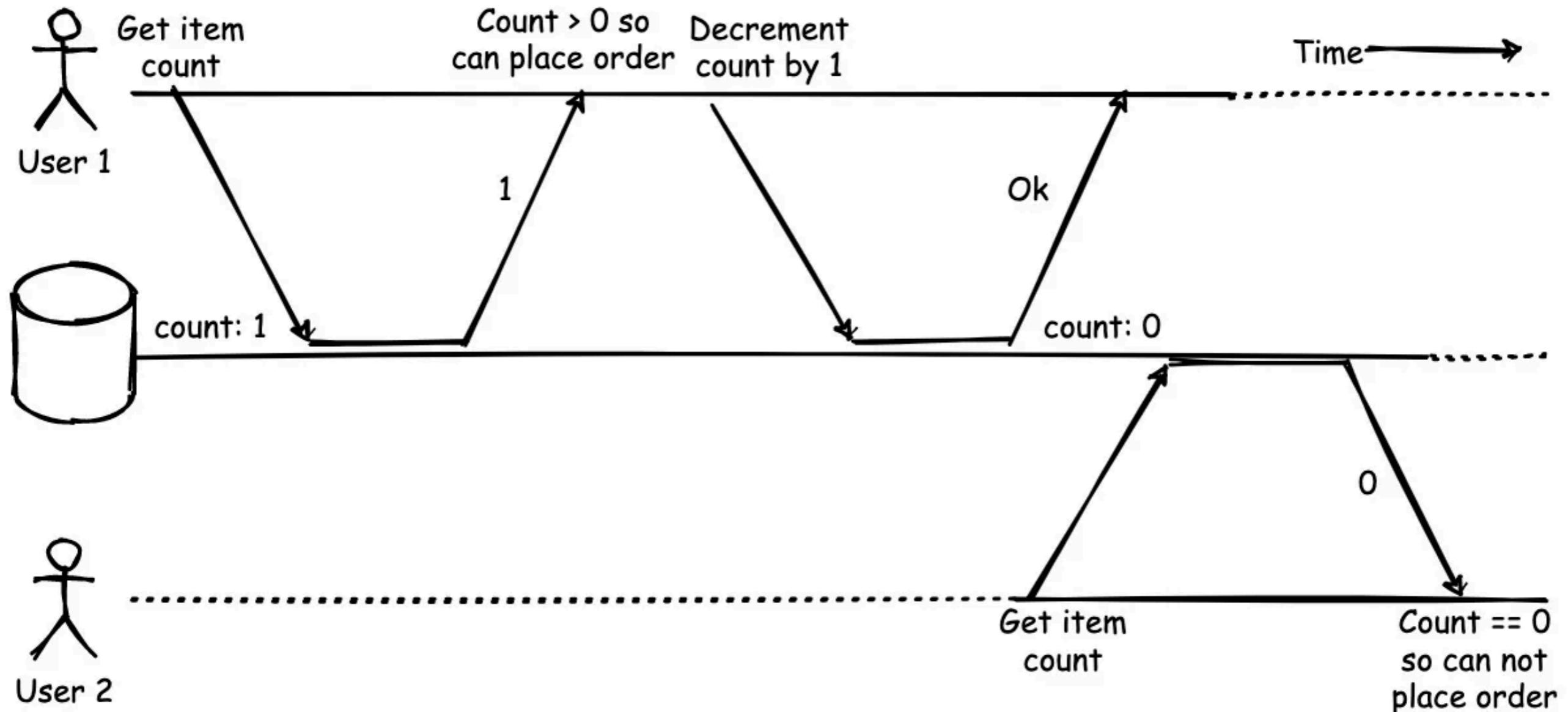
Race condition



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



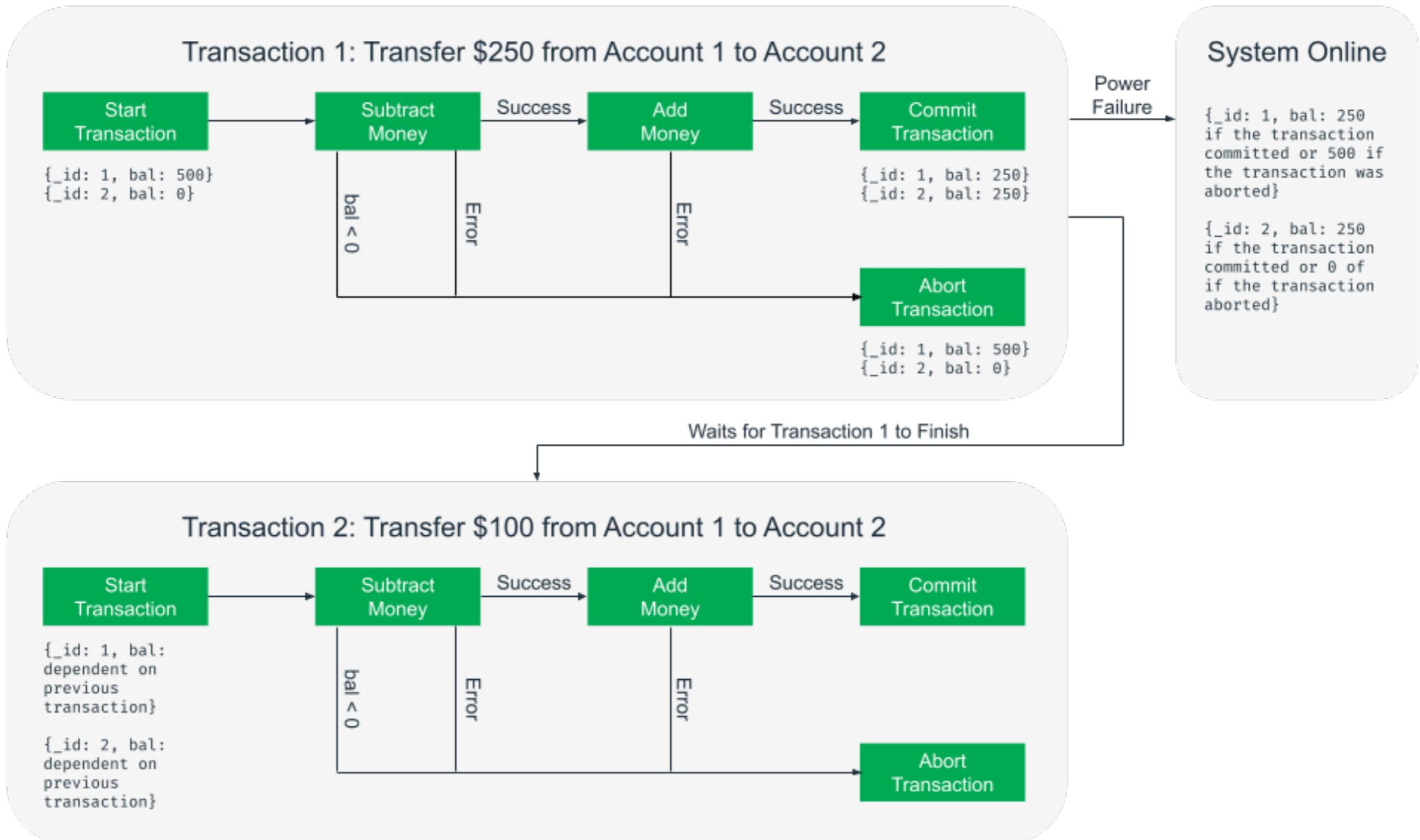
Sequencial



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



ACID in RDBMS



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



StructureQueryLanguage



SQL

Maintain data in **tables**

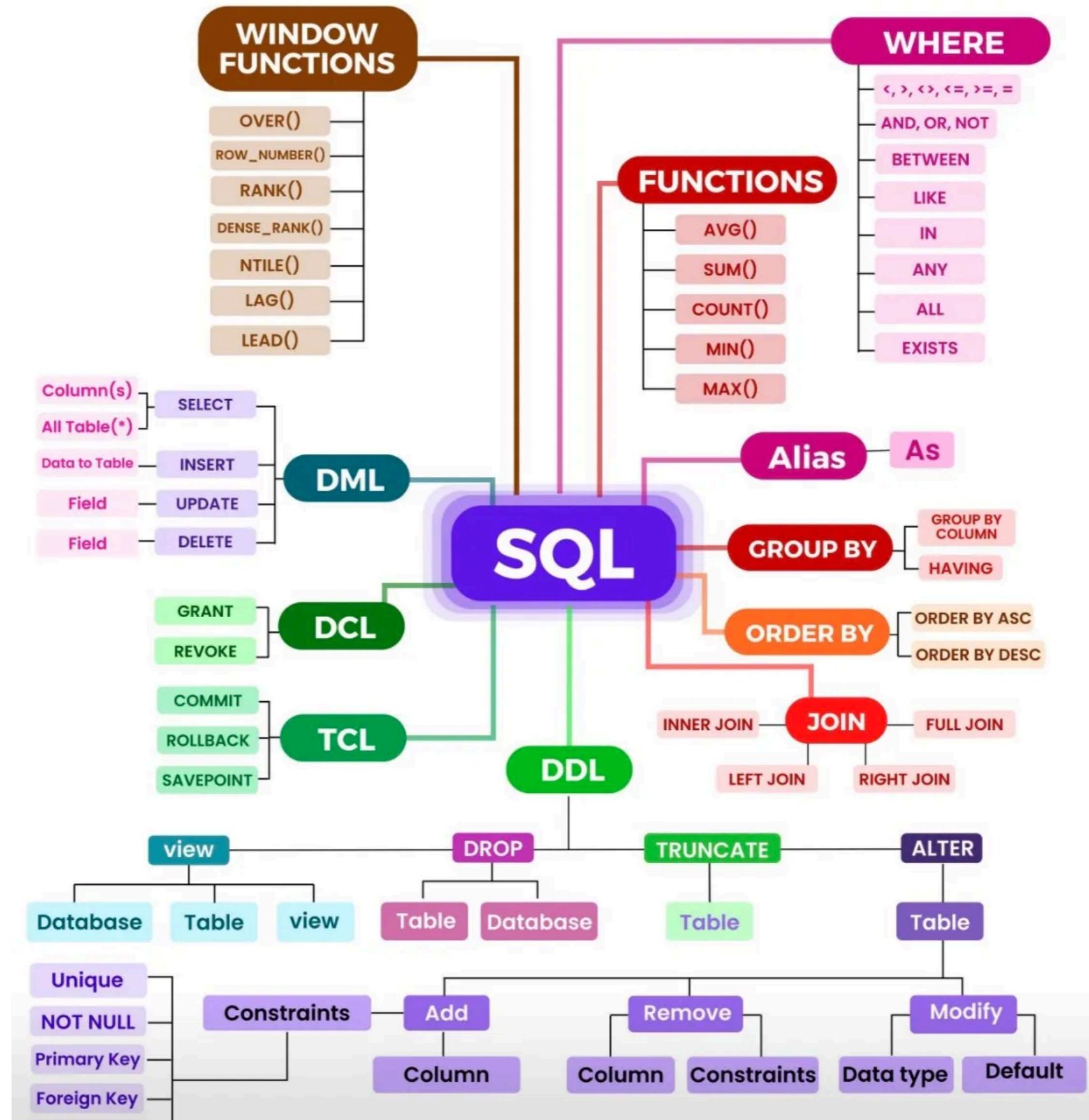
Pre-define structure

Each table has **Primary key**

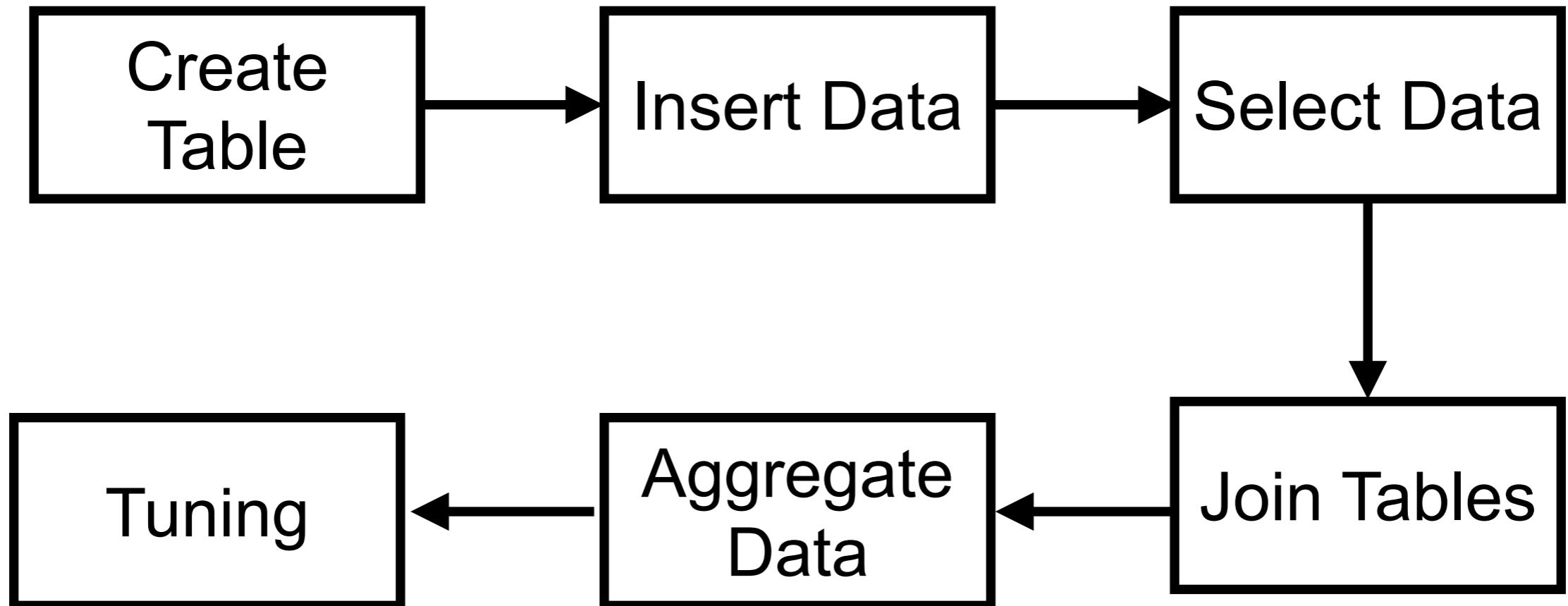
Many columns

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

Table A

Table B

Table C



Select Data process

Table A

Table B

Table C

FROM + JOIN
(merge)

Table A, B, C



Select Data process

Table A

Table B

Table C

FROM + JOIN
(merge)

Table A, B, C

WHERE
(filter)

Table A, B, C



Select Data process

Table A

Table B

Table C

FROM + JOIN
(merge)

Table A, B, C

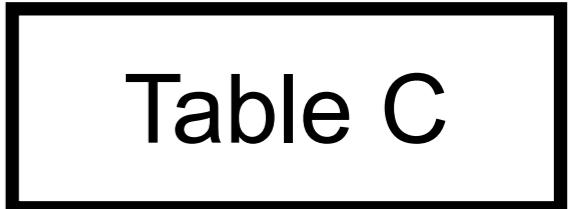
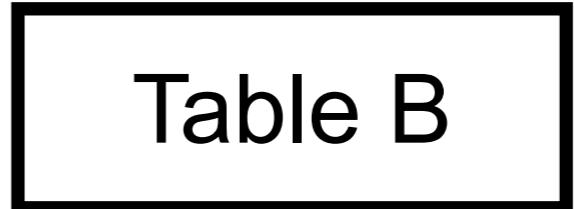
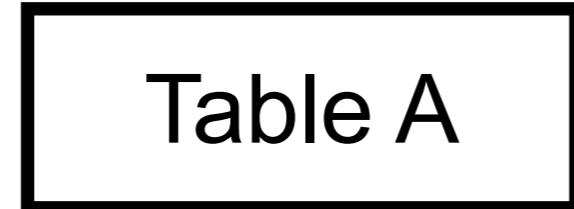
WHERE
(filter)

Table A, B, C

Group By
(Group)



Select Data process



FROM + JOIN
(merge)

Table A, B, C

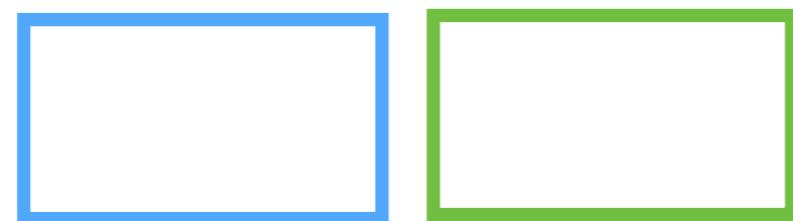
WHERE
(filter)

Table A, B, C

Group By
(Group)



Having
(filter)

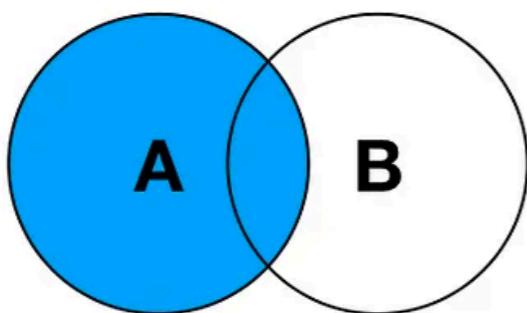


Joining

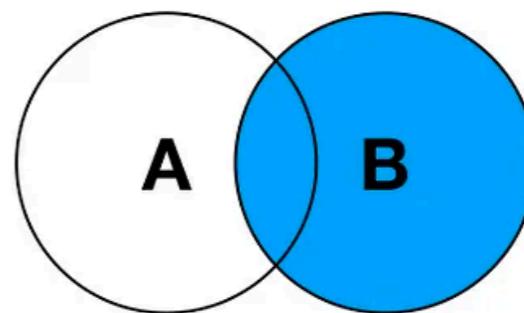
<https://www.postgresql.org/docs/current/tutorial-join.html>



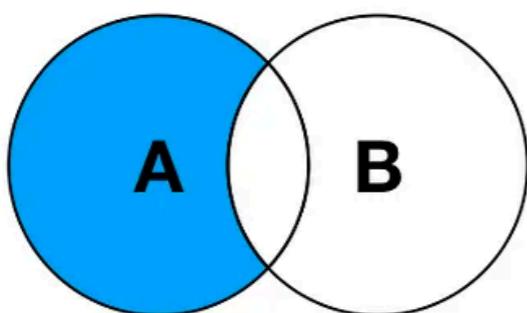
SQL JOINS



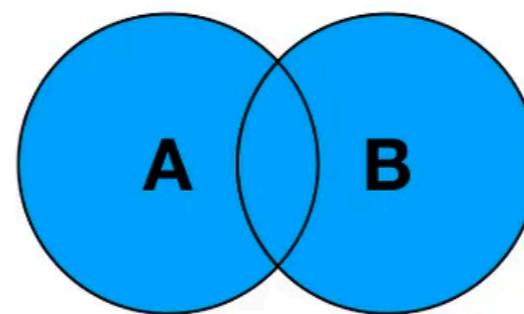
LEFT JOIN



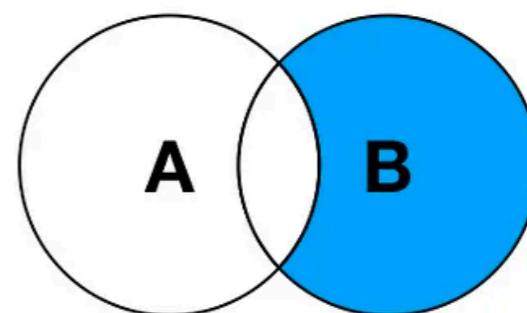
RIGHT JOIN



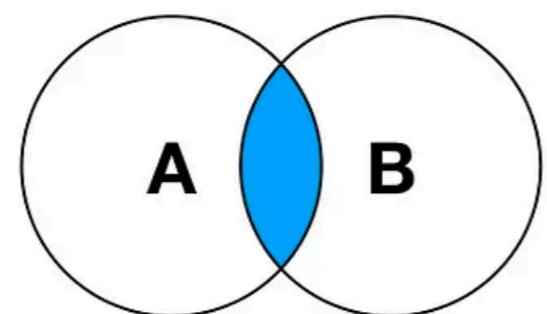
**LEFT JOIN EXCLUDING
INNER JOIN**



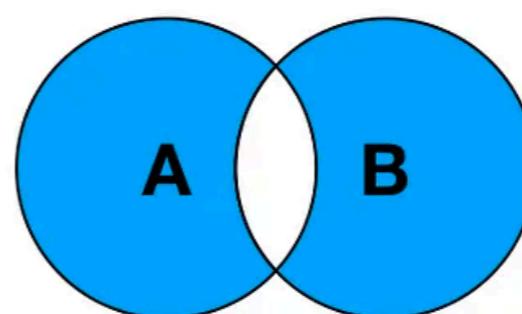
FULL OUTER JOIN



**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

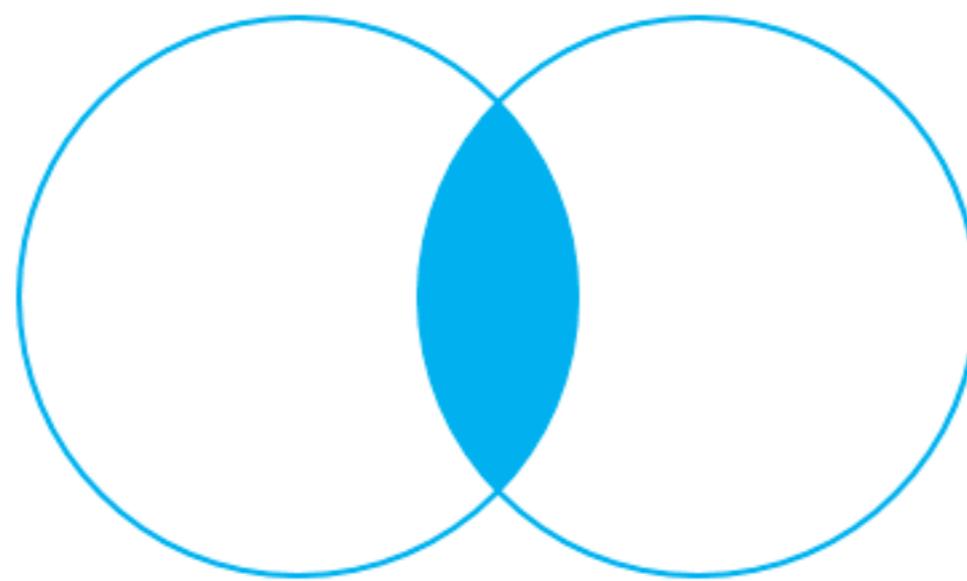
Apple
Orange
Banana
Cucumber

Basket B

Orange
Apple
Watermelon
Pear



Inner join



INNER JOIN



Inner join

Basket A

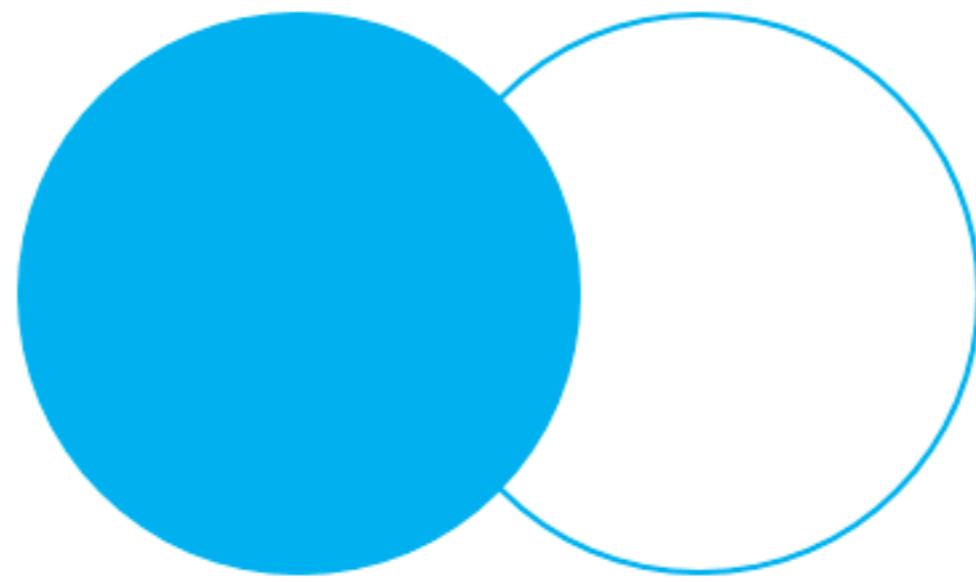
Apple
Orange
Banana
Cucumber

Basket B

Orange
Apple
Watermelon
Pear



Left join (1)



LEFT OUTER JOIN



Left join (1)

Basket A

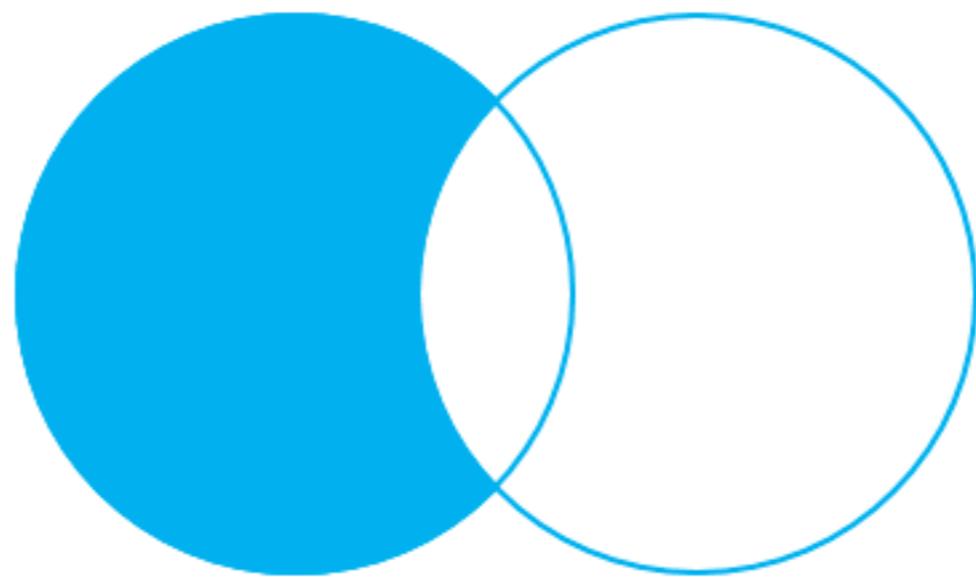
Apple
Orange
Banana
Cucumber

Basket B

Orange
Apple
Watermelon
Pear



Left join (2)



LEFT OUTER JOIN – only
rows from the left table



Left join (2)

Basket A

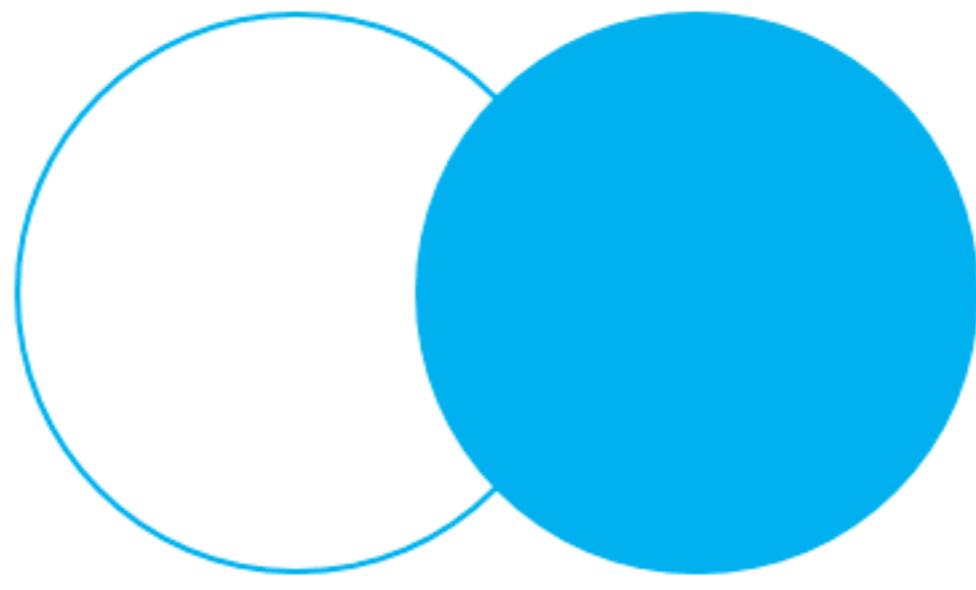
Apple
Orange
Banana
Cucumber

Basket B

Orange
Apple
Watermelon
Pear



Right join (1)



RIGHT OUTER JOIN



Right join (1)

Basket A

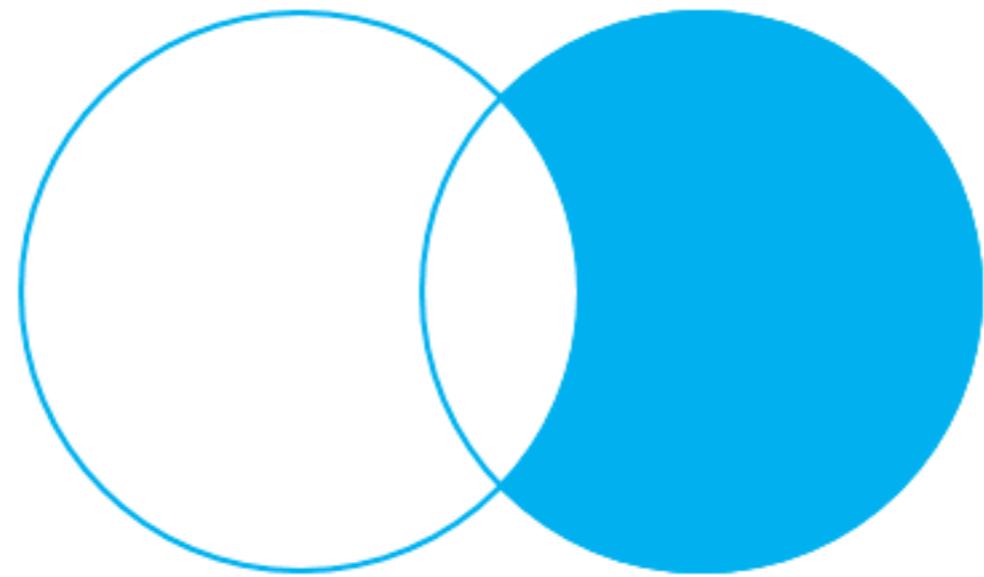
Apple
Orange
Banana
Cucumber

Basket B

Orange
Apple
Watermelon
Pear



Right join (2)



RIGHT OUTER JOIN – only
rows from the right table



Right join (2)

Basket A

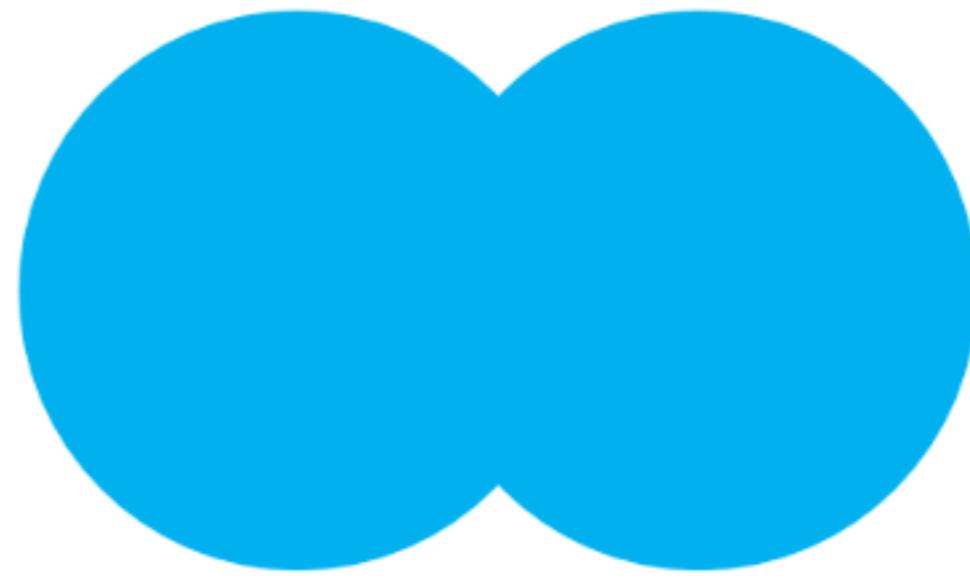
Apple
Orange
Banana
Cucumber

Basket B

Orange
Apple
Watermelon
Pear



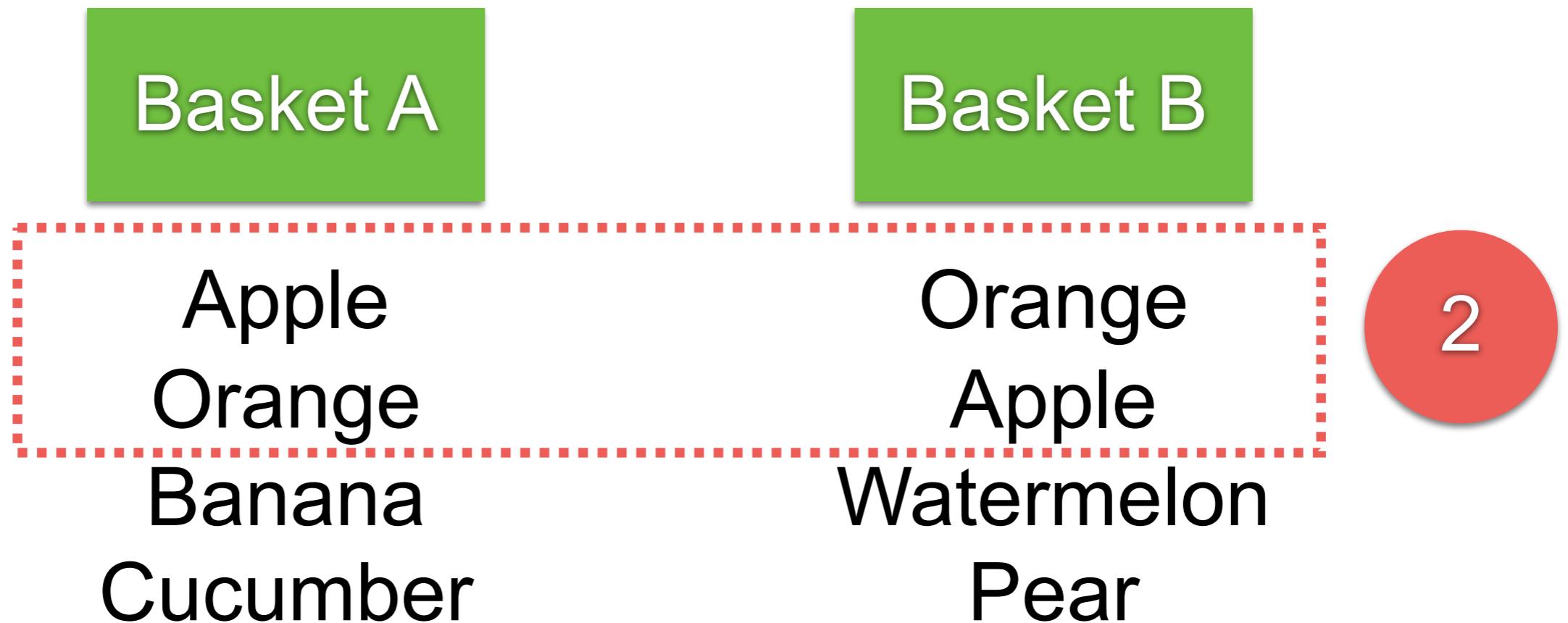
Full outer join (1)



FULL OUTER JOIN



Full outer join (1)



Full outer join (1)

Basket A

2

Apple
Orange
Banana
Cucumber

Basket B

Orange
Apple
Watermelon
Pear



Full outer join (1)

Basket A

Apple
Orange
Banana
Cucumber

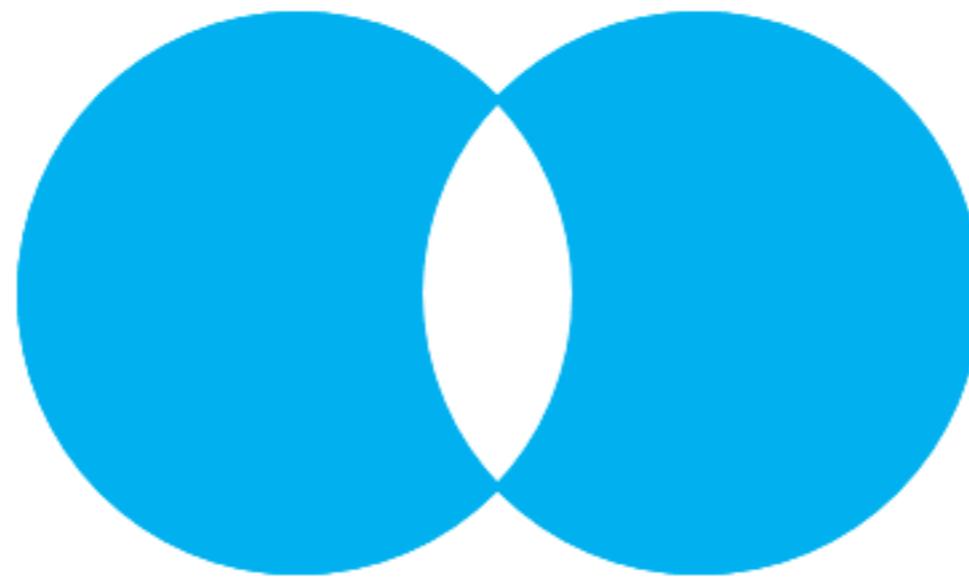
Basket B

Orange
Apple
Watermelon
Pear

2



Full outer join (2)



FULL OUTER JOIN – only
rows unique to both tables



Full outer join (2)

Basket A

2

Apple
Orange
Banana
Cucumber

Basket B

Orange
Apple
Watermelon
Pear



Full outer join (2)

Basket A

Apple
Orange
Banana
Cucumber

Basket B

Orange
Apple
Watermelon
Pear

2



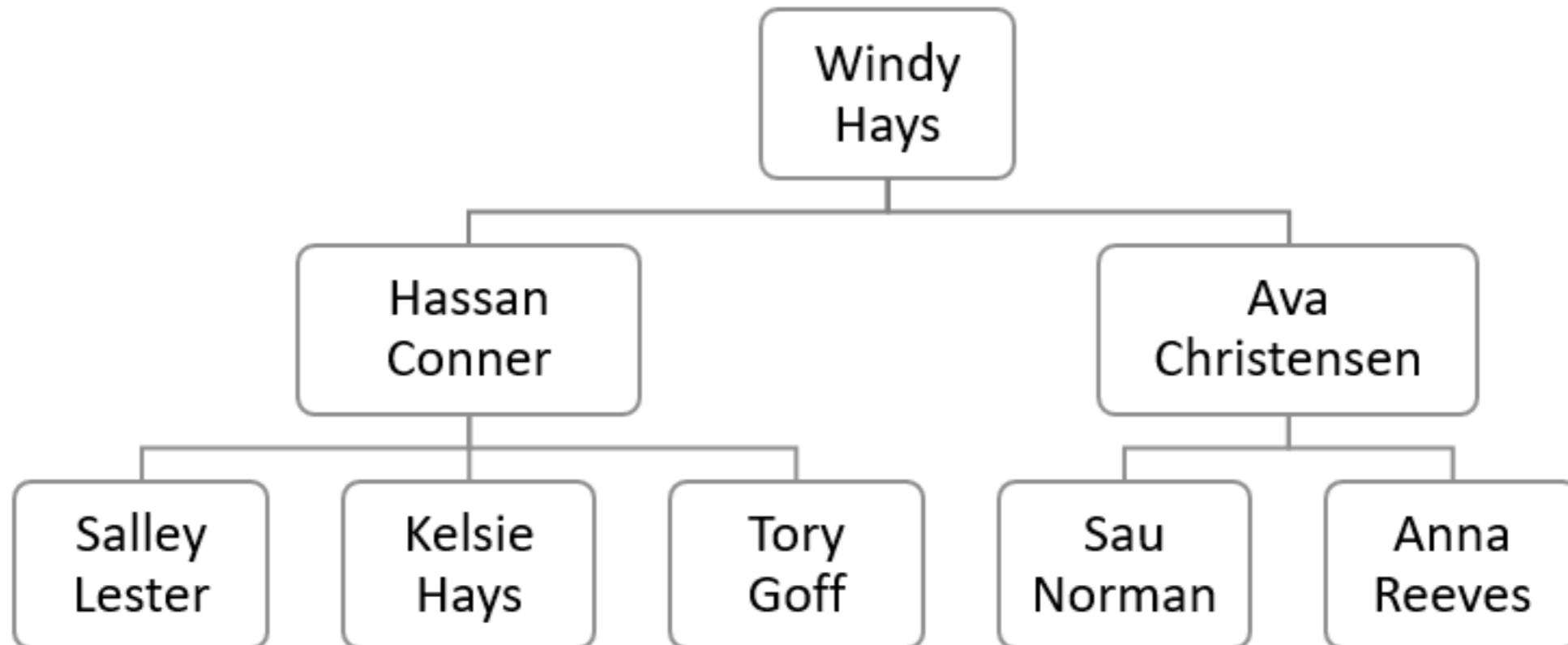
Workshop :: Joins

[https://github.com/up1/workshop-database/blob/main/
workshop/joins.md](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](https://github.com/up1/workshop-database/blob/main/workshop/self-join.md)



Problems ?



Problems

Slow query

Accurate knowledge is required

Add complexity

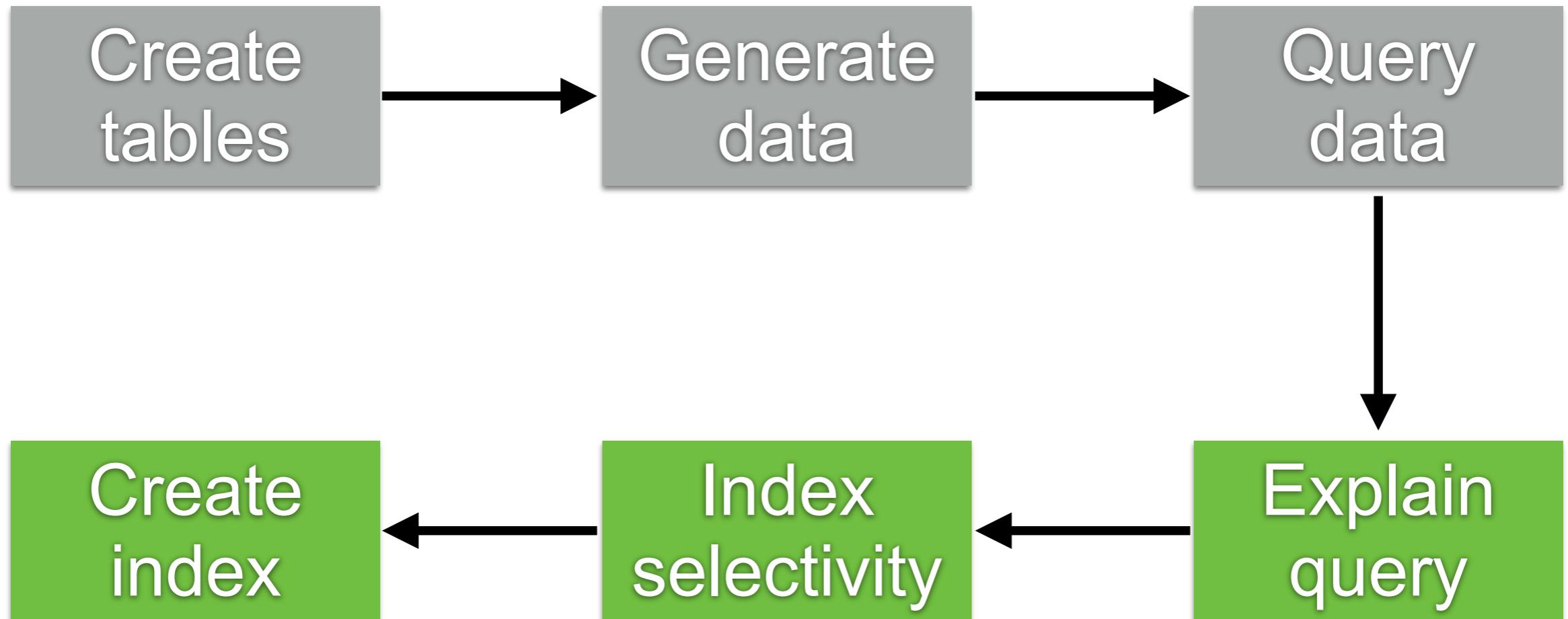
Need denormalization ?



Improvement to query data



Step in workshop



Query plan in PostgreSQL

EXPLAIN ANALYZE <SQL>

The screenshot shows a PostgreSQL query tool interface. At the top, there are tabs for "Query" and "Query History". Below the tabs, a query is entered: "EXPLAIN ANALYZE select * from book;". The "Data Output" tab is selected in the toolbar, which also includes "Messages", "Graph Visualiser", "Explain", and "Notifications". The "Explain" tab has an "X" icon next to it. Below the toolbar is a row of icons: a grid with a plus sign, a file icon, a dropdown arrow, a clipboard icon, another dropdown arrow, a trash bin, a folder icon, a download arrow, and a refresh/clock icon. The main area displays the query plan:

	QUERY PLAN	
text		lock icon
1	Seq Scan on book (cost=0.00..2334.00 rows=100000 width=89) (actual time=0.007..8.479 rows=100000 loops=1)	
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

Symbols

2PC, 219, 346, 347, 349–351

A

A* pathfinding algorithms, 64, 84
ACID, 197, 208, 217, 219, 321
ActiveMQ, 92
adjacency lists, 77
Advanced Message Queuing Protocol, 125
Aeron, 402
aggregation window, 164, 176
Airbnb, 193, 199, 337
Amazon, 137, 199, 317
Amazon API Gateway, 303
Amazon Web Services, 251, 302
AML/CFT, 318
AMM, 409
AMQP, 125
Apache James, 231
append-only, 362, 365
Apple, 393
Apple Pay, 315
application loop, 398
ask price, 380
asynchronous, 328
At-least once, 122
at-least once, 93, 122
at-least-once, 331
at-most once, 93, 122
at-most-once, 331

atomic commit, 167
atomic operation, 218
audit, 360
Automatic Market Making, 409
Availability Zone, 253
Availability Zones, 268
availability zones, 27
AVRO, 165
AWS, 251, 302, 303
AWS Lambda, 303
AZ, 268

B

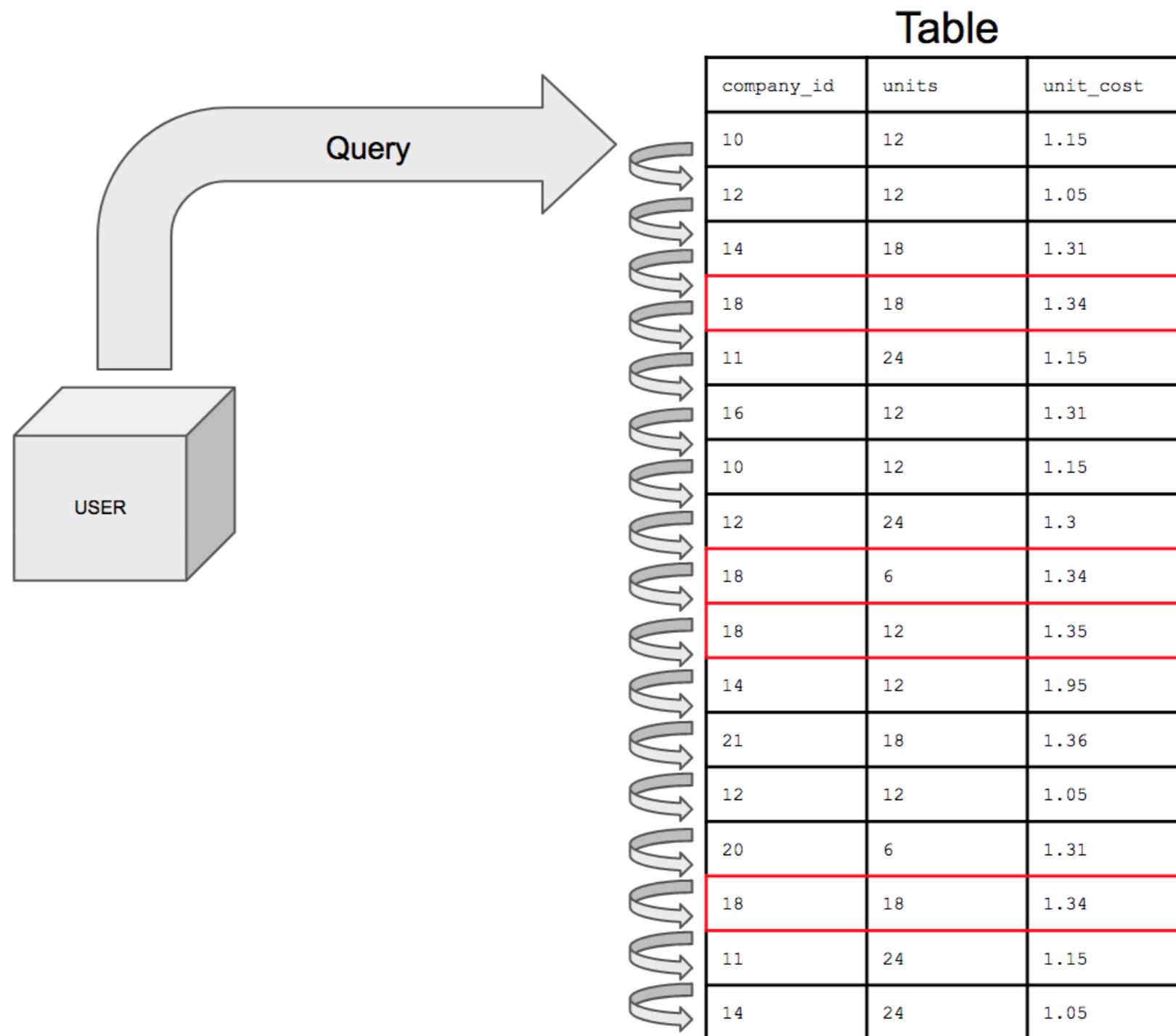
B+ tree, 267
Backblaze, 272
base32, 11
BEAM, 55
bid price, 380
Bigtable, 137, 235, 243
Blue/green deployment, 18
brokers, 95, 96, 98, 102, 105–107, 113, 118, 120, 122
buy order, 393

C

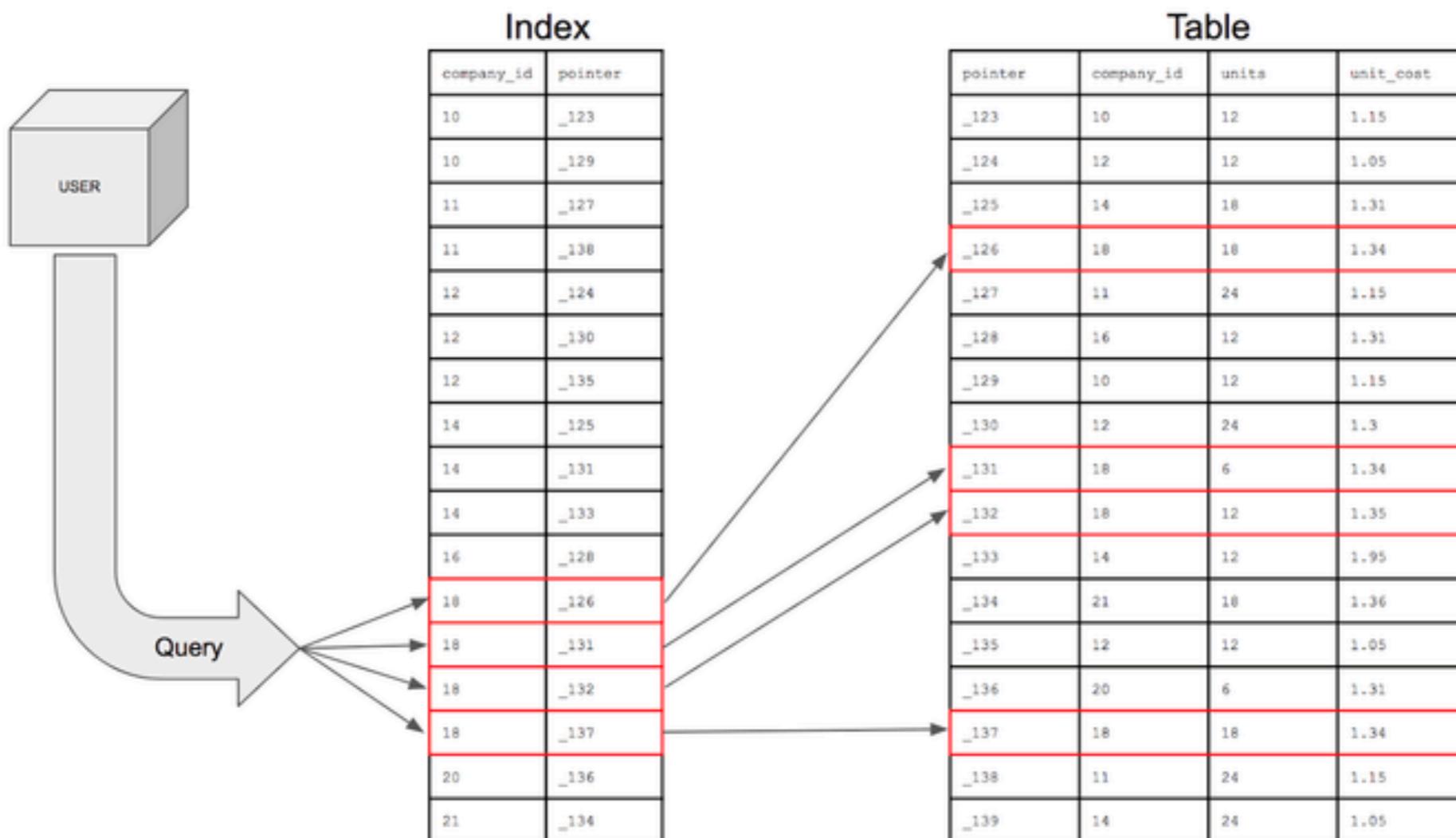
California Consumer Privacy Act, 2
candlestick chart, 381
candlestick charts, 384, 387, 396, 407
CAP theorem, 79



Without Index (full table scan)



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 by column "Column A"



Index selectivity

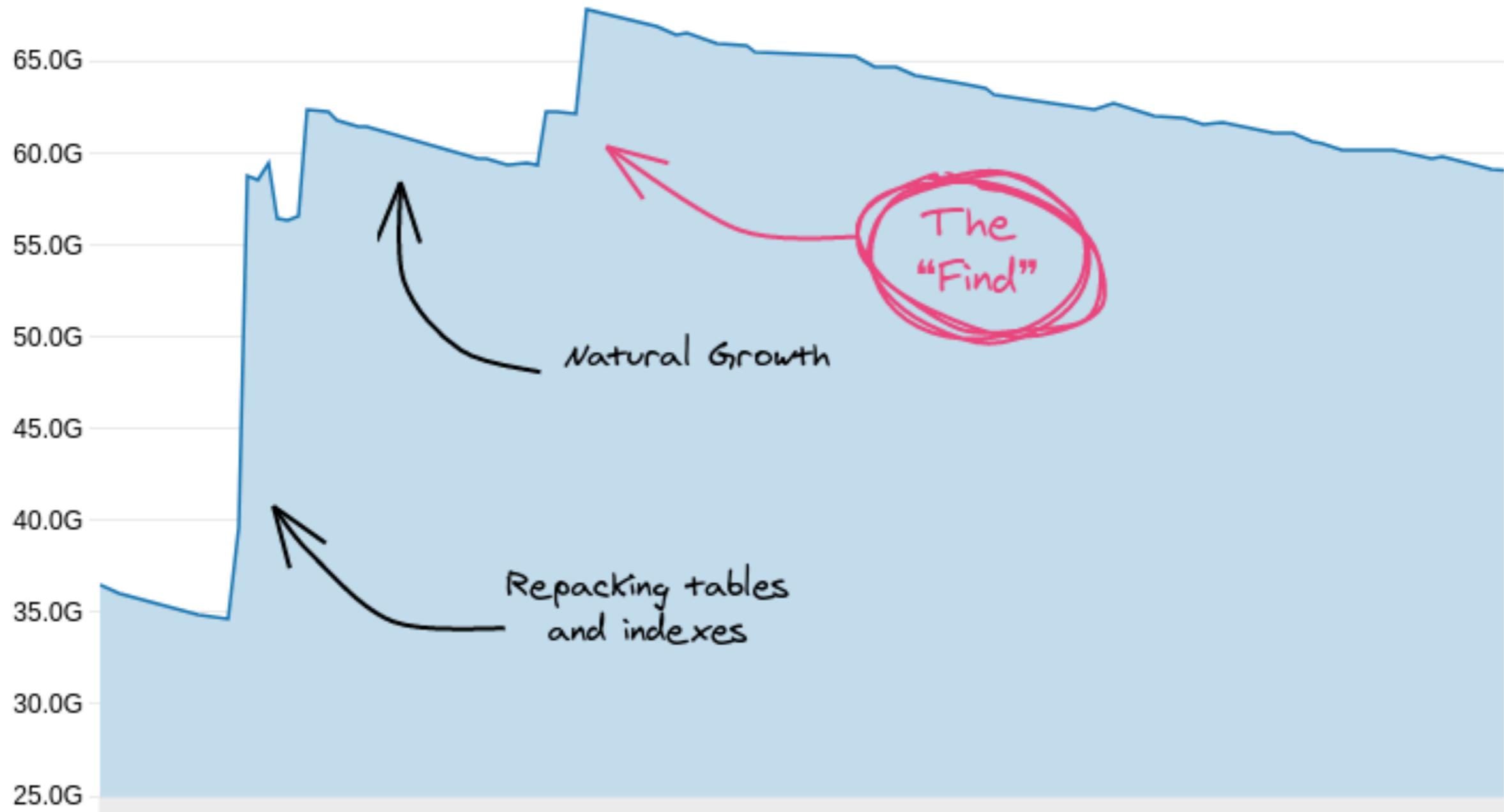
Factor to select a column to create an index

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)

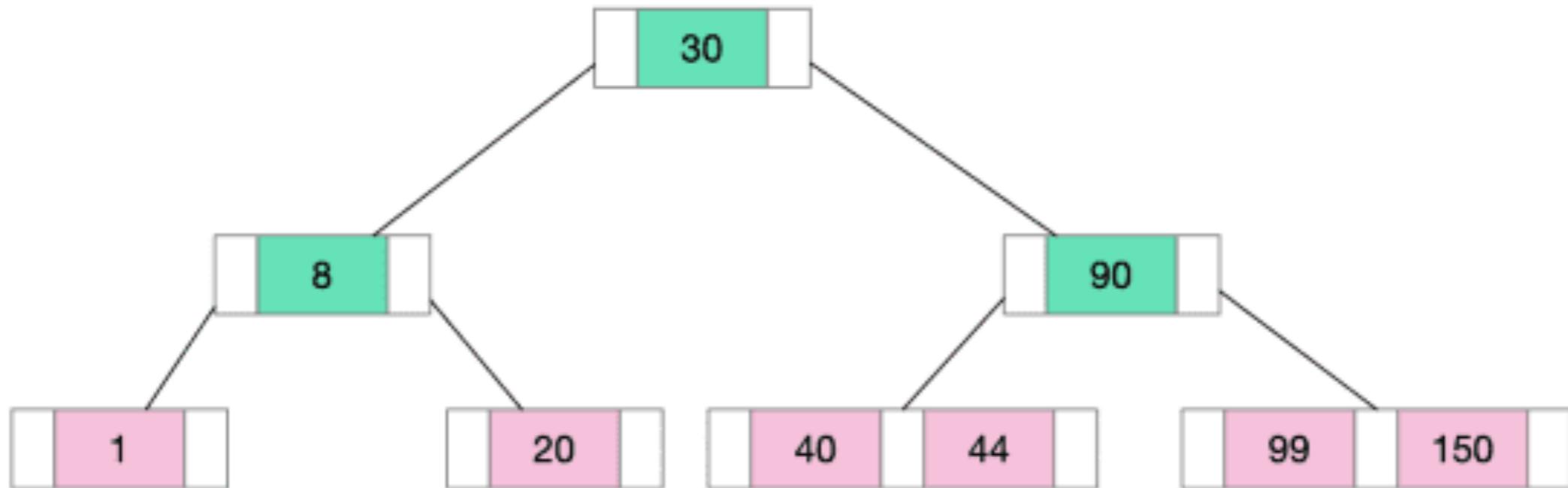
Generalized inverted search tree index (GiST)

Space partitioned GiST (SP-GiST)

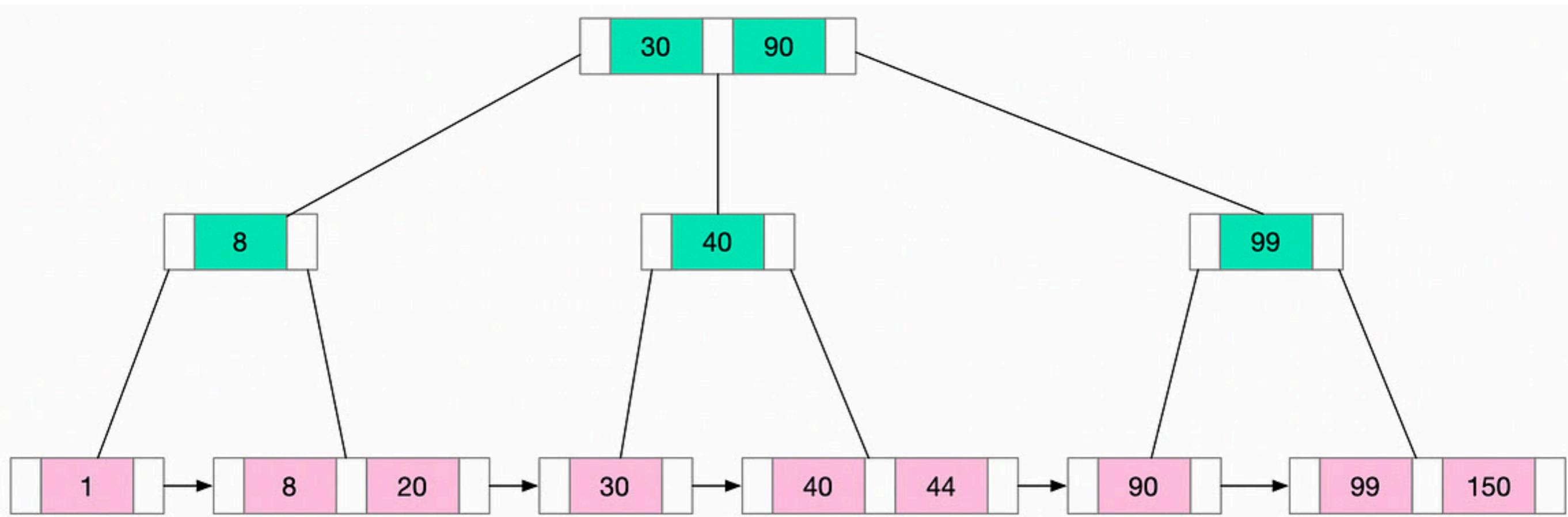
<https://www.postgresql.org/docs/current/indexes-types.html>



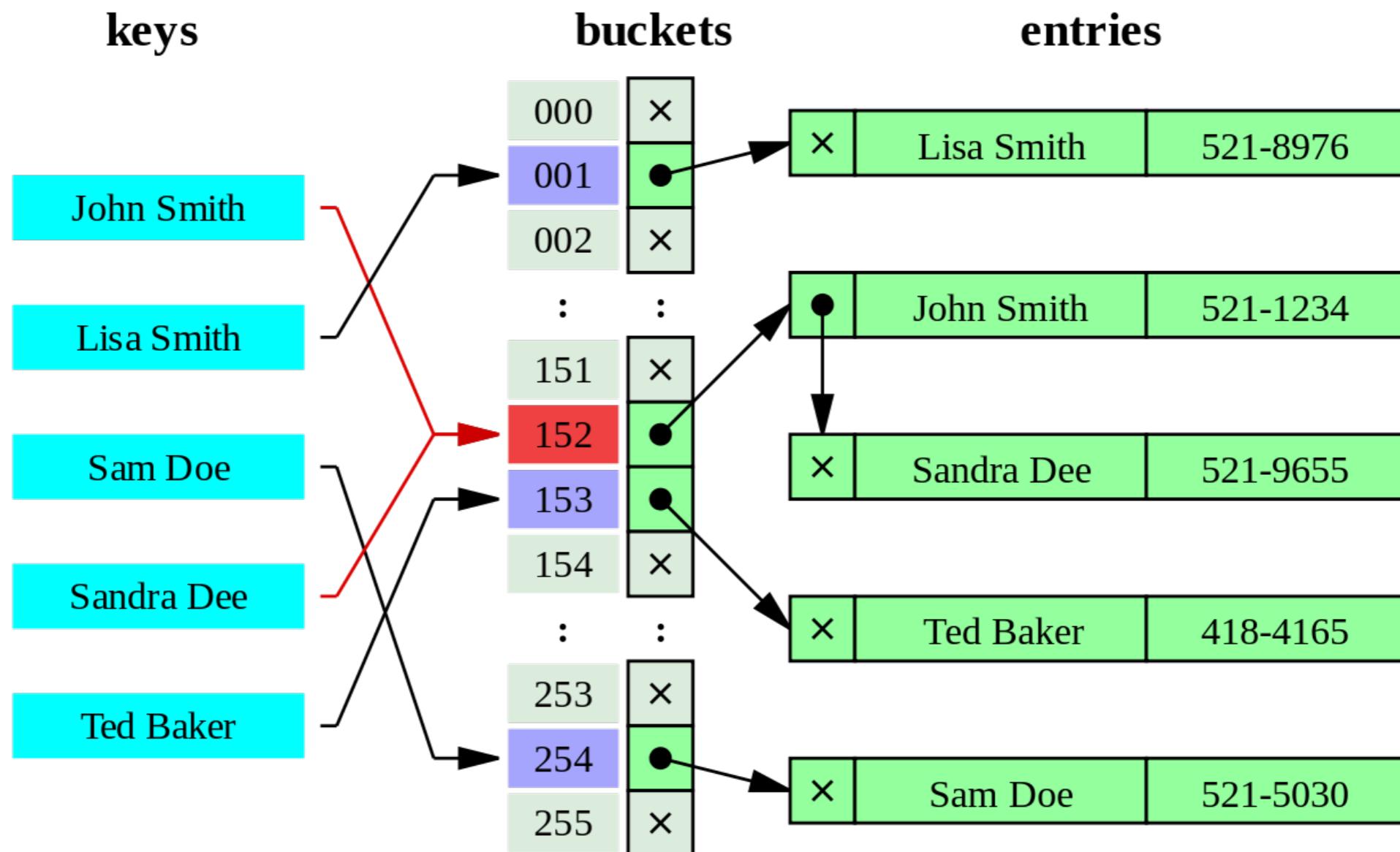
B-Tree



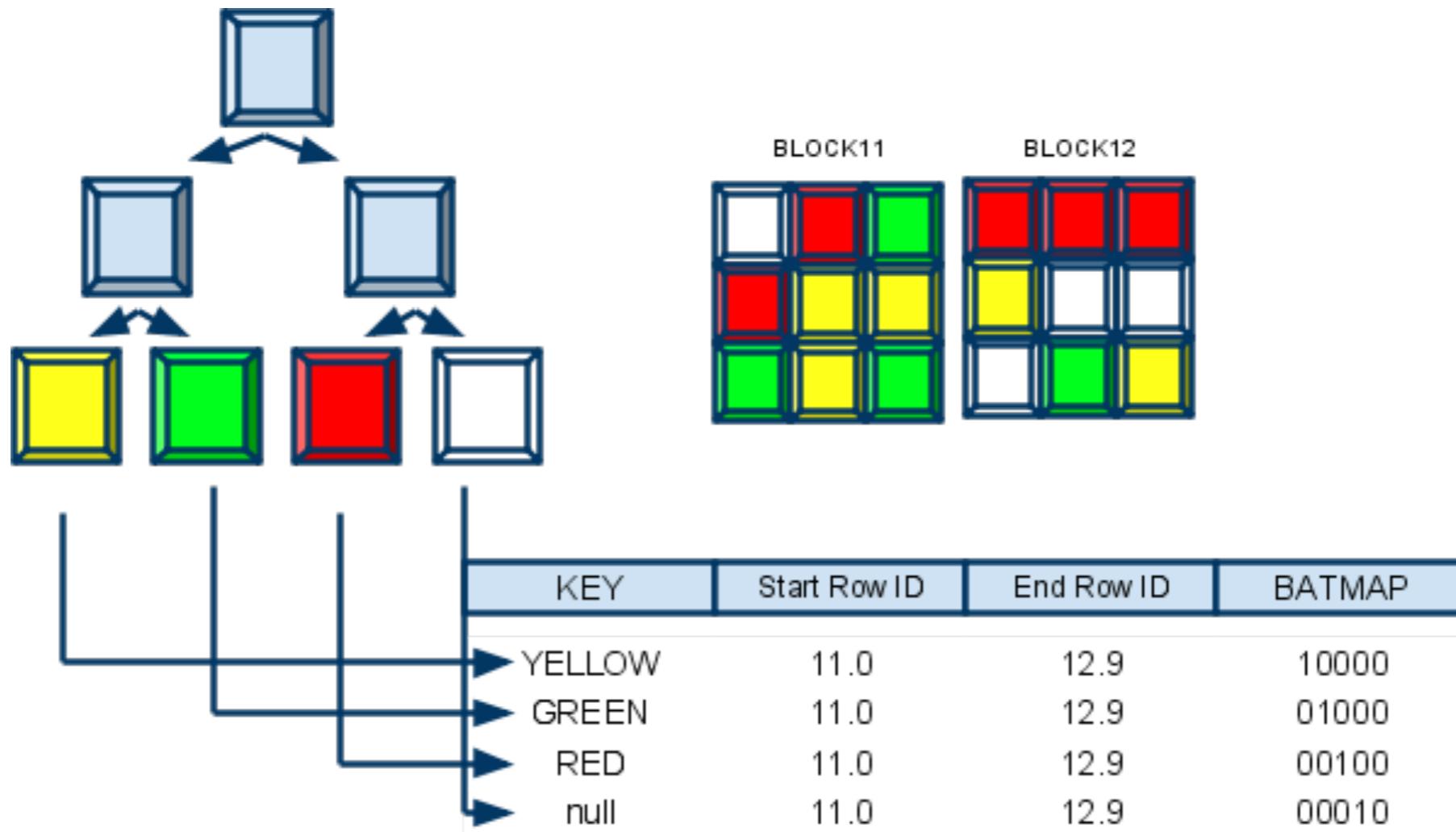
B-Tree +



Hash



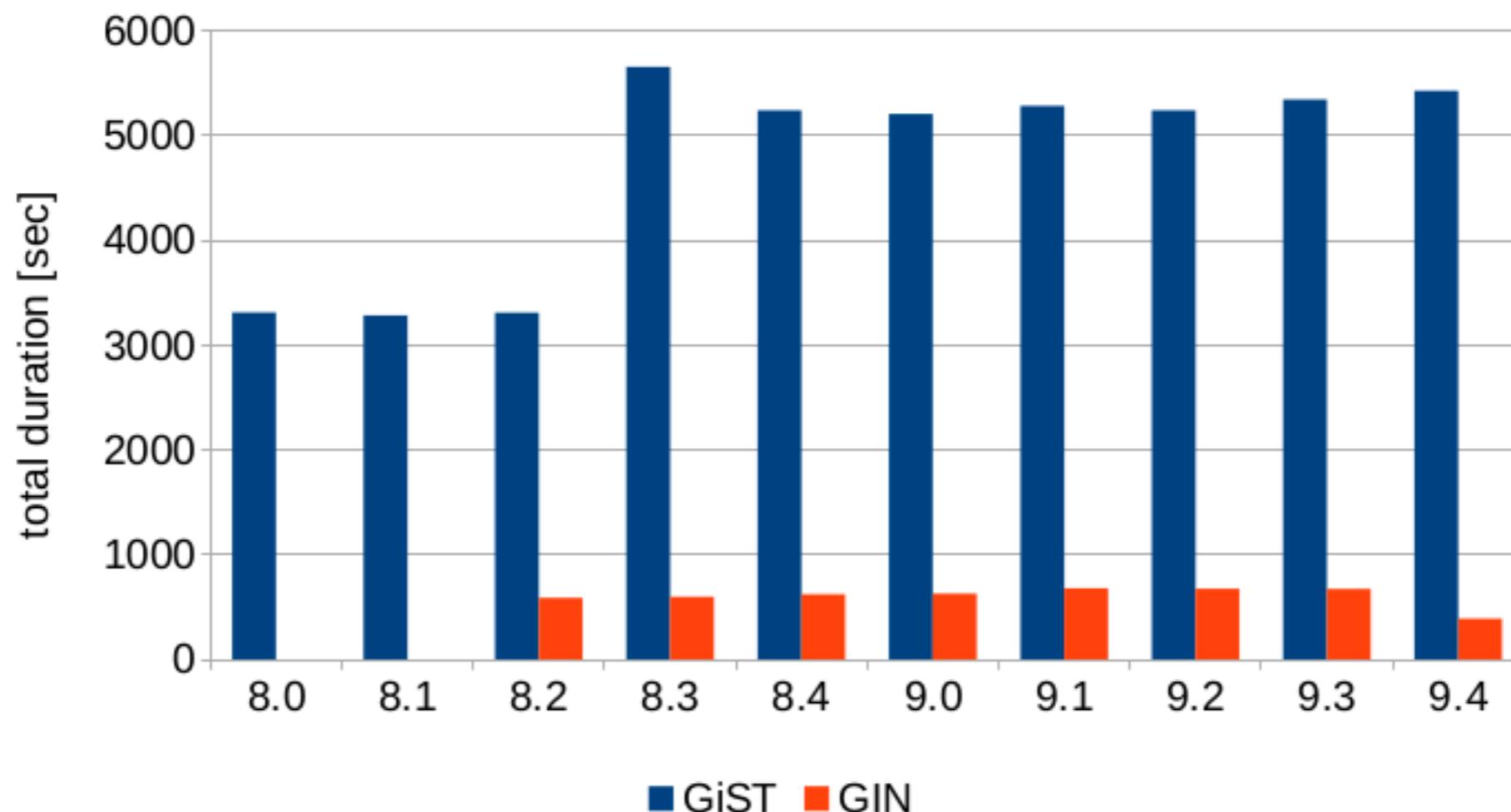
Bitmap



GIN vs GiST

Fulltext benchmark - GiST vs. GIN

33k queries from postgresql.org [TOP 100]



Don't over-index



Tuning Performance



Tuning performance

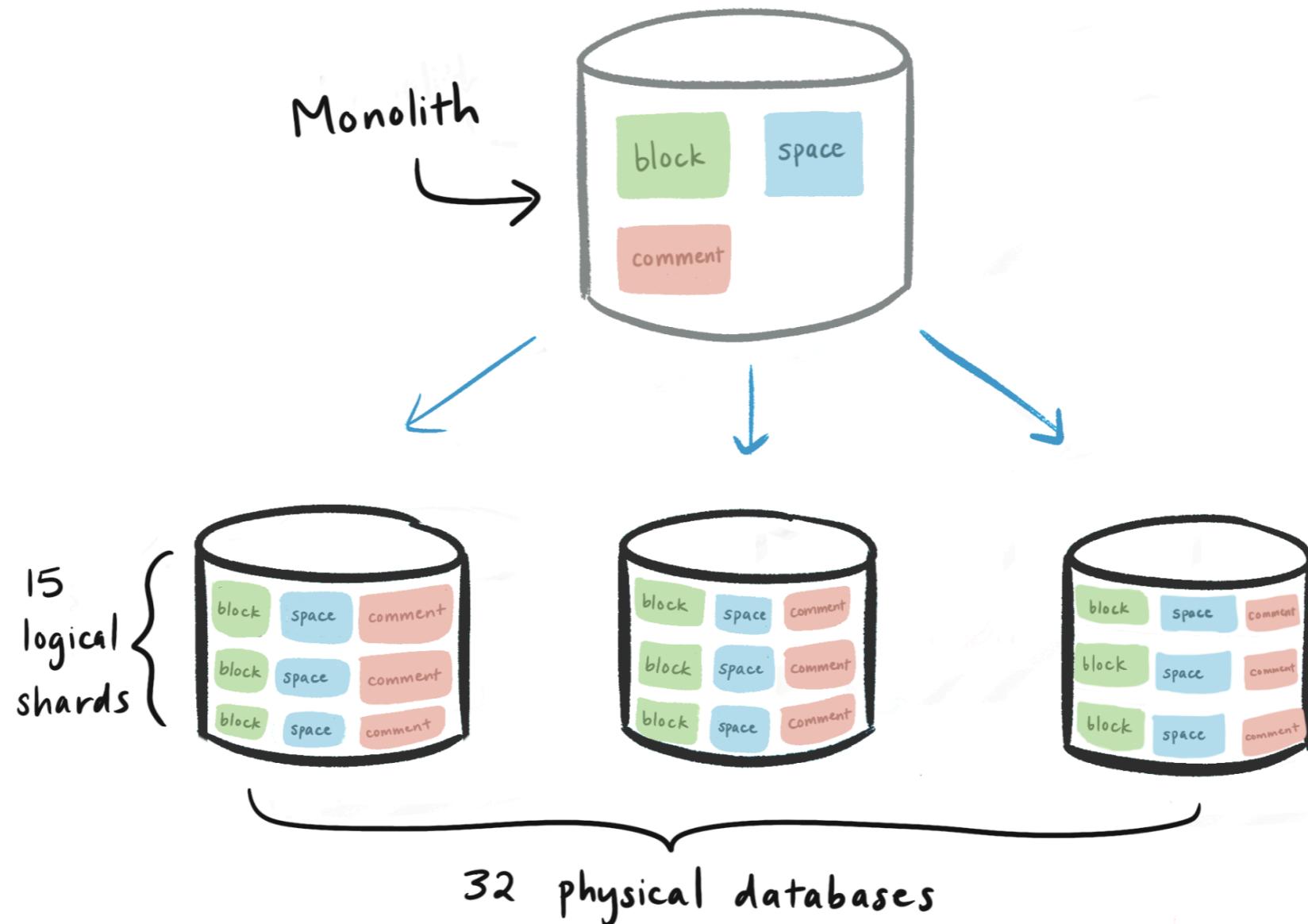
Sharding
Partition
Denormalization



Sharding



Sharding



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



Sharding techniques

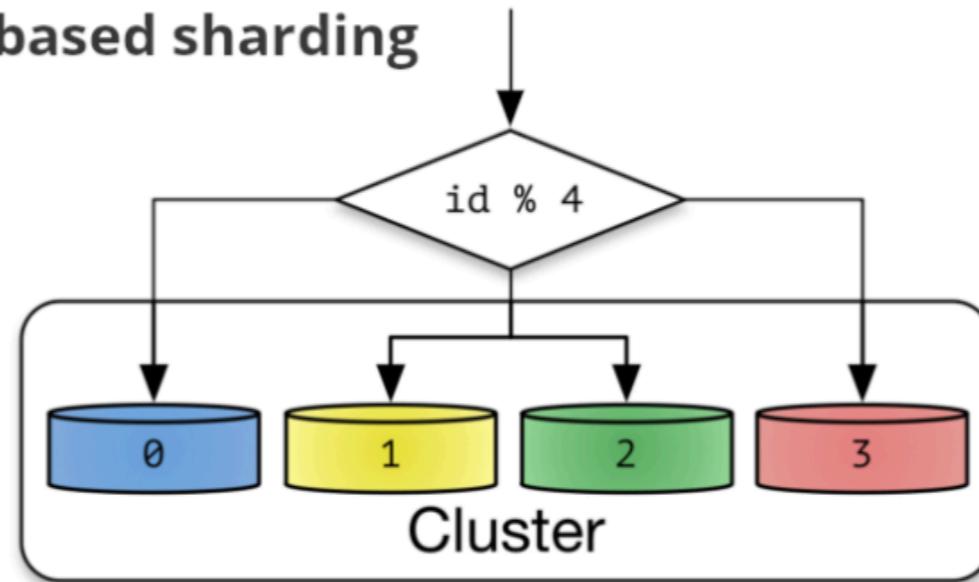
Hash
Range
Location
Key/Directory

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

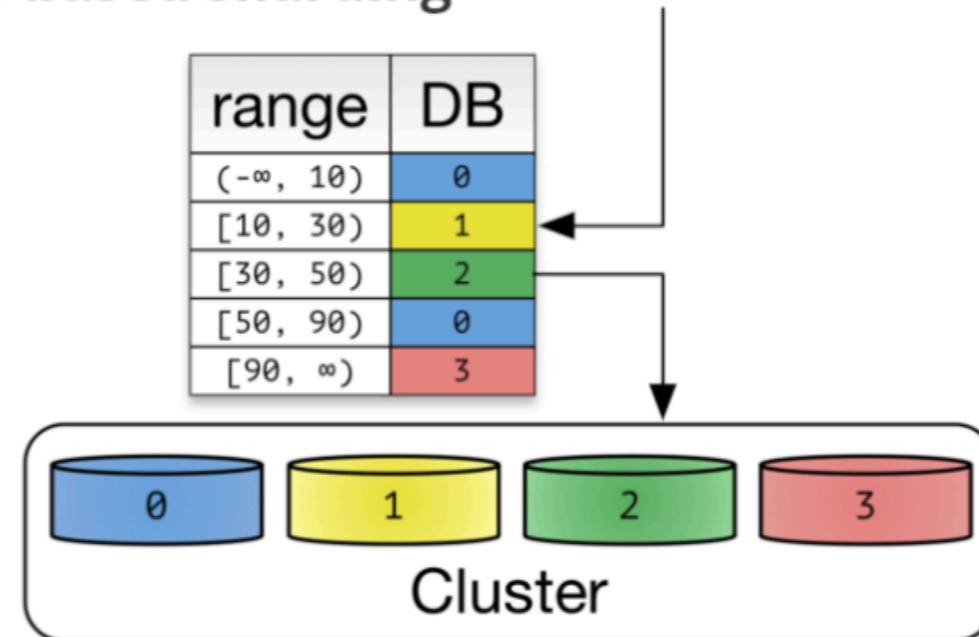


Sharding

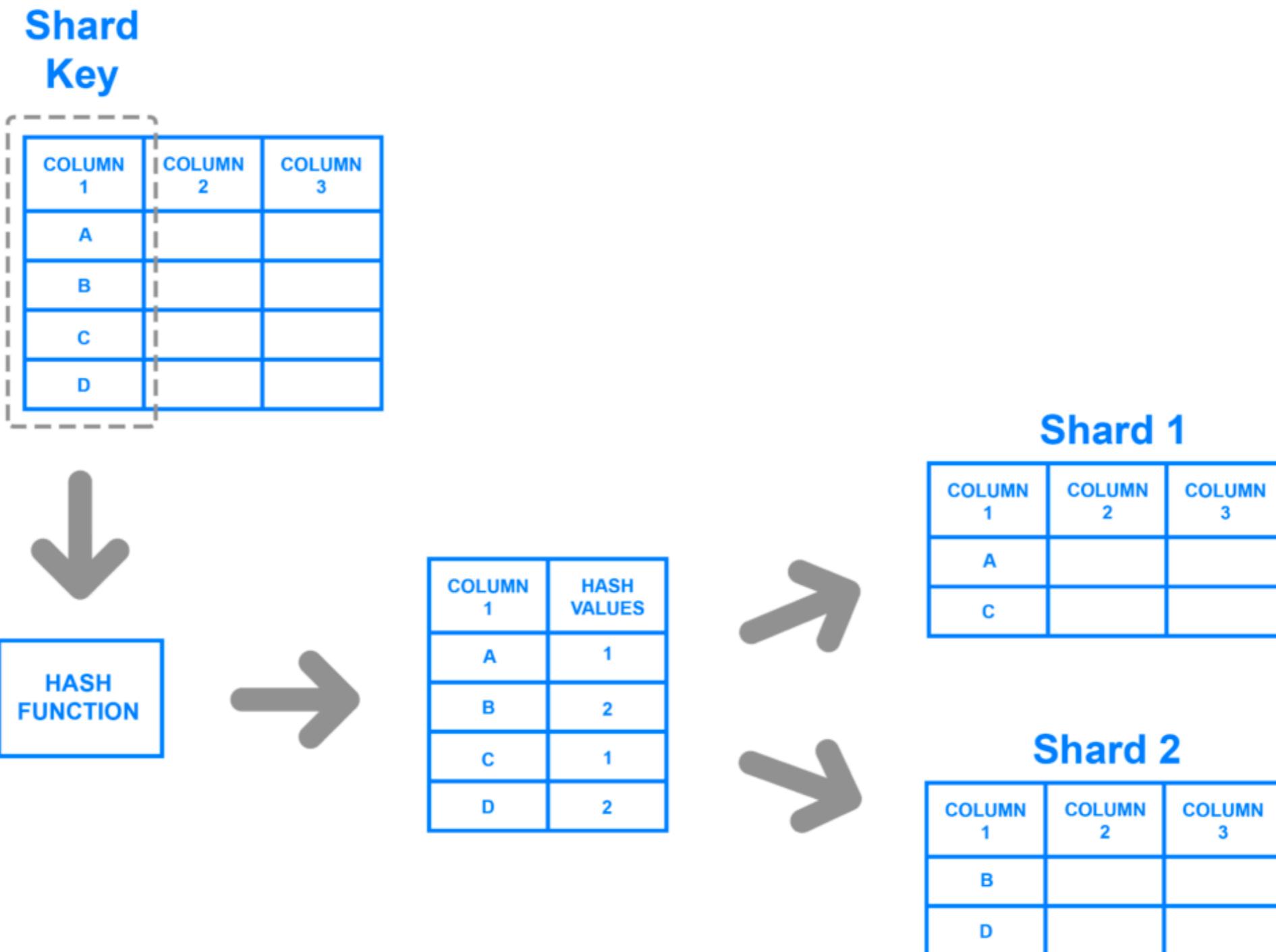
(a) Hash-based sharding



(b) Range-based 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



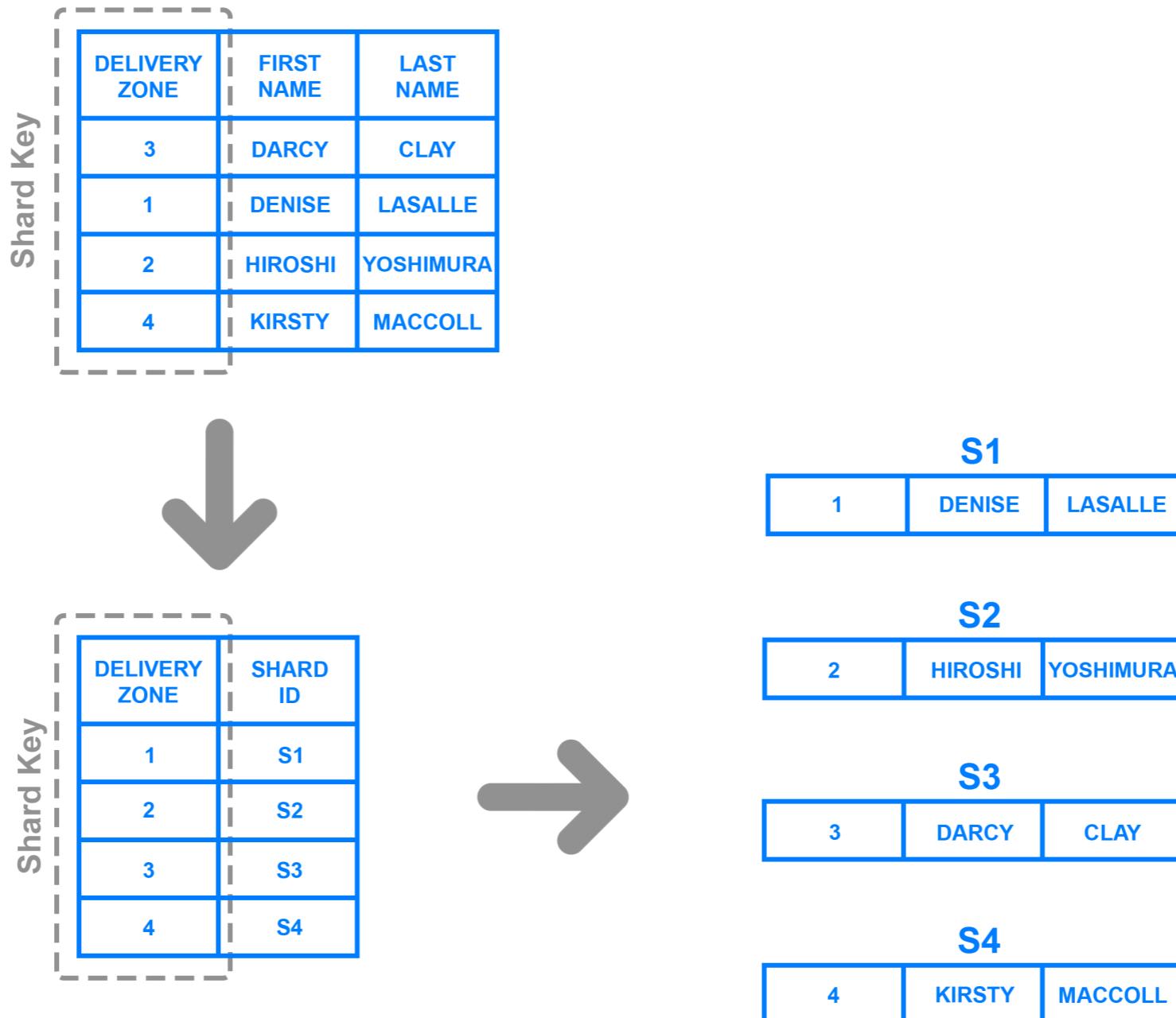
(\$100+)

PRODUCT	PRICE
WIDGET	\$118
TCHOTCHKE	\$999



Directory

Pre-Sharded Table



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



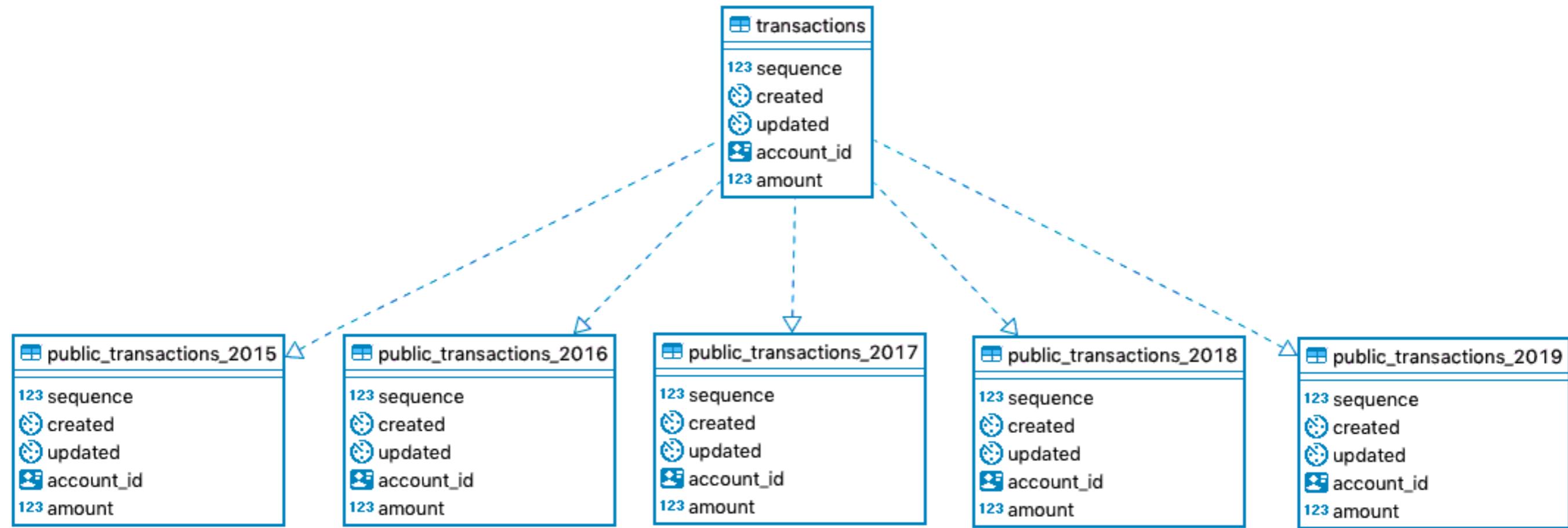
Partition

Horizontal partitioning
Vertical partitioning



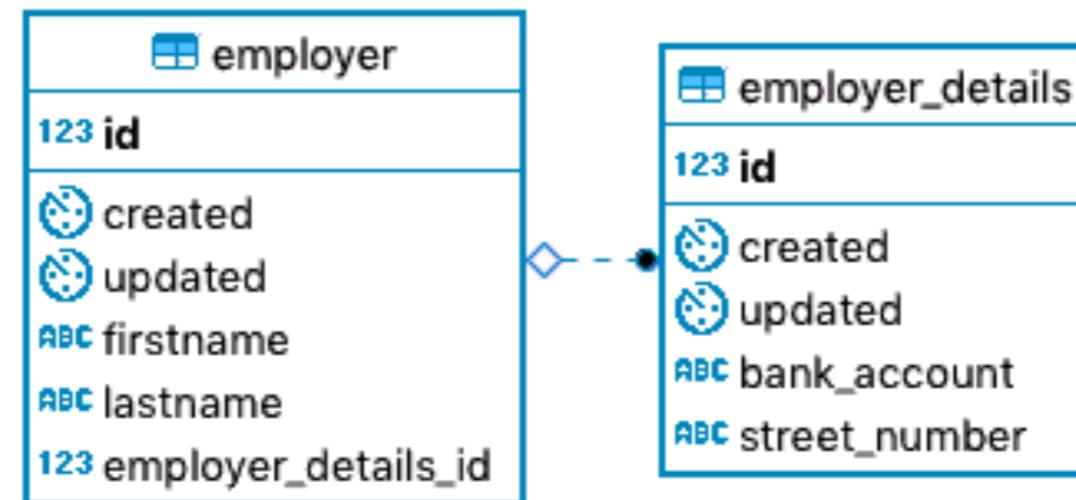
Horizontal partitioning

Putting different rows into different table



Vertical partitioning

Horizontal partitioning



Types Horizontal partitioning

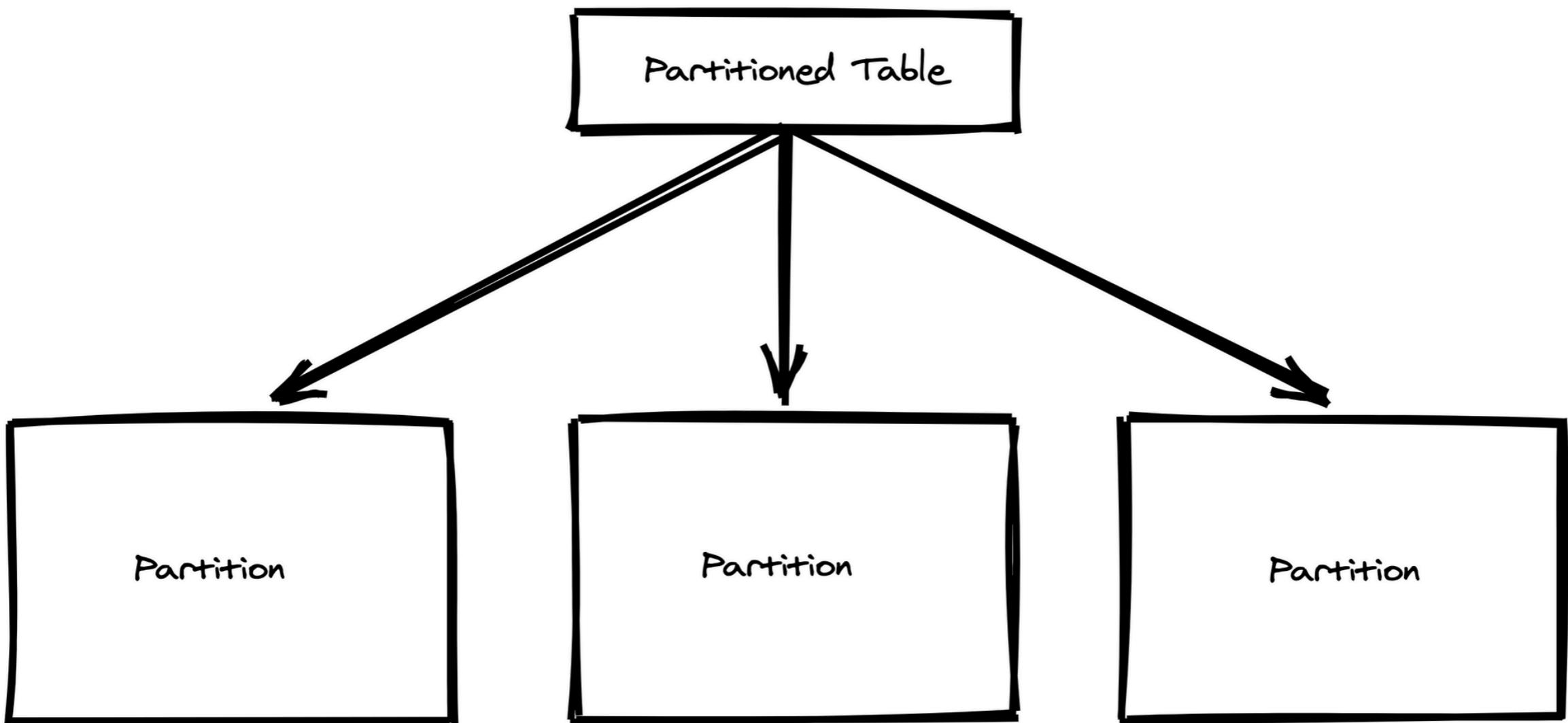
Range
List



Example



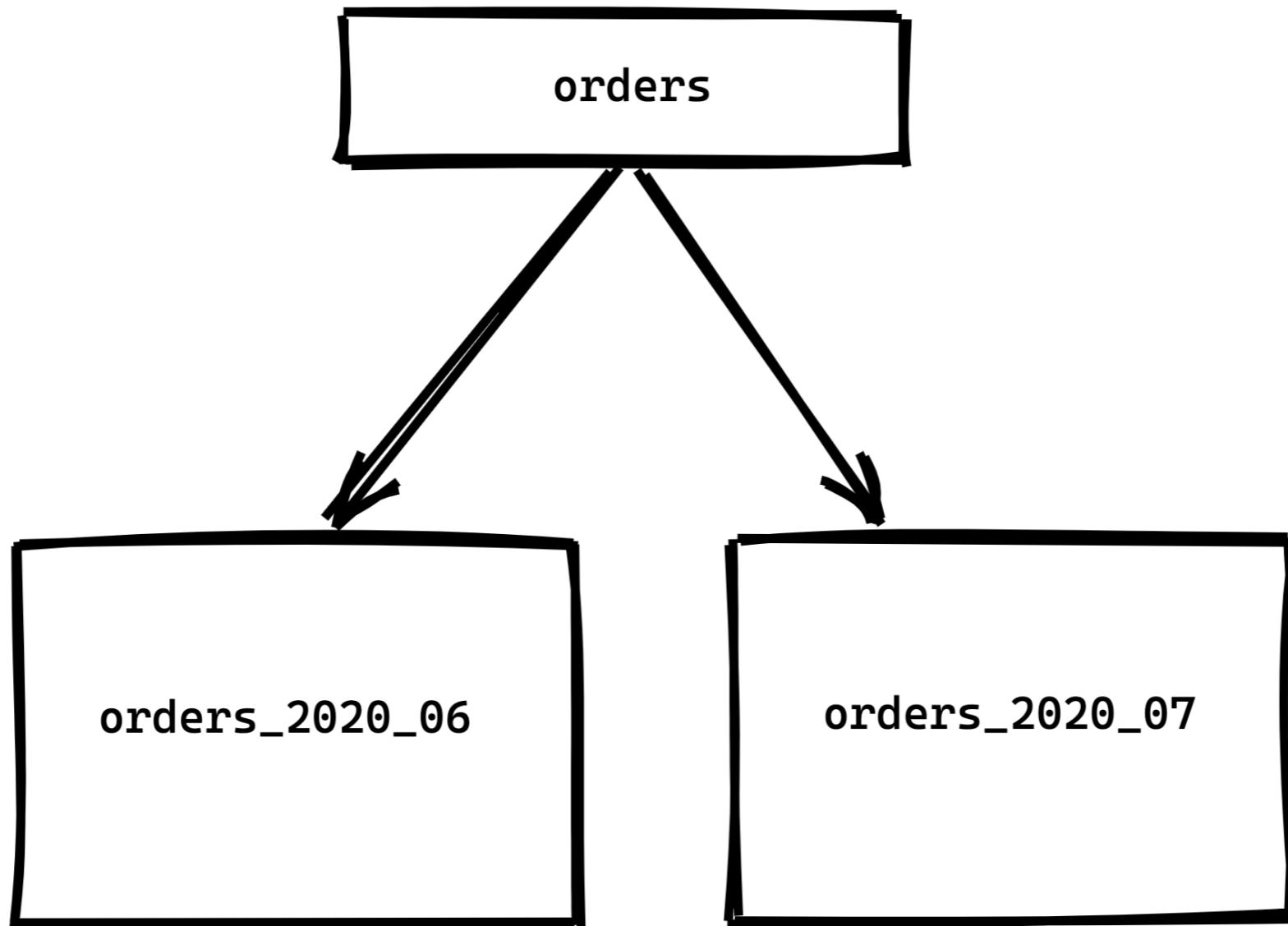
Partition tables



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



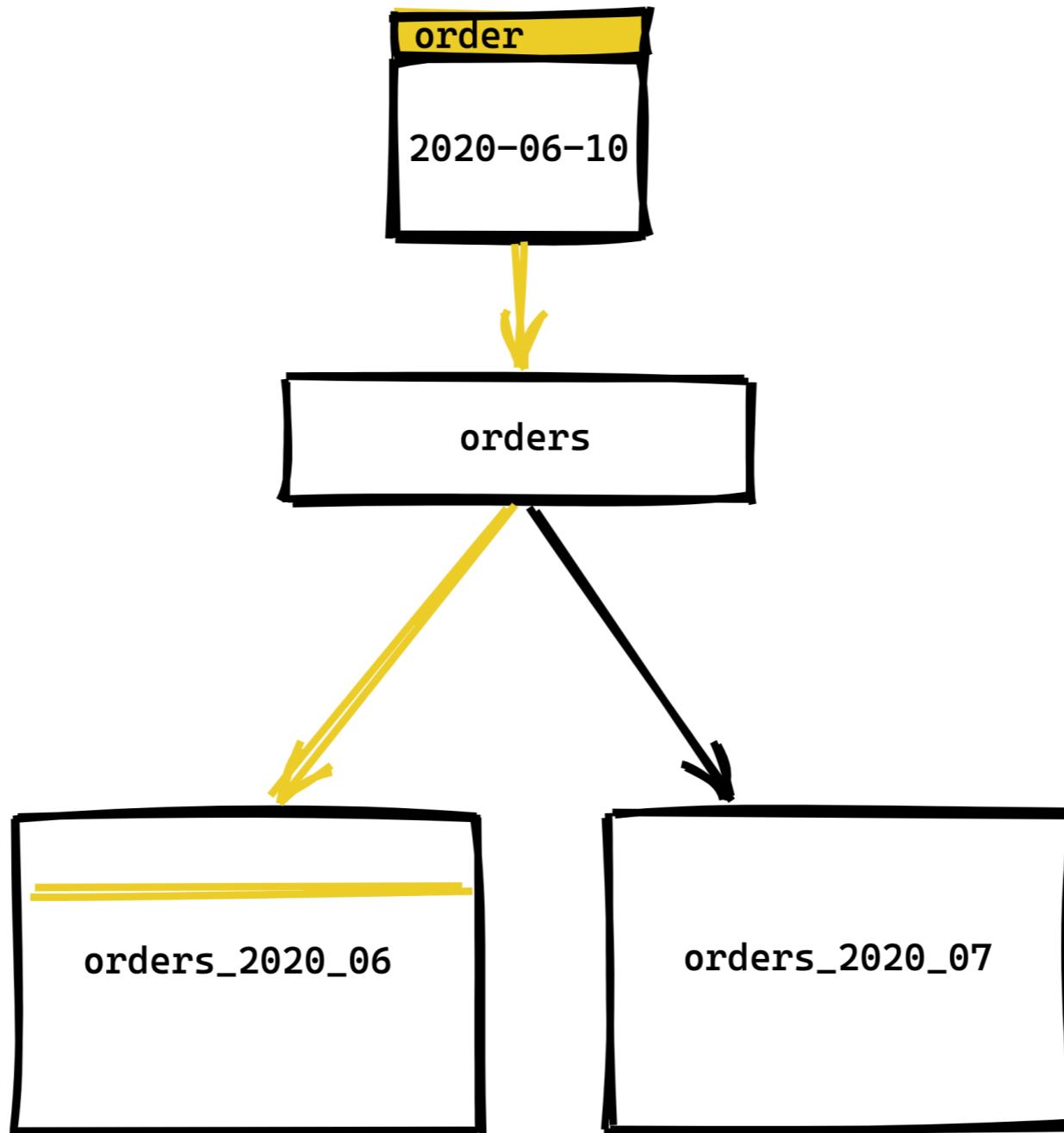
Orders table



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



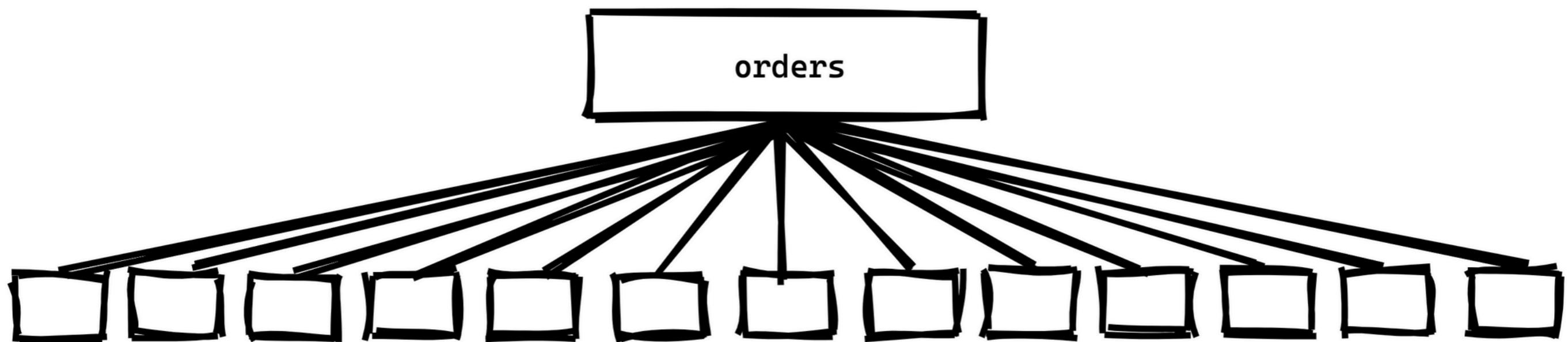
Insert data into orders



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



• • •



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



Workshop



Denormalization



Denormalization techniques

Redundant column/Pre-joining tables

Table splitting (horizontal/vertical)

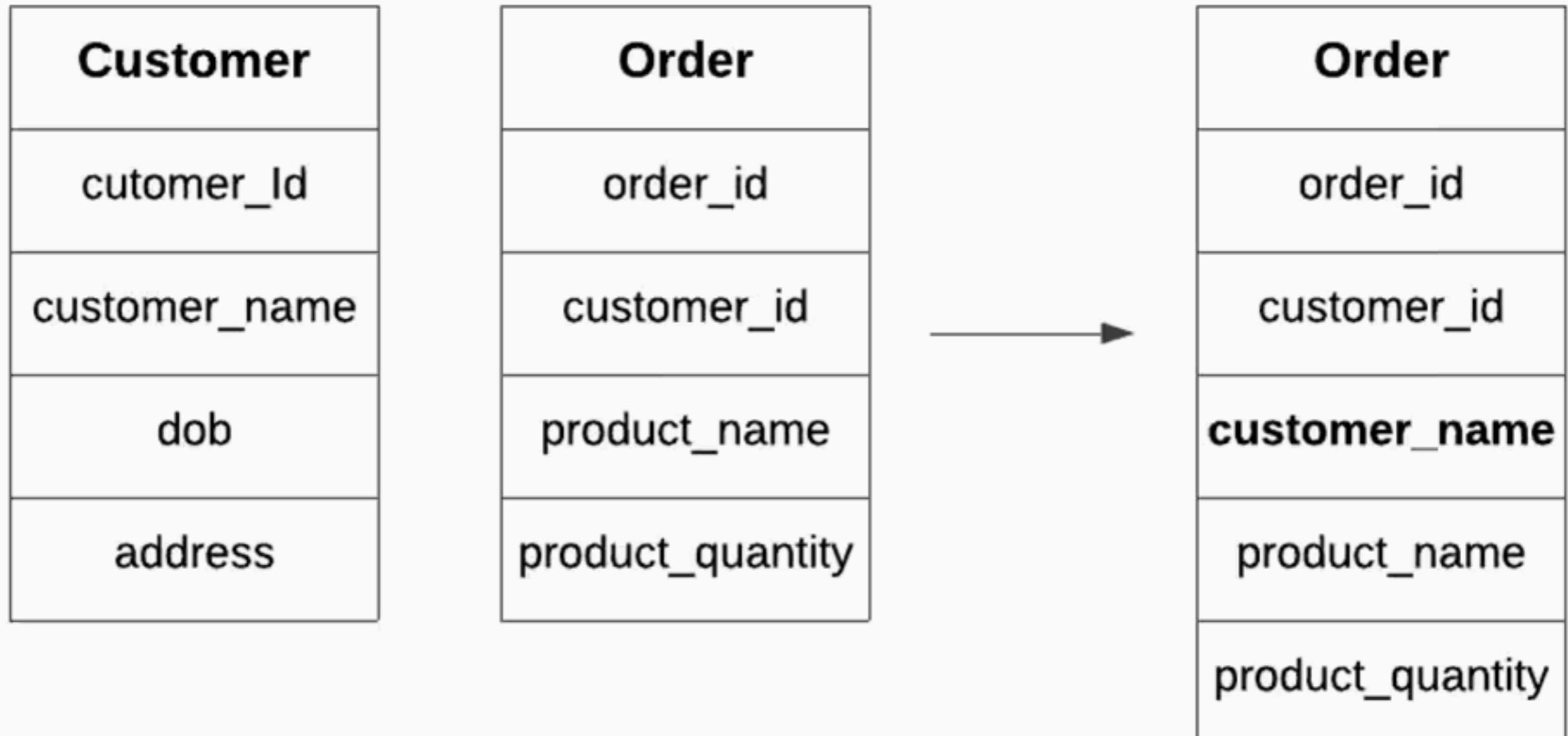
Add derived columns

Mirrored tables

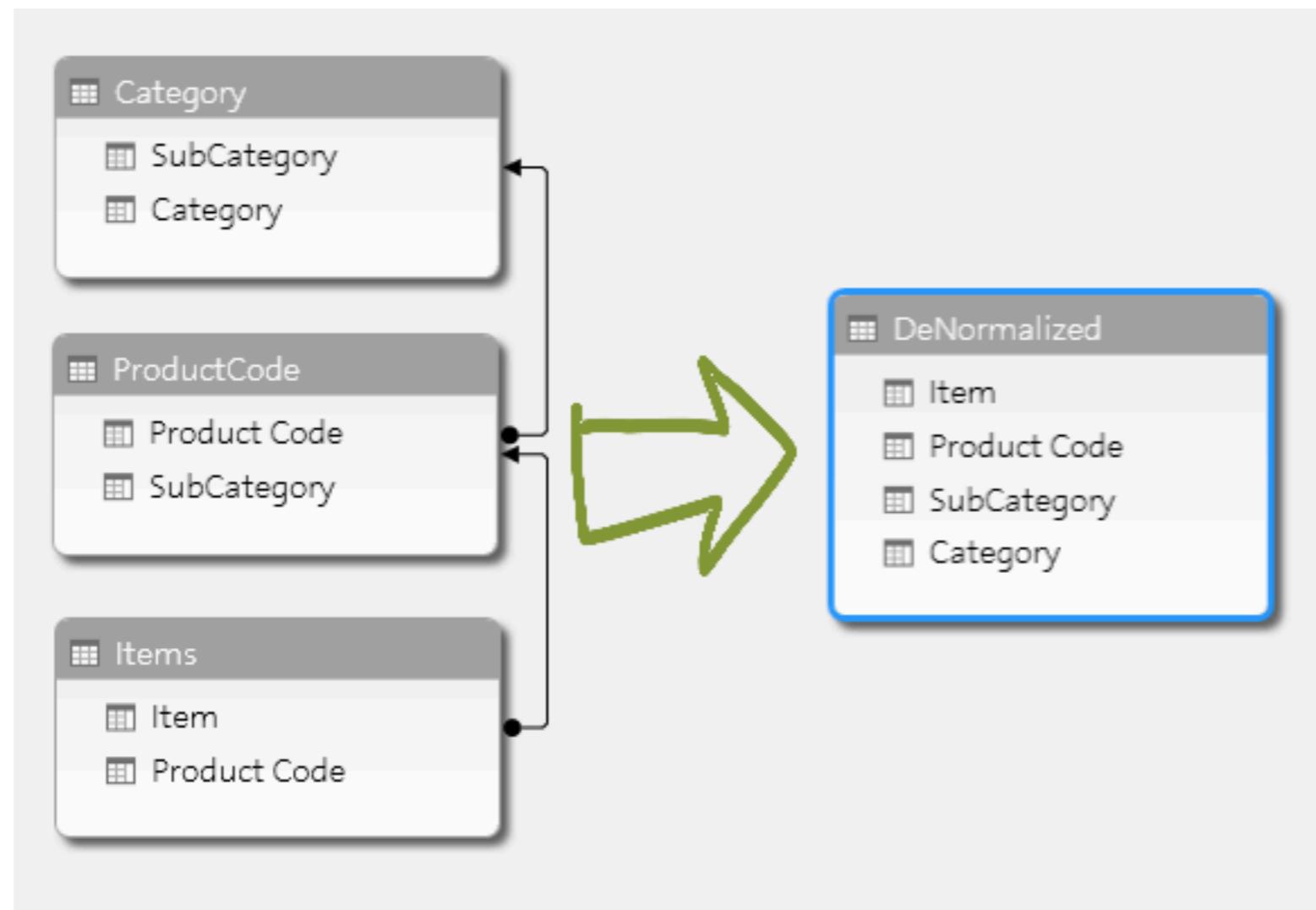
Materialized views



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

The diagram illustrates the process of horizontal splitting a single table into three separate tables based on departmental categories. Three arrows point from the original table to the resulting tables.

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

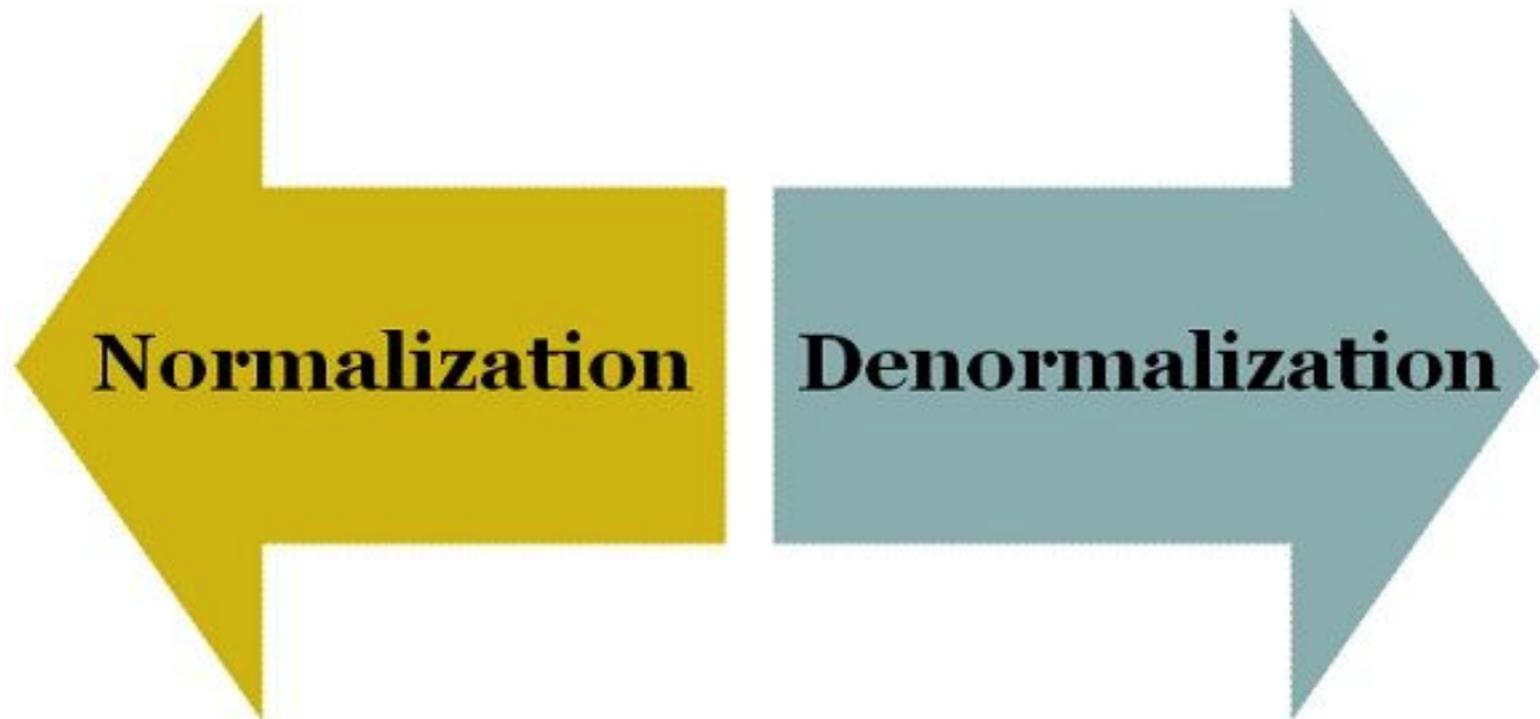
Require more disk space (split tables)

More data model (hard to maintain)

Difficult to insert and update data

Maintenance costs





Workshop

