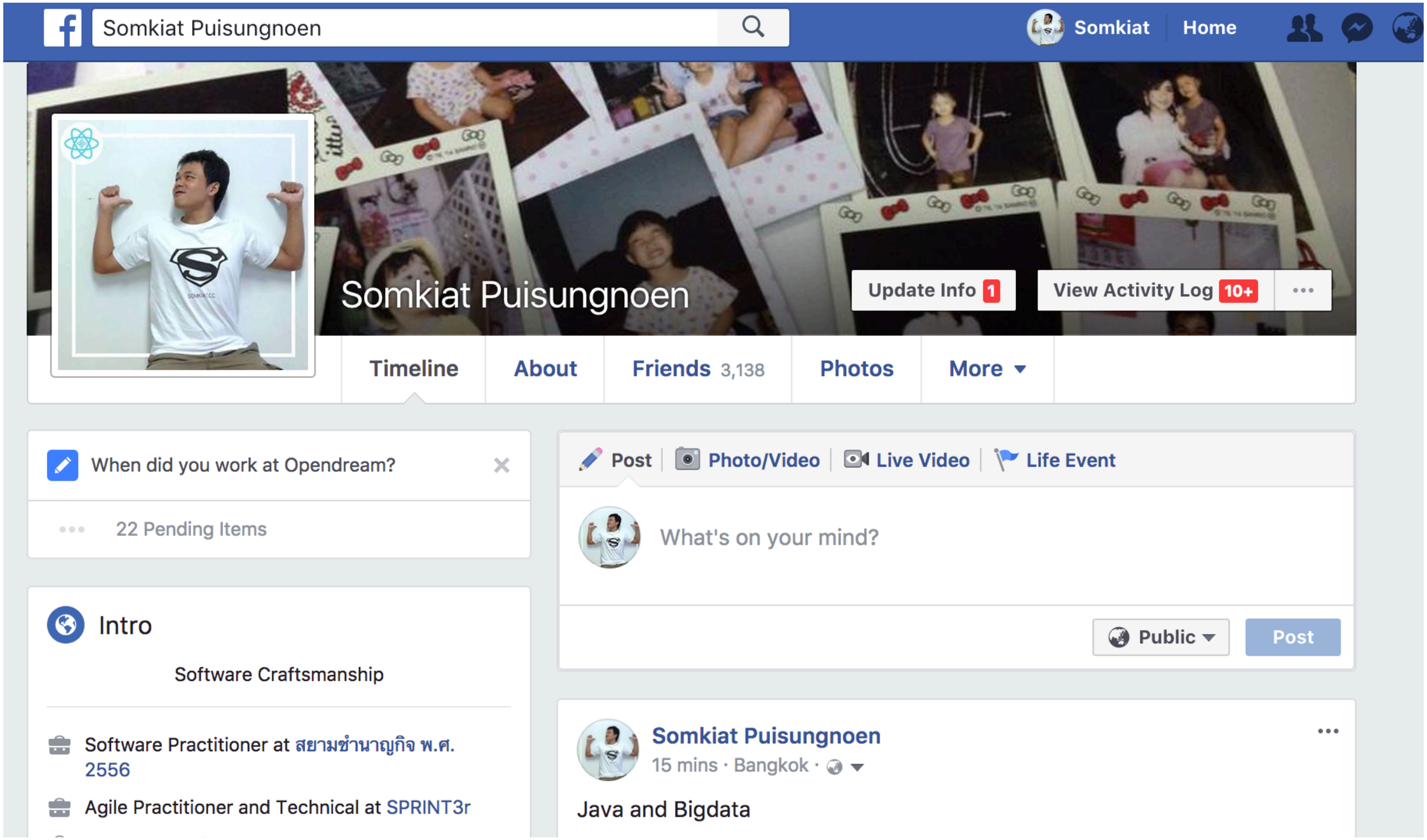
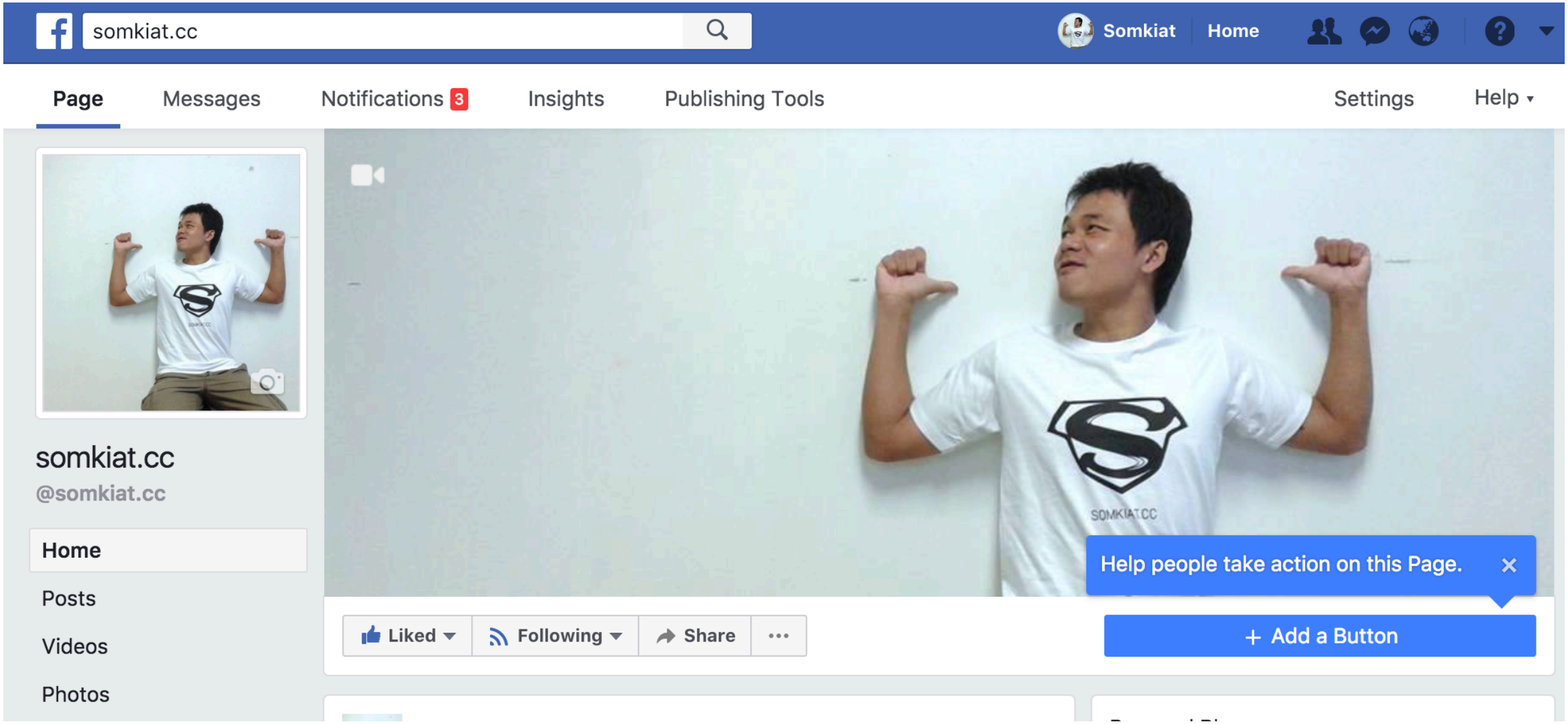
สยามชำนาญกิจ
SIAM CHAMNAN KIT

Introduction to SQL PostgreSQL





https://github.com/up1/course-sql

Topics

Overview of Database
SQL
SQL vs NoSQL
Relationship and JOINs
Database design
Normalization and De-nomalization
Workshop

Topics

Read vs Write operation with Database

Improve performance of SQL query

Benchmark and performance testing with Database (PostgreSQL)

Overview of Database

What is a Database ?

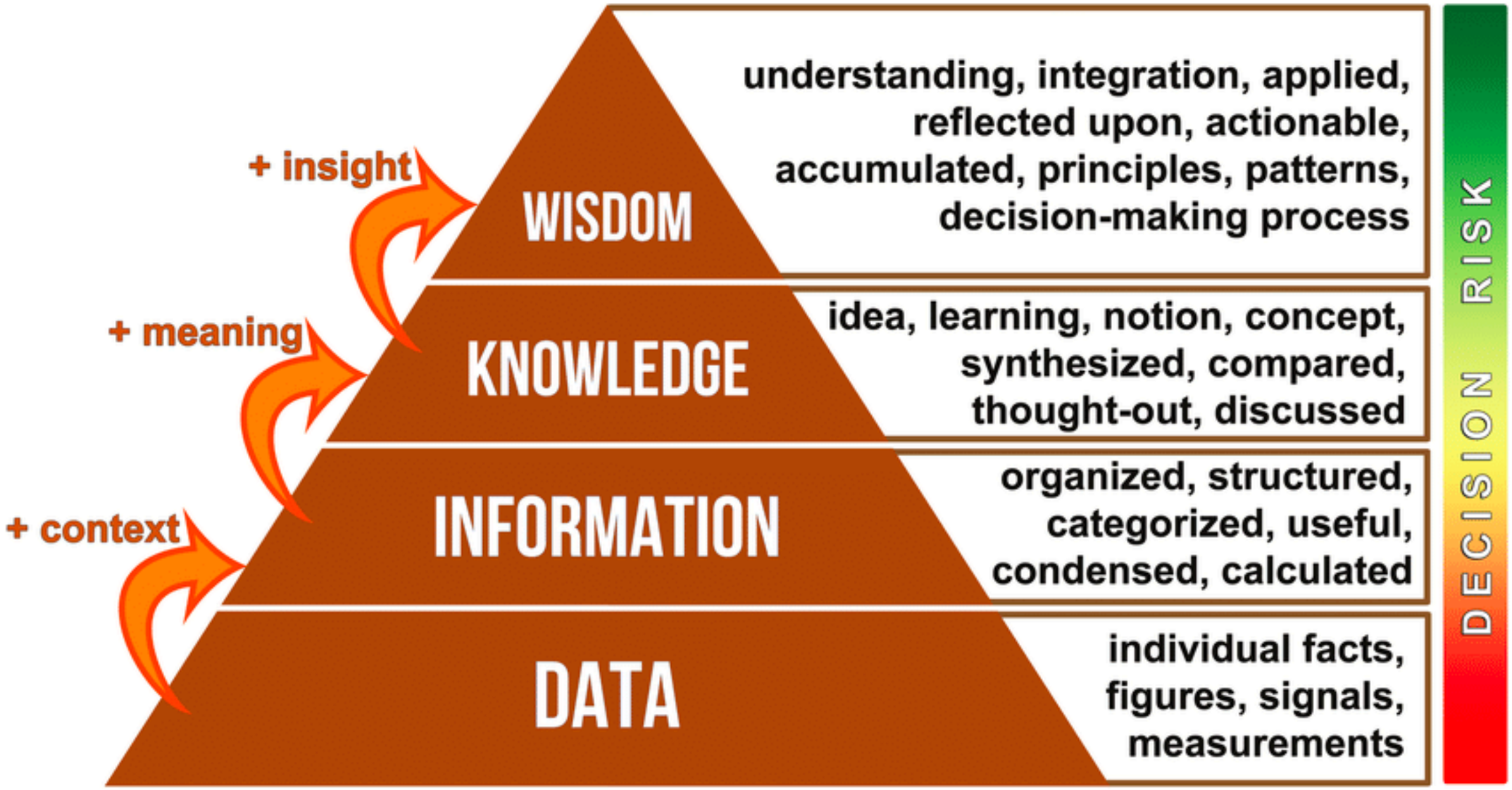
Any **collection** of related **information**

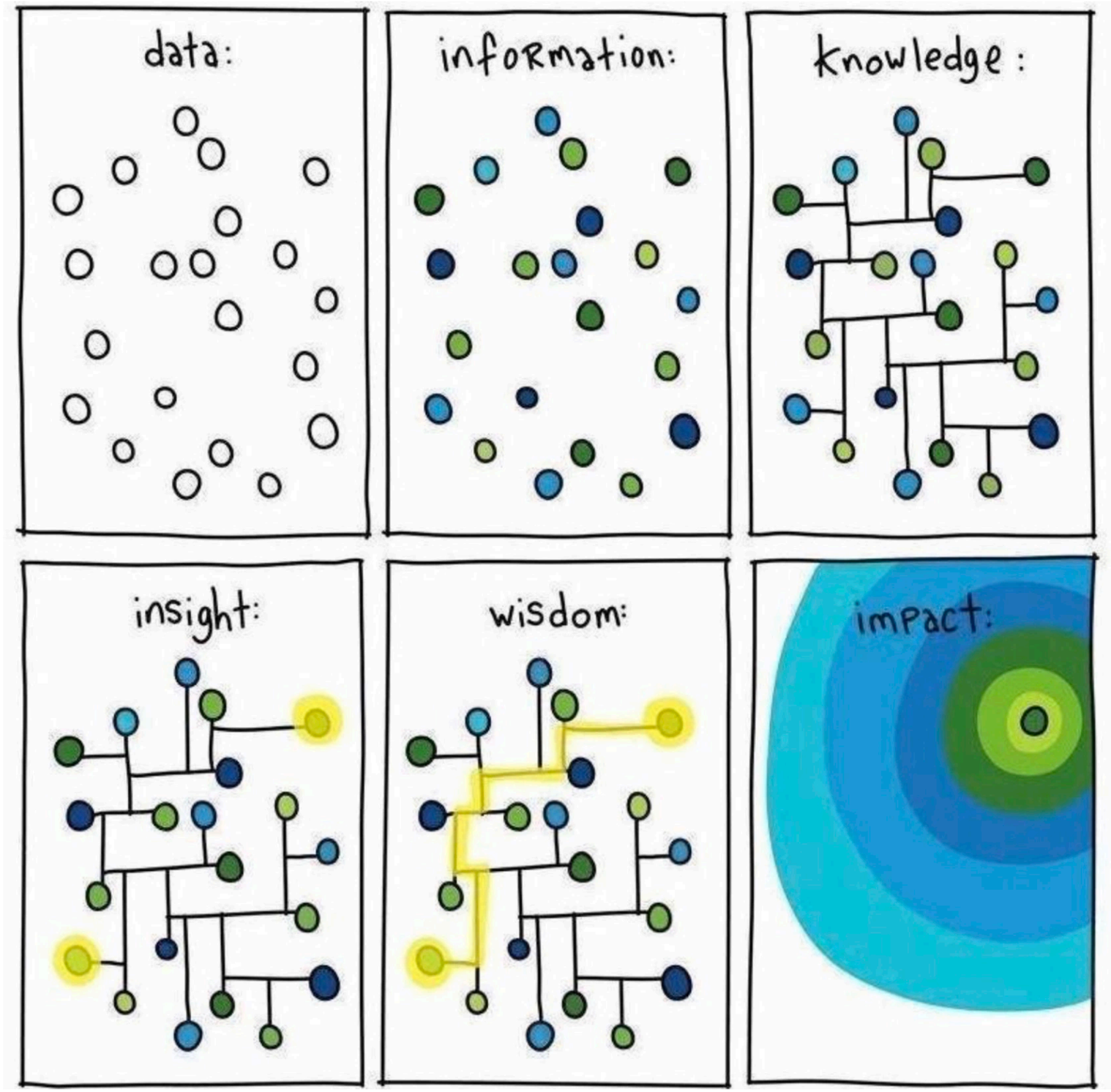
Phone numbers

Shopping list

Todo list

LINE group and friends





What is a Database ?

Database can be **stored** in different ways

On paper

In your mind

On computer

Slide

E-commerce

E-commerce

Keep track of products/reviews/orders/users
Trillions of information need to be stored
Information is extremely valuable
Security is essential
Information is stored on a computer

Computers a great at keeping track of large amounts of information

Database Management System (DBMS)

Database Management System

Special software program
Help users create and maintain a database

Database Management System

Easy ti manage large amounts of information

Handles security

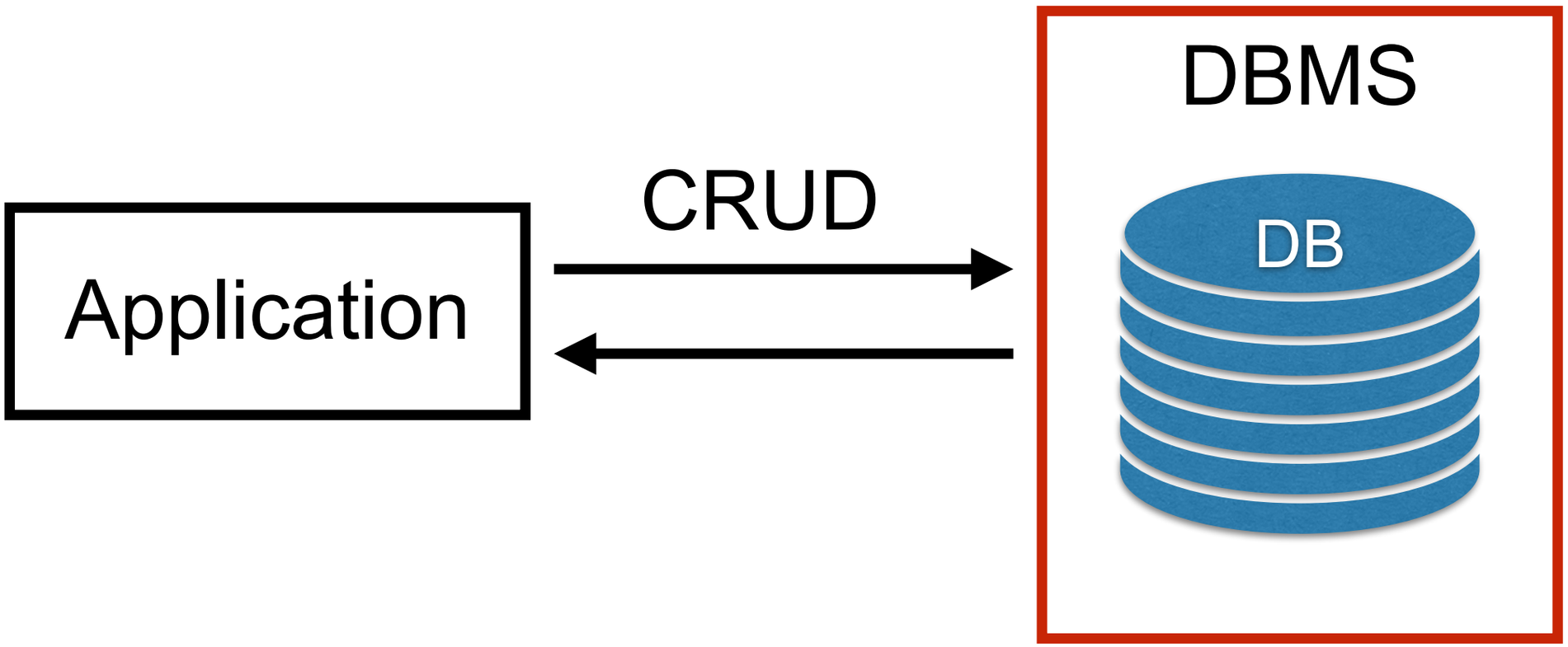
Backup

Import/export data

Concurrency

Interface with software applications

Database Management System



C.R.U.D

Create
Read
Update
Delete

Types of Databases

Types of Database

Relational Database (YesSQL)
Non-Relational Database (NoSQL)

1. Relational Database

Organize data into one or more tables
Each table has columns and rows
A unique key identified each row

Table customer

Column or attribute

Customer ID	Firstname	Lastname	Birthdate

Table customer

Row or tuple

Customer ID	Firstname	Lastname	Birthdate
1	Somkiat	Pui	April 18, 2000

Table customer

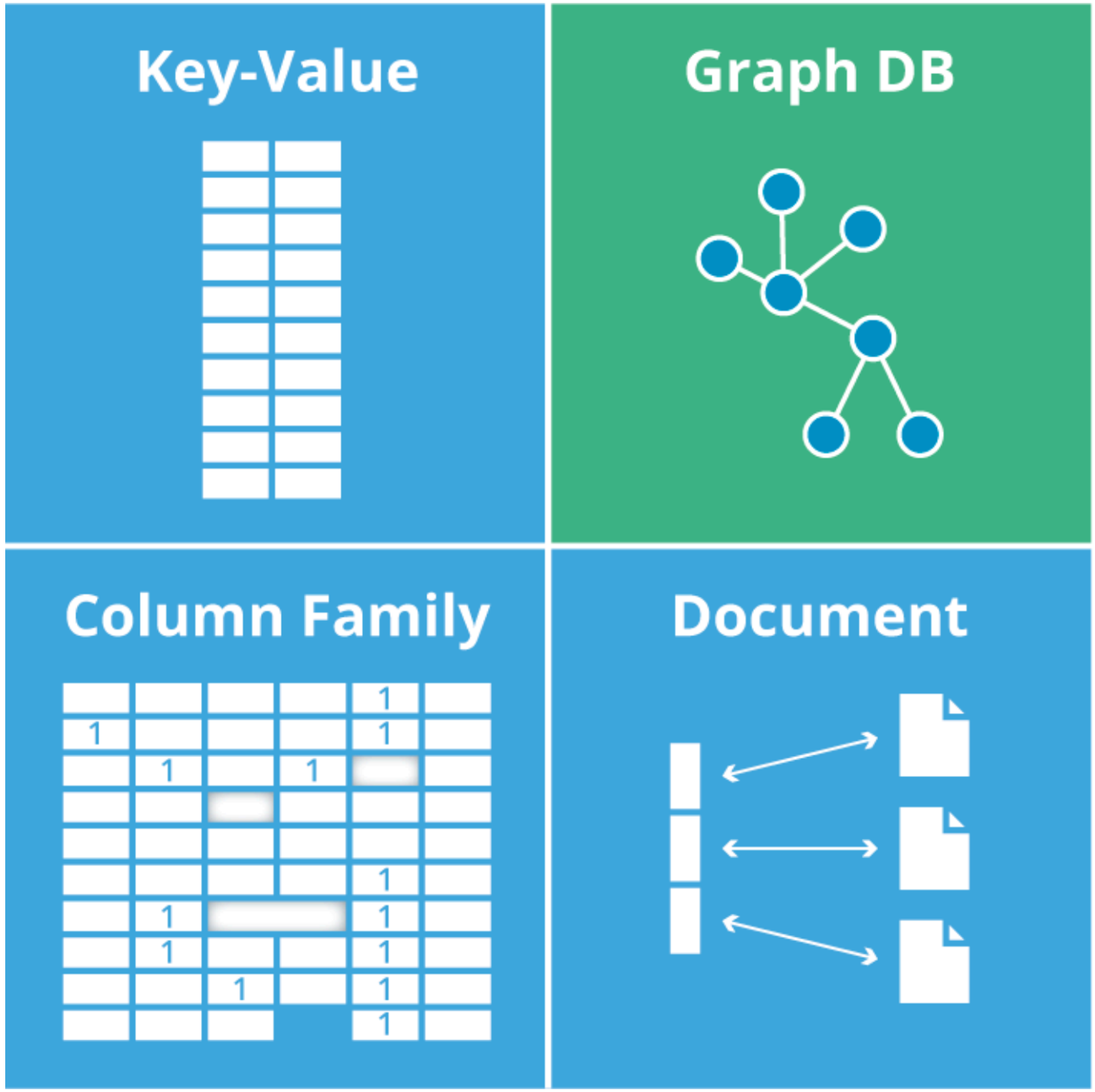
Unique key of Primary key for each row

Customer ID	Firstname	Lastname	Birthdate
1			
2			
3			
4			

2. Non-Relational Database

Organize data is anything
Key-value store
Document-based (JSON, XML)
Graph
Flexible table (Column-based)

NoSQL



Database ranking

Database ranking

358 systems in ranking, September 2020

Rank Sep 2020	Rank Aug 2020	Rank Sep 2019	DBMS	Database Model	Score Sep 2020	Score Aug 2020	Score Sep 2019
1.	1.	1.	Oracle	Relational, Multi-model	1369.36	+14.21	+22.71
2.	2.	2.	MySQL	Relational, Multi-model	1264.25	+2.67	-14.83
3.	3.	3.	Microsoft SQL Server	Relational, Multi-model	1062.76	-13.12	-22.30
4.	4.	4.	PostgreSQL	Relational, Multi-model	542.29	+5.52	+60.04
5.	5.	5.	MongoDB	Document, Multi-model	446.48	+2.92	+36.42
6.	6.	6.	IBM Db2	Relational, Multi-model	161.24	-1.21	-10.32
7.	7.	↑ 8.	Redis	Key-value, Multi-model	151.86	-1.02	+9.95
8.	8.	↓ 7.	Elasticsearch	Search engine, Multi-model	150.50	-1.82	+1.23
9.	9.	↑ 11.	SQLite	Relational	126.68	-0.14	+3.31
10.	↑ 11.	10.	Cassandra	Wide column	119.18	-0.66	-4.22

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

Relational Database

358 systems in ranking, September 2020

Rank Sep 2020	Rank Aug 2020	Rank Sep 2019	DBMS	Database Model	Score Sep 2020	Score Aug 2020	Score Sep 2019
1.	1.	1.	Oracle	Relational, Multi-model	1369.36	+14.21	+22.71
2.	2.	2.	MySQL	Relational, Multi-model	1264.25	+2.67	-14.83
3.	3.	3.	Microsoft SQL Server	Relational, Multi-model	1062.76	-13.12	-22.30
4.	4.	4.	PostgreSQL	Relational, Multi-model	542.29	+5.52	+60.04
5.	5.	5.	MongoDB	Document, Multi-model	446.48	+2.92	+36.42
6.	6.	6.	IBM Db2	Relational, Multi-model	161.24	-1.21	-10.32
7.	7.	↑ 8.	Redis	Key-value, Multi-model	151.86	-1.02	+9.95
8.	8.	↓ 7.	Elasticsearch	Search engine, Multi-model	150.50	-1.82	+1.23
9.	9.	↑ 11.	SQLite	Relational	126.68	-0.14	+3.31
10.	↑ 11.	10.	Cassandra	Wide column	119.18	-0.66	-4.22

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

Non-Relational Database

358 systems in ranking, September 2020

Rank Sep 2020	Rank Aug 2020	Rank Sep 2019	DBMS	Database Model	Score Sep 2020	Score Aug 2020	Score Sep 2019
1.	1.	1.	Oracle	Relational, Multi-model	1369.36	+14.21	+22.71
2.	2.	2.	MySQL	Relational, Multi-model	1264.25	+2.67	-14.83
3.	3.	3.	Microsoft SQL Server	Relational, Multi-model	1062.76	-13.12	-22.30
4.	4.	4.	PostgreSQL	Relational, Multi-model	542.29	+5.52	+60.04
5.	5.	5.	MongoDB	Document, Multi-model	446.48	+2.92	+36.42
6.	6.	6.	IBM Db2	Relational, Multi-model	161.24	-1.21	-10.32
7.	7.	↑ 8.	Redis	Key-value, Multi-model	151.86	-1.02	+9.95
8.	8.	↓ 7.	Elasticsearch	Search engine, Multi-model	150.50	-1.82	+1.23
9.	9.	↑ 11.	SQLite	Relational	126.68	-0.14	+3.31
10.	↑ 11.	10.	Cassandra	Wide column	119.18	-0.66	-4.22

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

Structured Query Language (SQL)

Structured Query Language

Standard language for interacting with RDBMS

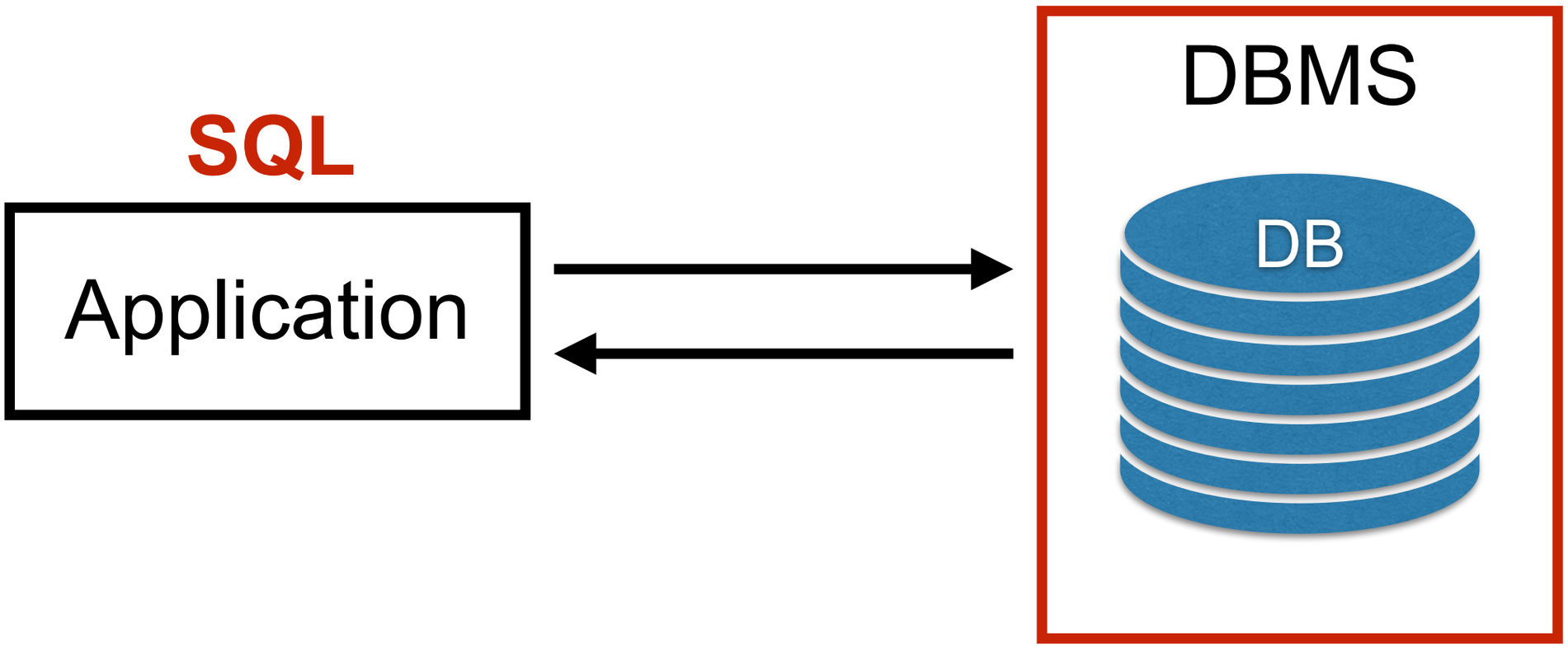
Use to perform **C.R.U.D** operations

Use to perform admin tasks (user, role, backup)

Structured Query Language

Data Definition Language (DDL)
Data Manipulation Language (DML)
Data Query Language (DQL)

SQL



Data Definition Language (DDL)

Define the structure of database/table/constraint

Create, Alter, Drop, Truncate

Data Manipulation Language (DML)

Used to manipulate data

Insert, Update, Delete

Data Query Language (DQL)

Used to performing queries on the data

Select …
From <table>
Where <condition>
Group by …
Having …
Order by …

Summary of Database

Database is any collection of information
Computer are great for storing database
DBMS make it easy to manage database

Summary of Database

2 types of database (relation and non-relation)
Relational database use SQL
Relational database store data in tables with rows and columns

Workshop

Create/Alter/Drop table
C.R.U.D operations

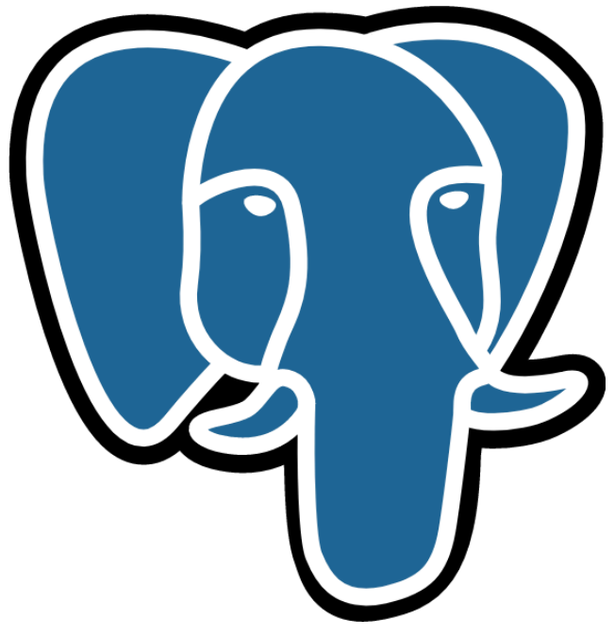
Table customer

Unique key of Primary key for each row

Customer ID	Firstname	Lastname	Birthdate
1			
2			
3			
4			

Create table customer

Define **data type** of each column
Working with **PostgreSQL**



Types of column in table

Consistency
Validation
Compactness
Performance

Data types

Numerics
Monetary
Character
Binary data
Data/Time

https://www.postgresql.org/docs/current/datatype.html

Data types

Enumerated
Geometric
Network address
XML
JSON
Text search
Array
Custom/composite
Range

https://www.postgresql.org/docs/current/datatype.html

Define data type of columns

Table customer

Customer ID	Firstname	Lastname	Birthdate
1			
2			
3			
4			

Define data type of columns

Table customer

Column name	Type	Unique	Nullable
Customer_id	Int	Yes	No
Firstname	varchar(100)	No	No
Lastname	varchar(100)	No	No
Birthdate	Date	No	Yes

1. Create table

```sql
CREATE TABLE customer_1 (
  customer_id INT PRIMARY KEY,
  firstname VARCHAR(100) Not Null,
  lastname VARCHAR(100) Not Null,
  birthdate DATE
);
```

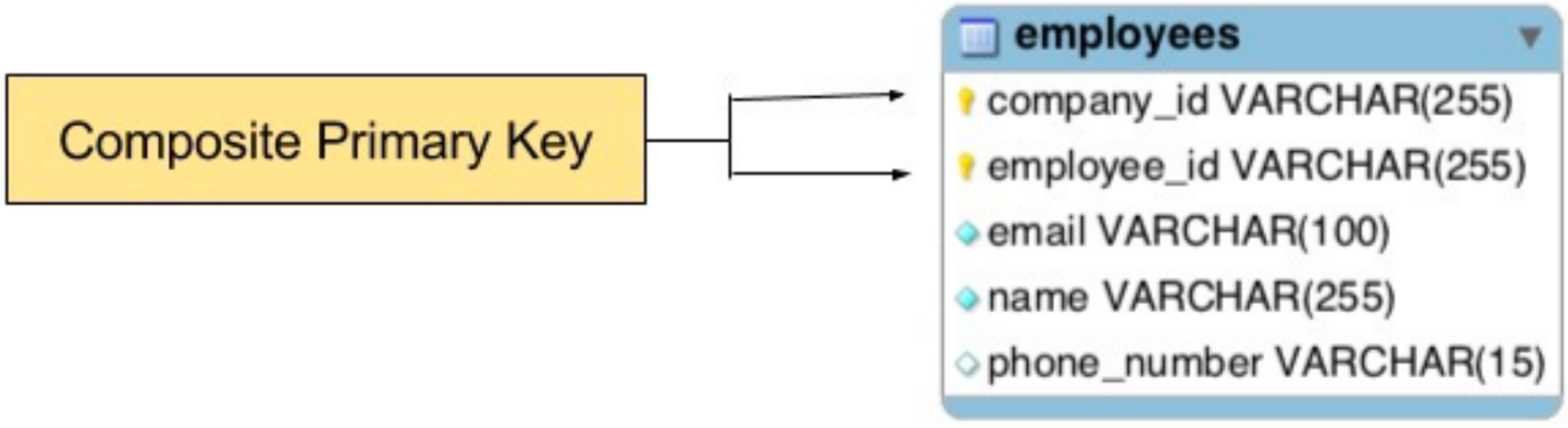
1. Create table

```sql
CREATE TABLE customer_1 (
  customer_id INT PRIMARY KEY,
  firstname VARCHAR(100) Not Null,
  lastname VARCHAR(100) Not Null,
  birthdate DATE
);
```

```sql
CREATE TABLE customer_2 (
  customer_id INT Not Null,
  firstname VARCHAR(100) Not Null,
  lastname VARCHAR(100) Not Null,
  birthdate DATE,
  PRIMARY KEY(customer_id)
);
```

Composite Primary Key

Composite Primary Key



Constraints in SQL

Constraints in SQL

Key and reference integrity
Restrictions on column and NULL
Constraints on each row within a relation

Key and Referential Integrity

PRIMARY KEY (PK)
UNIQUE
FOREIGN KEY (FK)

Foreign Key (FK)

Default operation: reject update in violation
Attach referential trigger action

On Delete
On Update
Cascade option for relationship

Constraints in SQL

NULL / NOT NULL

DEFAULT <value>

CHECK <condition>

https://docs.microsoft.com/en-us/sql/relational-databases/tables/create-check-constraints

CHECK Constraints

```
CREATE TABLE employees(
   employee_id INT NOT NULL,
   last_name VARCHAR(50) NOT NULL,
   first_name VARCHAR(50),
   salary MONEY,
   CONSTRAINT check_employee_id
     CHECK (employee_id BETWEEN 1 and 10000)
);
```

CHECK Constraints

```
CREATE TABLE employees(
  employee_id INT NOT NULL,
  last_name VARCHAR(50) NOT NULL,
  first_name VARCHAR(50),
  salary MONEY,
  CONSTRAINT check_salary
    CHECK (salary > 0)
);
```

Default value of column

```
CREATE TABLE Orders (
    ID int NOT NULL,
    OrderNumber int NOT NULL,
    OrderDate date DEFAULT GETDATE()
);
```

2. Alter table

Add new column

ALTER TABLE customer_1 **ADD score DECIMAL**;

Delete existing column

ALTER TABLE customer_2 **DROP COLUMN birthdate**;

2. Alter table

Add new column with default value

```
ALTER TABLE Persons
ADD CONSTRAINT city
DEFAULT 'Bangkok' FOR City;
```

3. Drop table

Delete data and structure of table from database

```
Drop table customer_1;
```

4. Truncate table

Delete all data from table

```
Truncate table customer_2;
```

C.R.U.D operations

C.R.U.D operations

Operation	Command
Create	**INSERT**
Read	**SELECT**
Update	**UPDATE**
Delete	**DELETE**

INSERT

INSERT INTO <table> VALUES (values);

INSERT INTO <table>(columns) VALUES (values);

INSERT INTO <table> SELECT <condition>;

SELECT

SELECT * FROM <table>

SELECT count(1) FROM <table>;

SELECT * FROM <table> WHERE <condition>;

UPDATE

UPDATE <table> SET <column>=<value>

UPDATE <table> SET <column>=<value>
WHERE <condition>

DELETE

DELETE FROM <table>;

DELETE FROM <table> WHERE <condition>;

Delete vs Truncate ?

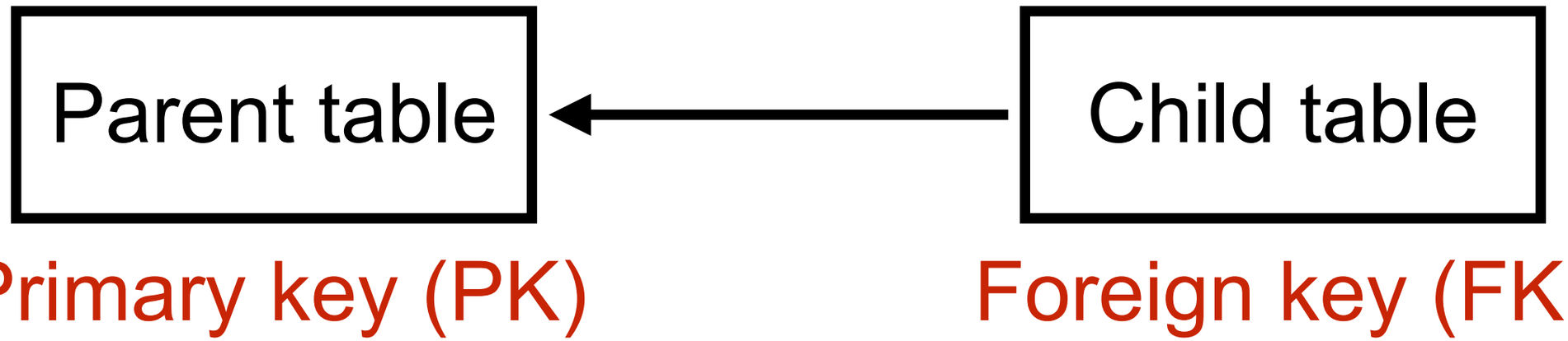
Delete vs Truncate

	Delete	Truncate
Command Type	DML	DDL
Where condition	Support	Not support
Reset identify column	No	Yes
Acquired lock	Row lock	Table lock
Performance	Slow	Faster than delete

Workshop C.R.U.D 01

Relationships

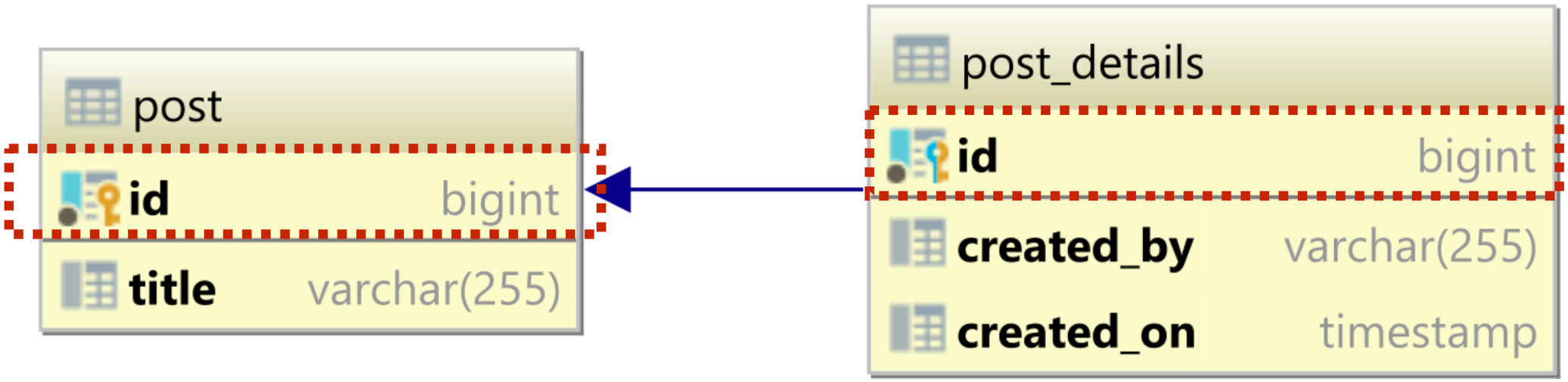
Relationship



Types of Relationship

One-to-One
One-to-Many
Many-to-Many

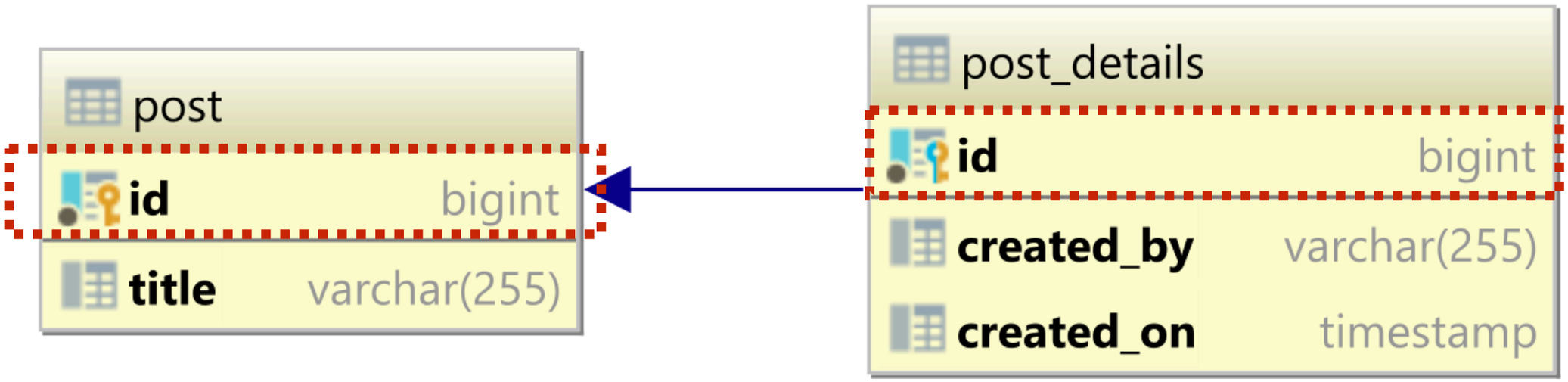
One-to-One



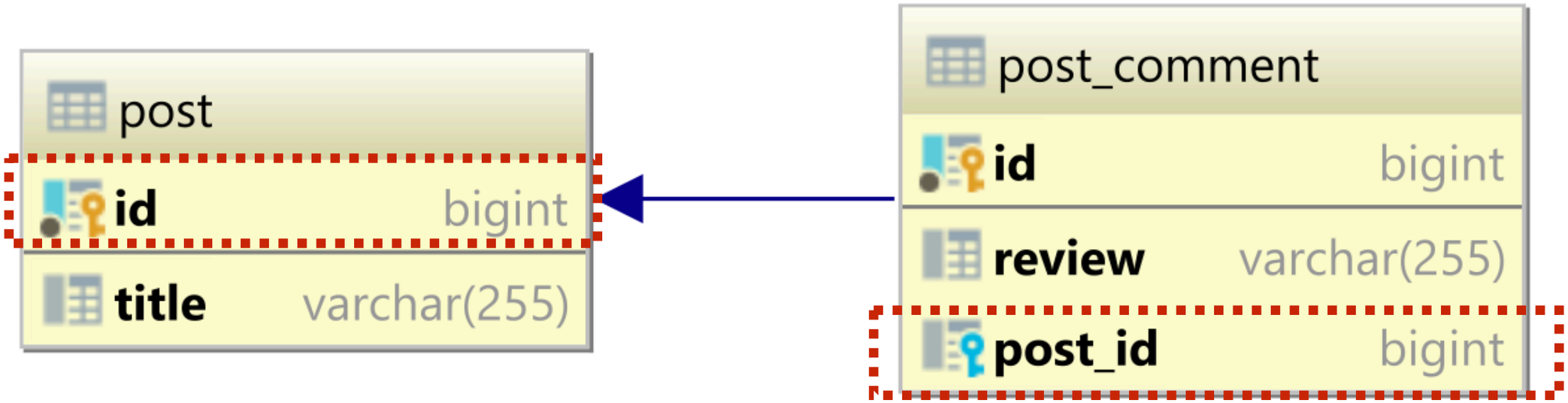
One-to-One

Primary key (PK)

Foreign key (FK)



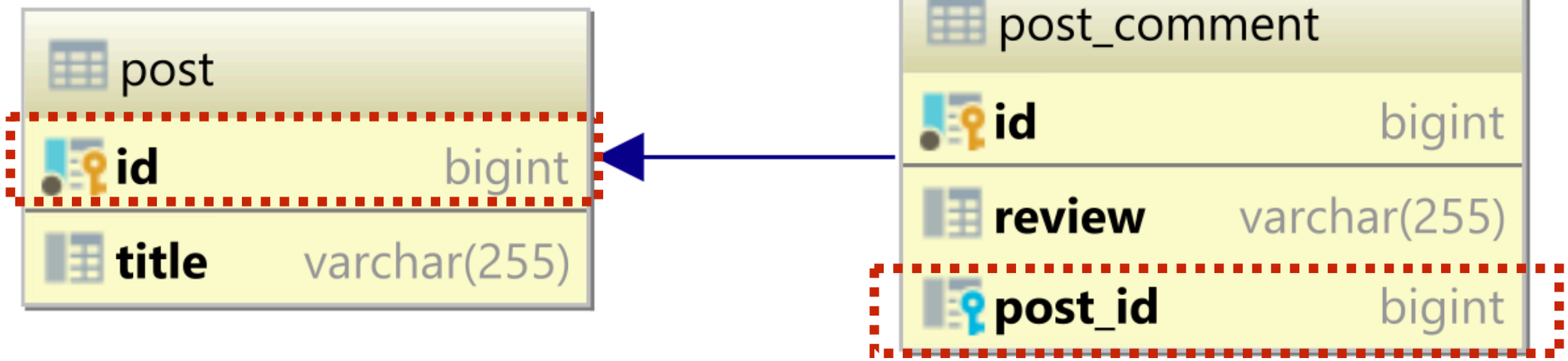
One-to-Many



One-to-Many

Primary key (PK)

Foreign key (FK)



Many-to-Many

Many-to-Many

Primary key (PK)

Primary key (PK)

Foreign key (FK)

Many-to-Many

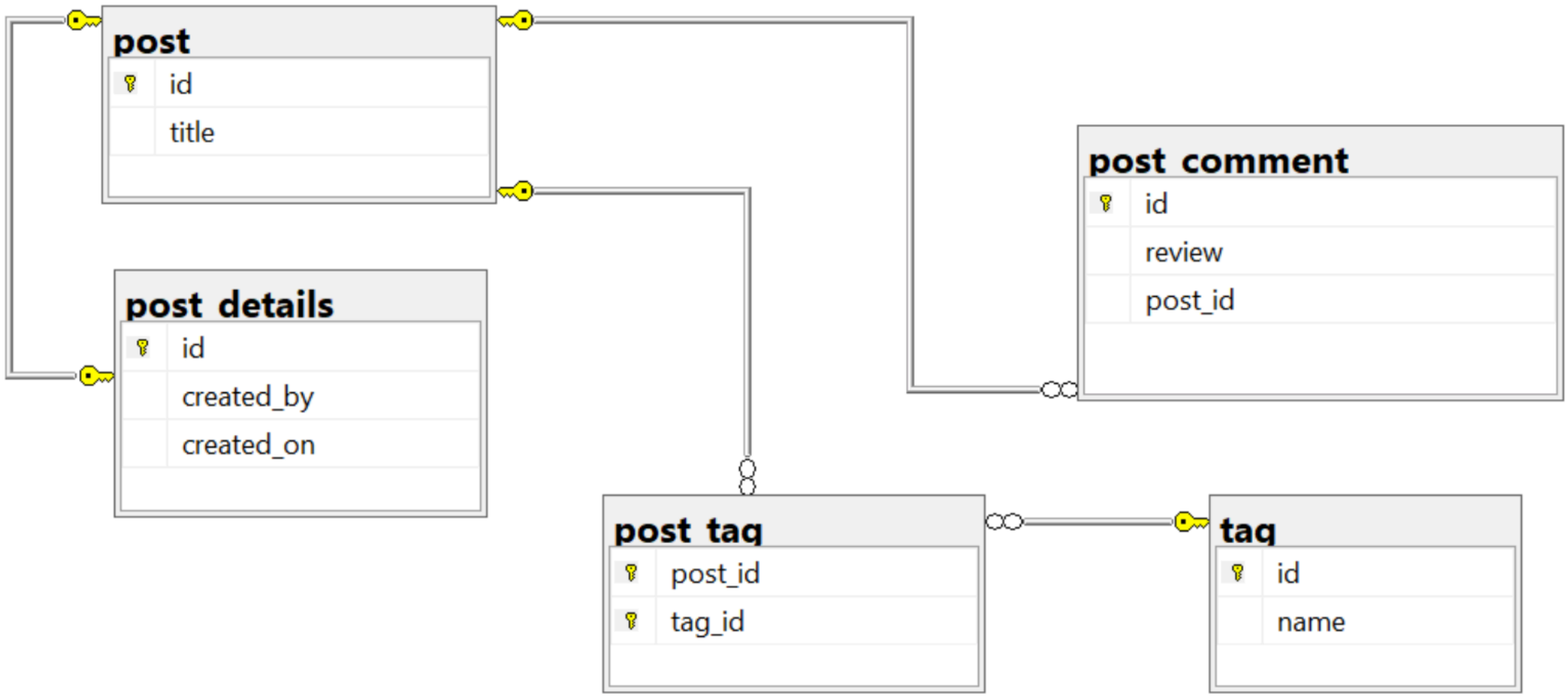
Primary key (PK)

Primary key (PK)

Composite Primary key (FK)

Workshop with relationship

Create Database Diagram



Workshop
Insert data into all tables

Data Query Language (DQL)

Data Query Language (DQL)

Used to performing queries on the data

Select …
From <table>
Where <condition>
Group by …
Having …
Order by …

Basic of SELECT statement

Select <column list>
From <table list>
Where <condition> AND/OR <condition>

Condition in Where clause

Logical comparison operators

=, <, <=, >, >=, <>

LIKE comparison operator

Use string pattern matching

% = zero or more characters

_ = one character

BETWEEN comparison operator

Example

Table customer

Customer_id	Firstname	Lastname	Age
1	Ant	Wallace	1
2	Bat	Dune	5
3	Cat	Puisu	3
4	Dog	Morad	10

List of customer

Age less than 5

```
Select *
From customer
Where  age < 5
```

List of customer

Firstname start with A character

```
Select *
From customer
Where  first name like ‘A%’
```

List of customer

Age more than 3 and less than 20

```
Select *
From customer
Where  age between 3 and 20
```

List of customer

Age more than 3 and less than 20

```
Select *
From customer
Where  age between 3 and 20
```

>= 3 and <= 20 !!

List of customer

Age more than 3 and less than 20

```
Select *
From customer
Where  age > 3 and age < 20
```

Use ORDER BY clause

Select <column list>
From <table list>
Where <condition>
Order by <column list>

Use ORDER BY clause

Keyword **DESC** to see result in a descending order of values

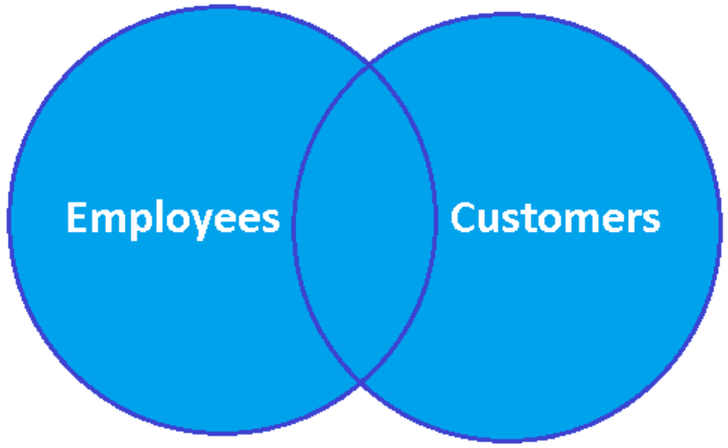
Keyword **ASC** to see result in a ascending order of values

“Order By name DESC, age ASC”

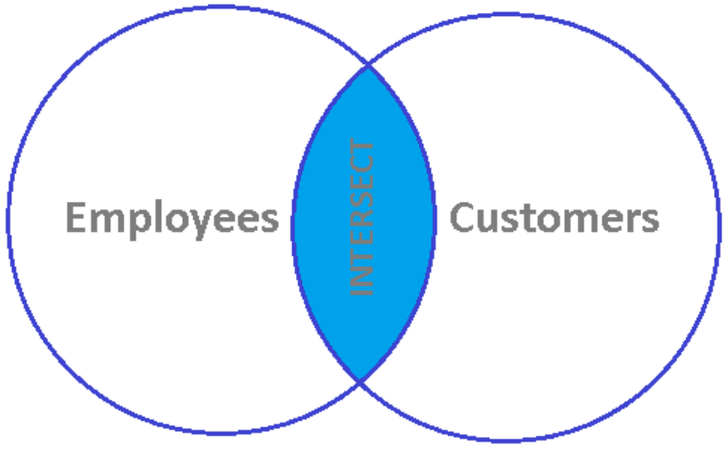
Set operations

Union/ Union All
Except
Intersect

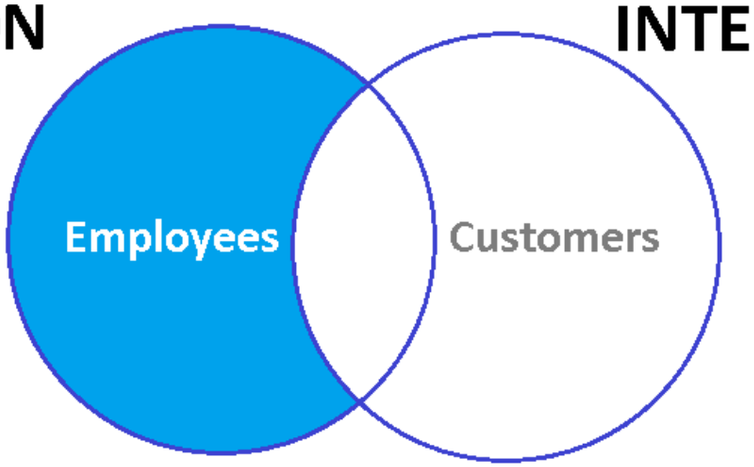
Set operations



UNION



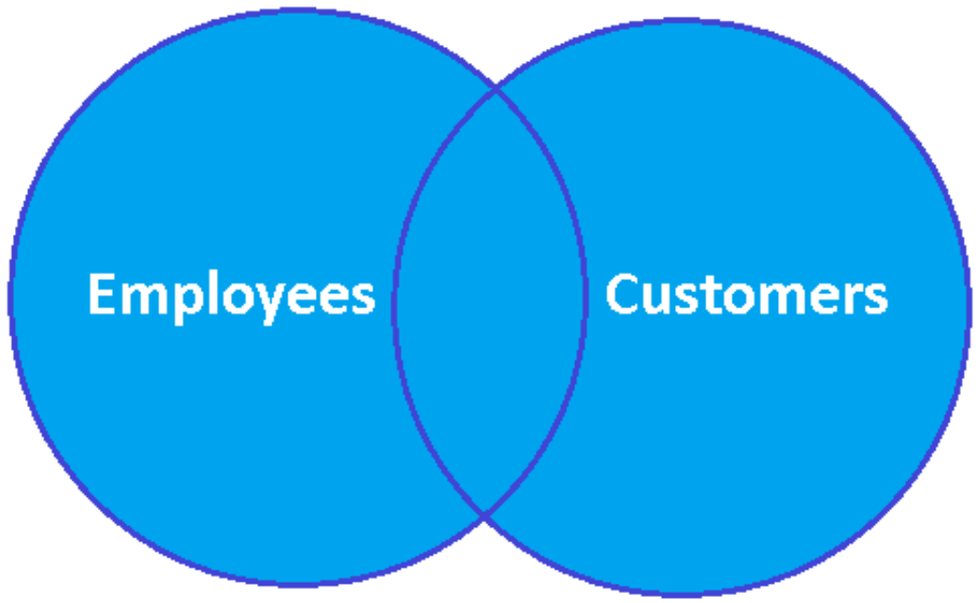
INTERSECT



EXCEPT

Union

Combine multiple inputs into a single result

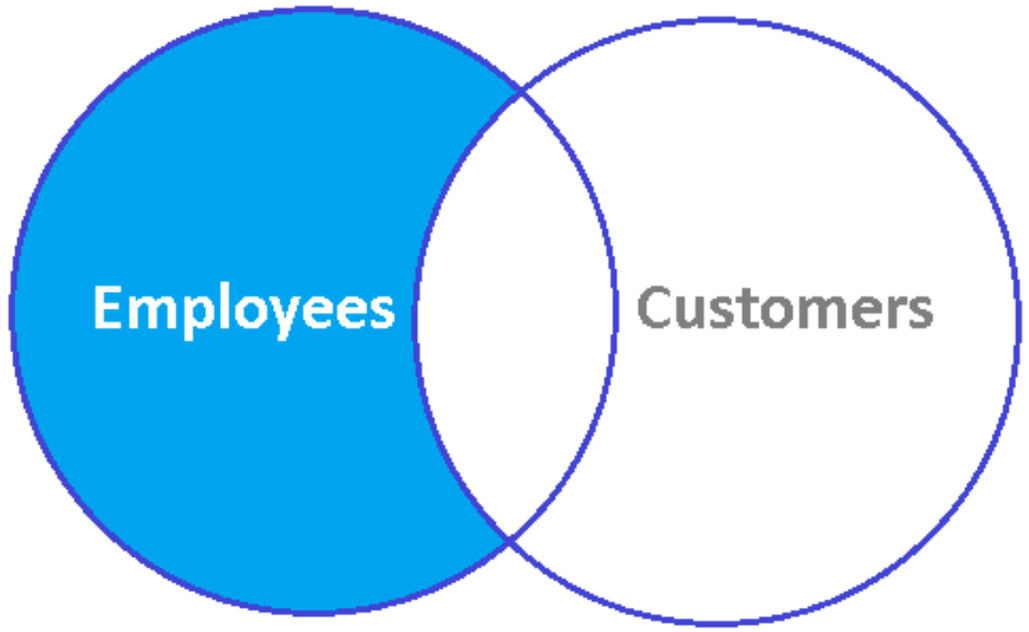


Union All

Like Union operator **but not filter out duplicated rows**

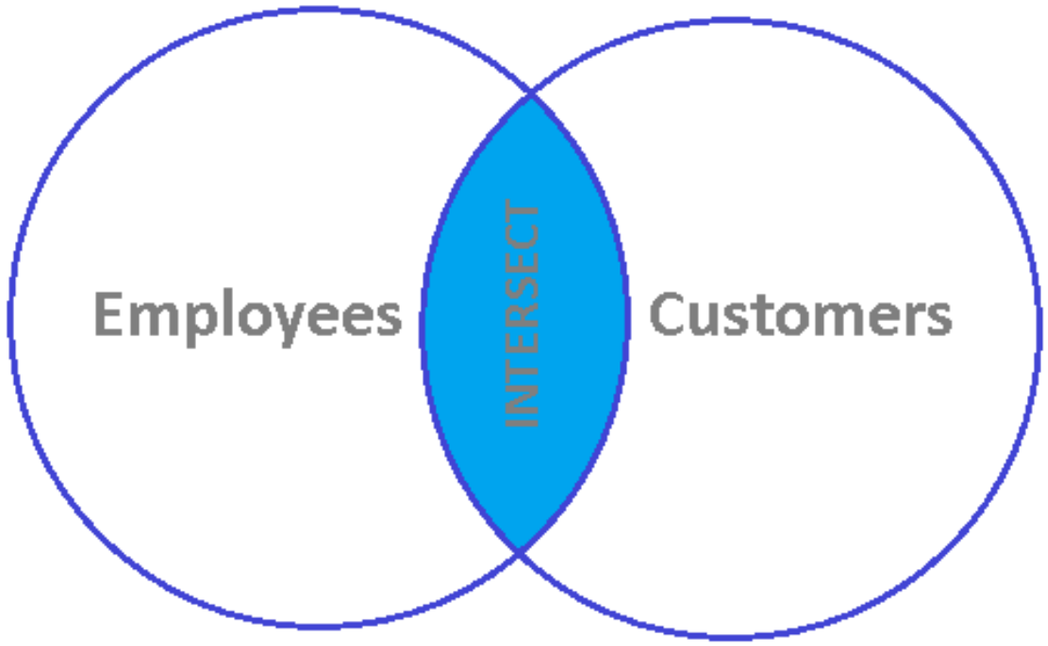
Except

Return distinct rows that only in first input



Intersect

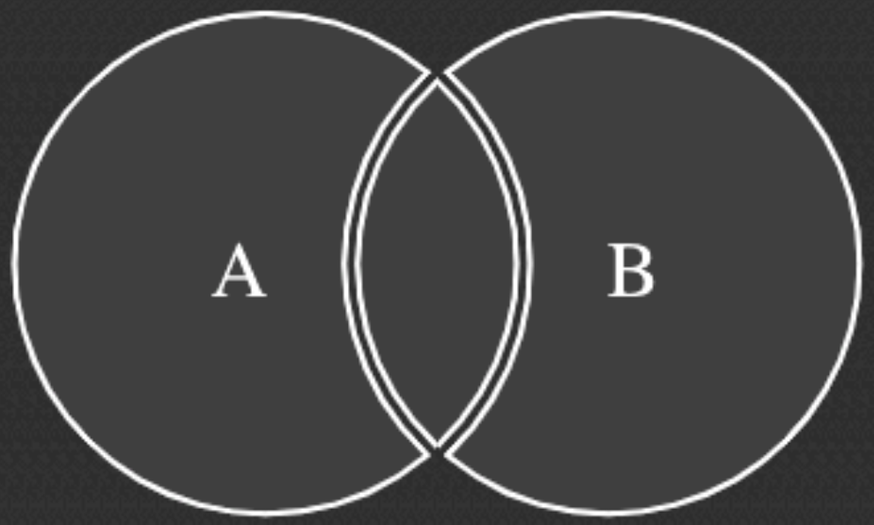
Return only distinct rows that are present in both input



Joining

SQL Joins Visualizer

SQL Joins Visualizer help to you build SQL JOIN between two tables by using of Venn diagrams.
Working offline and as mobile app.

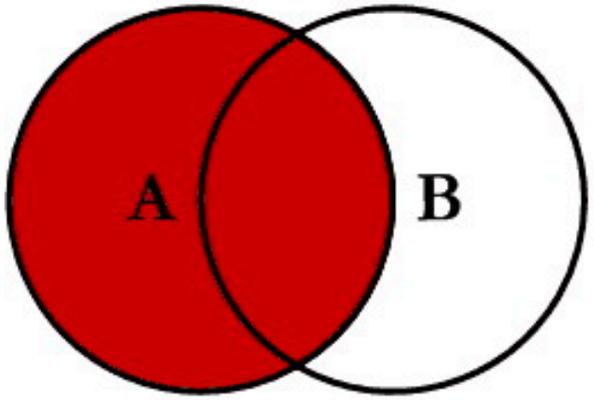


Please select how do you want to do SQL JOIN between two table

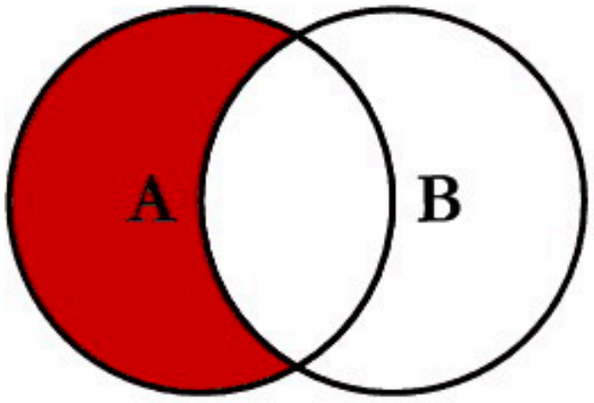
Copy SQL

https://sql-joins.leopard.in.ua/

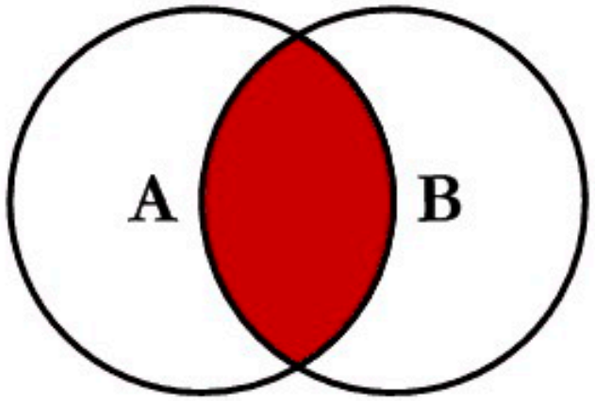
SQL JOINS



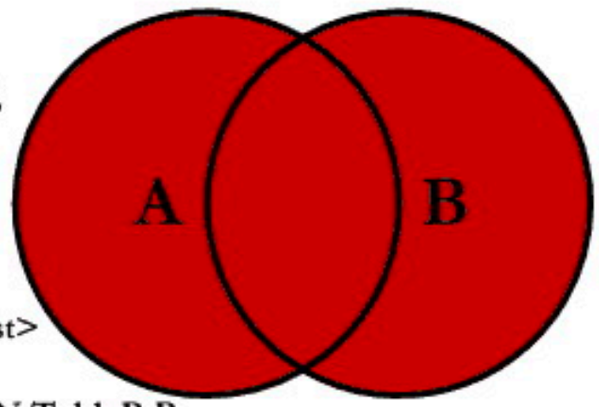

```
SELECT <select_list>
FROM TableA A
LEFT JOIN TableB B
ON A.Key = B.Key
```



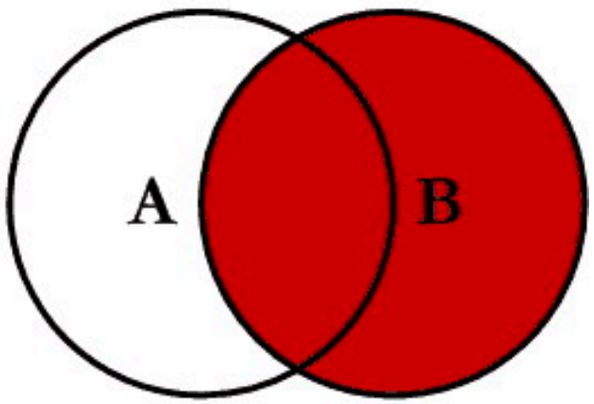

```
SELECT <select_list>
FROM TableA A
LEFT JOIN TableB B
ON A.Key = B.Key
WHERE B.Key IS NULL
```



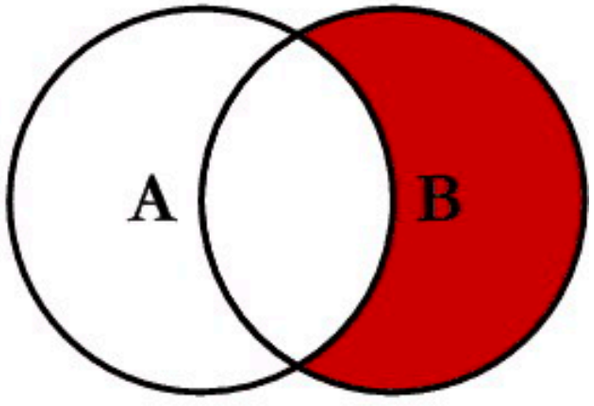

```
SELECT <select_list>
FROM TableA A
INNER JOIN TableB B
ON A.Key = B.Key
```



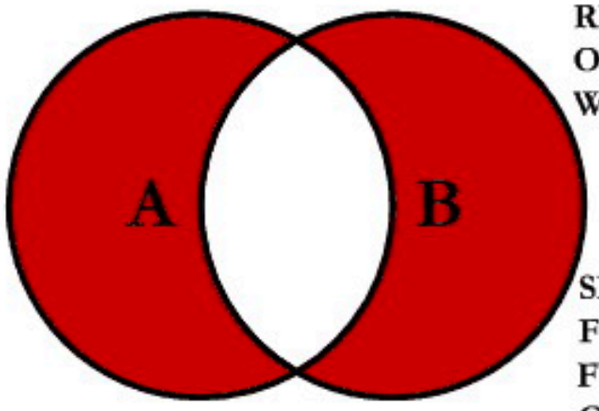

```
SELECT <select_list>
FROM TableA A
FULL OUTER JOIN TableB B
ON A.Key = B.Key
```




```
SELECT <select_list>
FROM TableA A
RIGHT JOIN TableB B
ON A.Key = B.Key
```



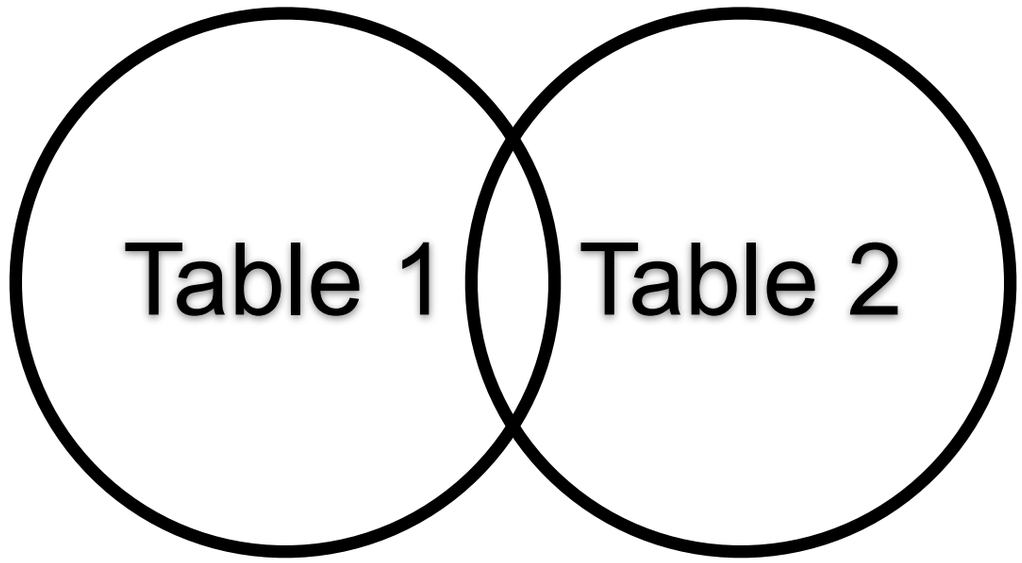

```
SELECT <select_list>
FROM TableA A
RIGHT JOIN TableB B
ON A.Key = B.Key
WHERE A.Key IS NULL
```




```
SELECT <select_list>
FROM TableA A
FULL OUTER JOIN TableB B
ON A.Key = B.Key
WHERE A.Key IS NULL
OR B.Key IS NULL
```

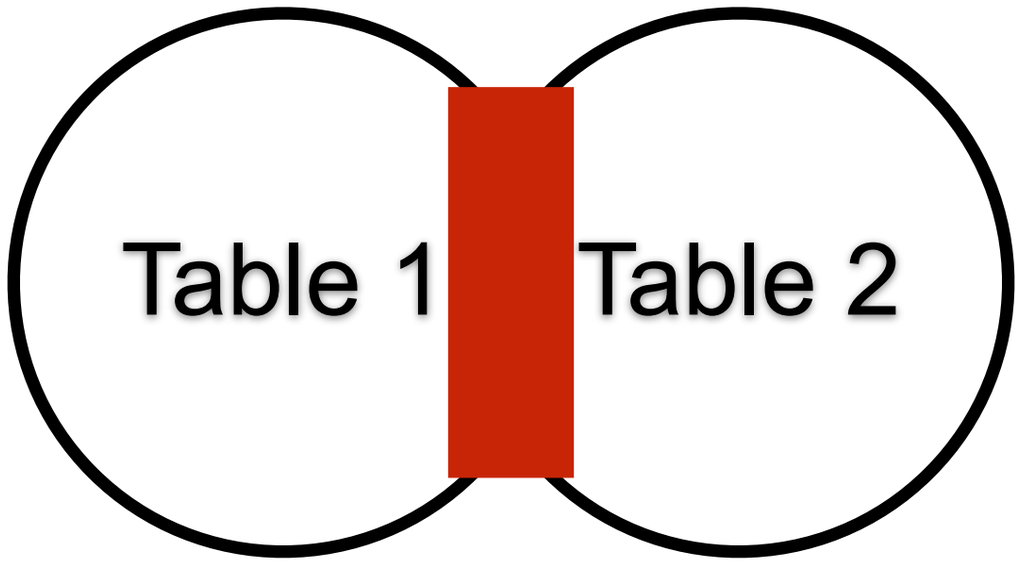
(Inner) Join

Returns rows that have matching values in both tables



(Inner) Join

Returns rows that have matching values in both tables



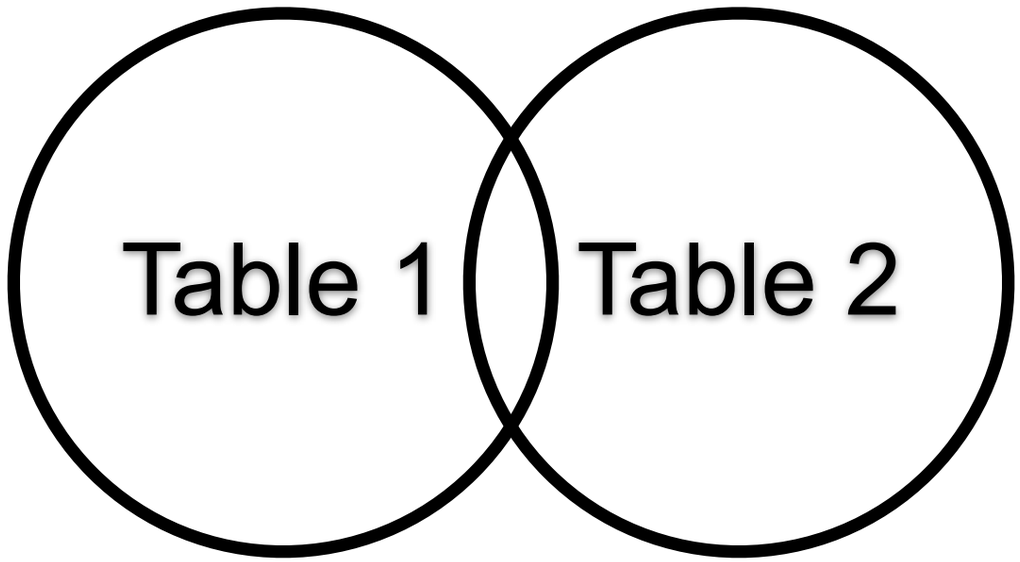
(Inner) Join

```
SELECT *
FROM Table1
INNER JOIN Table2 ON Table1.id = Table2.id
```

```
SELECT *
FROM   Table1, Table2
WHERE  Table1.id = Table2.id
```

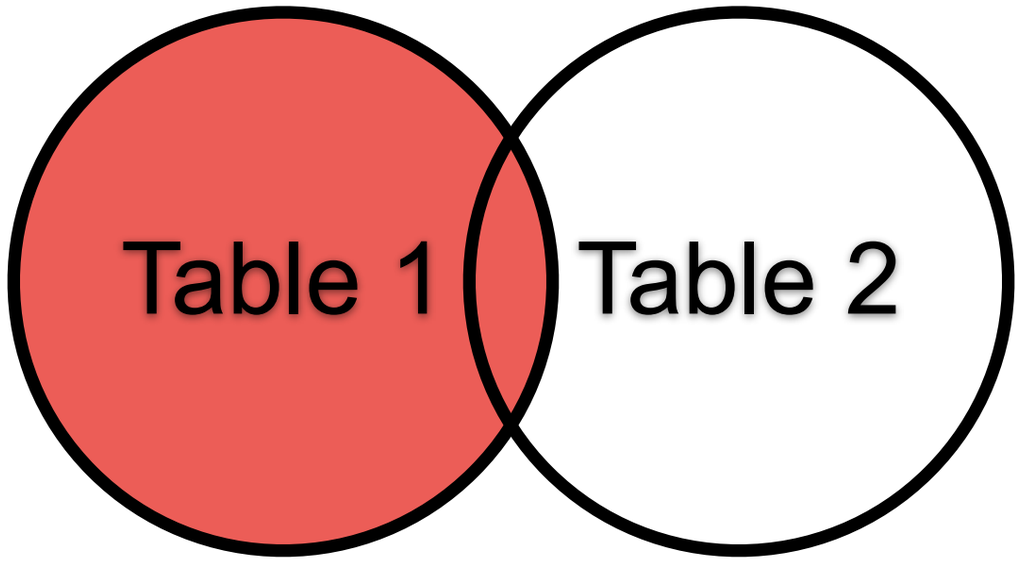
Left (outer) Join

Returns all rows from the left table and matched rows from the right table



Left (outer) Join

Returns all rows from the left table and matched rows from the right table

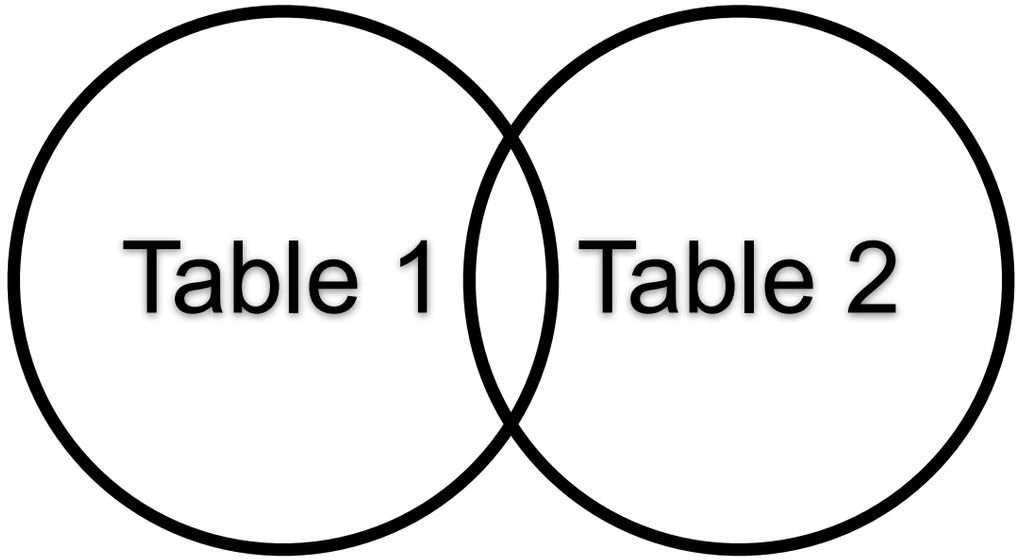


Left (outer) Join

```
SELECT *
FROM table1
LEFT JOIN table2
ON table1.column_name = table2.column_name;
```

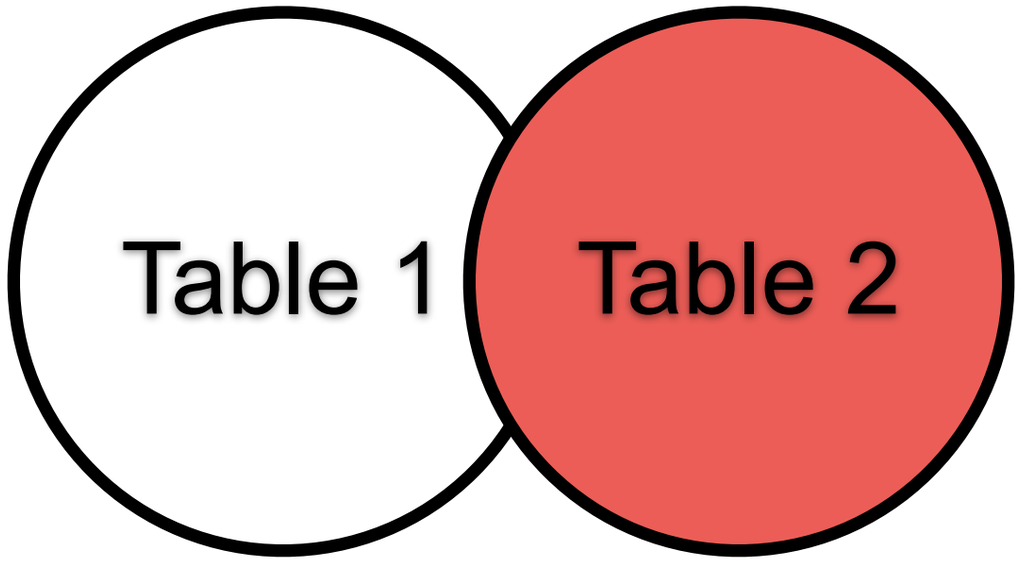
Right (outer) Join

Returns all rows from the right table and matched rows from the left table



Right (outer) Join

Returns all rows from the right table and matched rows from the left table

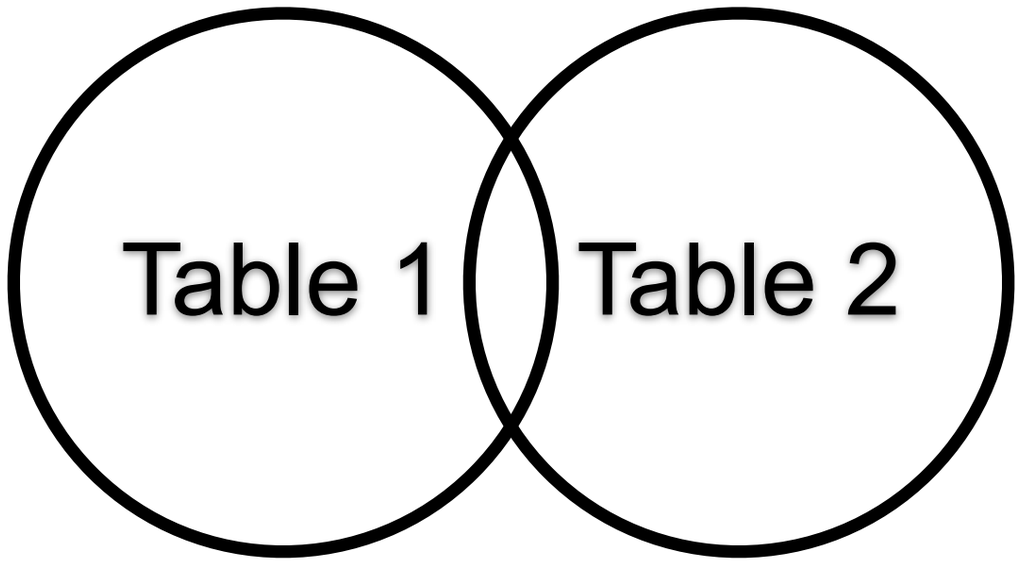


Right (outer) Join

```
SELECT *
FROM table1
RIGHT JOIN table2
ON table1.column_name = table2.column_name;
```

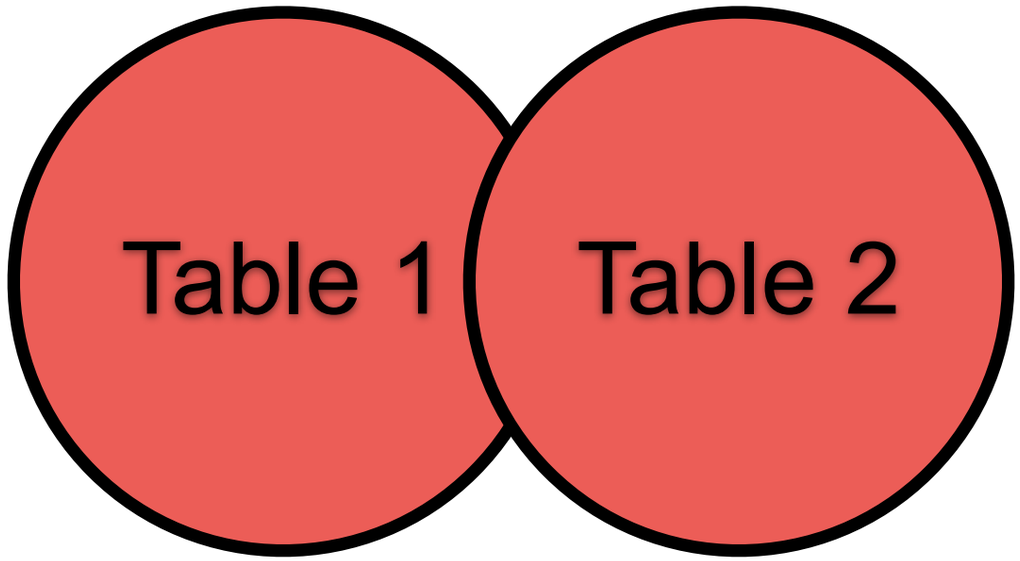
Full (outer) Join

Returns all rows when there a match in either left or right table



Full (outer) Join

Returns all rows when there a match in either left or right table



Full (outer) Join

```
SELECT *
FROM table1
FULL OUTER JOIN table2
ON table1.column_name = table2.column_name;
```

Aggregation in SQL

Use to summarize information from multiple rows into single

Try to grouping data with **GROUPY BY**

Build-in aggregation function ?

Aggregation functions

Count
Sum
Max
Min
Avg

Modifier in Aggregation

ALL (not remove duplicate)
DISTINCT (remove duplicate)
Max
Min
Avg

C.R.U.D with more tables

Workshop with post data

Workshop as a Team Company database

Company Database

Employee

emp_id	first_name	last_name	birth_date	sex	salary	super_id	branch_id
100	David	Wallace	1967-11-17	M	250,000	NULL	1
101	Jan	Levinson	1961-05-11	F	110,000	100	1
102	Michael	Scott	1964-03-15	M	75,000	100	2
103	Angela	Martin	1971-06-25	F	63,000	102	2
104	Kelly	Kapoor	1980-02-05	F	55,000	102	2
105	Stanley	Hudson	1958-02-19	M	69,000	102	2
106	Josh	Porter	1969-09-05	M	78,000	100	3
107	Andy	Bernard	1973-07-22	M	65,000	106	3
108	Jim	Halpert	1978-10-01	M	71,000	106	3

Branch

branch_id	branch_name	mgr_id	mgr_start_date
1	Corporate	100	2006-02-09
2	Scranton	102	1992-04-06
3	Stamford	106	1998-02-13

Client

client_id	client_name	branch_id
400	Dunmore Highschool	2
401	Lackawana Country	2
402	FedEx	3
403	John Daly Law, LLC	3
404	Scranton Whitepages	2
405	Times Newspaper	3
406	FedEx	2

Works_With

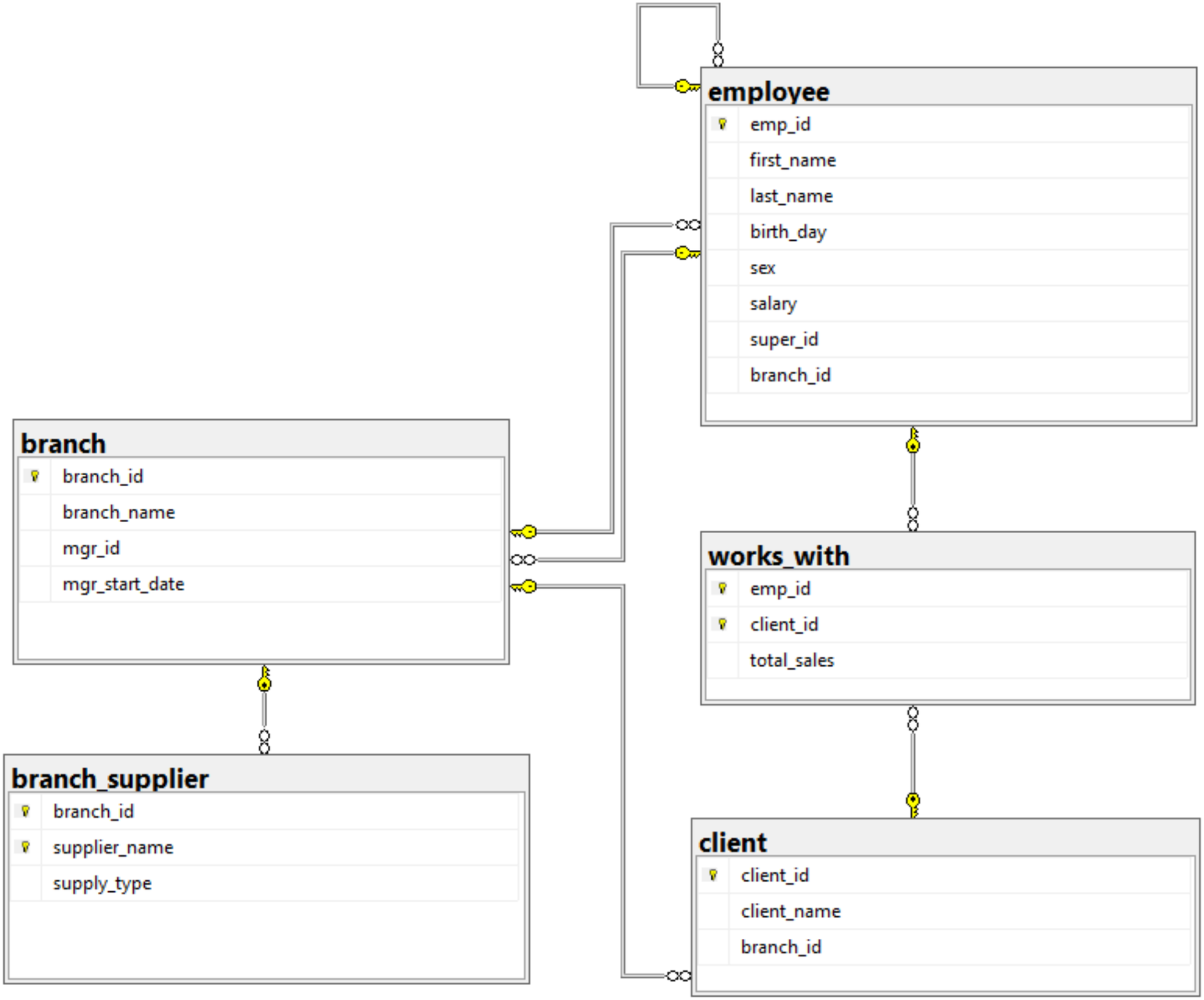
emp_id	client_id	total_sales
105	400	55,000
102	401	267,000
108	402	22,500
107	403	5,000
108	403	12,000
105	404	33,000
107	405	26,000
102	406	15,000
105	406	130,000

Branch Supplier

branch_id	supplier_name	supply_type
2	Hammer Mill	Paper
2	Uni-ball	Writing Utensils
3	Patriot Paper	Paper
2	J.T. Forms & Labels	Custom Forms
3	Uni-ball	Writing Utensils
3	Hammer Mill	Paper
3	Stamford Lables	Custom Forms

Labels

	Primary Key
	Foreign Key
	Attribute



Database design

Nomalization Form

What id Normalization ?

Process to organize data with **efficiency**

What id Normalization ?

Process to organize data with **efficiency**
Eliminate redundancy
Ensure data is stored in the right table

Eliminate need for restructure database when data is added

Benefits

Data deduplication
Consistency
Data Integrity

Drawbacks

Divide data into many tables
Complex JOINs
Hard to change the structure of data
Impedance mismatch with modern applications

5 levels of Normal form

Third normal form is sufficient for most typical database applications

First Normal Form (1NF)

No repeat or duplicate columns
Each column contains only a single value
Each row is unique

Example

Item	Colors	Price	Tax
T-shirt	Red, blue	12.00	0.60
Polo	Red, yellow	12.00	0.60
T-shirt	Red, blue	12.00	0.60
Other	Blue, black	25.00	1.25

Duplication rows

Item	Colors	Price	Tax
T-shirt	Red, blue	12.00	0.60
Polo	Red, yellow	12.00	0.60
T-shirt	Red, blue	12.00	0.60
Other	Blue, black	25.00	1.25

No Primary Key

Item	Colors	Price	Tax
T-shirt	Red, blue	12.00	0.60
Polo	Red, yellow	12.00	0.60
T-shirt	Red, blue	12.00	0.60
Other	Blue, black	25.00	1.25

Multiple-value in colors

Item	Colors	Price	Tax
T-shirt	Red, blue	12.00	0.60
Polo	Red, yellow	12.00	0.60
T-shirt	Red, blue	12.00	0.60
Other	Blue, black	25.00	1.25

Improvement with 1NF

Item	Colors	Price	Tax
T-shirt	Red	12.00	0.60
T-shirt	Blue	12.00	0.60
Polo	Red	12.00	0.60
Polo	Yellow	12.00	0.60
Other	Blue	25.00	1.25
Other	Black	25.00	1.25

Second Normal Form (2NF)

All non-key columns depend on all components of the primary key

Price and Tax depend on Item

Item	Colors	Price	Tax
T-shirt	Red	12.00	0.60
T-shirt	Blue	12.00	0.60
Polo	Red	12.00	0.60
Polo	Yellow	12.00	0.60
Other	Blue	25.00	1.25
Other	Black	25.00	1.25

Improvement with 2NF

Item	Colors
T-shirt	Red
T-shirt	Blue
Polo	Red
Polo	Yellow
Other	Blue
Other	Black

Item	Price	Tax
T-shirt	12.00	0.60
Polo	12.00	0.60
Other	25.00	1.25

Third Normal Form (3NF)

No non-key columns depend upon another of the primary key

Tax depend on price, not item

Item	Colors
T-shirt	Red
T-shirt	Blue
Polo	Red
Polo	Yellow
Other	Blue
Other	Black

Item	Price	Tax
T-shirt	12.00	0.60
Polo	12.00	0.60
Other	25.00	1.25

Improvement with 3NF

Item	Colors
T-shirt	Red
T-shirt	Blue
Polo	Red
Polo	Yellow
Other	Blue
Other	Black

Item	Price
T-shirt	12.00
Polo	12.00
Other	25.00

Price	Tax
12.00	0.60
25.00	1.25

Add Relationship ?

Item	Colors
T-shirt	Red
T-shirt	Blue
Polo	Red
Polo	Yellow
Other	Blue
Other	Black

Item	Price
T-shirt	12.00
Polo	12.00
Other	25.00

Price	Tax
12.00	0.60
25.00	1.25

Workshop

Workshop with 3NF

Name	Assignment 1	Assignment 2
Somkiat Pui	Article summary	Paper prototype
John At	Article summary	Paper prototype
Jane Scott	Article summary	Paper prototype

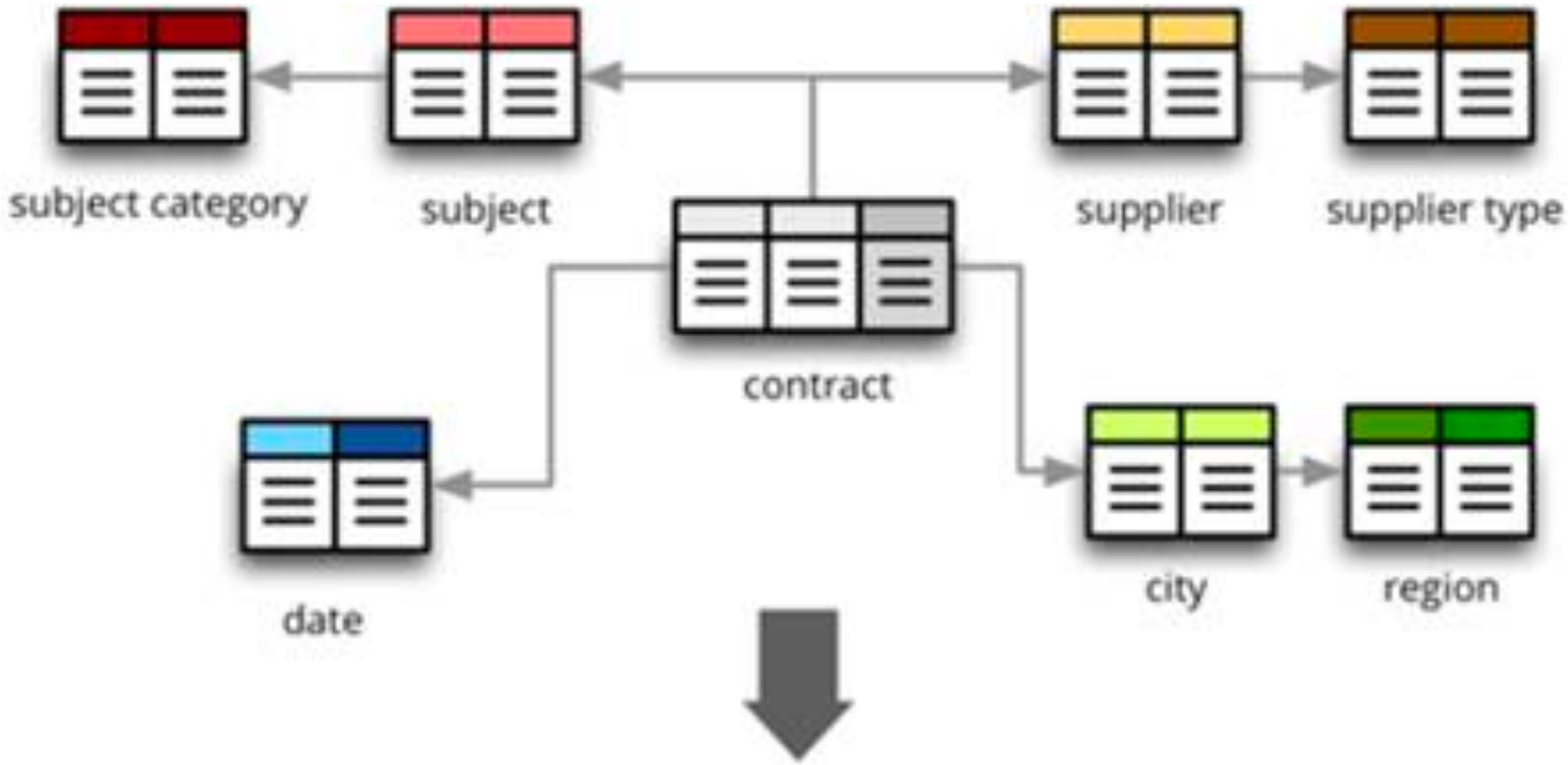
Workshop with 3NF

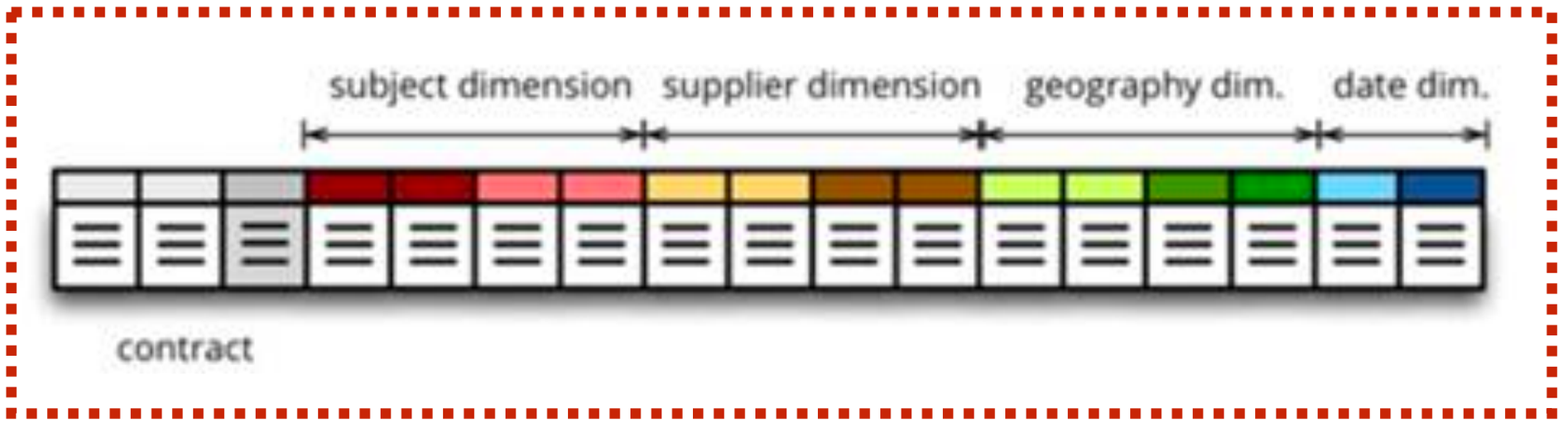
ID	Firstname	Lastname
1	Somkiat	Pui
2	John	At
3	Jane	Scott

ID	Description
1	Article summary
2	Paper prototype

student_id	assignment_id
1	1
1	2
1	3
2	1
2	3
3	2

De-Nomalization ?





Benefits

Reduce **impedance mismatch**
Improve performance
Remove/reduce JOINs
To facilitate and accelerate reporting

Drawbacks

Data duplication
Data inconsistency
Hard to model complex relationships

Impedance mismatch

John Smith

4490 Oak Drive
Albany, NY 12210

INVOICE

Bill To
Jessie M Horne
4312 Wood Road
New York, NY 10031

Ship To
Jessie M Horne
2019 Redbud Drive
New York, NY 10011

Invoice #	INT-001
Invoice Date	11/02/2019
P.O.#	2412/2019
Due Date	26/02/2019

QTY	DESCRIPTION	UNIT PRICE	AMOUNT
1	Front and rear brake cables	100.00	100.00
2	New set of pedal arms	25.00	50.00
3	Labor 3hrs	15.00	45.00
		Subtotal	195.00
		Sales Tax 5.0%	9.75
		TOTAL	**$204.75**

Common mistake in Database design !!

Mistakes

Poor preplanning
Failure to understand the purpose of data
Poor Normalization
Poor indexing
Redundant records
Inconsistent naming convention
Poor documentation
Inadequate testing

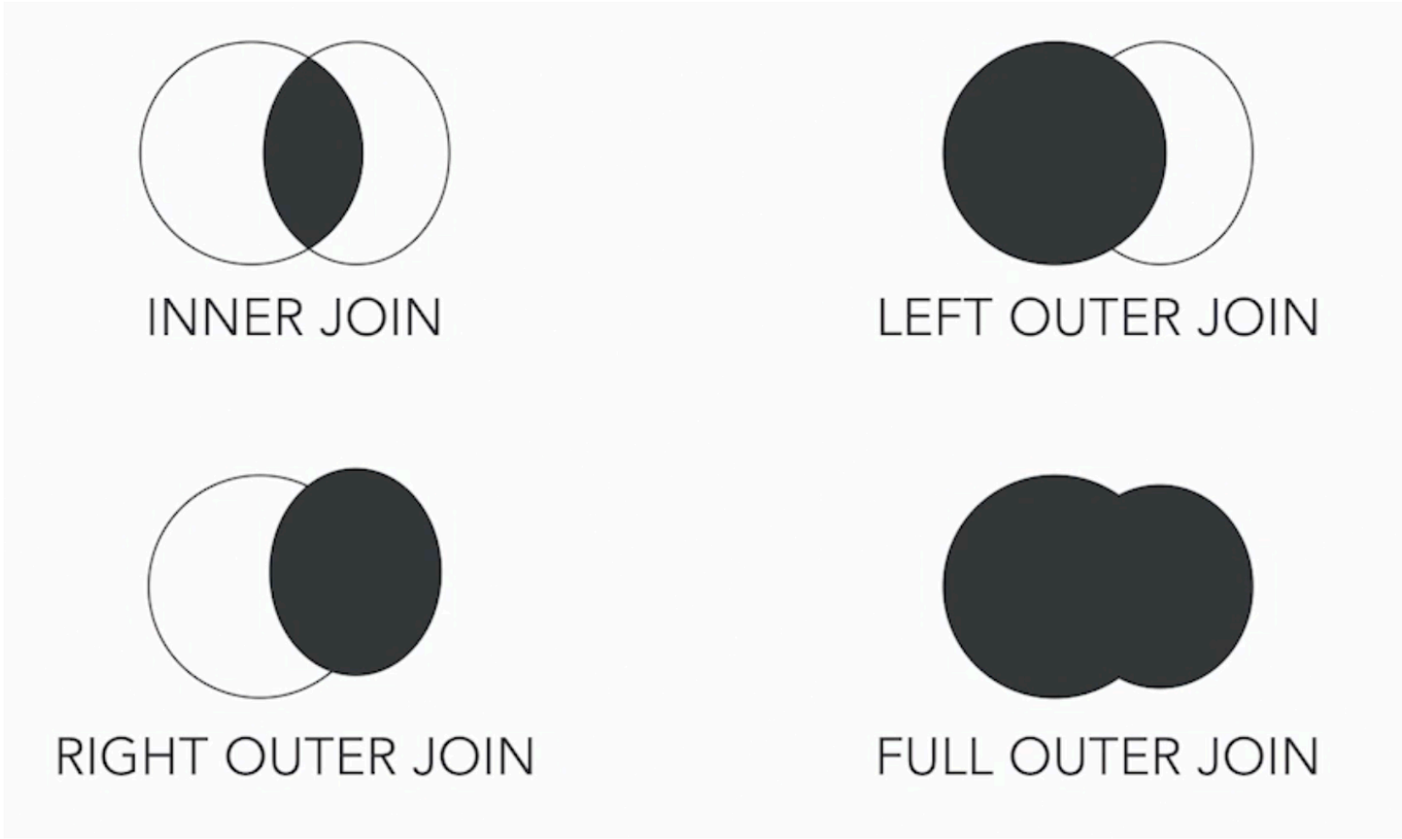
Improve query performance

Benchmark and performance testing with PostgreSQL

TODO

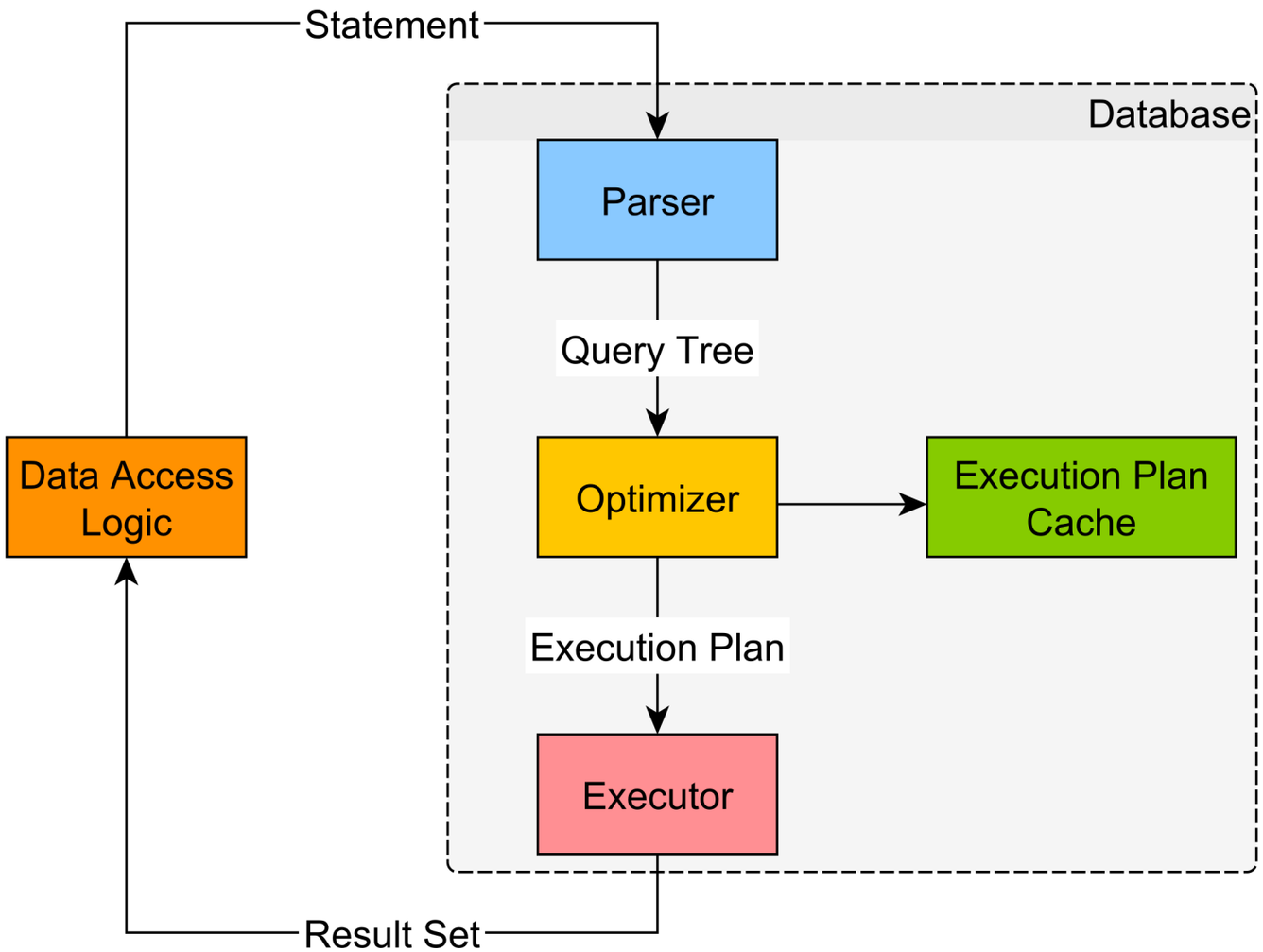
Tuning ?

Reduce query response time with query tuning
Explain query ?



SQL use execute plan

Set of step that scan, filter and join data



Efficient execution plan ?

Good or Bad ?

Good

Using index to find data

Bad

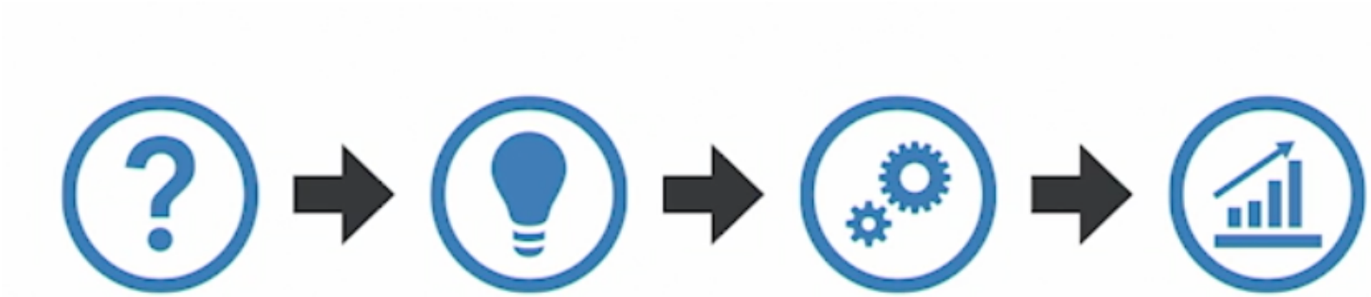
Scan 1M rows to find 100 rows

Step to learn

Understand query plan steps

Understand trade-offs in way to implement query plan

Learn techniques that lead to efficient plans



Scanning tables

customer_id	fname	lname	last_purchase_date	status_level
1234	Alice	Smith	23-Jan-2016	silver
2345	Bob	Washington	14-July-2018	bronze
2352	Charlie	Johnson	20-Dec-2018	gold
2678	Chendong	Mao	19-Dec-2017	gold
3422	Javier	Valdez	23-Dec-2018	gold
3577	Avery	Winston	5-Jan-2019	silver
4240	Charles	Davis	15-Jan-2017	bronze
6235	Vihaan	Patel	12-Nov-2018	gold
2352	Katherine	Coltrane	04-Apr-2016	silver

Scanning tables

Scanning looks at each row

Fetch data block containing row

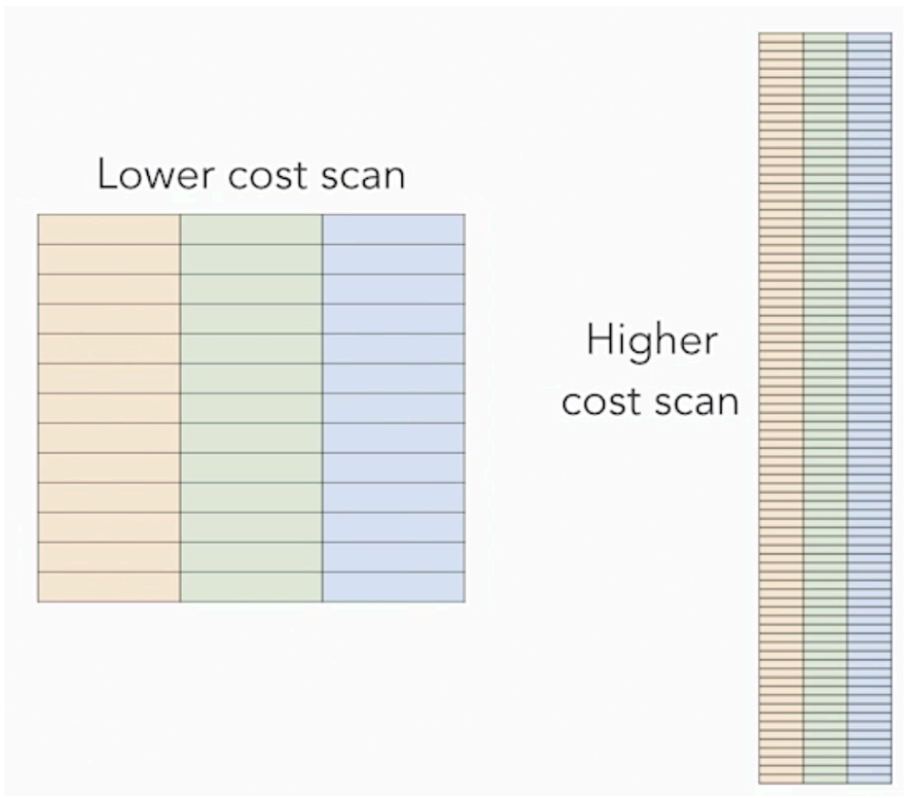
Apply filters or conditions

Cost of query based on number of rows in the table

Cost based on number of rows

Scanning on small tables is efficient

Scanning on large tables can be efficient if have few queries



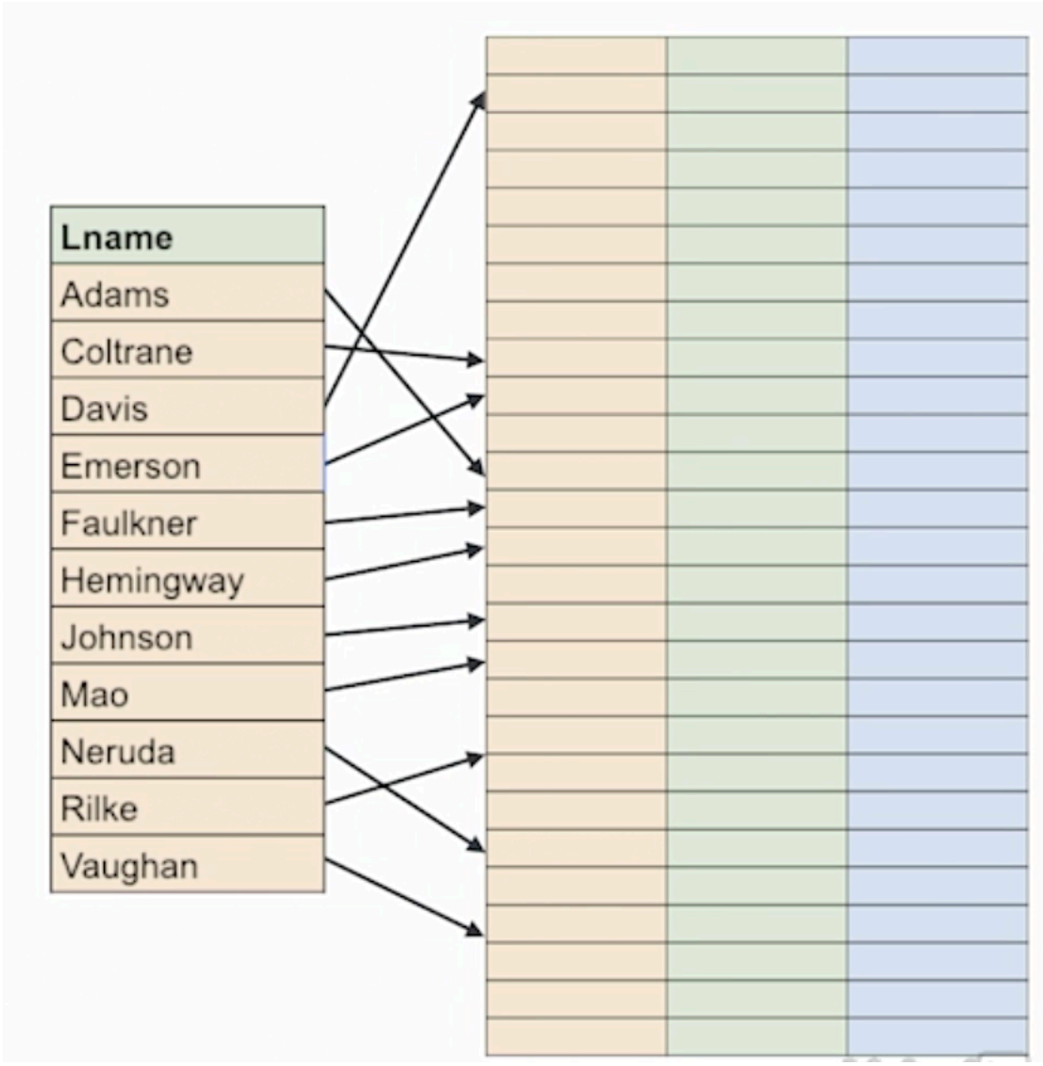
Indexing reduce full table scan

Indexes are **ordered**

Faster to search index for an attribute value

Point to location of row

Indexing reduce full table scan



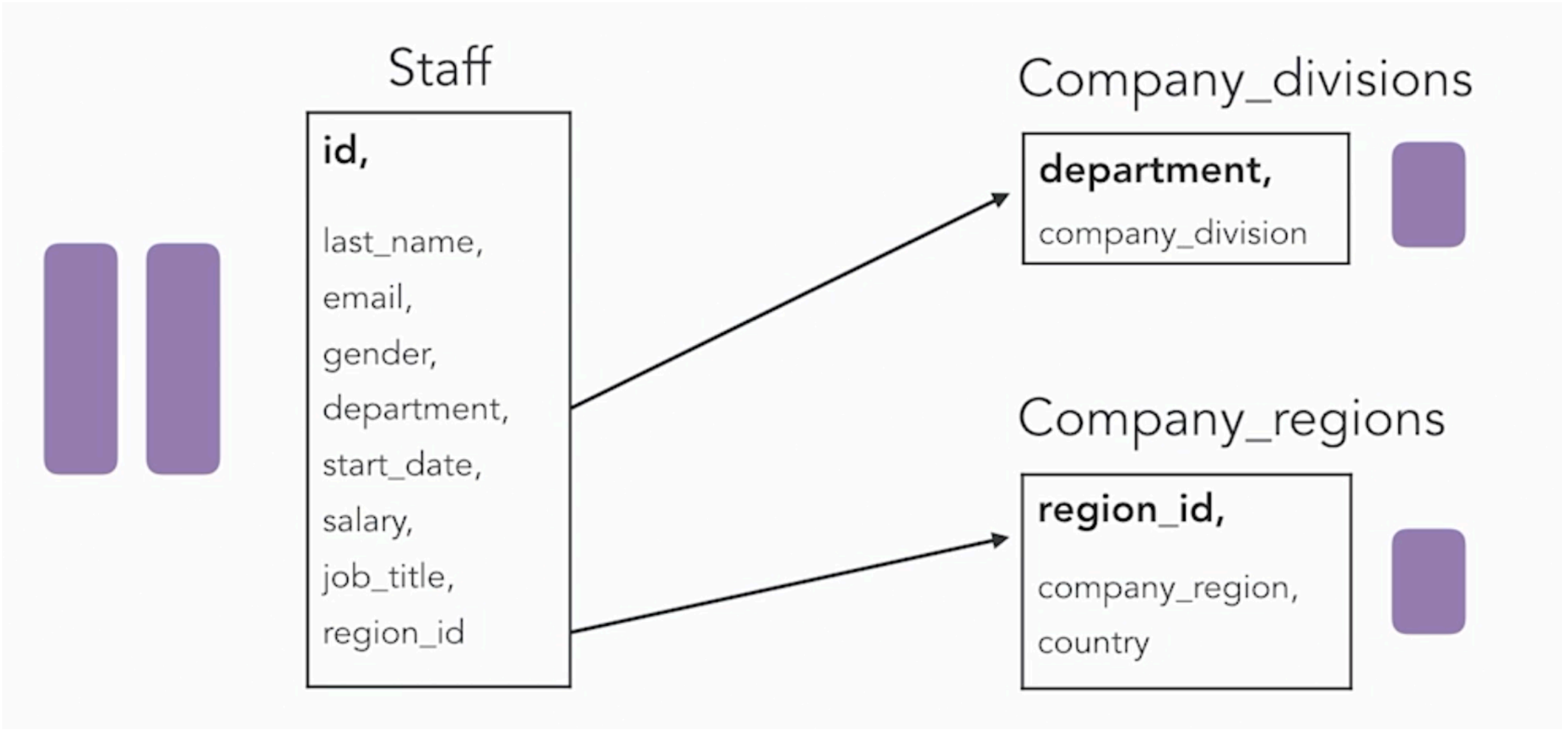
Purposes of indexes

Speed up access to data
Help enforce constraints
Indexes are ordered
Typically smaller than tables

Implementation of indexes

Data structure separate from table
Sometime duplicate some data (e.g. key)
Organised differently than table data

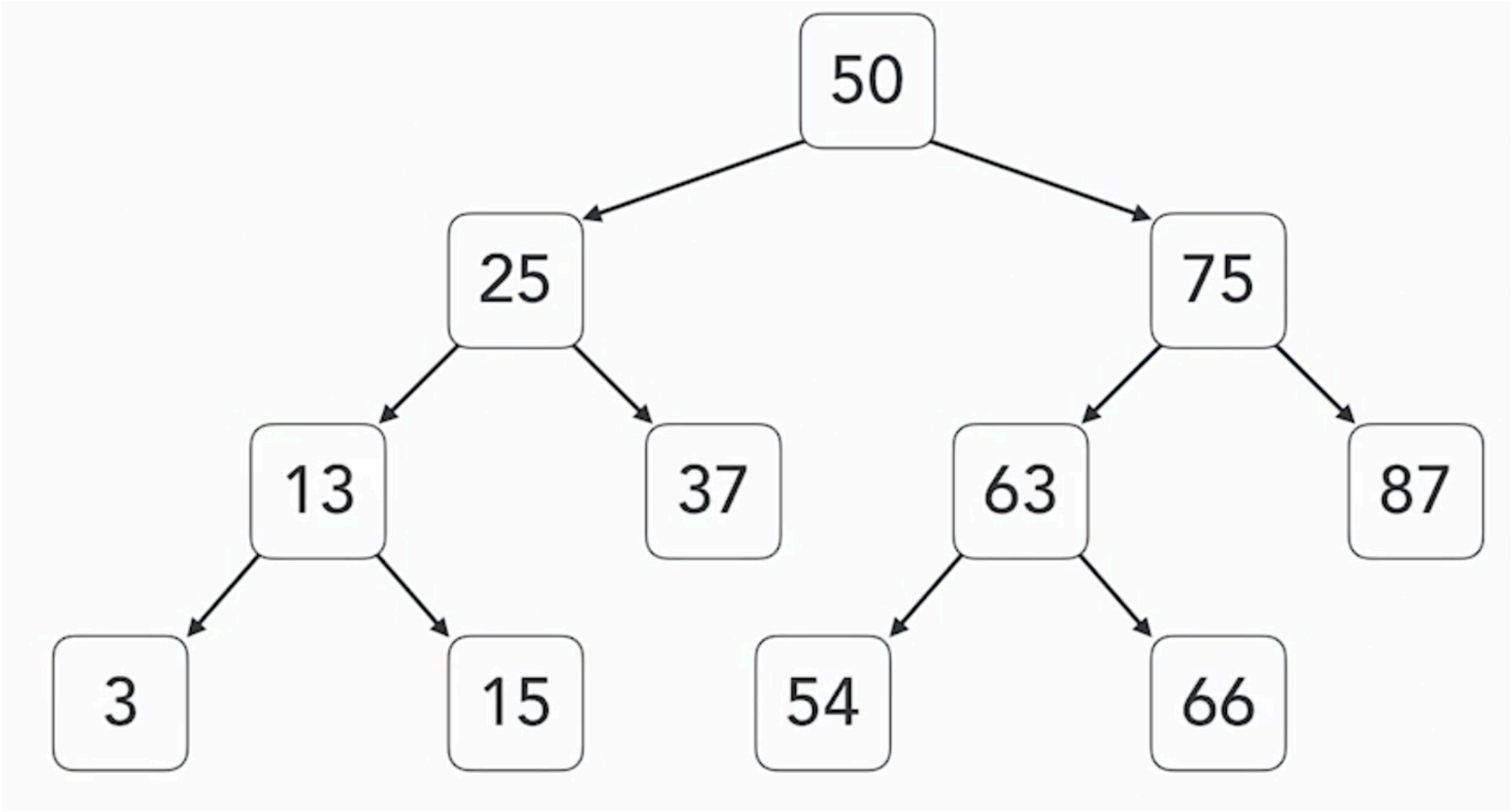
Implementation of indexes



Types of indexes

B-tree for equality and range query
Hash for equality query
Bitmap for inclusion
Specialize for geo-spatial or user-defined

B-tree indexes



B-tree indexes

Most common type of index

Use when large number of possible values in a column

Rebalances as need

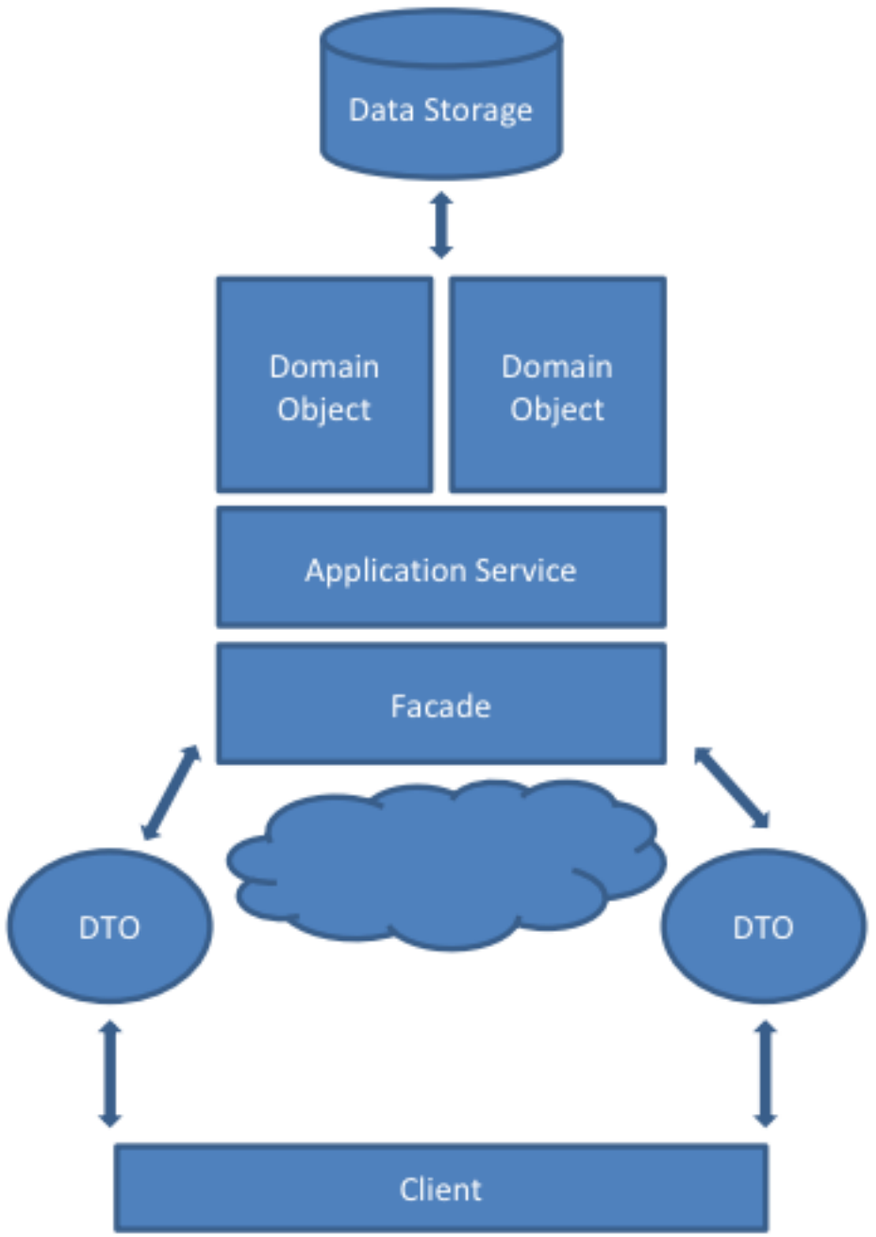
Time to access is based on **depth of tree**

Benchmark and performance testing with PostgreSQL

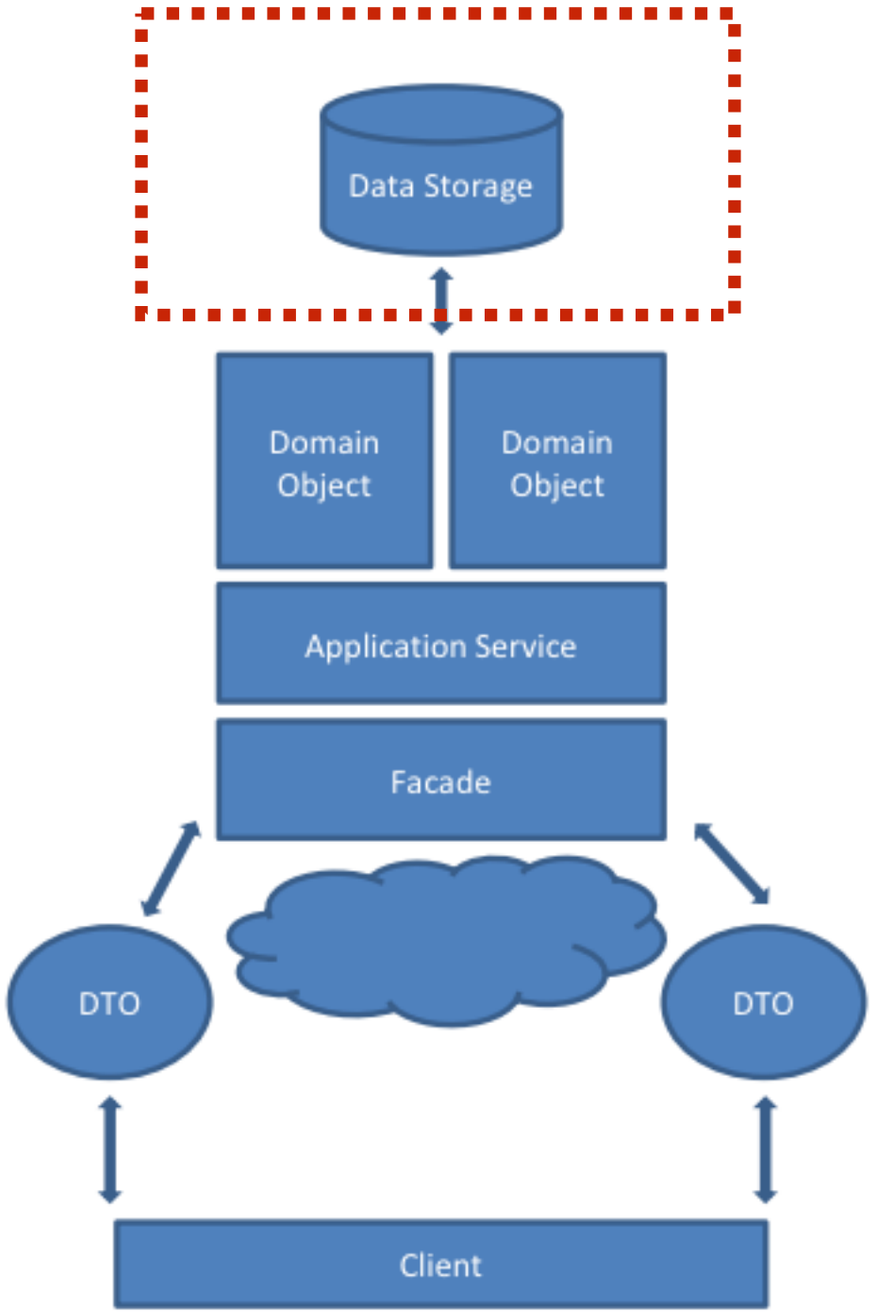
Reporting problem ?

Read vs Write operation with Database

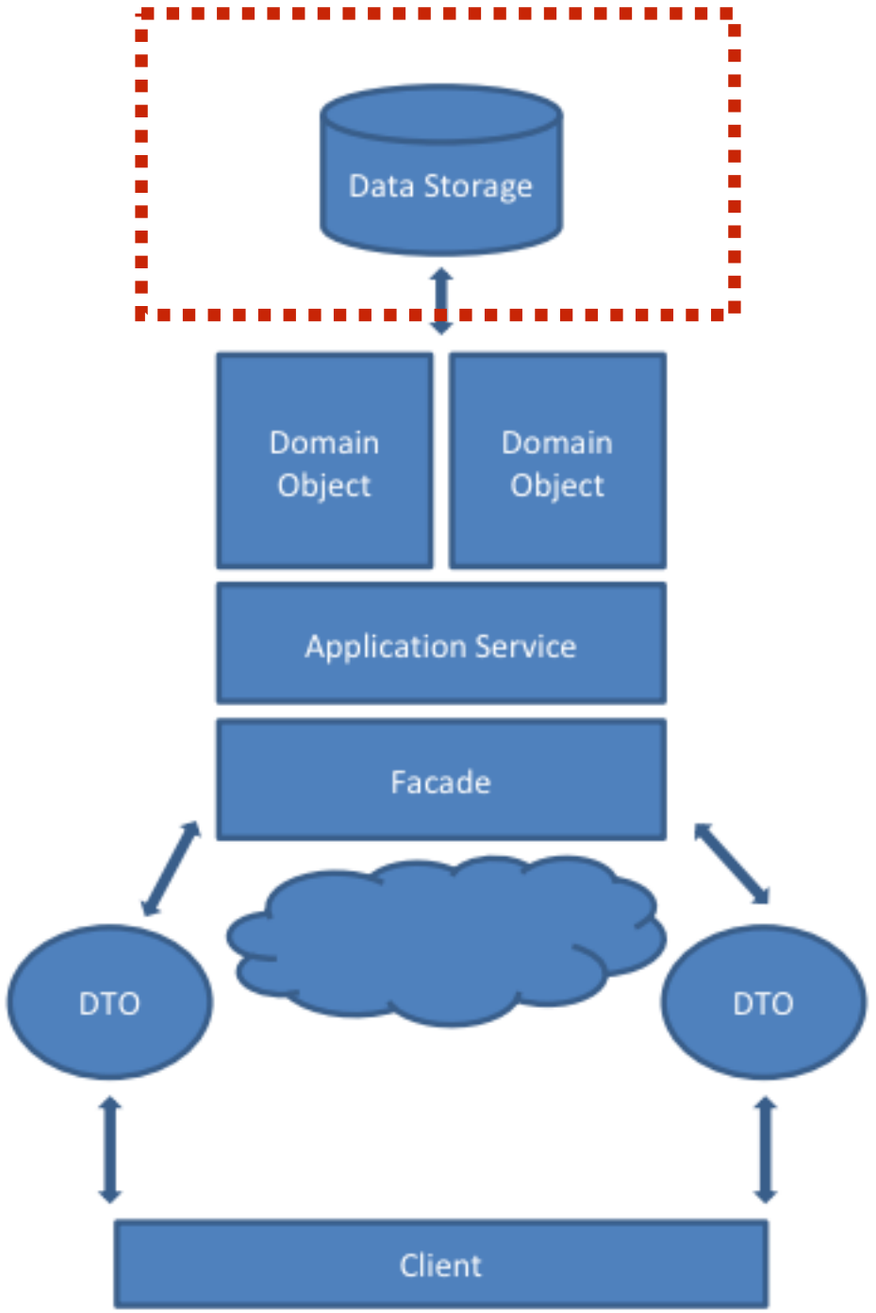
Simple Architecture



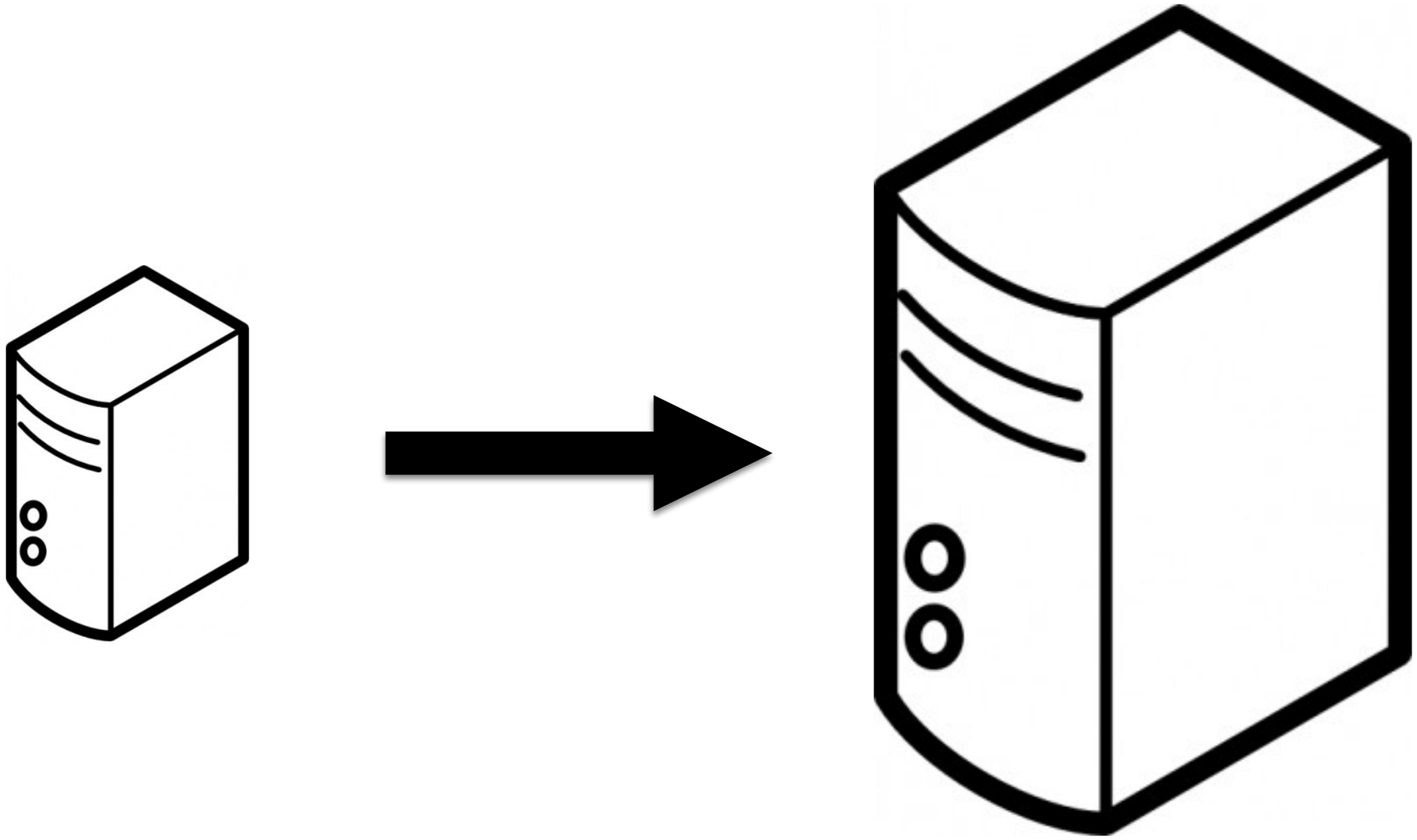
Read and Write



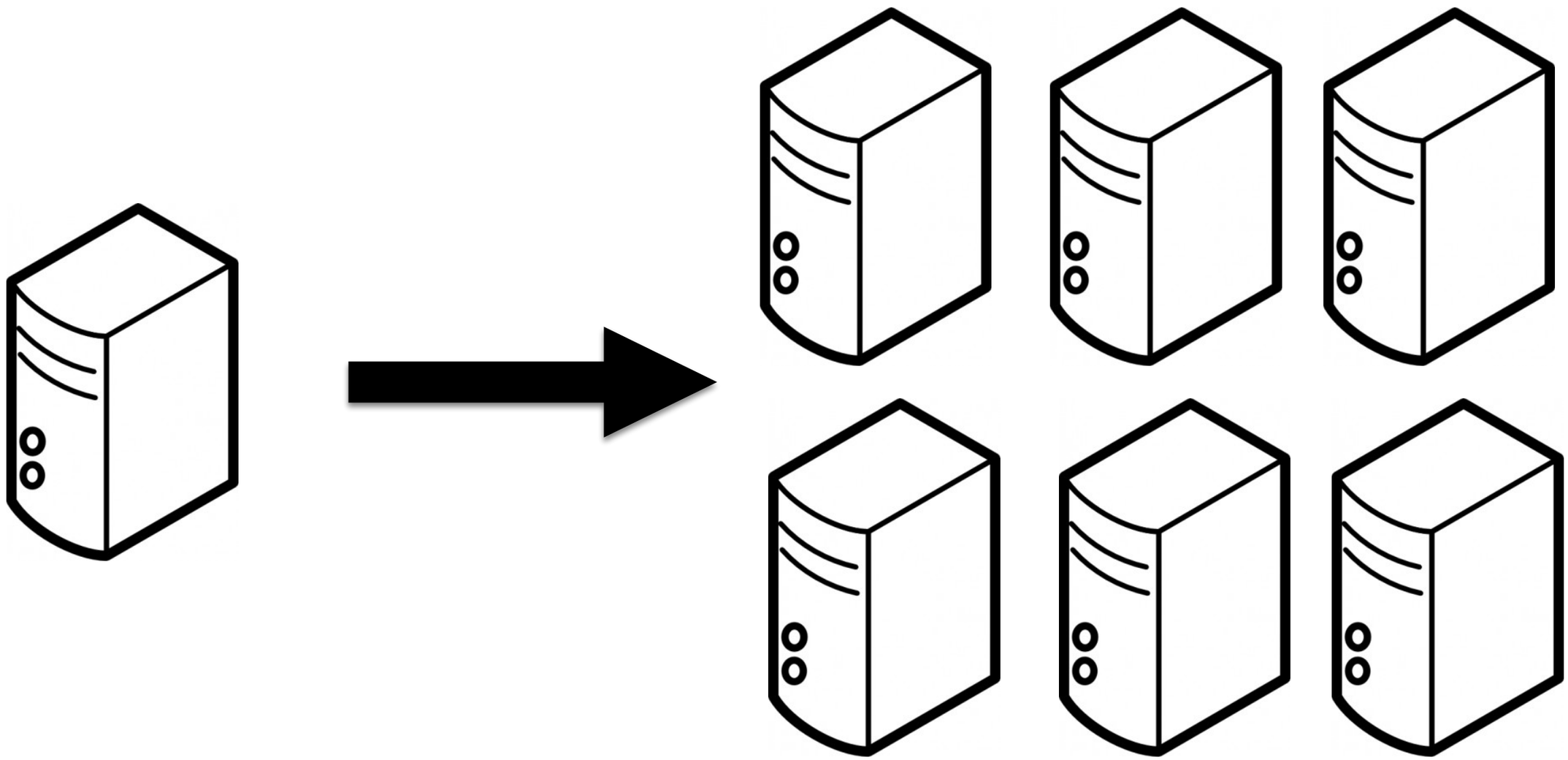
How to scale ?



Scale up (Vertical)

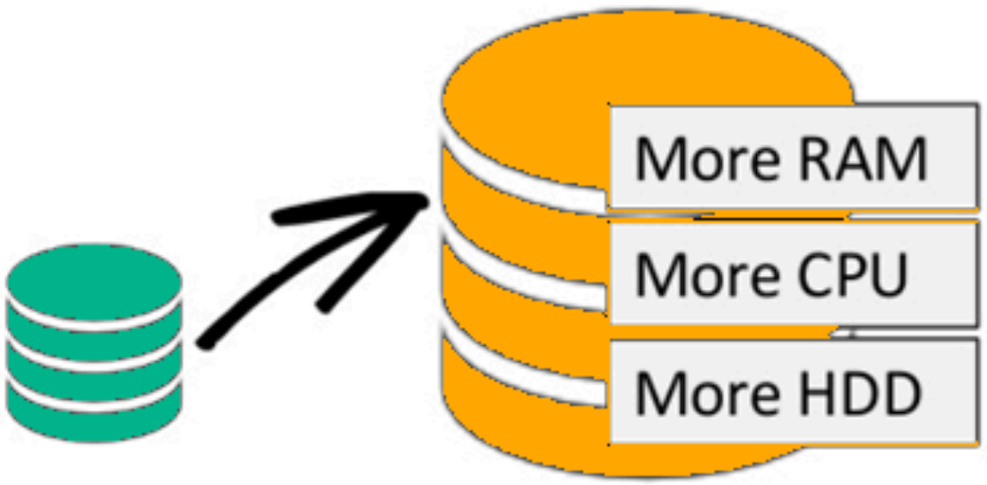


Scale out (Horizontal)

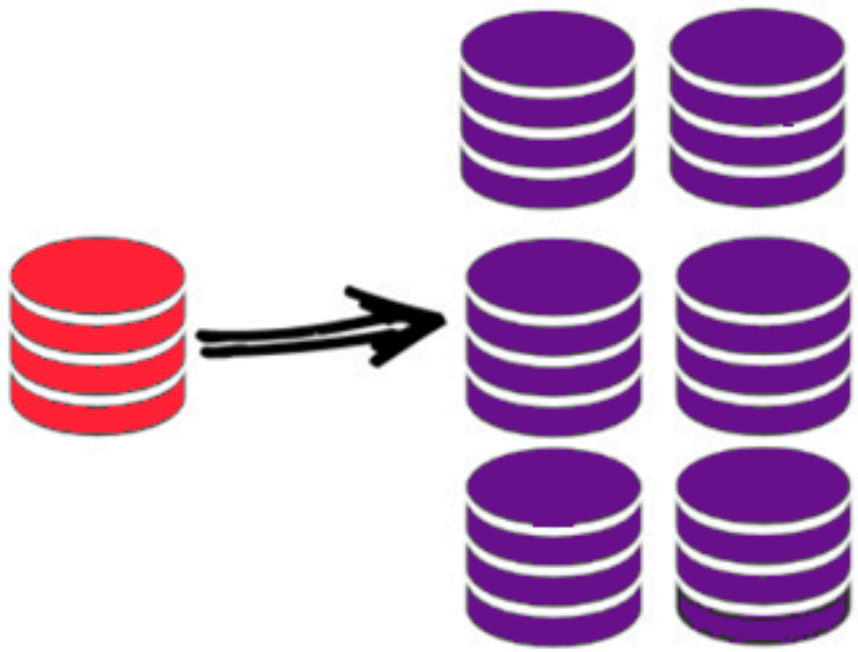


Scale up vs out ?

Scale-Up (*vertical* scaling):



Scale-Out (*horizontal* scaling):

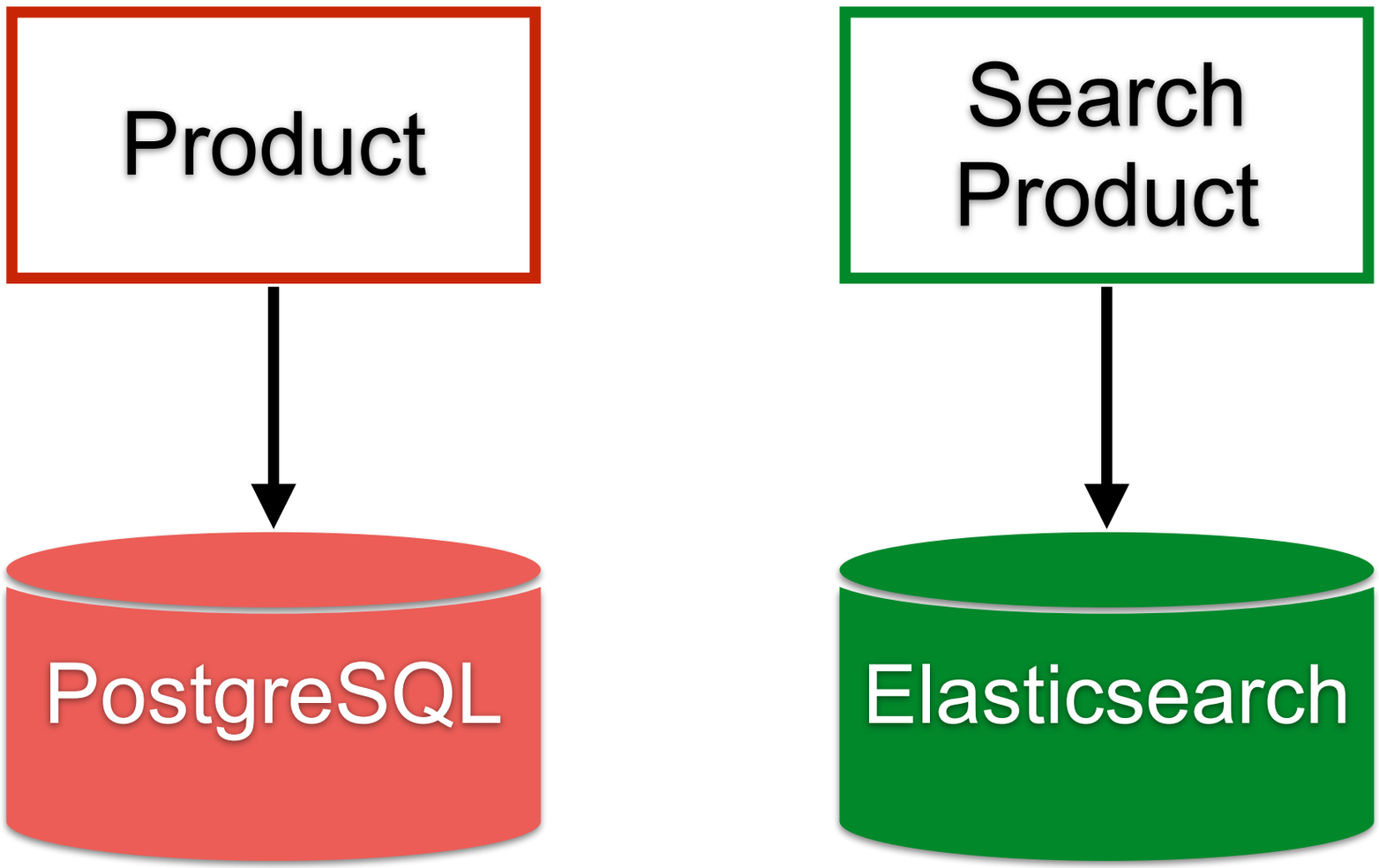


Scale up

Command **Q**uery **R**esponsibility **S**egregation

Design for read
Design for write

Separate DB for read and write



Command Query Separation

Every method should either be a **command**
that performs an action,
or a **query** that returns data to the caller
but not both

Simple service

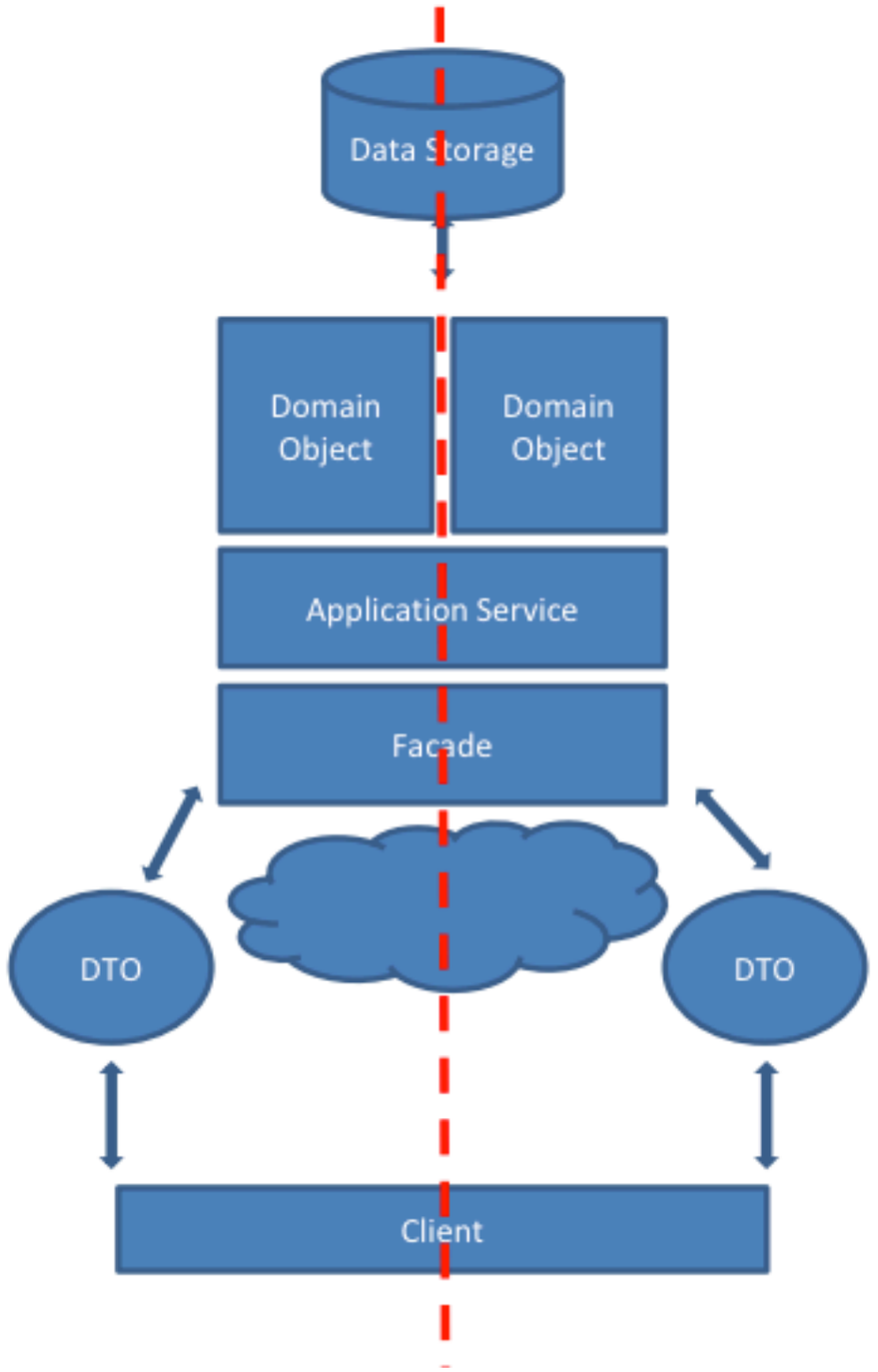
```
BankAccountService
{
   BankAccount CreateBankAccount(BankAccount);
   BankAccount GetBankAccount(AccountId);
   decimal UpdateBalance(AccountId, Balance);
   void UpdateBankAccount(BankAccount);
   decimal GetBankAccountBalance(AccountId);
}
```

CQS service

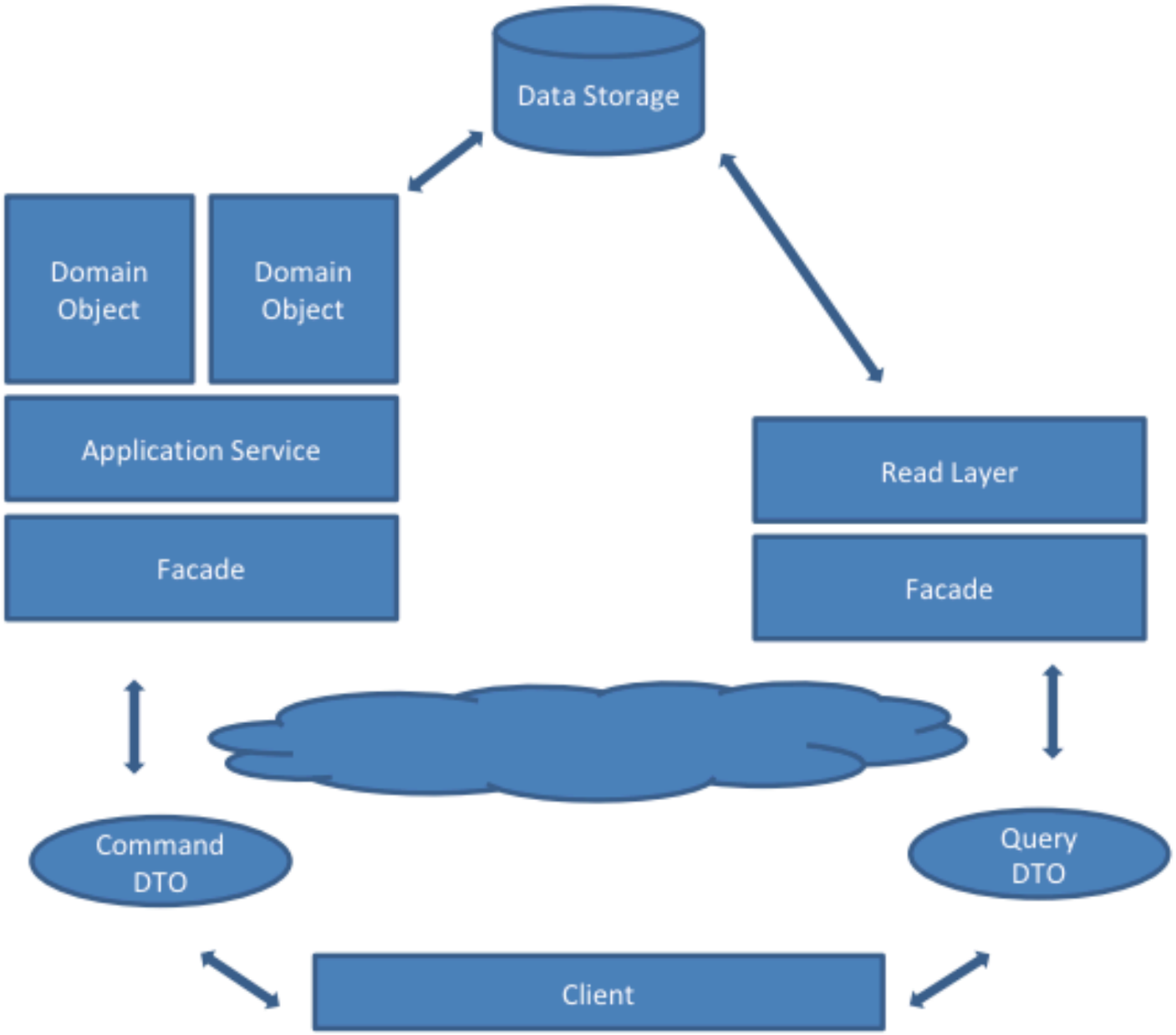
```
BankAccountReadService
{
    BankAccount GetBankAccount(AccountId);
    decimal GetBankAccountBalance(AccountId);
}
```

```
BankAccountWriteService
{
    BankAccount CreateBankAccount(BankAccount);
    decimal UpdateBalance(AccountId, Balance);
    void UpdateBankAccount(BankAccount);
}
```

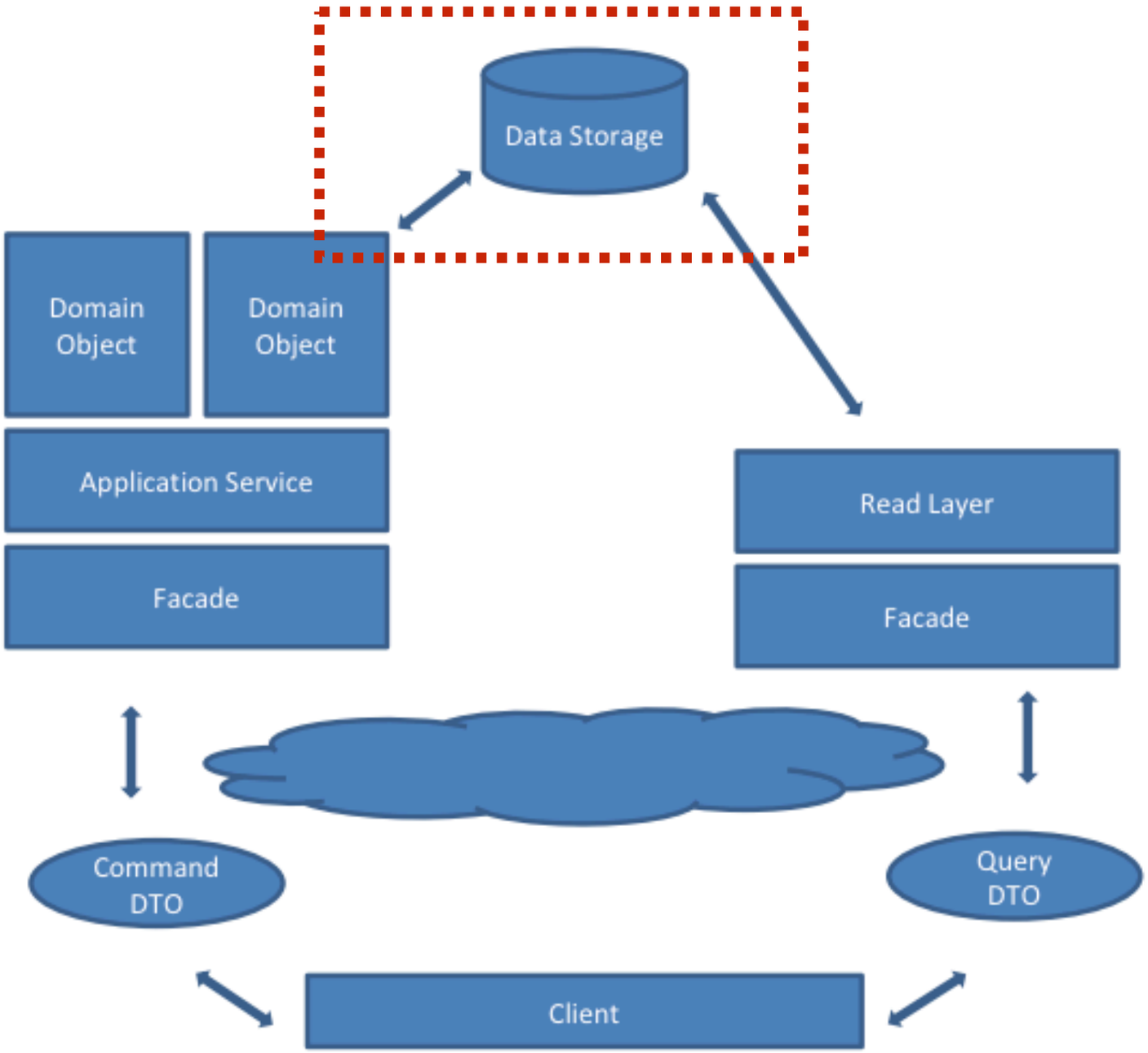
Separate responsibility



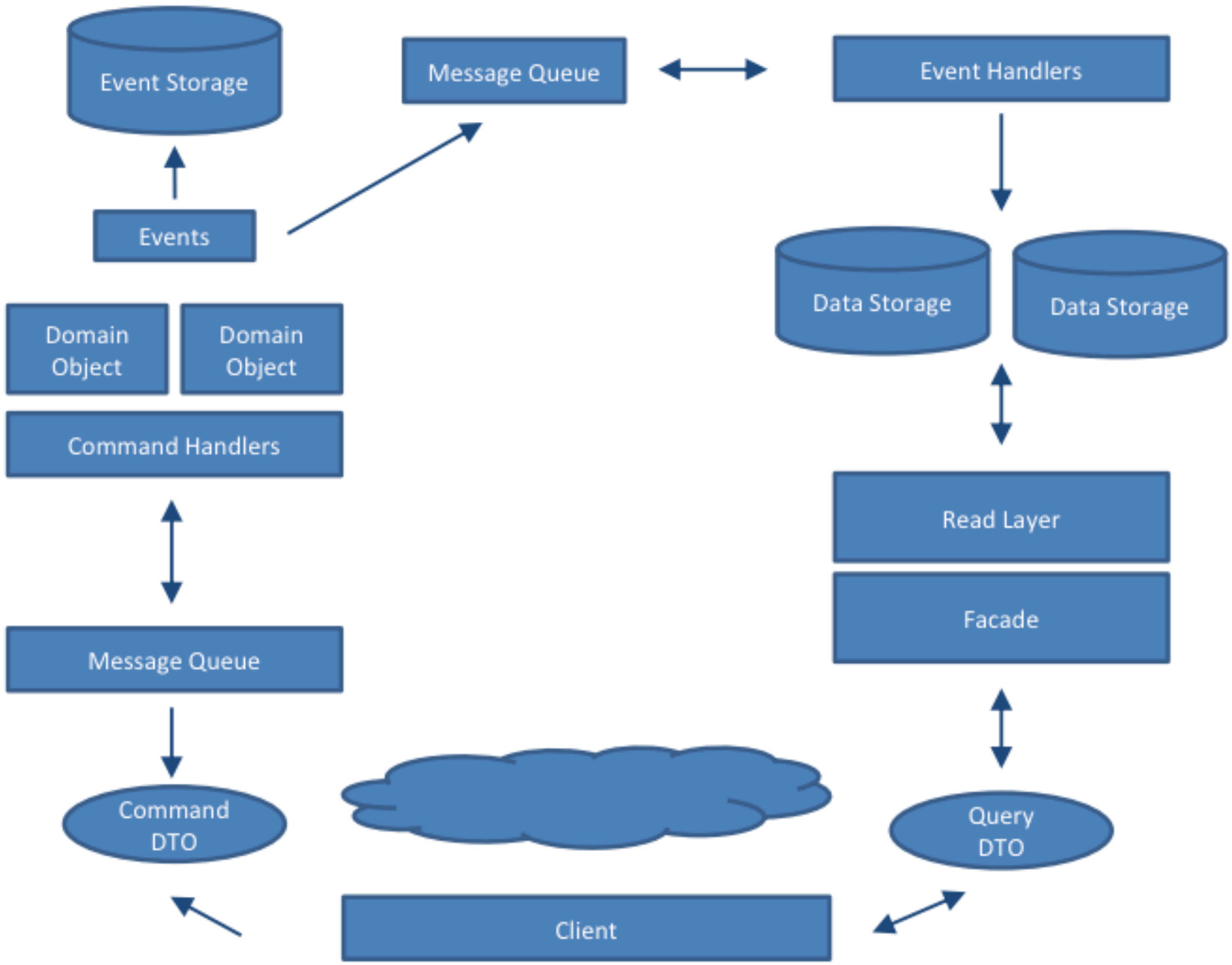
Separate Responsibility



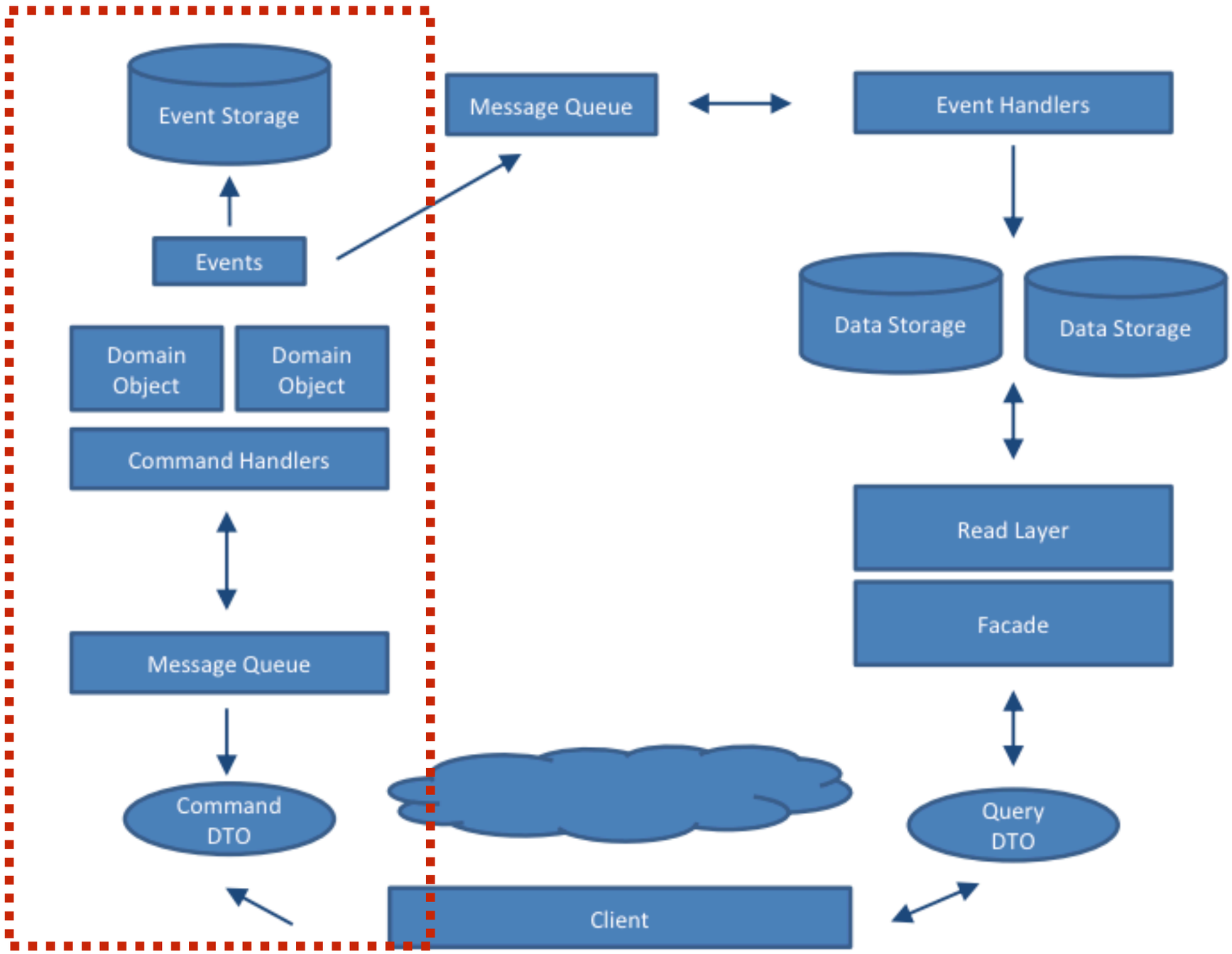
Separate responsibility



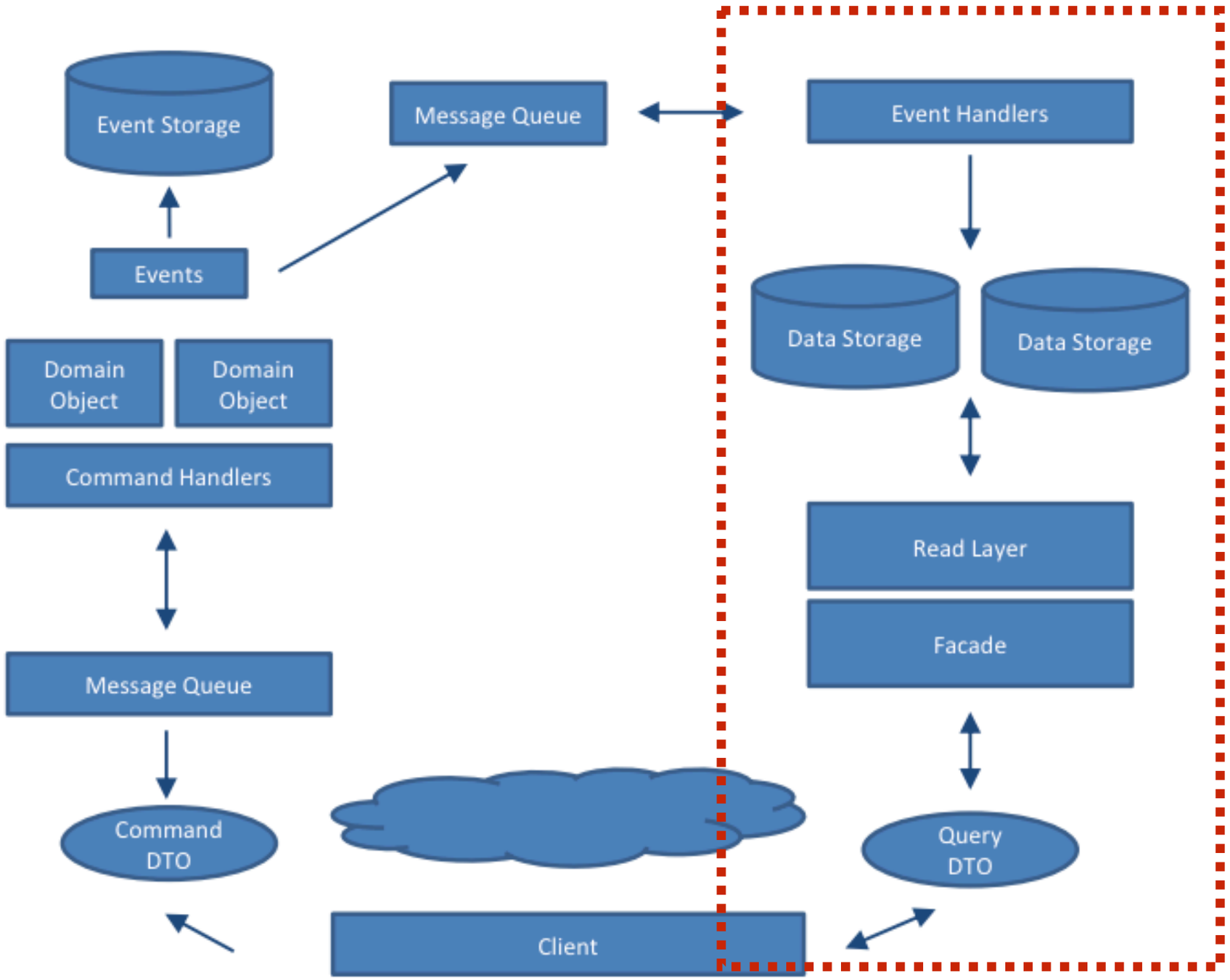
Scalability



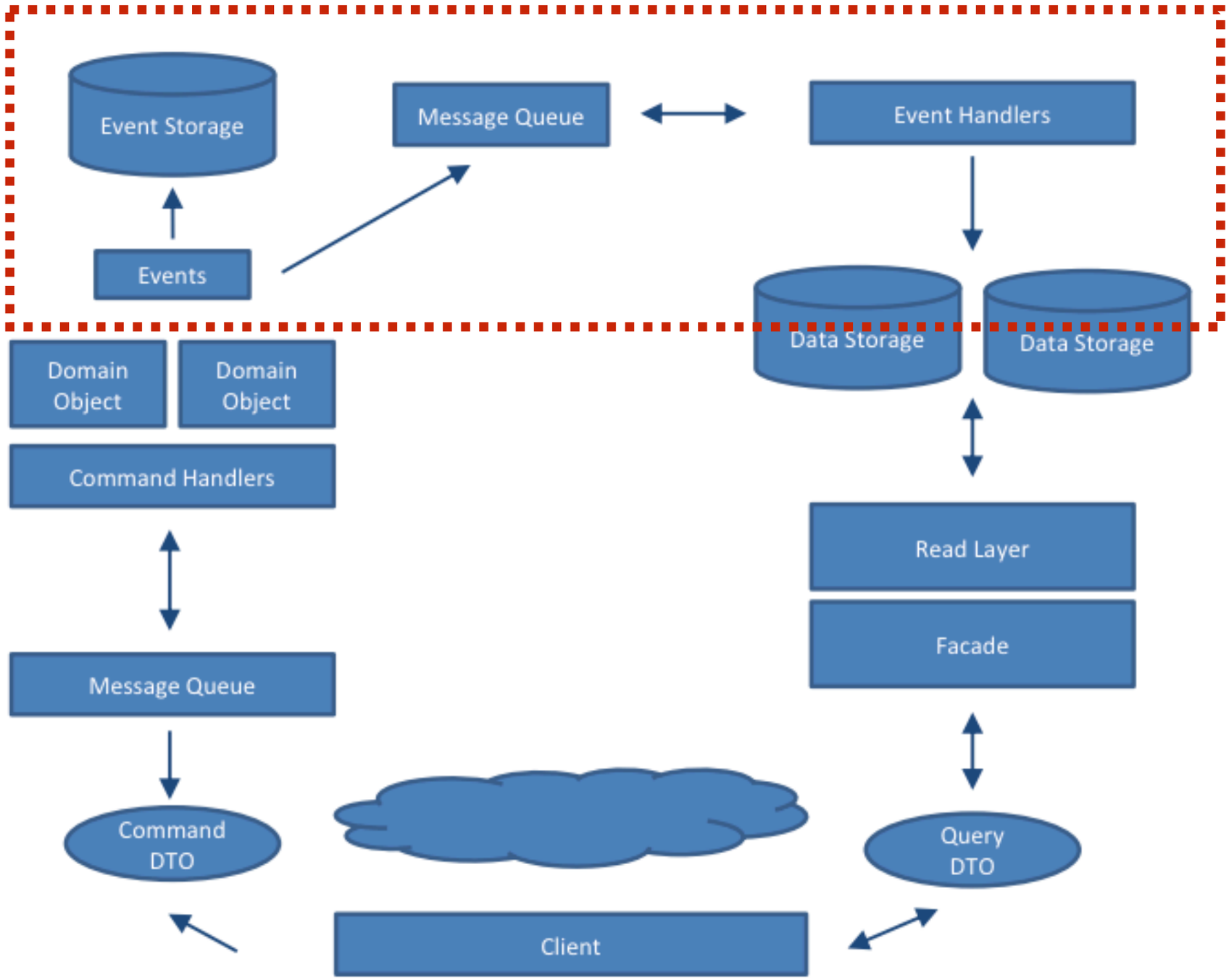
Command



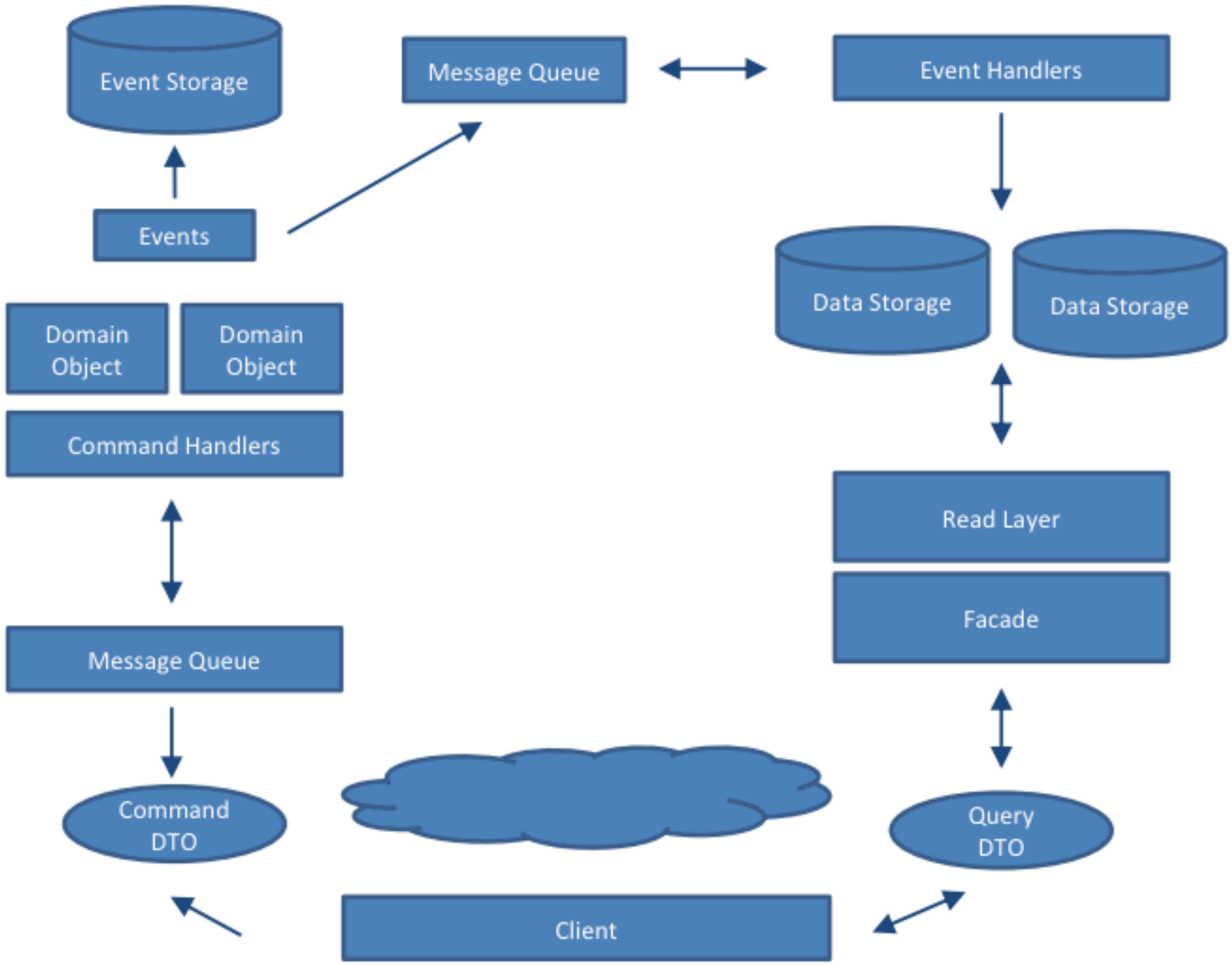
Query



Sync data



CQRS



CQRS

Command **Q**uery **R**esponsibility **S**egregation

Scale with PostgreSQL

Partitioning
Replication
Clustering

Partitioning

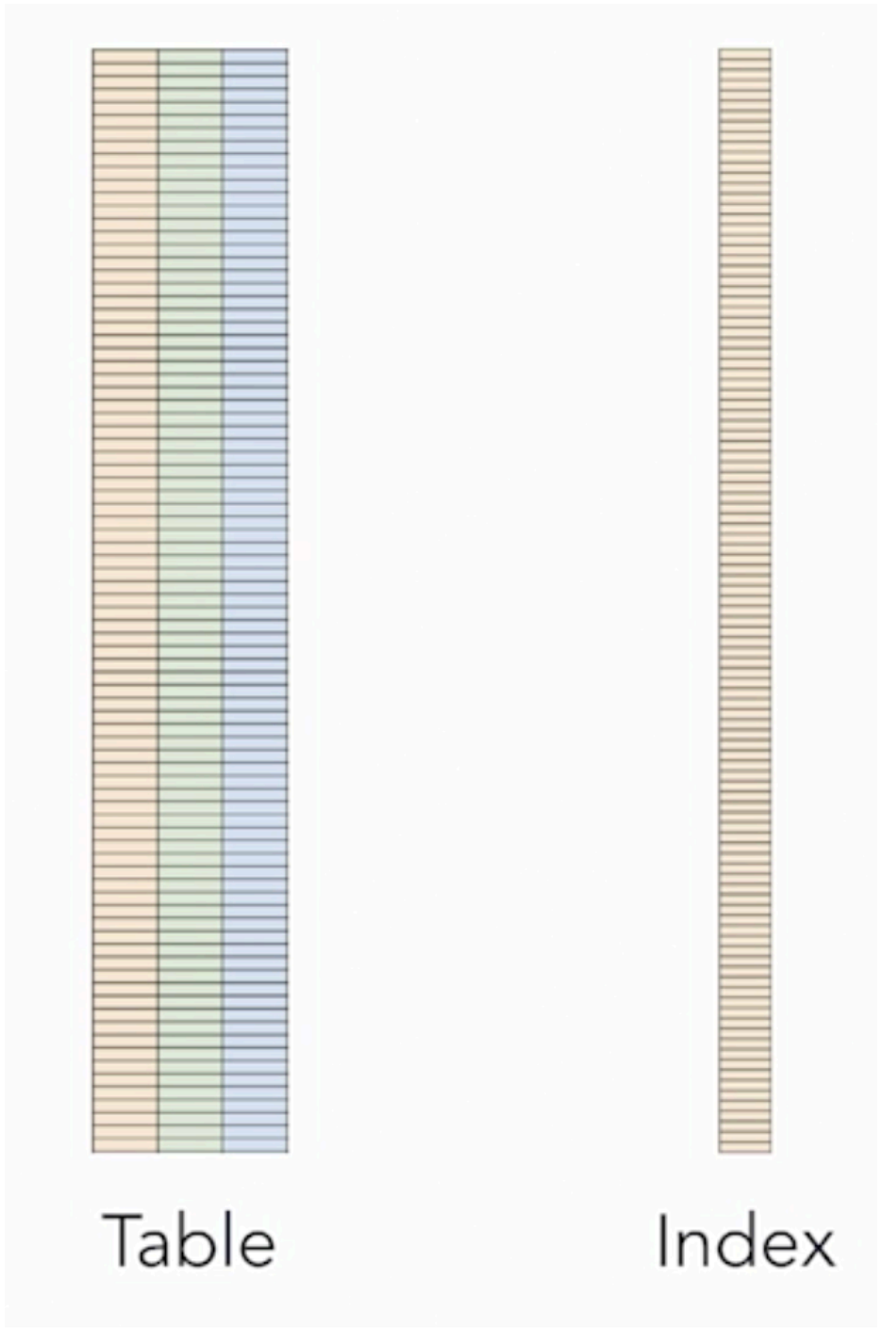
Storing table data in multiple sub-tables
Called “**Partition**”

Partitioning

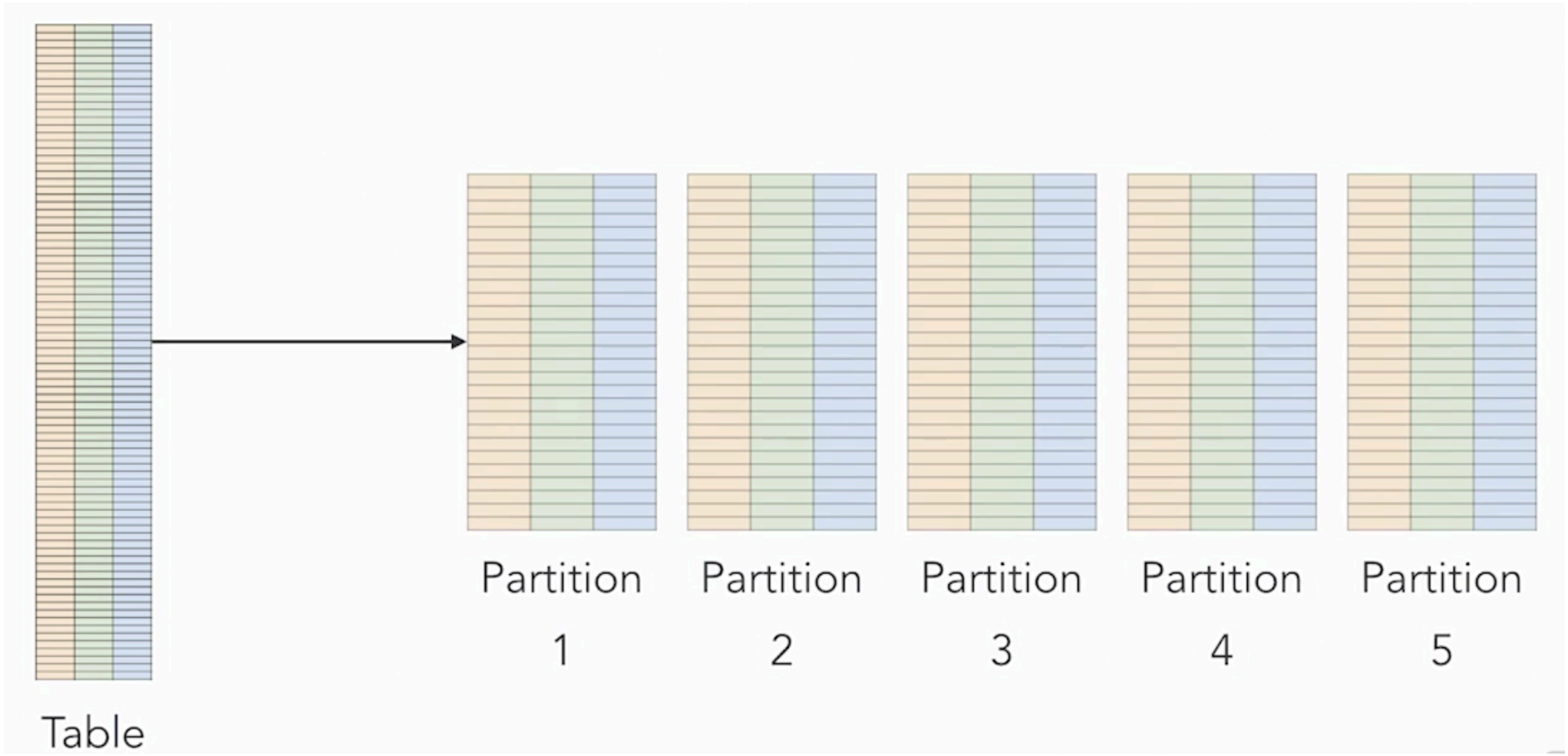
Used to improve query, load and delete operation

Used for **large tables**

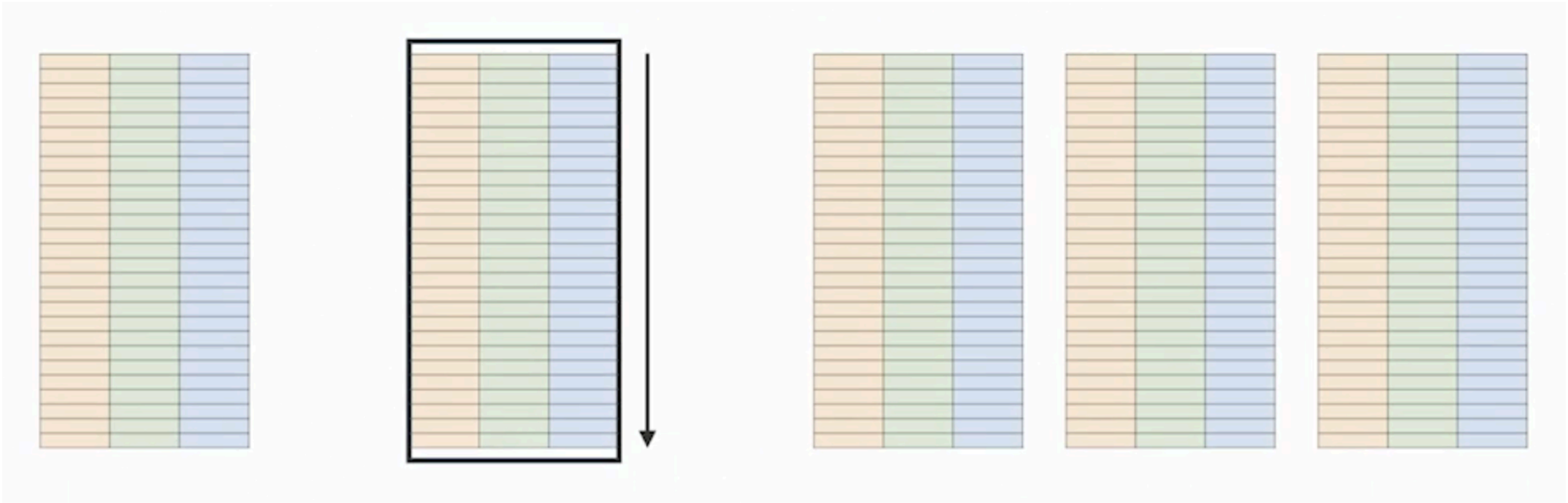
Large table == Large index



Partition data



Faster scan

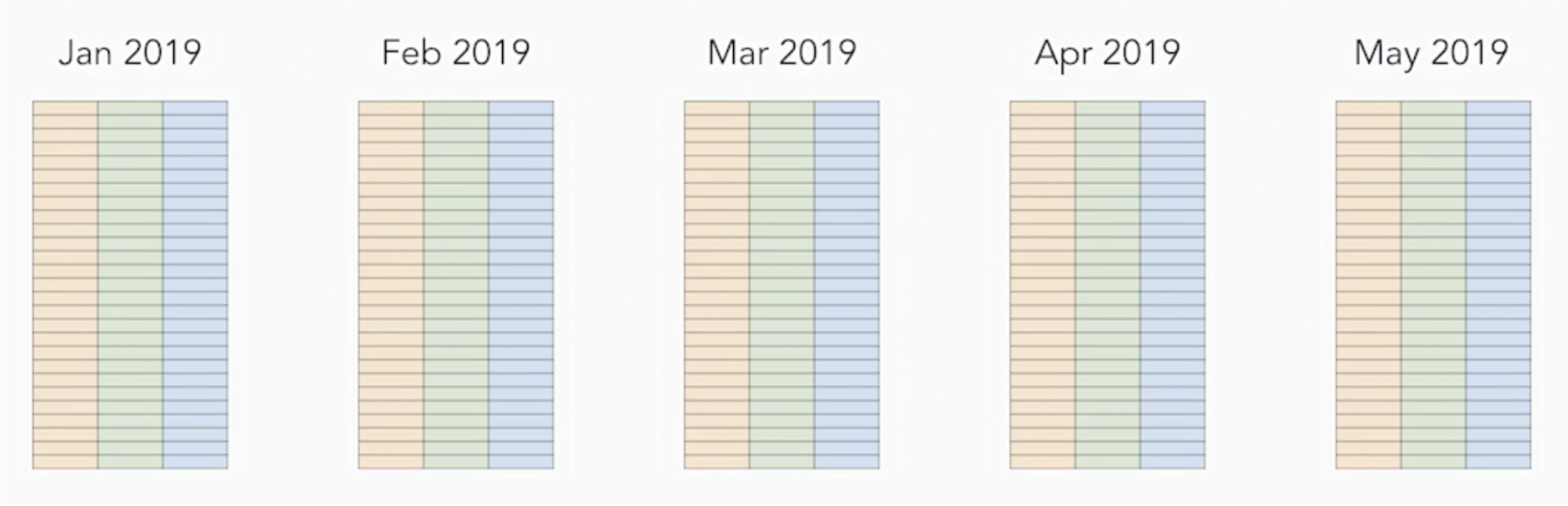


Partitioning

Range of values
List of values
Hash of values

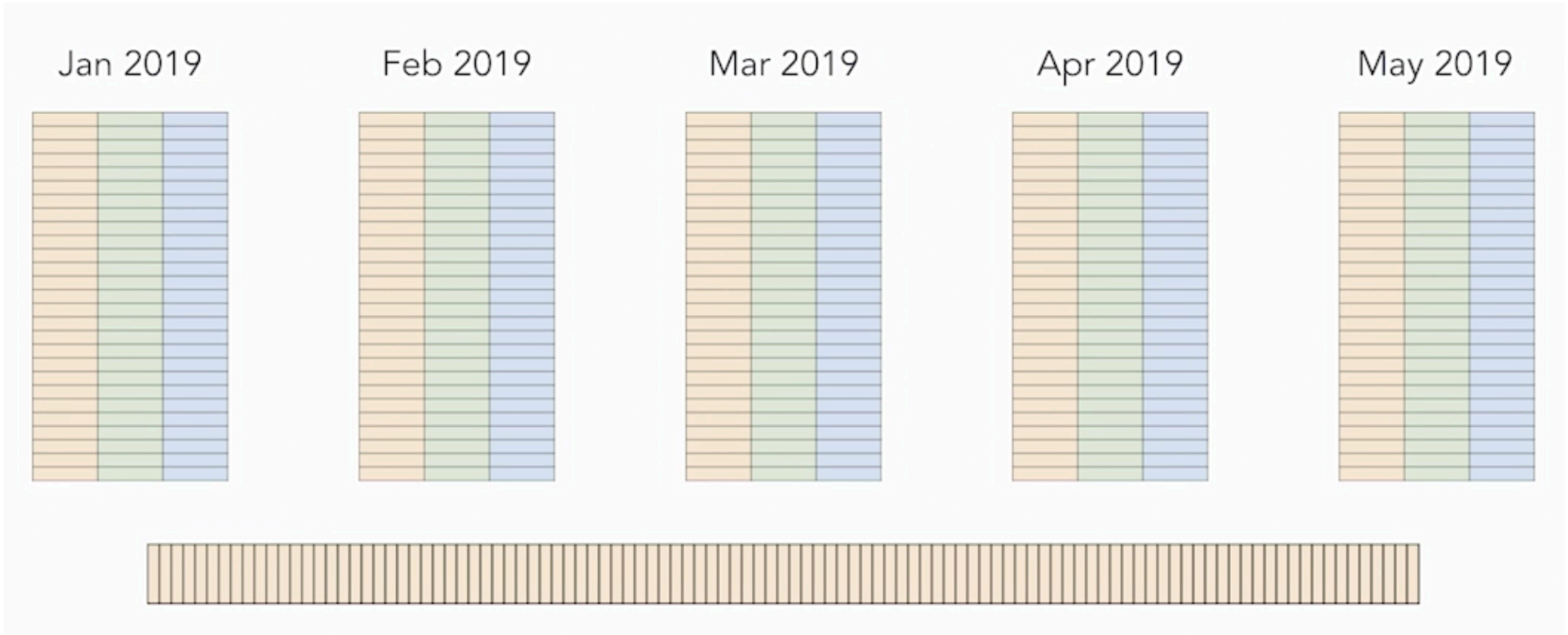
https://github.com/up1/course-sql/blob/master/workshop-postgresql/partition.md

Partition by range

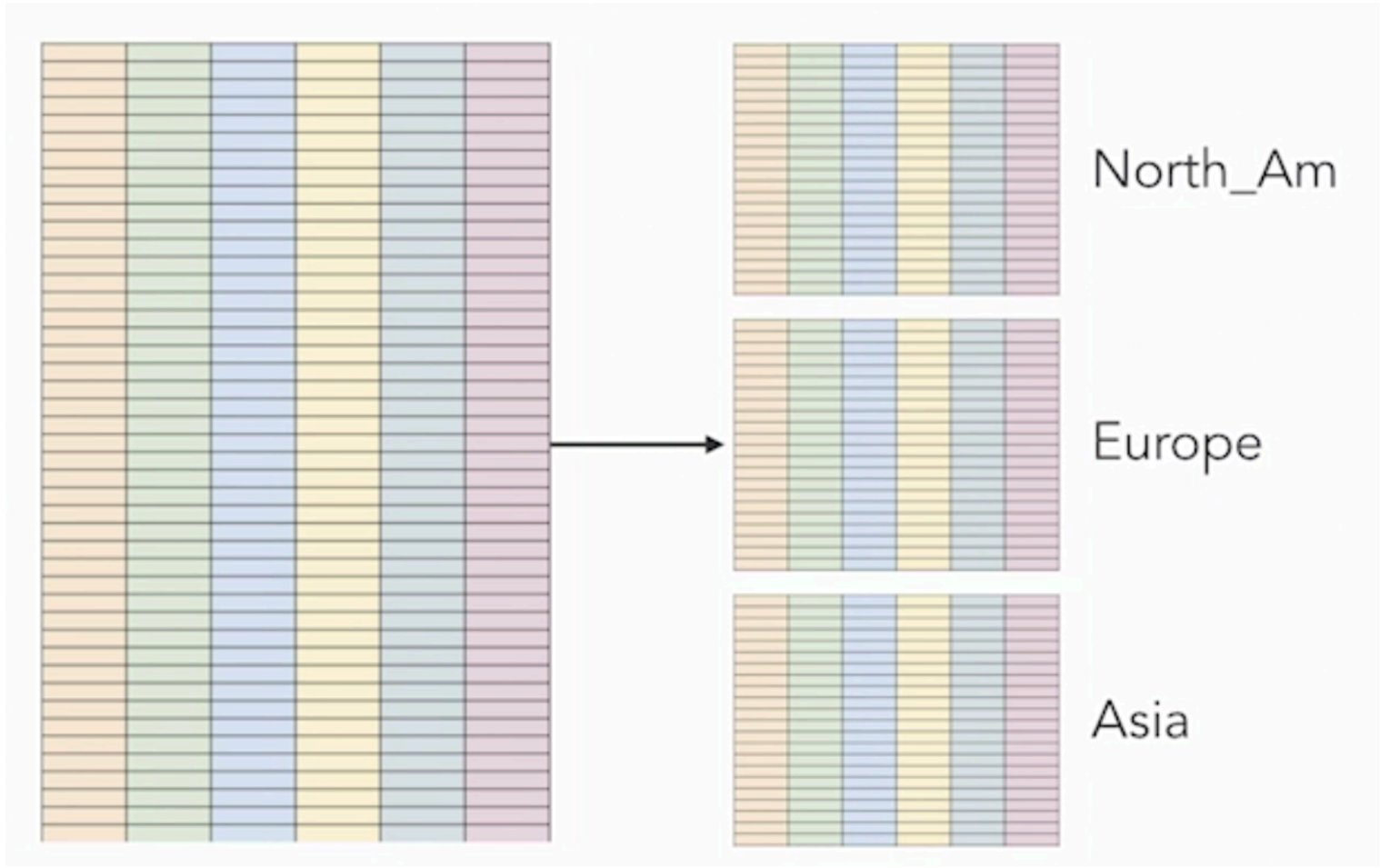


Local index in each partition

Global index for all partitions



Partition by list of values



Example

Partition by list of product catalog

```
CREATE TABLE products
    (prod_id    int not null,
     prod_name text not null,
     prod_short_descr text not null,
     prod_long_descr text not null,
     prod_category varchar)
PARTITION BY LIST (prod_category);
```

Example

Partition by list of product catalog

```
CREATE TABLE products
    (prod_id    int not null,
     prod_name text not null,
     prod_short_descr text not null,
     prod_long_descr text not null,
     prod_category varchar)
PARTITION BY LIST (prod_category);
```

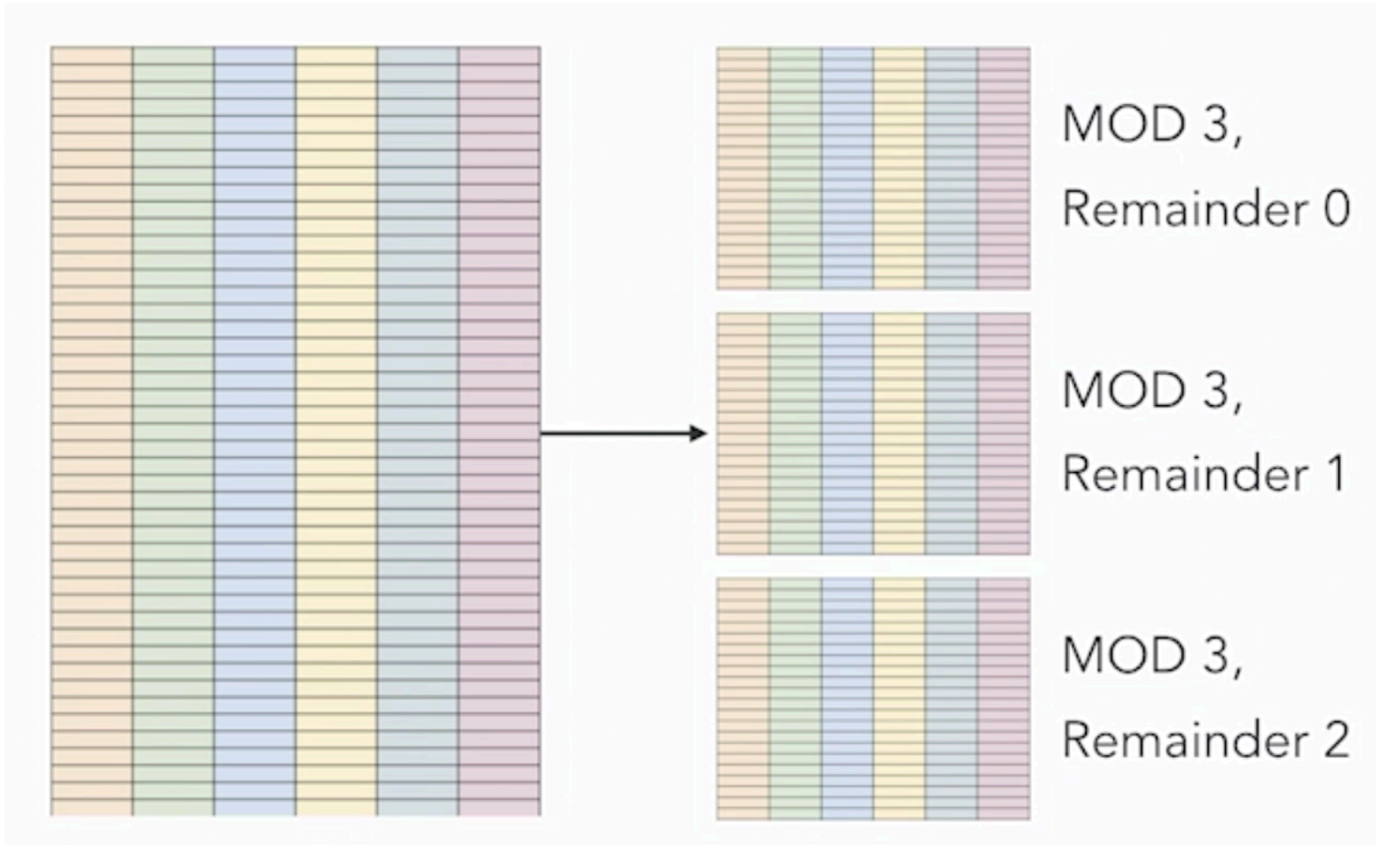
```
CREATE TABLE product_clothing PARTITION OF products
 FOR VALUES IN ('casual_clothing', 'business_attire', 'formal_clothing');

CREATE TABLE product_electronics PARTITION OF products
  FOR VALUES IN ('mobile_phones', 'tablets', 'laptop_computers');

CREATE TABLE product_kitchen PARTITION OF products
  FOR VALUES IN ('food_processor', 'cutlery', 'blenders');
```

Partition by hash

Partition on modulus of hash of partition keys



Example

Hash function on ci_id

```
CREATE TABLE customer_interaction
    (ci_id int not null,
     ci_url text not null,
     time_at_url int not null,
     click_sequence int not null)
 PARTITION BY HASH (ci_id);
```

Example

Hash function on ci_id

```
CREATE TABLE customer_interaction
   (ci_id int not null,
    ci_url text not null,
    time_at_url int not null,
    click_sequence int not null)
PARTITION BY HASH (ci_id);
```

```
CREATE TABLE customer_interaction_1 PARTITION OF customer_interaction
  FOR VALUES WITH (MODULUS 5, REMAINDER 0);
CREATE TABLE customer_interaction_2 PARTITION OF customer_interaction
  FOR VALUES WITH (MODULUS 5, REMAINDER 1);
CREATE TABLE customer_interaction_3 PARTITION OF customer_interaction
  FOR VALUES WITH (MODULUS 5, REMAINDER 2);
CREATE TABLE customer_interaction_4 PARTITION OF customer_interaction
  FOR VALUES WITH (MODULUS 5, REMAINDER 3);
CREATE TABLE customer_interaction_5 PARTITION OF customer_interaction
  FOR VALUES WITH (MODULUS 5, REMAINDER 4);
```

Scale with PostgreSQL

Partitioning

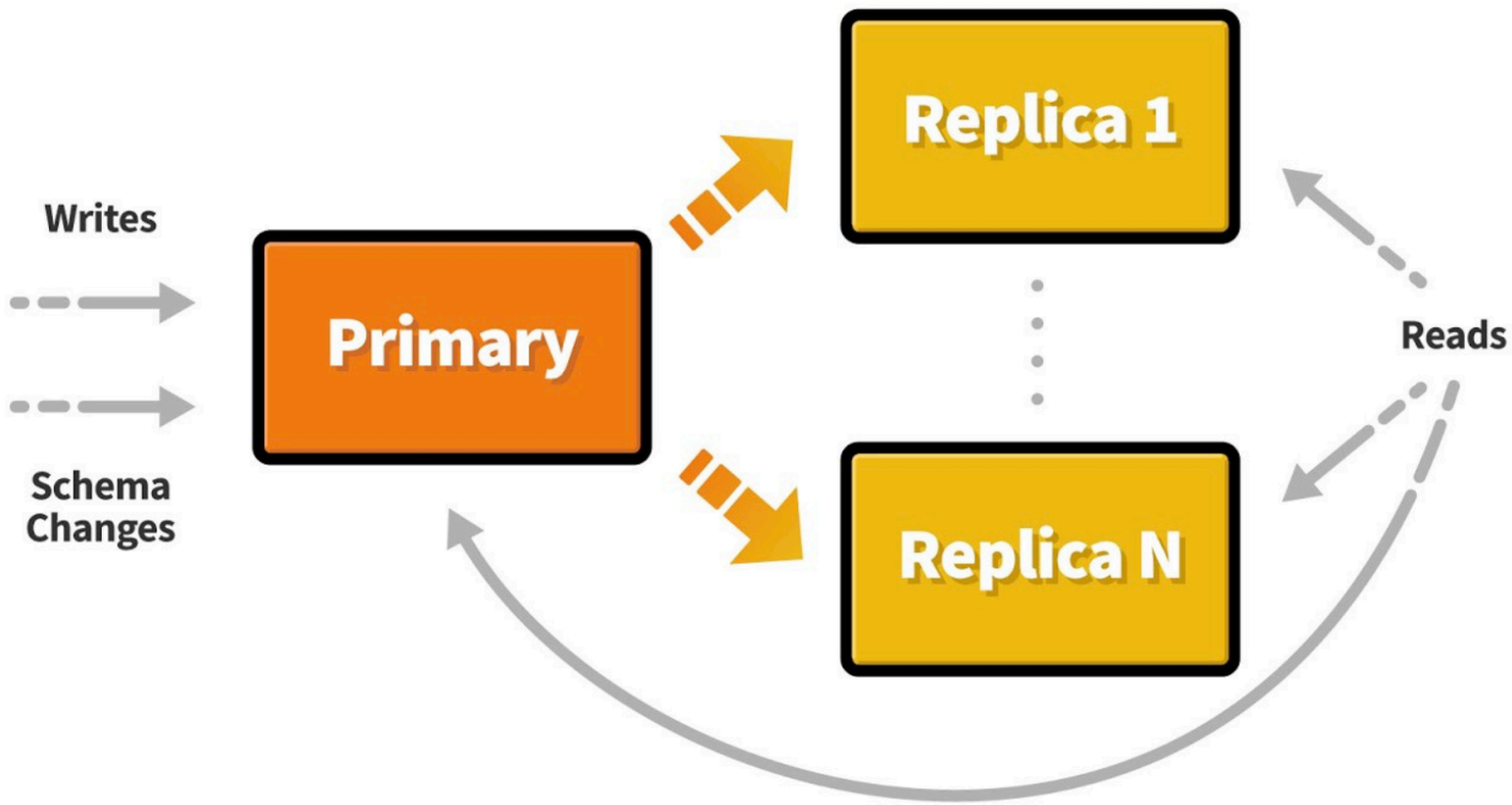
Replication

Clustering

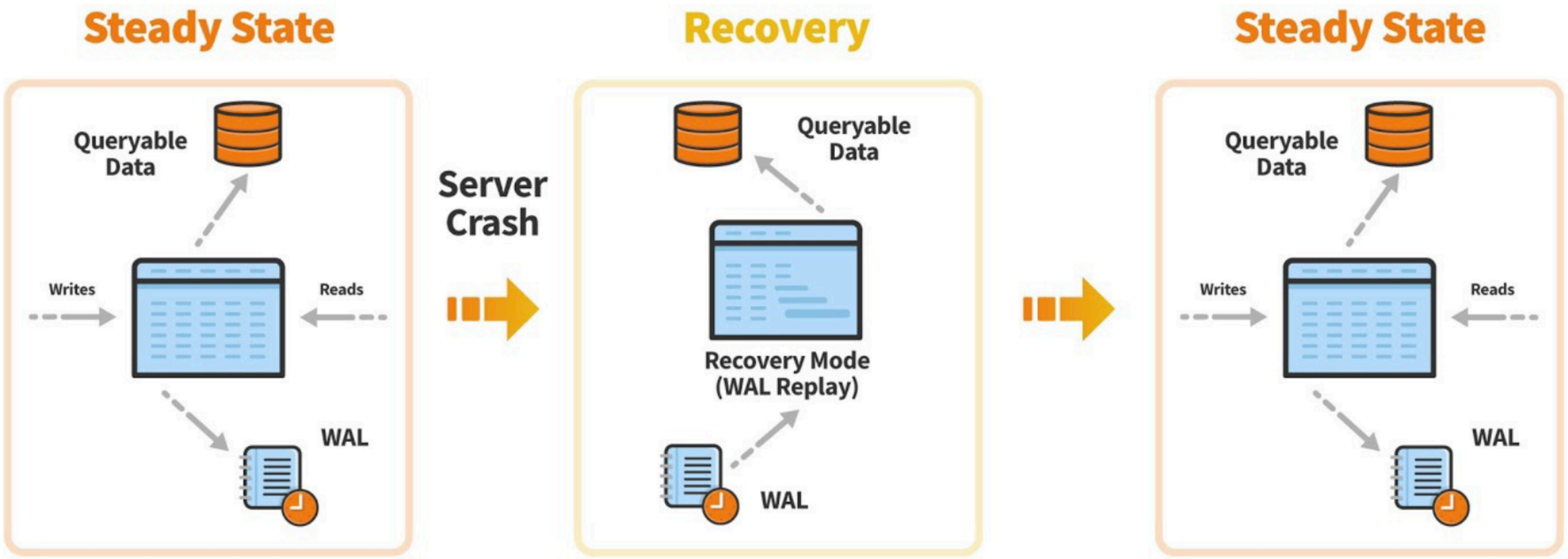
Replication

How an (N+1) node PostgreSQL cluster works ?

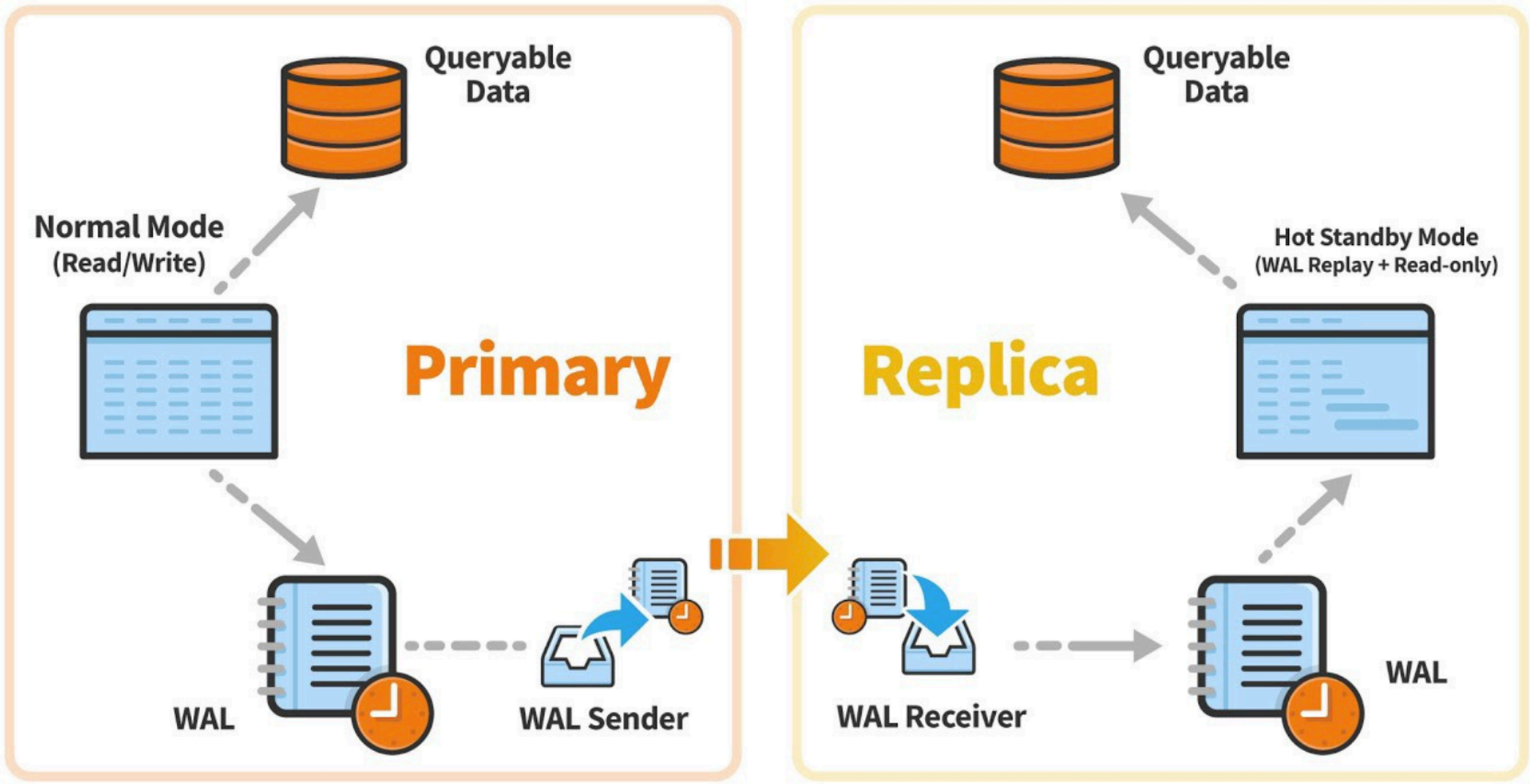
Using streaming replication



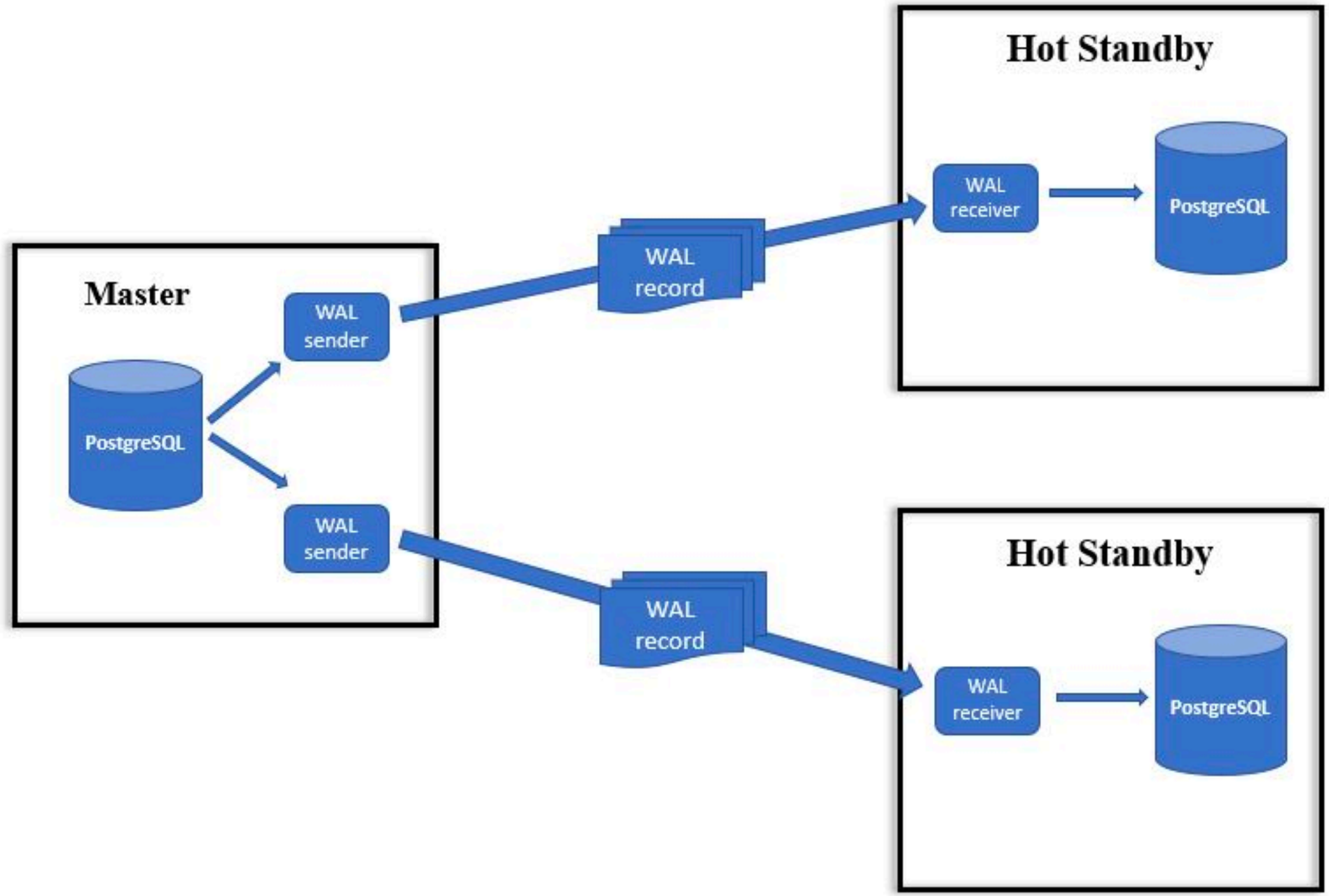
WAL (Write Ahead Log)



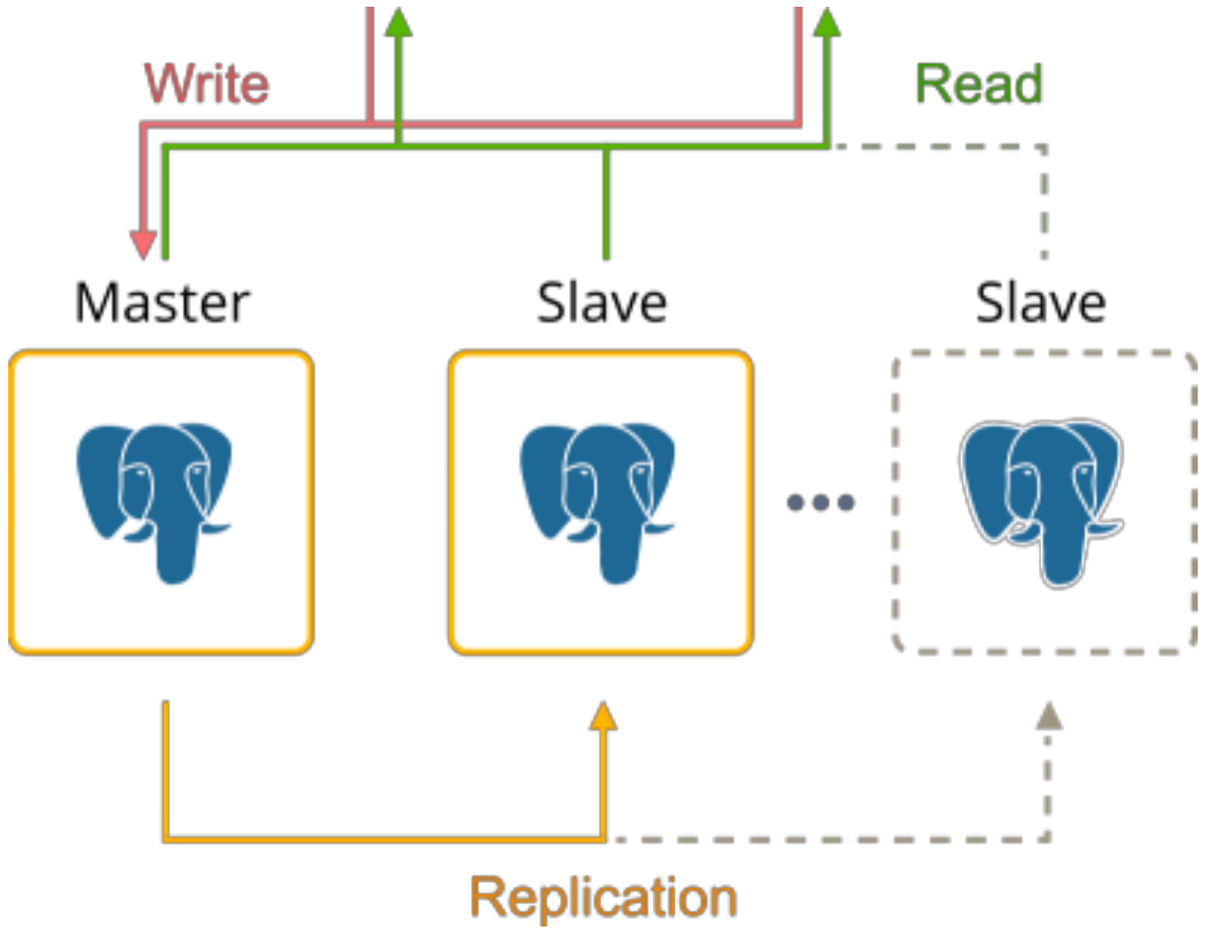
WAL and Replication



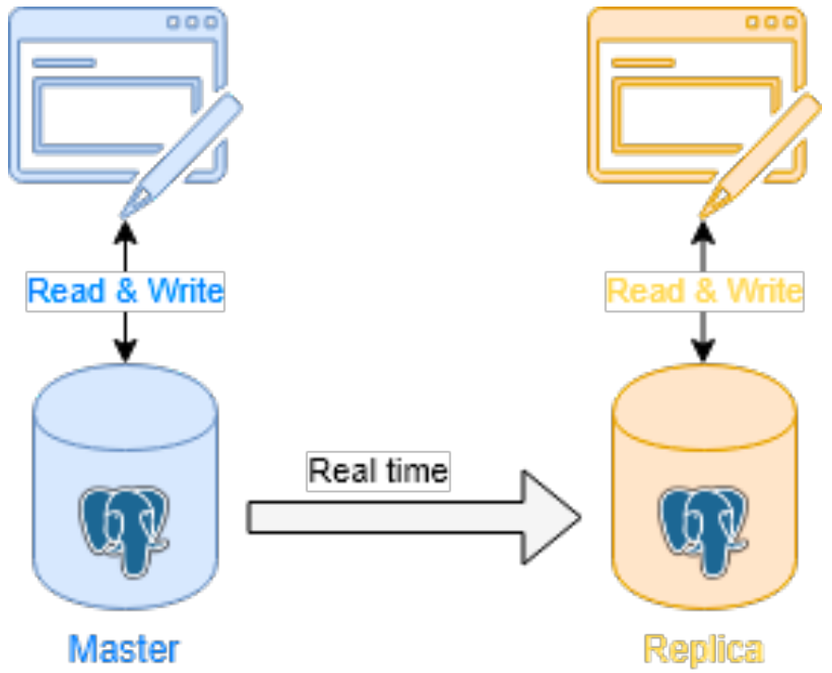
Master(primary) - Slave (standby)



Master - Slave



Logical replication

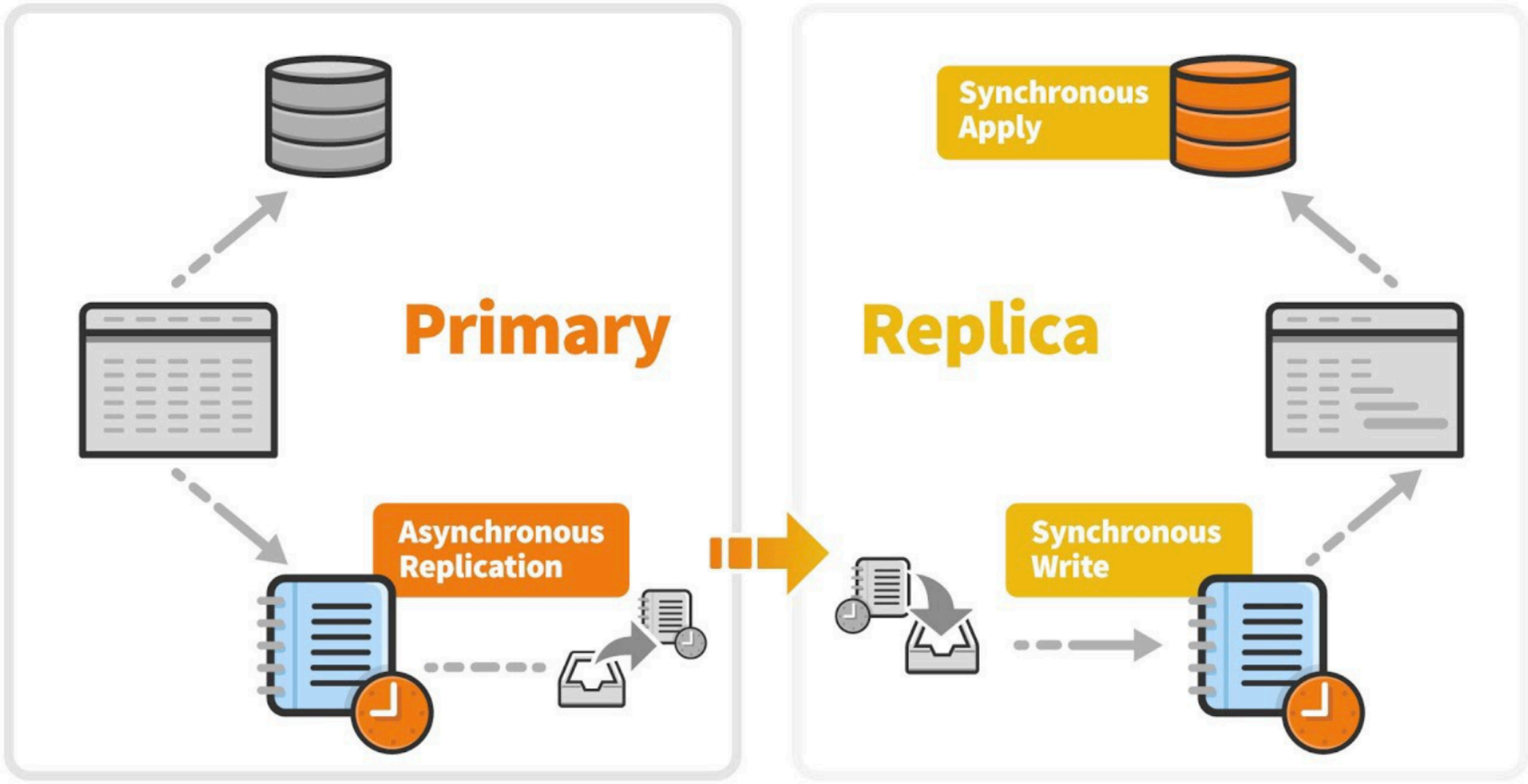


https://www.postgresql.org/docs/current/logical-replication.html

Replication modes

Asynchronous replication
Synchronous write replication
Synchronous apply replication

Replication modes



Performance

Replication mode	Write performance	Read consistency	Data loss
Async replication	Highest	Weakest, eventually consistent	Highest risk
Sync write replication	Medium	eventually consistent BUT less lag	Low risk
Sync apply replication	Lowest	Strong consistency	Lowest risk

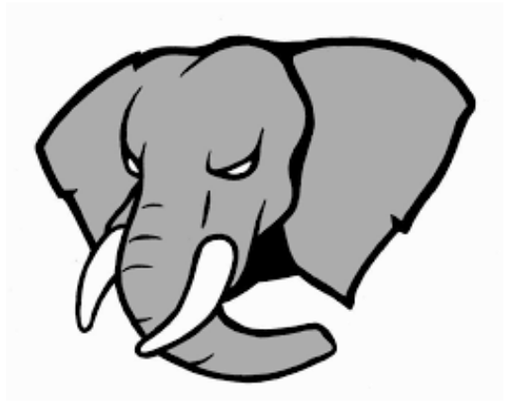
Replication solutions

Slony-I

Pg-pool-II

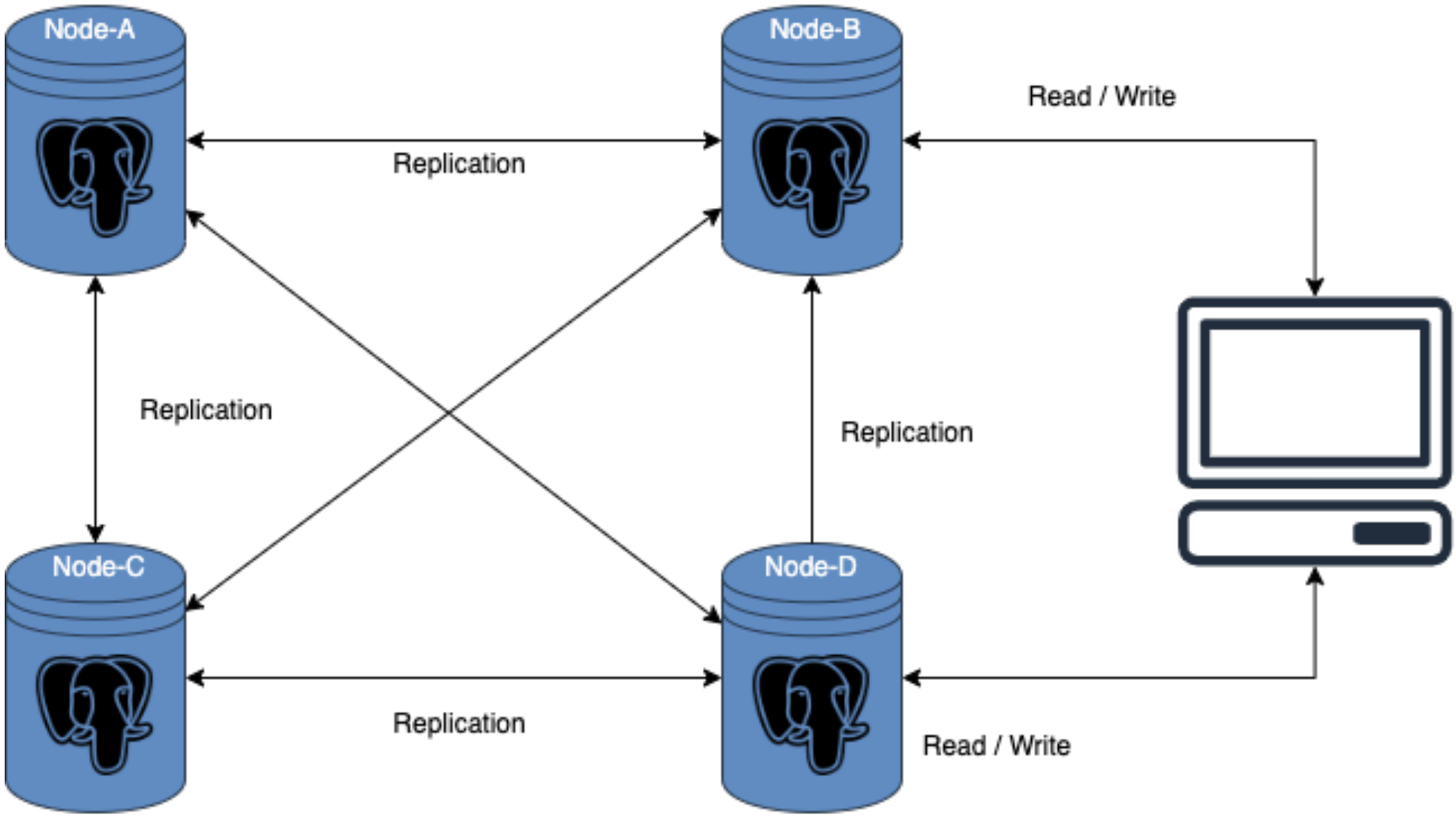
Bucardo

Londiste



https://www.percona.com/blog/2020/06/09/multi-master-replication-solutions-for-postgresql/

Multi-master



Scale with PostgreSQL

Partitioning
Replication
Clustering

Clustering