BITS Pilani
Pilani Campus

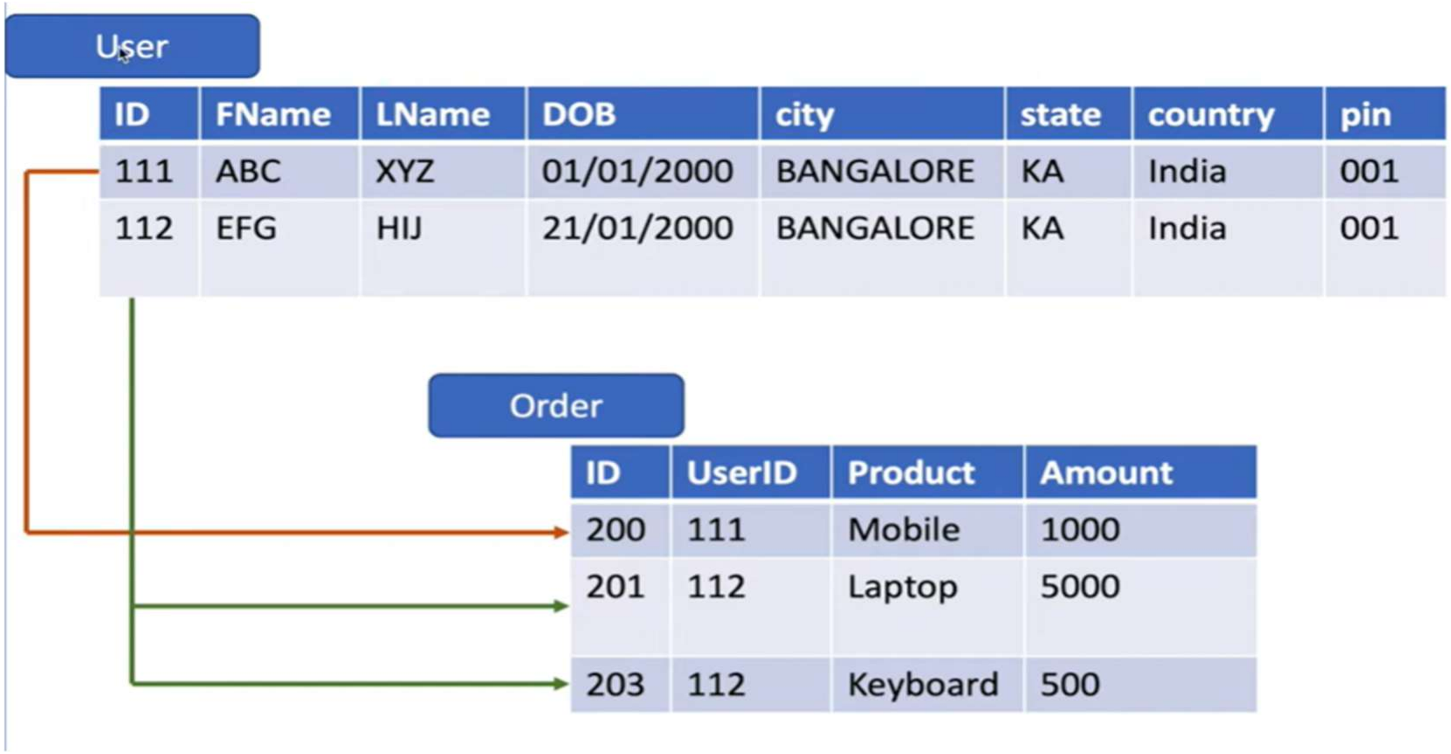
BITS Pilani presentation

Dr. Vivek V. Jog
Dept. of Computer Engineering

BITS Pilani
Pilani Campus

Big Data Systems (S1-24_CCZG522)
Lecture No.11

Traditional RDBMS



User

ID	FName	LName	DOB	city	state	country	pin
111	ABC	XYZ	01/01/2000	BANGALORE	KA	India	001
112	EFG	HIJ	21/01/2000	BANGALORE	KA	India	001

Order

ID	UserID	Product	Amount
200	111	Mobile	1000
201	112	Laptop	5000
203	112	Keyboard	500

SQL Joins

SQL JOIN operations are used to combine rows from two or more tables, and are of two types:

•Conditional Join: Combine rows based on a condition over one or more common columns (typically primary or foreign keys).

•There are 4 types of conditional joins: inner, left, right, and full.

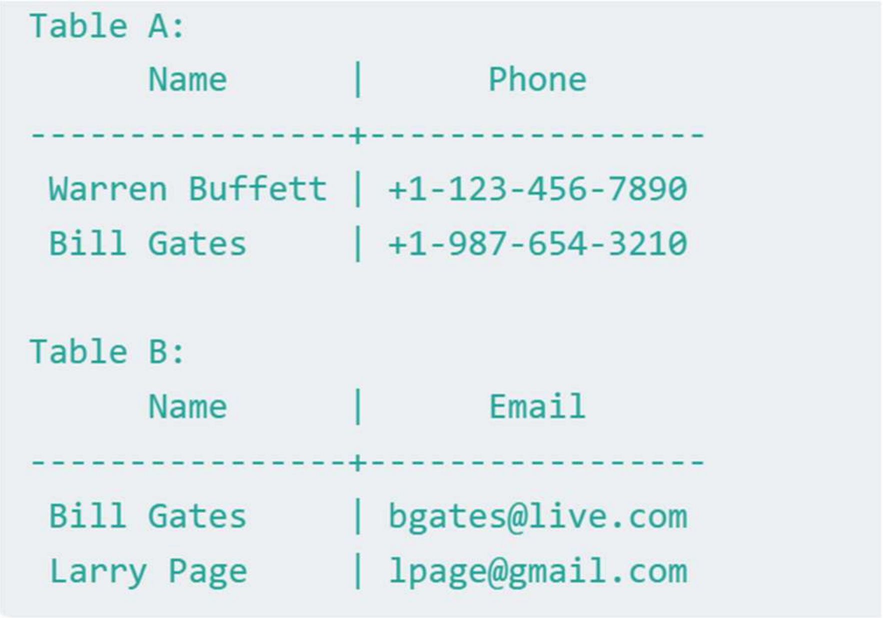
•Cross Join: Cartesian Product of two tables.

Inner Join

An Inner Join produces a row combining a row from both tables only if the key is present in *both* tables.

For example, if you had run two marketing campaigns on different channels that captured potential customer info in two tables:

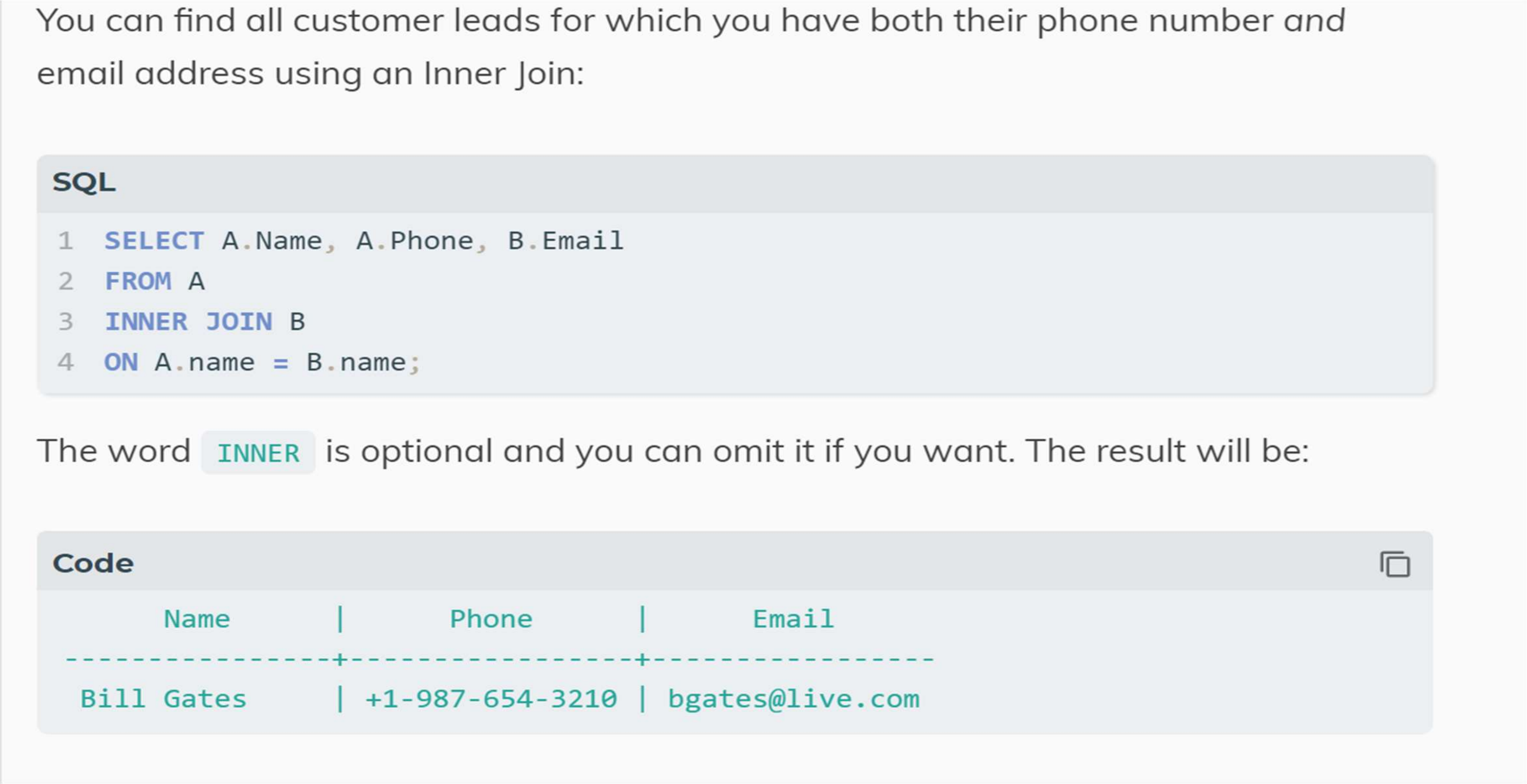
- Table A: Name and phone number
- Table B: Name and email



```
Table A:
      Name      |       Phone
----------------+-----------------
 Warren Buffett | +1-123-456-7890
 Bill Gates     | +1-987-654-3210

Table B:
      Name      |       Email
----------------+-----------------
 Bill Gates     | bgates@live.com
 Larry Page     | lpage@gmail.com
```

Inner Join Cont....



You can find all customer leads for which you have both their phone number *and* email address using an Inner Join:

SQL

```sql
SELECT A.Name, A.Phone, B.Email
FROM A
INNER JOIN B
ON A.name = B.name;
```

The word `INNER` is optional and you can omit it if you want. The result will be:

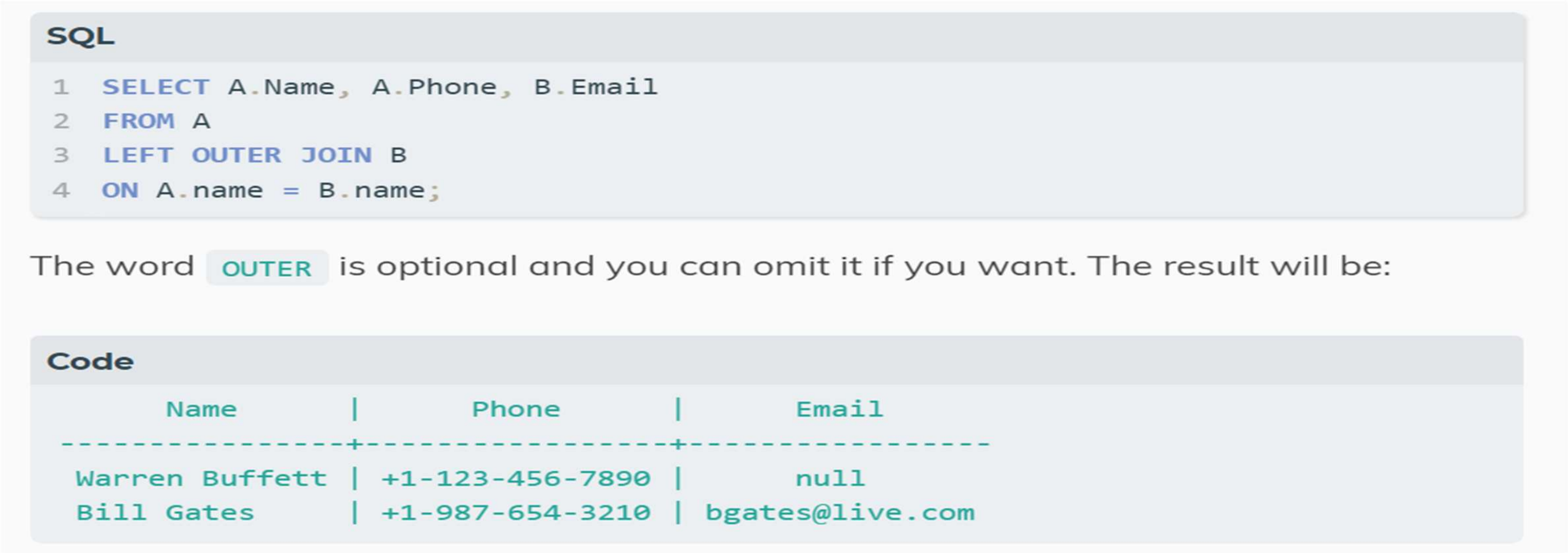
Code

```
      Name      |      Phone      |      Email
----------------+-----------------+------------------
 Bill Gates     | +1-987-654-3210 | bgates@live.com
```

Left Outer Join

A Left Join returns all rows from the left table and the matched rows from the right table. Say, your sales team decides to reach out to all customer leads over the phone, but, if possible, wants to follow up by email too.

With a Left Join, you can create a list of customers where you have their phone numbers for sure but may also have their email:



SQL

```sql
SELECT A.Name, A.Phone, B.Email
FROM A
LEFT OUTER JOIN B
ON A.name = B.name;
```

The word `OUTER` is optional and you can omit it if you want. The result will be:

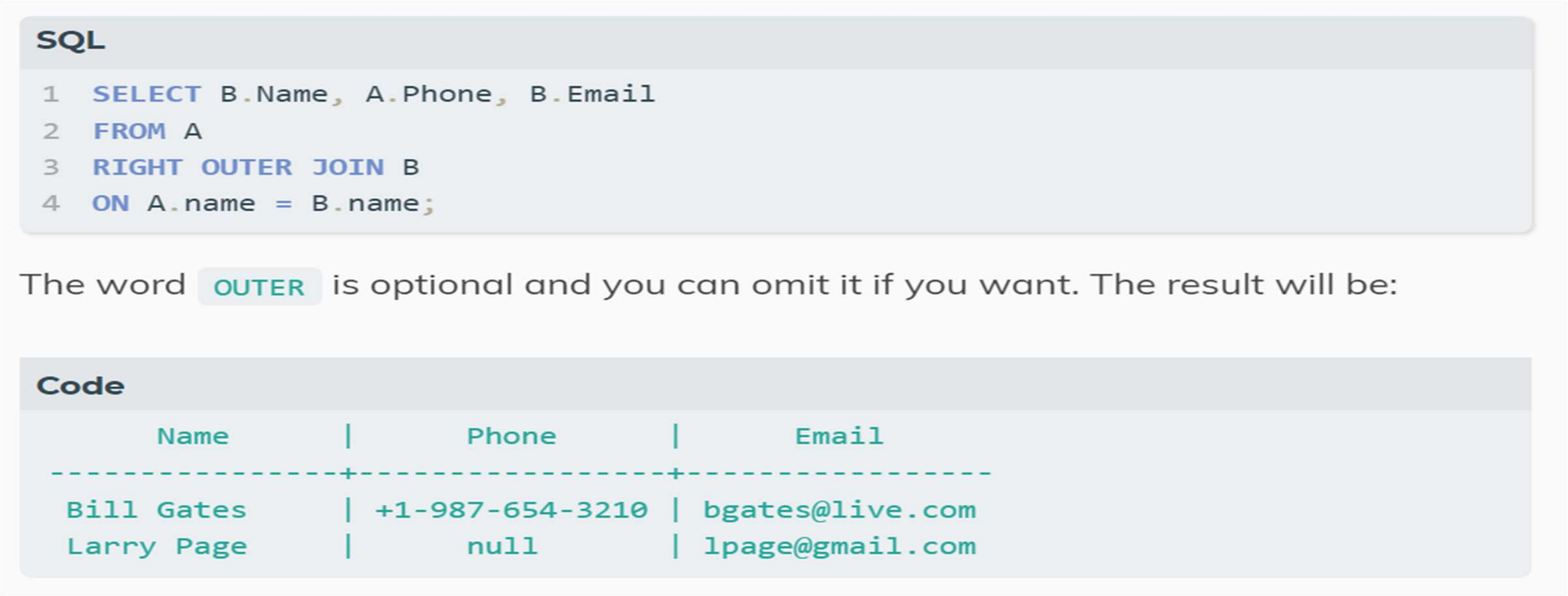
Code

```
      Name       |      Phone       |      Email
-----------------+------------------+-----------------
 Warren Buffett | +1-123-456-7890 |       null
 Bill Gates     | +1-987-654-3210 | bgates@live.com
```

Right Outer Join

A Right Join returns all rows from the right table and the matched rows from the left table.

Say, you want the opposite: all customers with an email address and, if possible, their phone numbers too:



SQL

```sql
SELECT B.Name, A.Phone, B.Email
FROM A
RIGHT OUTER JOIN B
ON A.name = B.name;
```

The word `OUTER` is optional and you can omit it if you want. The result will be:

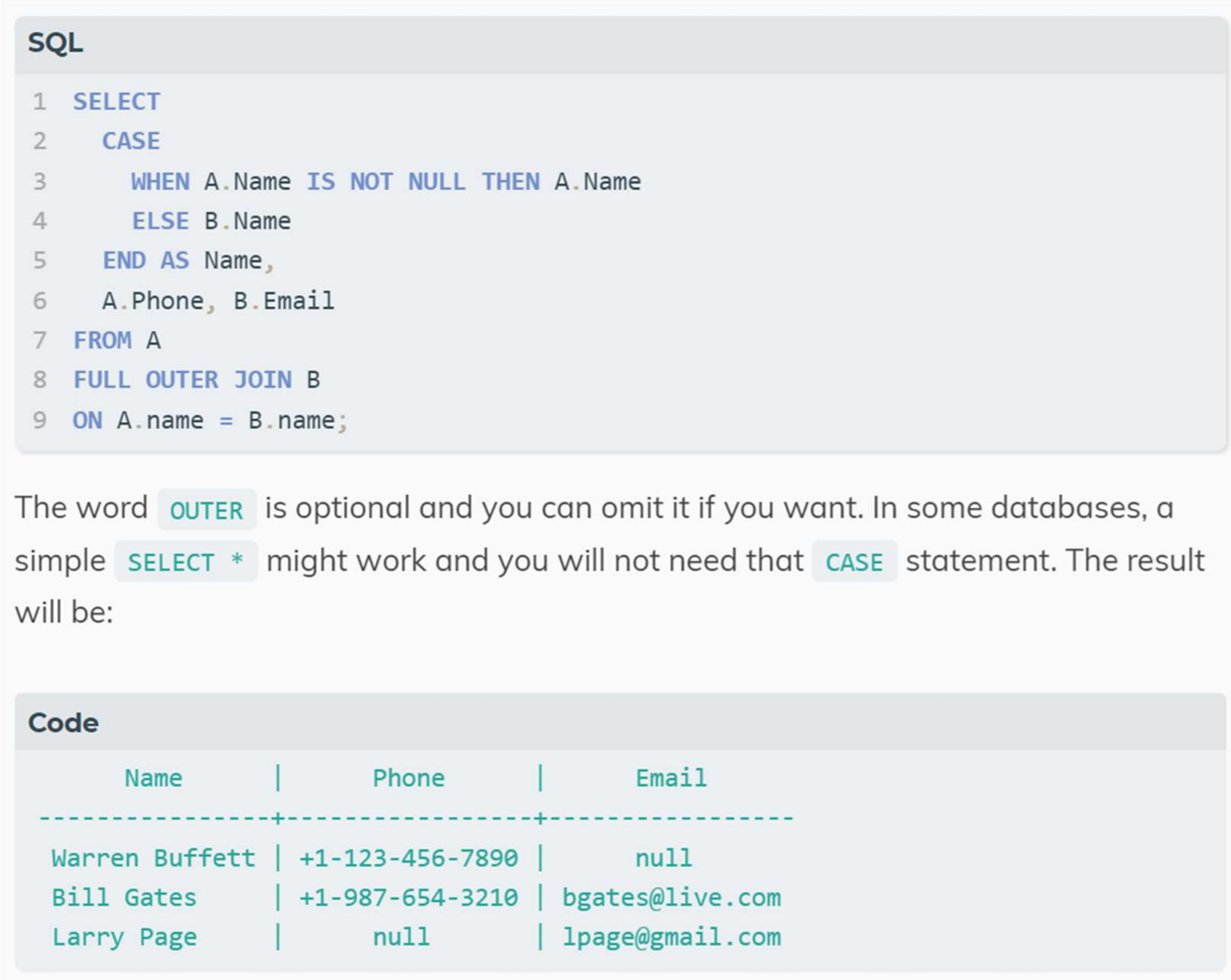
Code

```
      Name       |       Phone       |       Email
-----------------+-------------------+-----------------
  Bill Gates     | +1-987-654-3210 | bgates@live.com
  Larry Page     |       null        | lpage@gmail.com
```

Full Outer Join

A Full Join returns all rows from both tables matching them whenever possible.

Say, you want to have a consolidated list of all customers having a phone number or email address or, if possible, both:



SQL

```sql
SELECT
  CASE
    WHEN A.Name IS NOT NULL THEN A.Name
    ELSE B.Name
  END AS Name,
  A.Phone, B.Email
FROM A
FULL OUTER JOIN B
ON A.name = B.name;
```

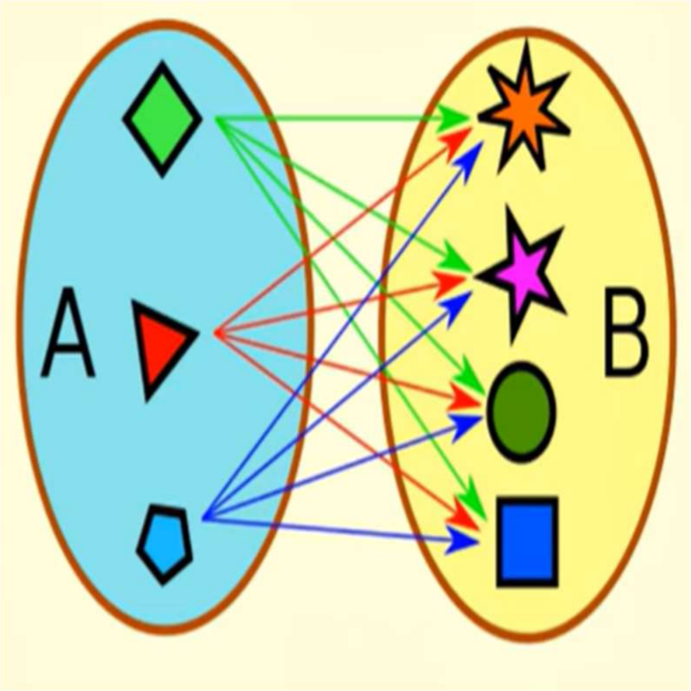
The word `OUTER` is optional and you can omit it if you want. In some databases, a simple `SELECT *` might work and you will not need that `CASE` statement. The result will be:

Code

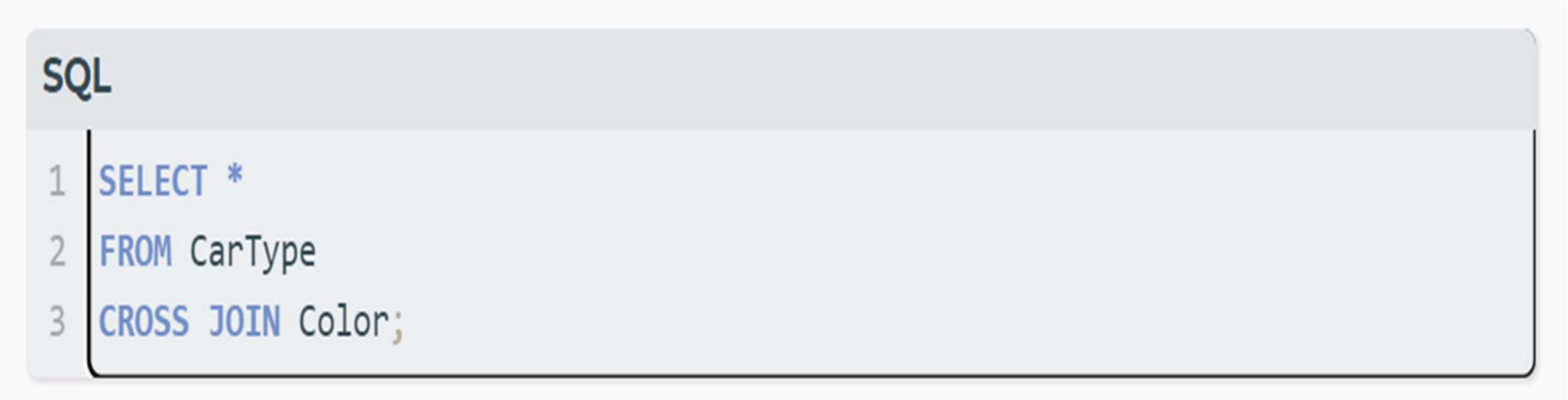
```
     Name       |      Phone       |      Email
----------------+------------------+-----------------
 Warren Buffett | +1-123-456-7890 |      null
 Bill Gates     | +1-987-654-3210 | bgates@live.com
 Larry Page     |      null        | lpage@gmail.com
```

Cross Join

- QL Cross Join is the Cartesian Product of two tables. It pairs each row of the left table with each row of the right table.
- Say, you have a table CarType with 7 rows having values: Sedan, Hatchback, Coupe, MPV, SUV, Pickup Truck, and Convertible.
- And you also have a table Color with 5 rows: Red, Black, Blue, Silver Grey, and Green.
- A Cross Join between the two will generate a table with 35 rows having all possible combinations of car types and colors:

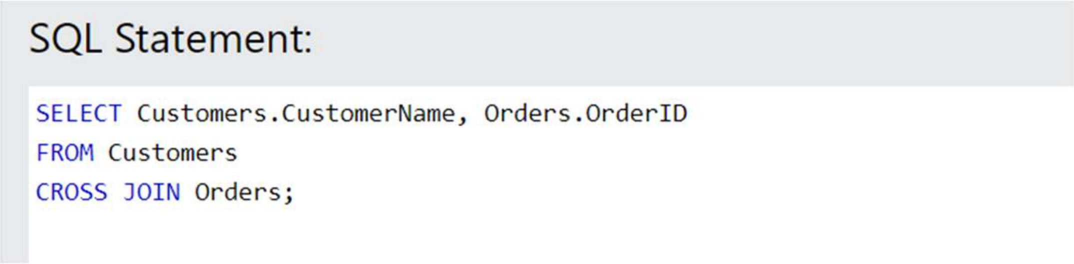


Cross Join Example



SQL

```sql
SELECT *
FROM CarType
CROSS JOIN Color;
```

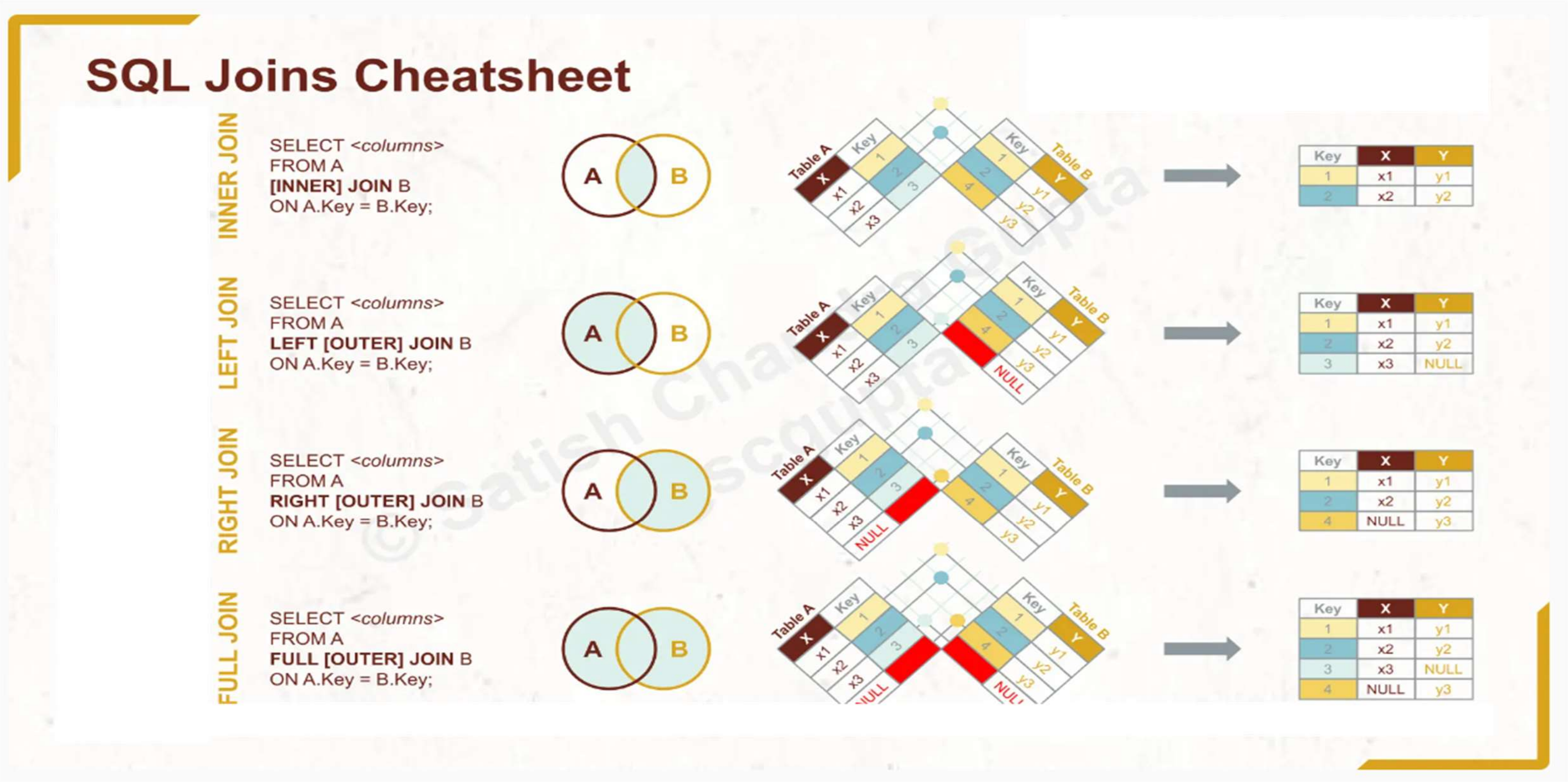


SQL Statement:

```sql
SELECT Customers.CustomerName, Orders.OrderID
FROM Customers
CROSS JOIN Orders;
```

For total 3 customers in customers table
and 4 Orders in Orders table total 12 row output
Will be generated

SQL Joins - Cheatsheet



SQL Joins Cheatsheet

INNER JOIN

```
SELECT <columns>
FROM A
[INNER] JOIN B
ON A.Key = B.Key;
```

Key	X	Y
1	x1	y1
2	x2	y2

LEFT JOIN

```
SELECT <columns>
FROM A
LEFT [OUTER] JOIN B
ON A.Key = B.Key;
```

Key	X	Y
1	x1	y1
2	x2	y2
3	x3	NULL

RIGHT JOIN

```
SELECT <columns>
FROM A
RIGHT [OUTER] JOIN B
ON A.Key = B.Key;
```

Key	X	Y
1	x1	y1
2	x2	y2
4	NULL	y3

FULL JOIN

```
SELECT <columns>
FROM A
FULL [OUTER] JOIN B
ON A.Key = B.Key;
```

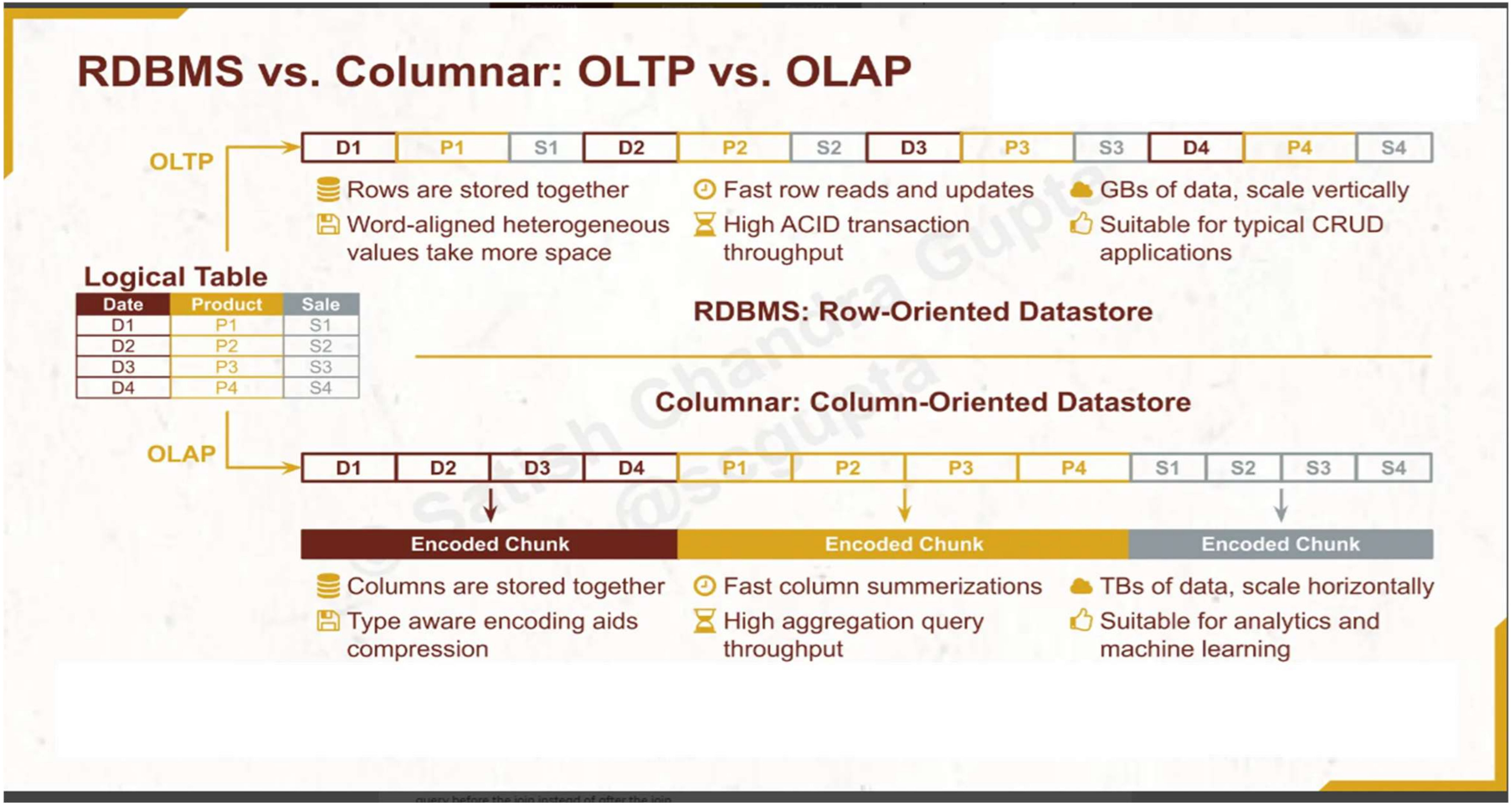
Key	X	Y
1	x1	y1
2	x2	y2
3	x3	NULL
4	NULL	y3

POINTER

Data warehouses are Columnar databases, i.e., values in a column are stored together (in RDBMS, the rows are stored together).

Therefore, SELECT * is a particularly bad idea.

Real Time use of Row & Columnar DB -OLAP VS OLTP



RDBMS vs. Columnar: OLTP vs. OLAP

- Rows are stored together
- Word-aligned heterogeneous values take more space
- Fast row reads and updates
- High ACID transaction throughput
- GBs of data, scale vertically
- Suitable for typical CRUD applications

RDBMS: Row-Oriented Datastore

Logical Table

Date	Product	Sale
D1	P1	S1
D2	P2	S2
D3	P3	S3
D4	P4	S4

Columnar: Column-Oriented Datastore

- Columns are stored together
- Type aware encoding aids compression
- Fast column summerizations
- High aggregation query throughput
- TBs of data, scale horizontally
- Suitable for analytics and machine learning

query before the join instead of after the join

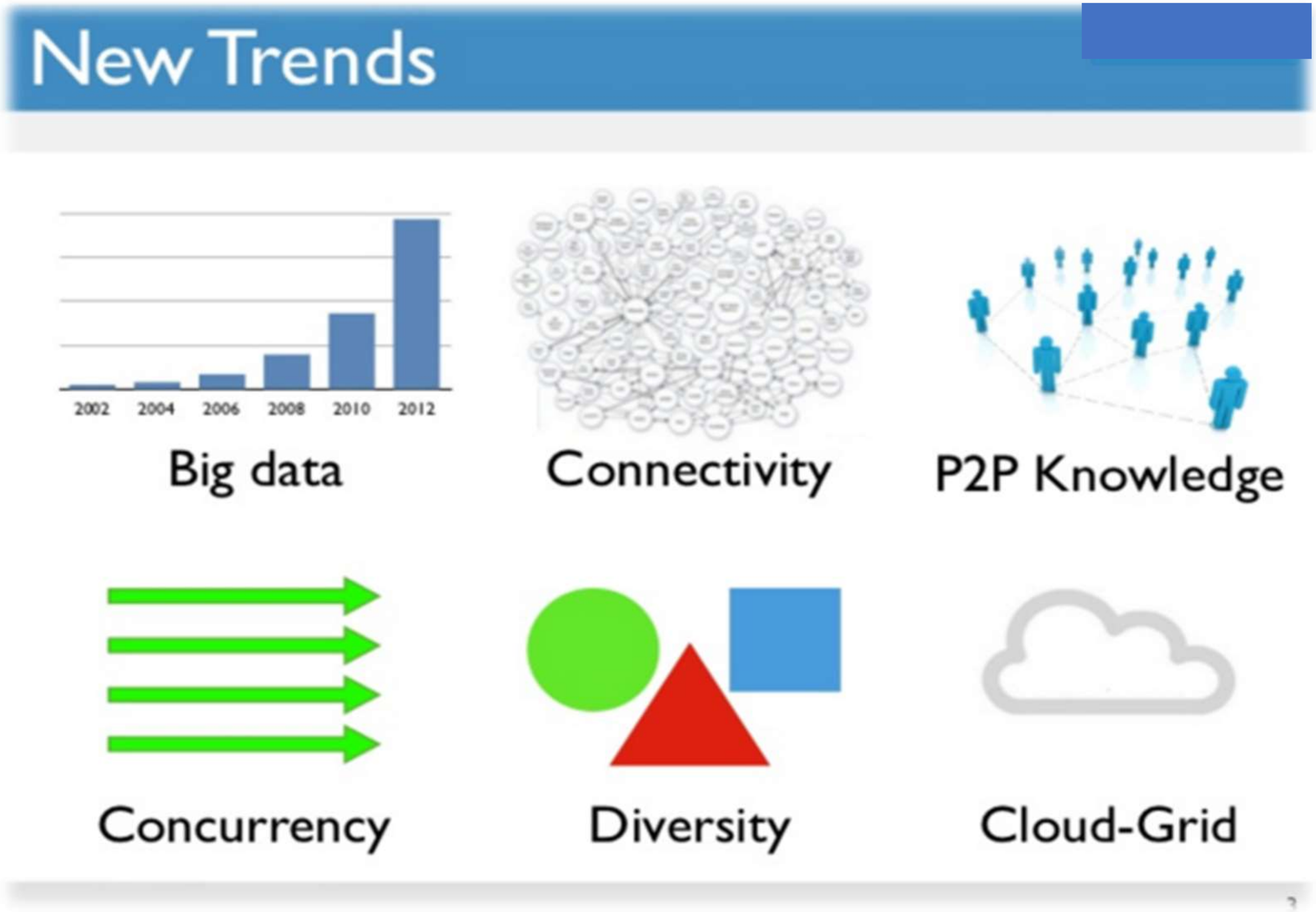
Why NoSQL

Relational databases are optimized for transaction operations. Transactions often update multiple records in multiple tables. Indexes are optimized for frequent low-latency writes of ACID Transactions.

While transactions are on rows (records), analytics properties are computed on columns (attributes). OLAP applications need an optimized column-read operation on a table. Columnar databases are designed for high-throughput of column aggregations. That’s why Columnar DBs are row-oriented databases.

However, the primary RDBMS operation is low-latency high-frequency ACID transactions. That does not scale to the Big Data scale common in analytics applications.

Why NOSQL now?? Ans. Driving Trends



New Trends

Big data

Connectivity

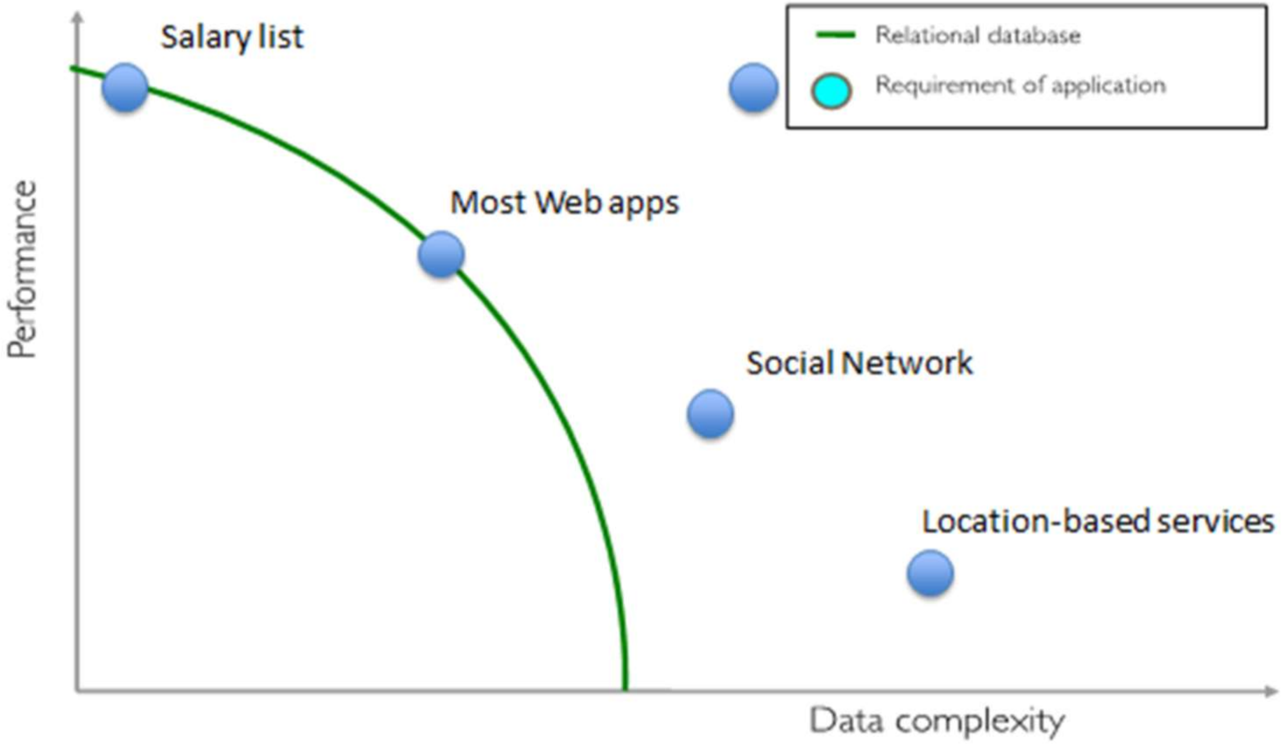
P2P Knowledge

Concurrency

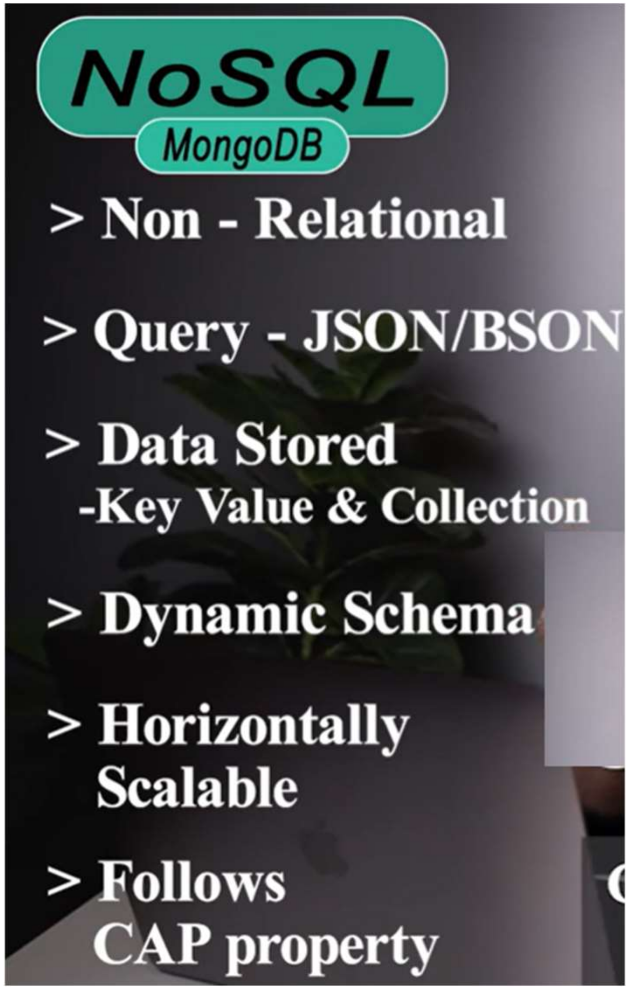
Diversity

Cloud-Grid

Side note: RDBMS performance



Contrast



NoSQL

MongoDB

> Non - Relational

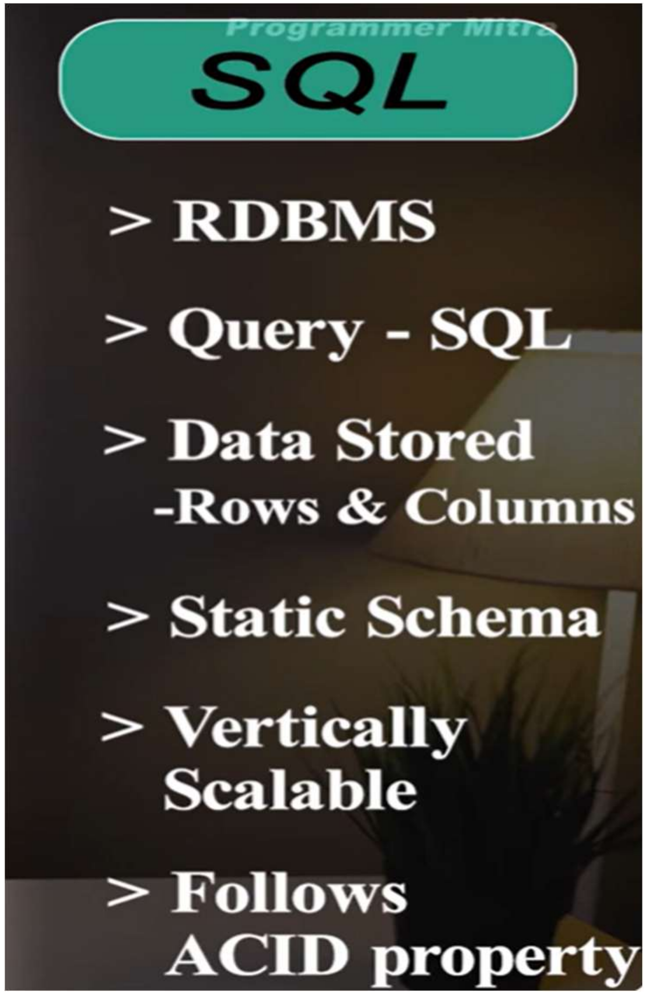
> Query - JSON/BSON

> Data Stored
-Key Value & Collection

> Dynamic Schema

> Horizontally
Scalable

> Follows
CAP property



SQL

> RDBMS

> Query - SQL

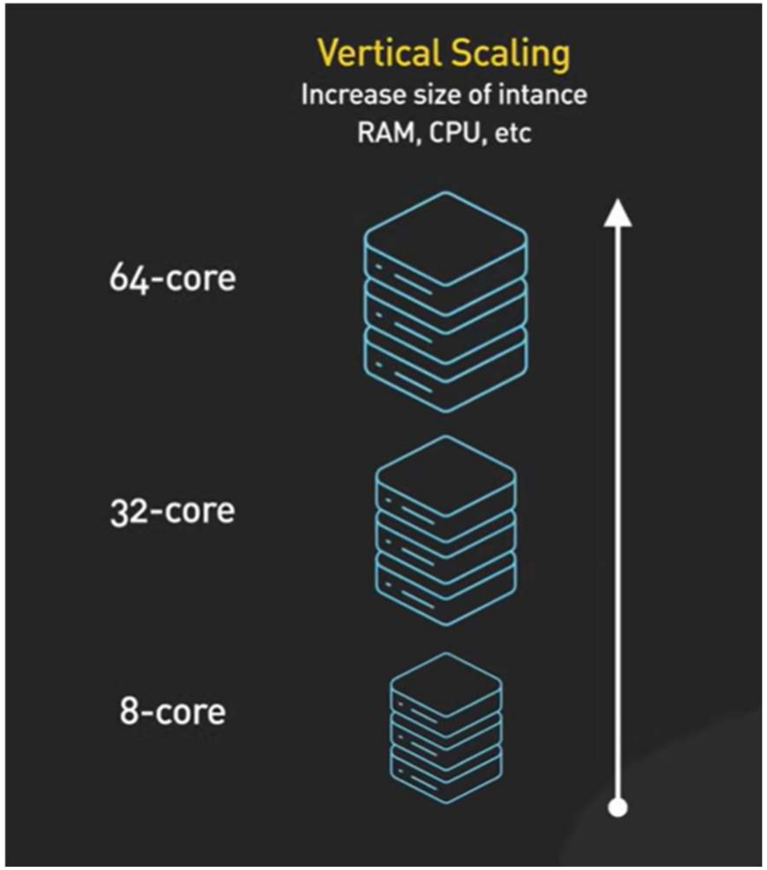
> Data Stored
-Rows & Columns

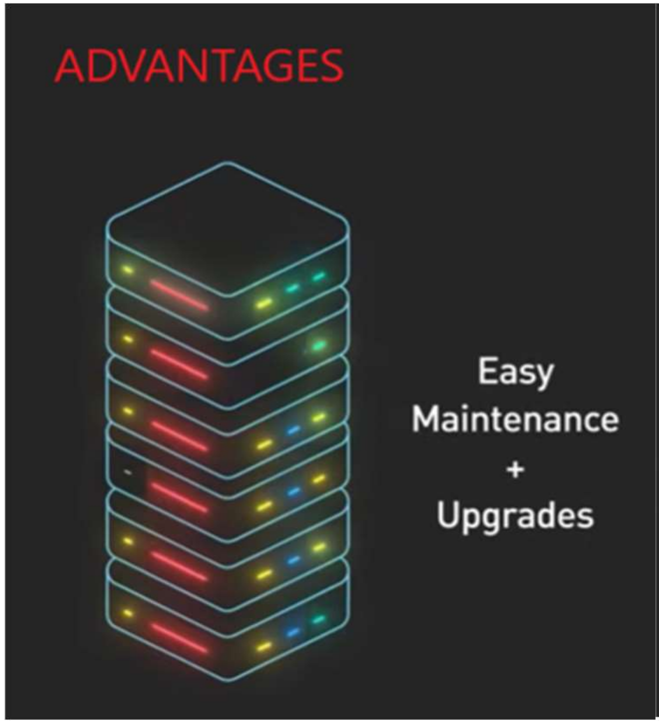
> Static Schema

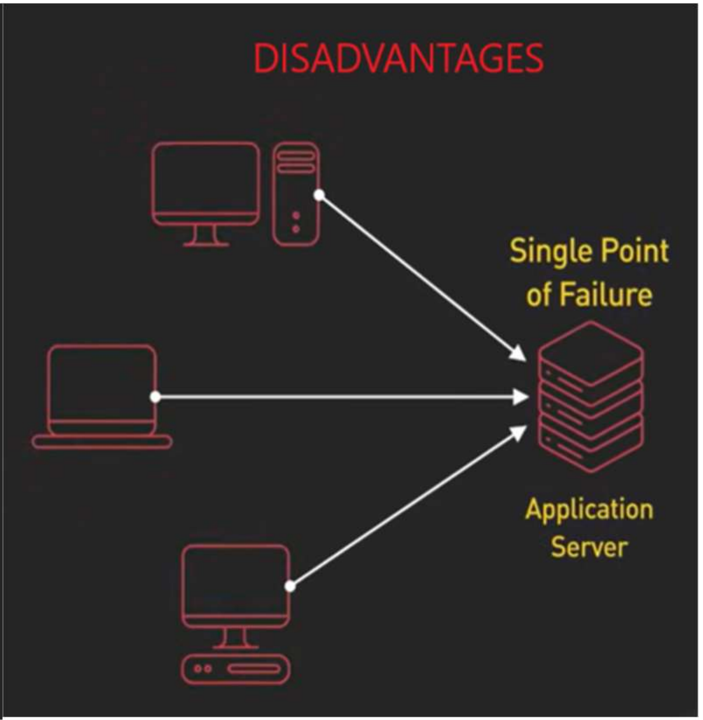
> Vertically
Scalable

> Follows
ACID property

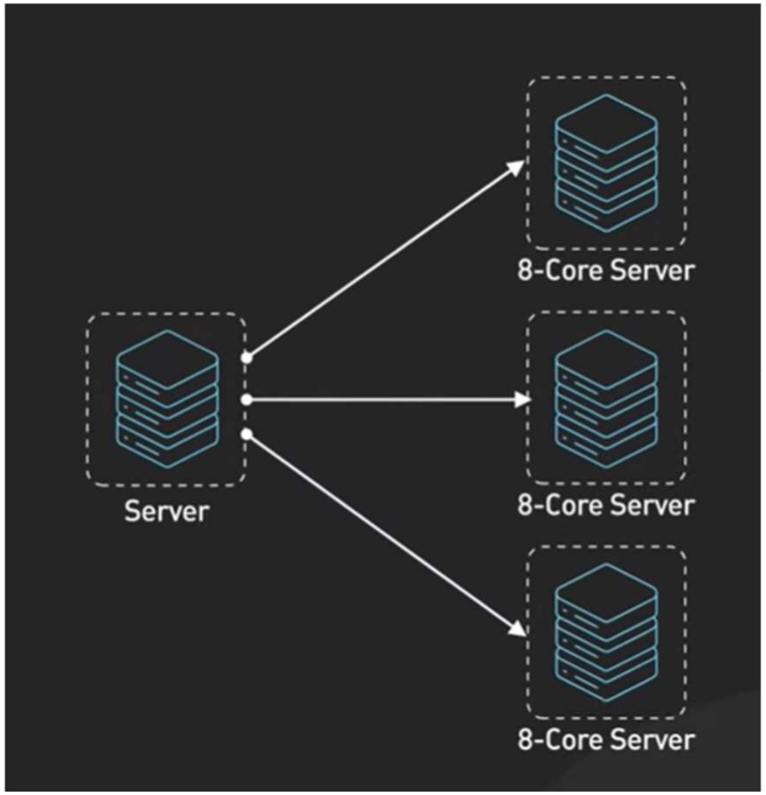
RDBMS = Vertical Scaling

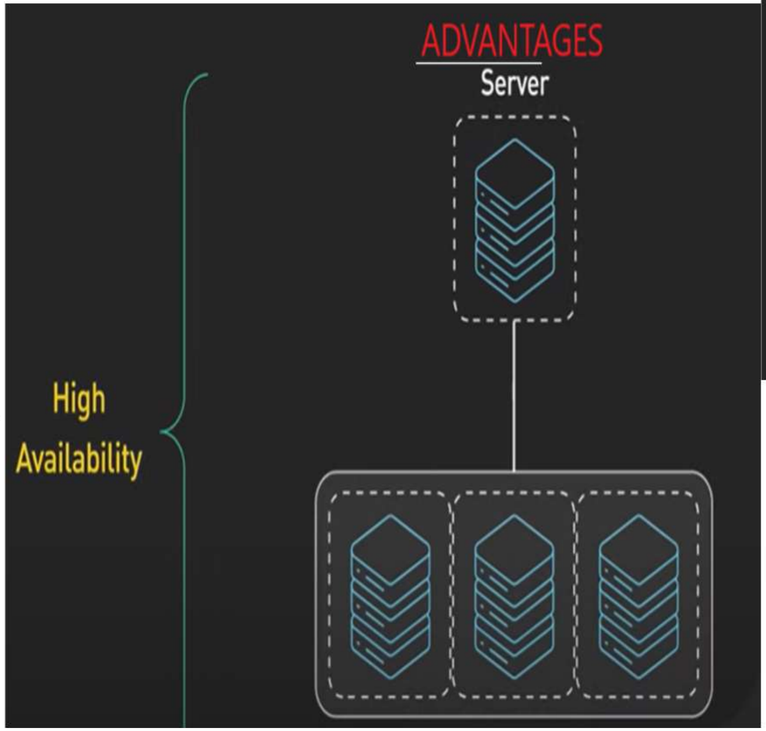


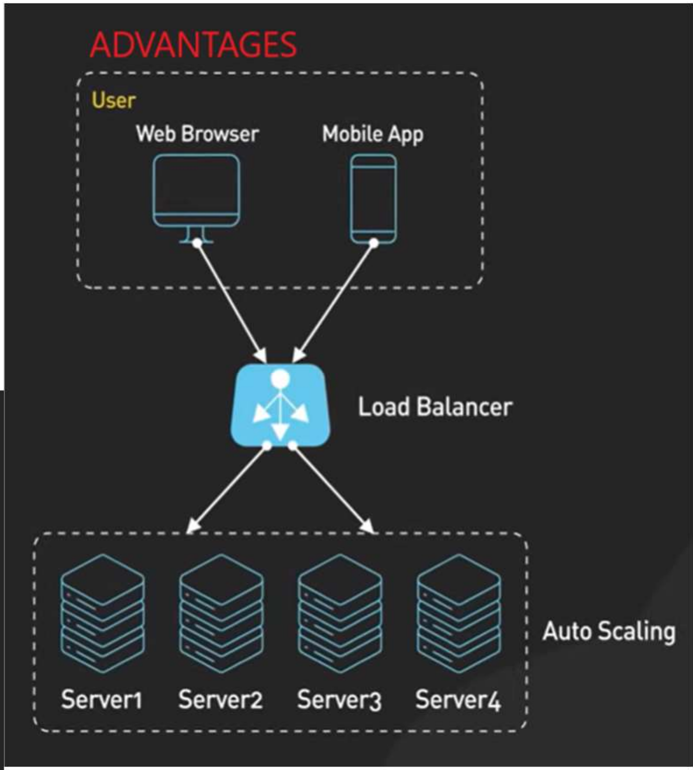




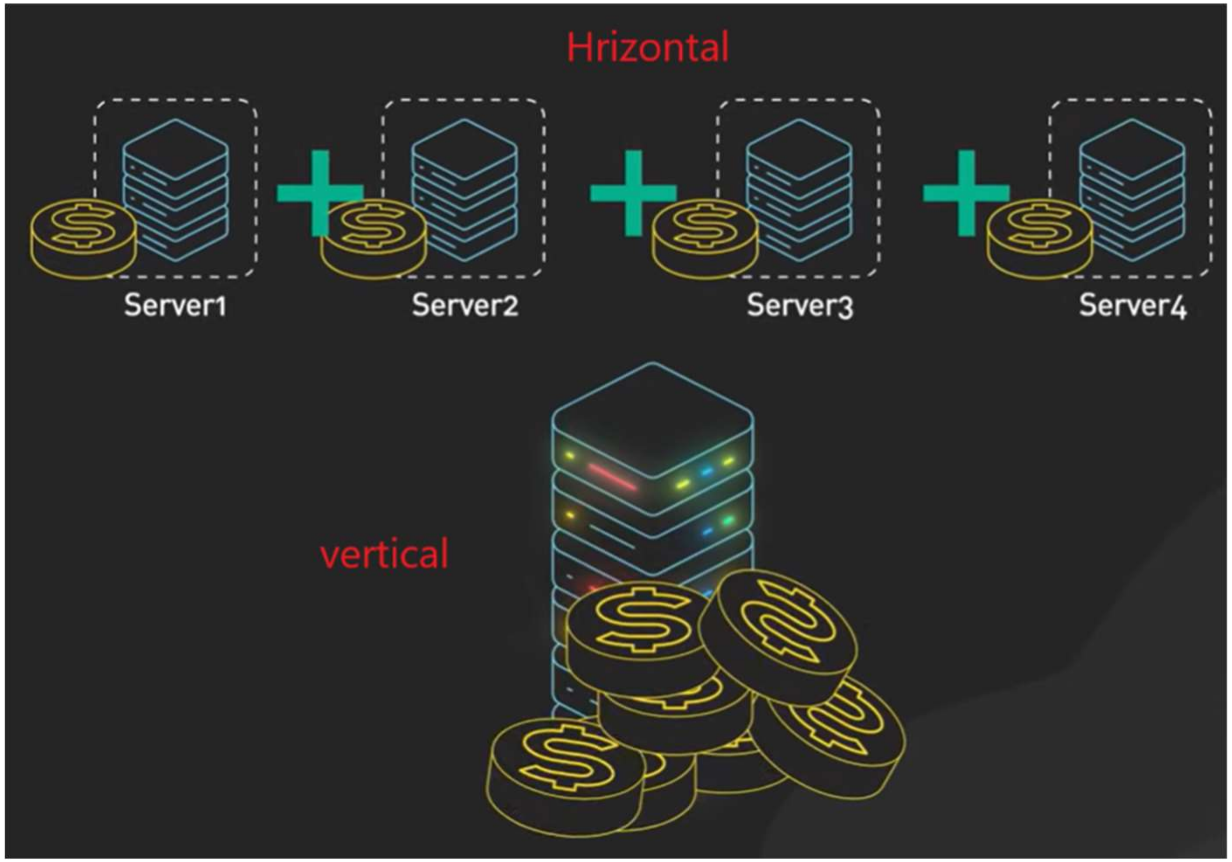
NoSQL = Horizontal Scaling



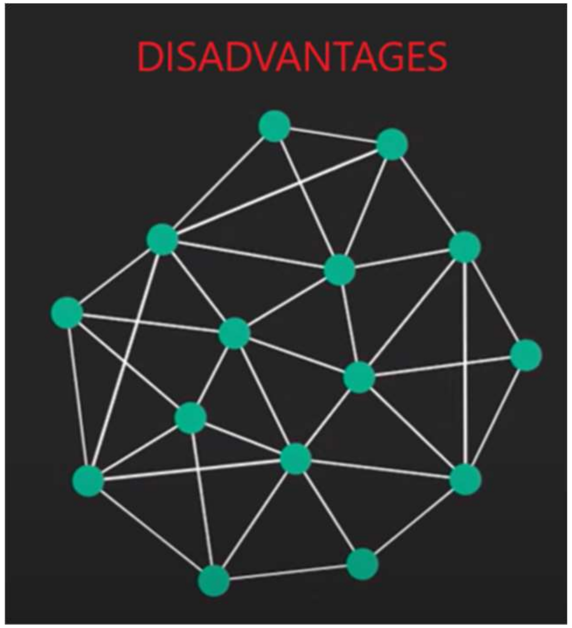


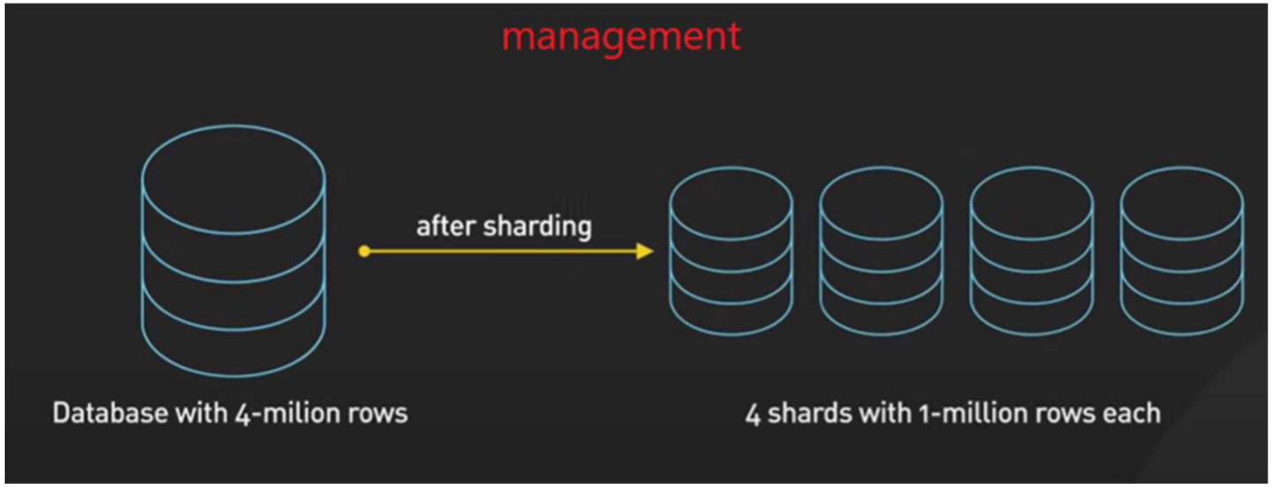


Cost…



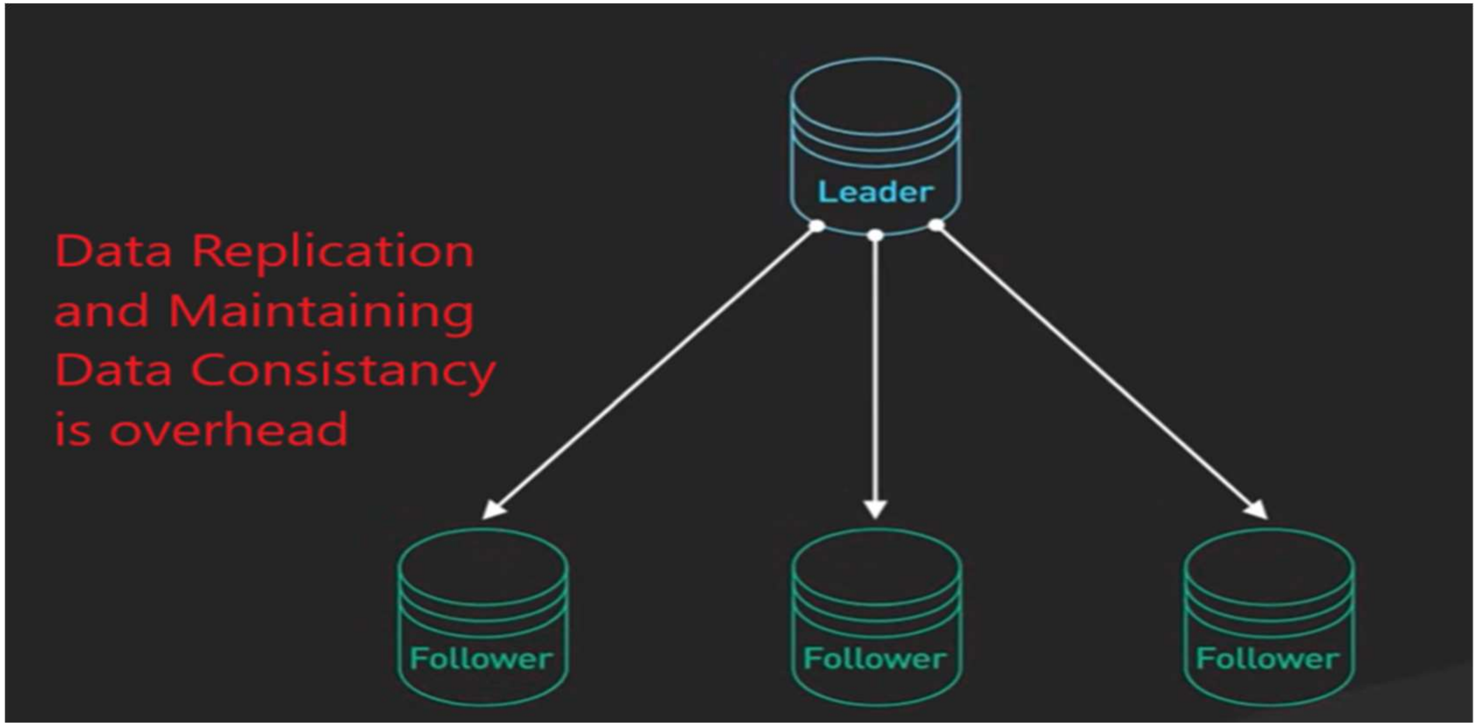
Disadvantages -- Horizontal





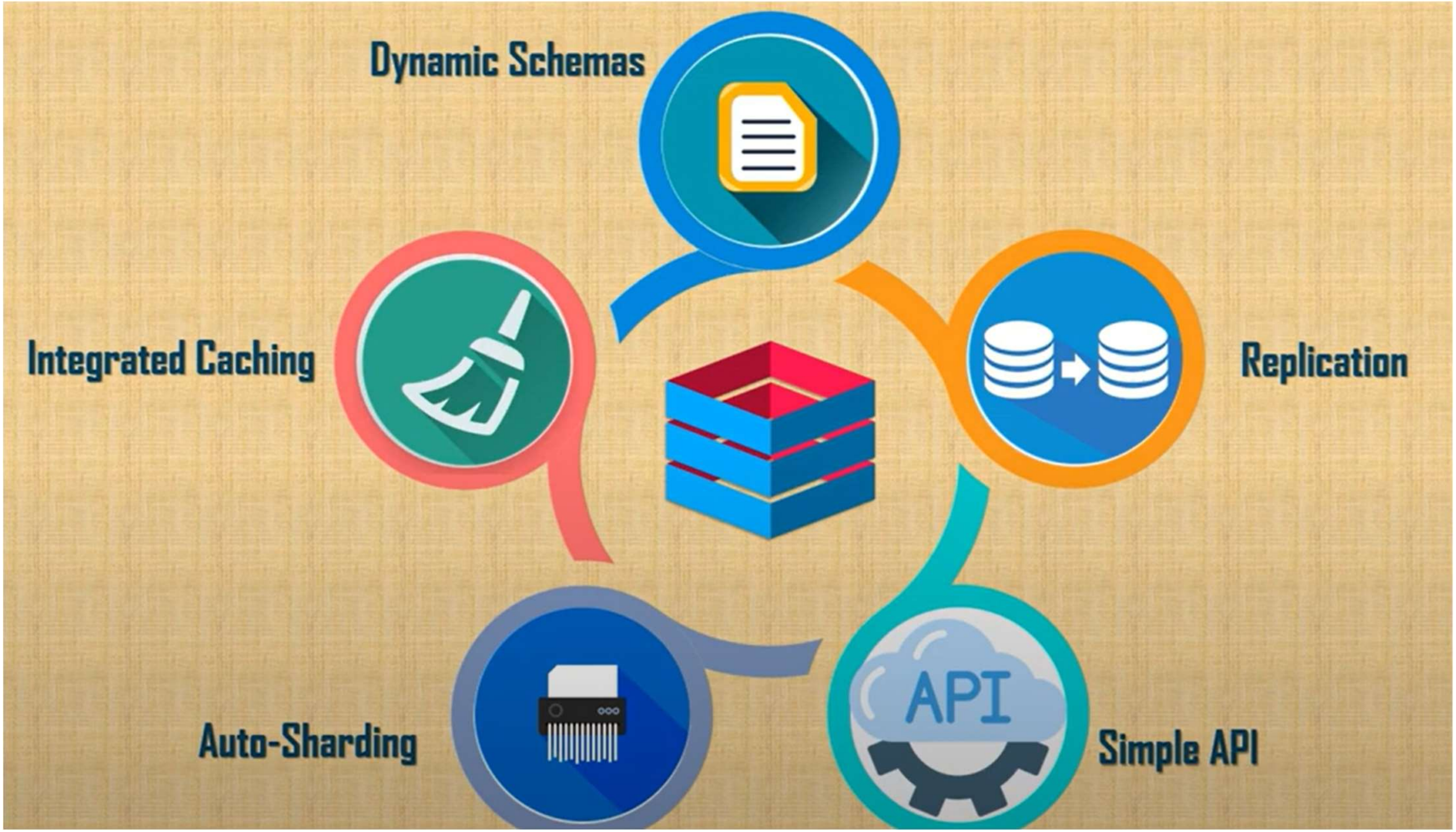
management
after sharding
Database with 4-milion rows
4 shards with 1-million rows each

Disadvantages

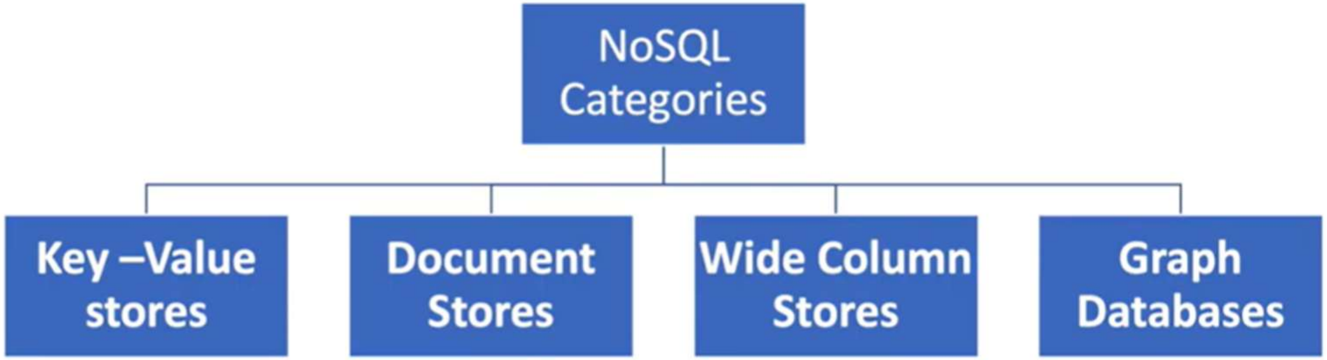


Data Replication and Maintaining Data Consistancy is overhead

NoSQL Features



NoSQL DB Types



NoSQL Categories

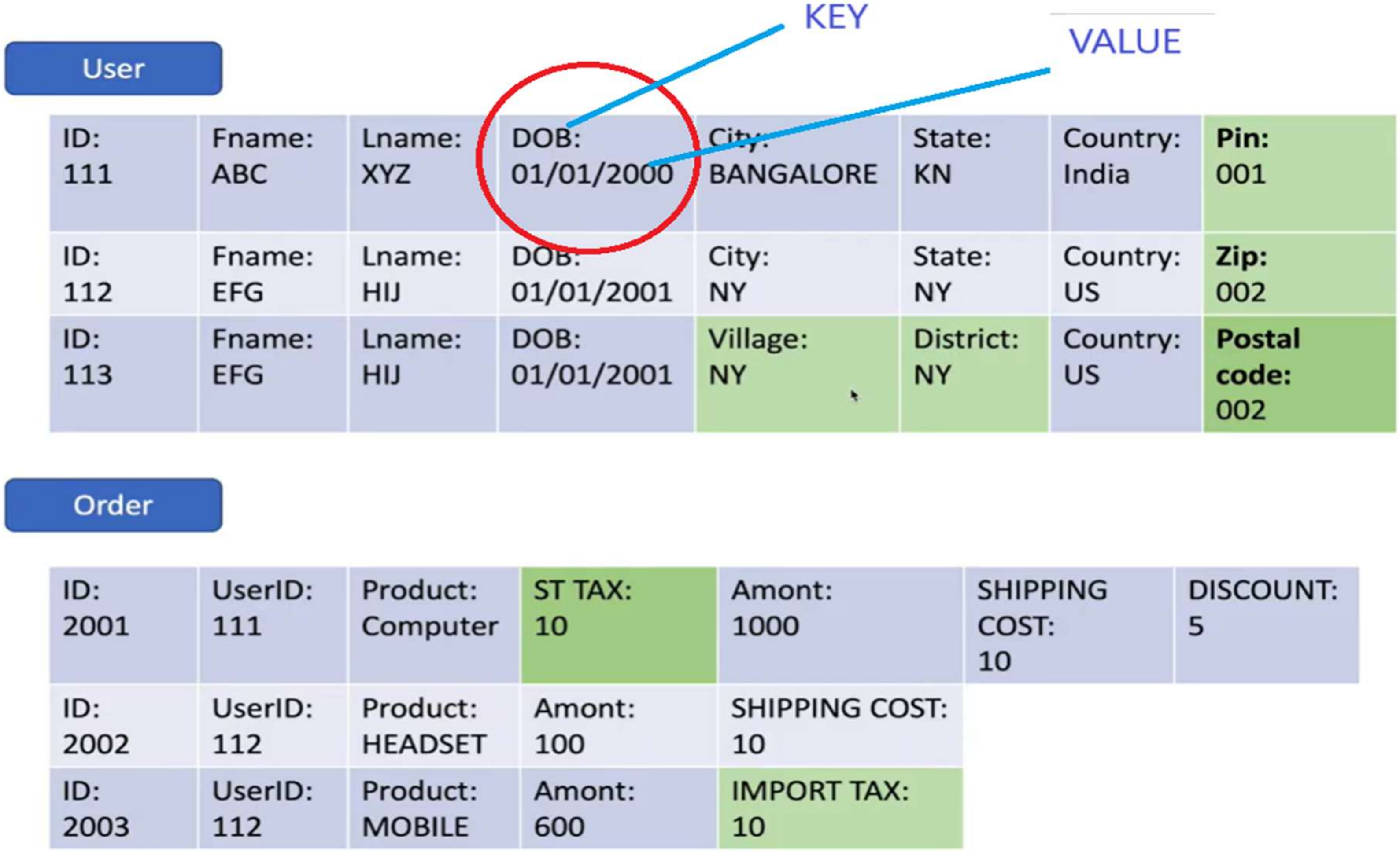
Key –Value stores

Document Stores

Wide Column Stores

Graph Databases

NoSQL DB



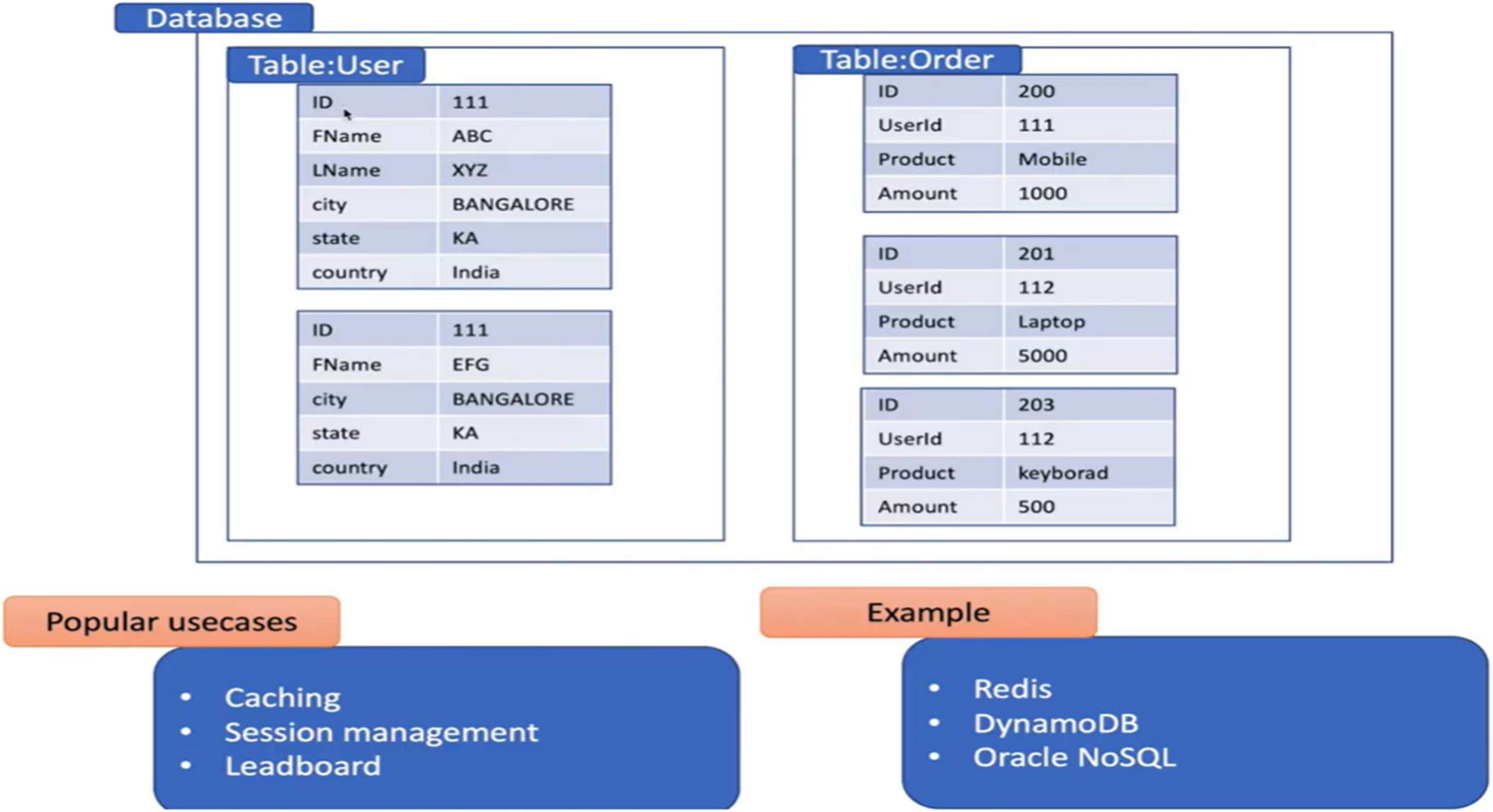
User

ID: 111	Fname: ABC	Lname: XYZ	DOB: 01/01/2000	City: BANGALORE	State: KN	Country: India	**Pin:** 001
ID: 112	Fname: EFG	Lname: HIJ	DOB: 01/01/2001	City: NY	State: NY	Country: US	**Zip:** 002
ID: 113	Fname: EFG	Lname: HIJ	DOB: 01/01/2001	Village: NY	District: NY	Country: US	**Postal code:** 002

Order

ID: 2001	UserID: 111	Product: Computer	ST TAX: 10	Amont: 1000	SHIPPING COST: 10	DISCOUNT: 5
ID: 2002	UserID: 112	Product: HEADSET	Amont: 100	SHIPPING COST: 10		
ID: 2003	UserID: 112	Product: MOBILE	Amont: 600	IMPORT TAX: 10		

Key-Value Pair



Database

Table:User

ID	111
FName	ABC
LName	XYZ
city	BANGALORE
state	KA
country	India

ID	111
FName	EFG
city	BANGALORE
state	KA
country	India

Table:Order

ID	200
UserId	111
Product	Mobile
Amount	1000

ID	201
UserId	112
Product	Laptop
Amount	5000

ID	203
UserId	112
Product	keyborad
Amount	500

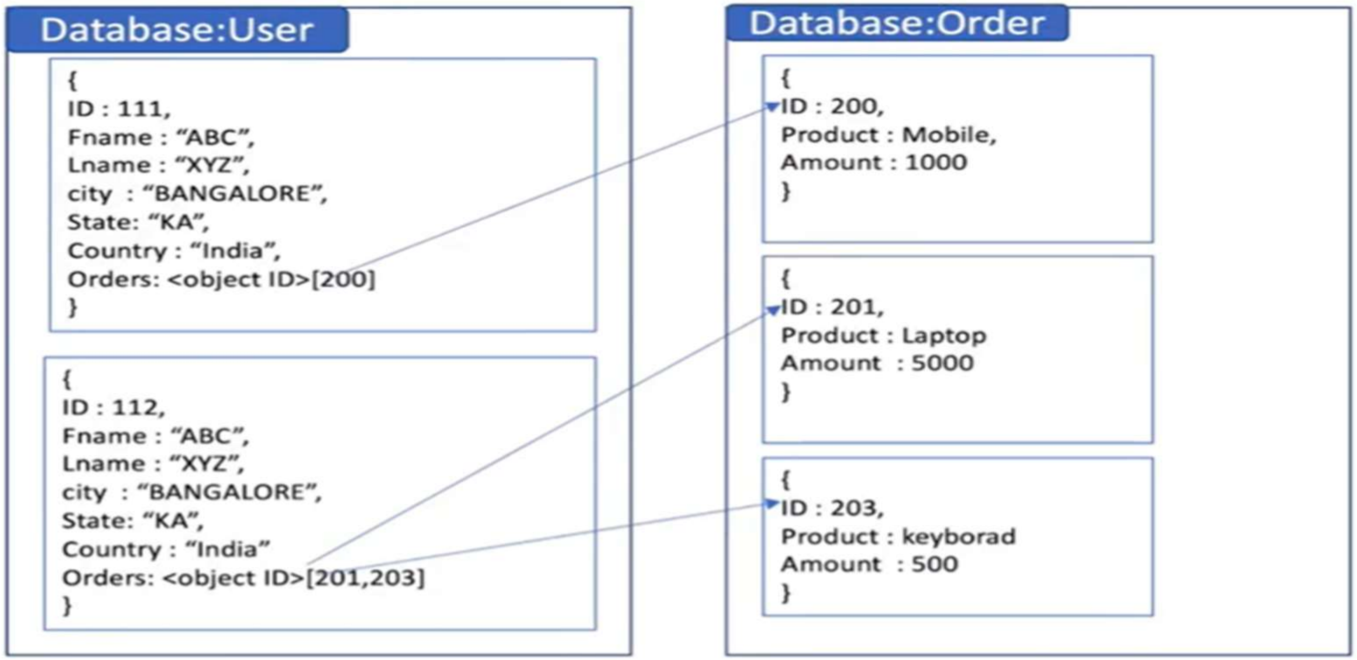
Popular usecases

- Caching
- Session management
- Leadboard

Example

- Redis
- DynamoDB
- Oracle NoSQL

Document DB



Database:User

```
{
ID : 111,
Fname : "ABC",
Lname : "XYZ",
city  : "BANGALORE",
State: "KA",
Country : "India",
Orders: <object ID>[200]
}
```

```
{
ID : 112,
Fname : "ABC",
Lname : "XYZ",
city  : "BANGALORE",
State: "KA",
Country : "India"
Orders: <object ID>[201,203]
}
```

Database:Order

```
{
ID : 200,
Product : Mobile,
Amount : 1000
}
```

```
{
ID : 201,
Product : Laptop
Amount  : 5000
}
```

```
{
ID : 203,
Product : keyborad
Amount  : 500
}
```

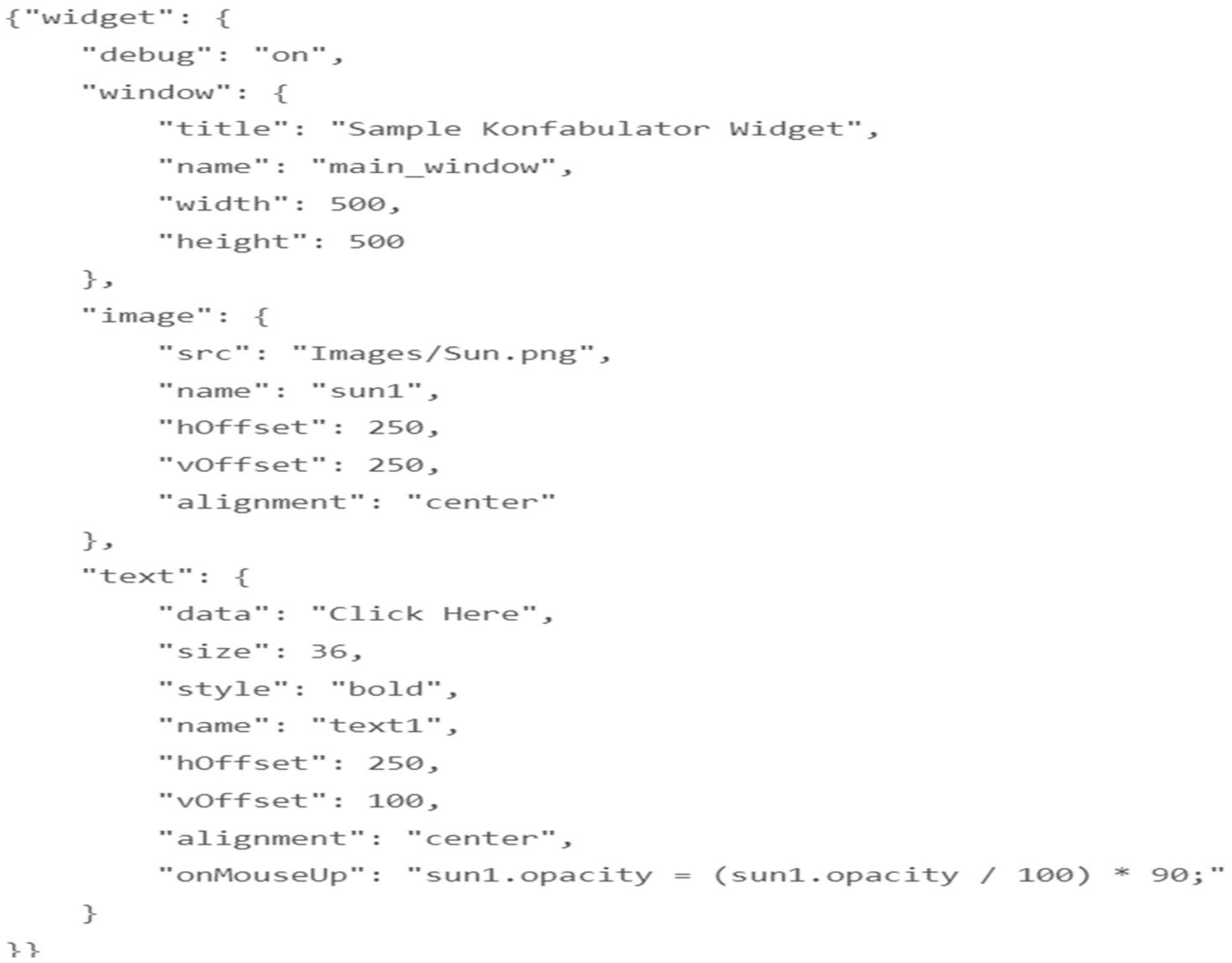
Popular usecases

- Blog/ website CMS
- Products catalog
- Big Data
- Analytics

Example

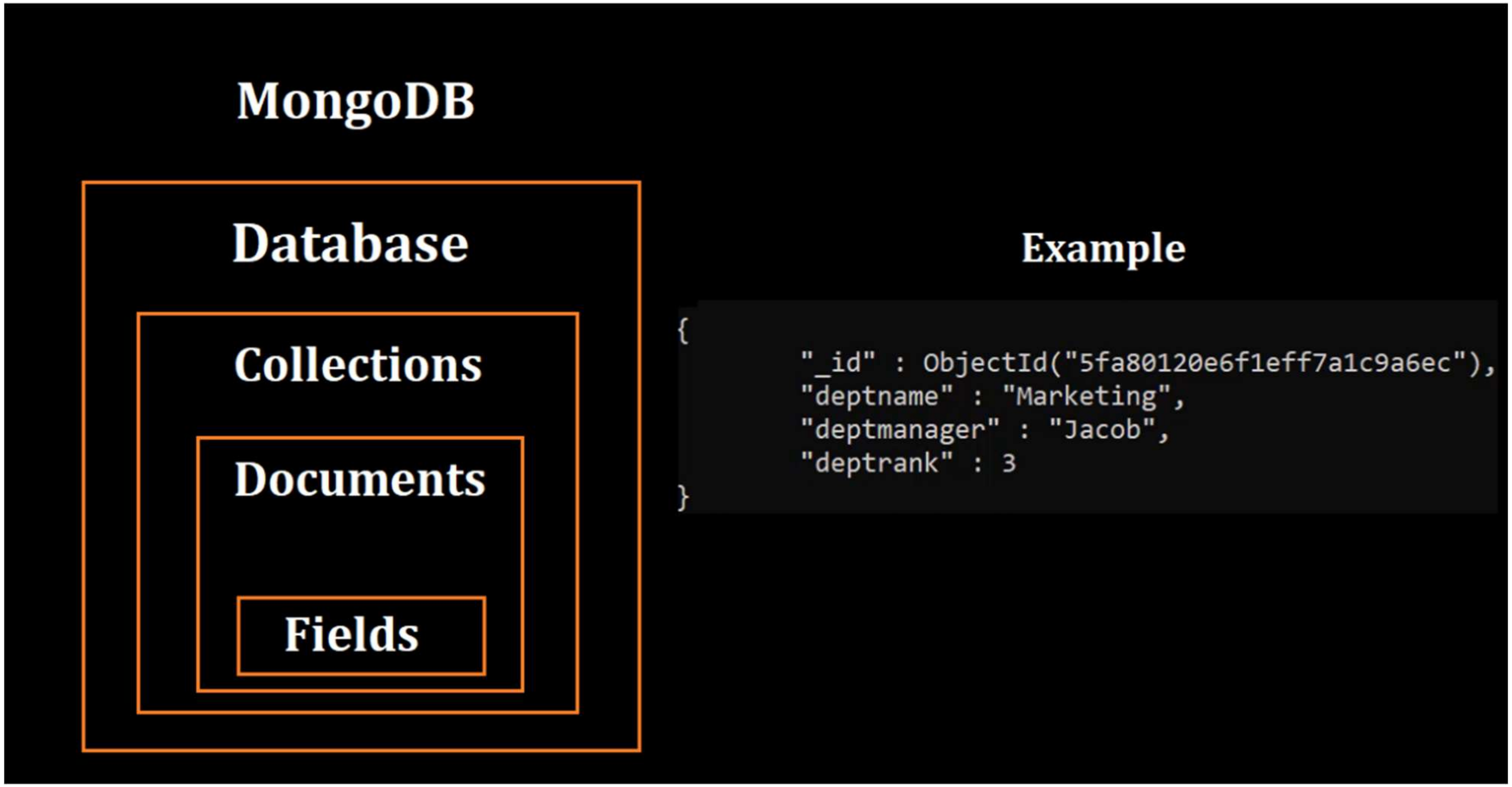
- MongoDB
- Apache CouchDB
- Azure Cosmos DB

MongoDB JSON



```
{"widget": {
    "debug": "on",
    "window": {
        "title": "Sample Konfabulator Widget",
        "name": "main_window",
        "width": 500,
        "height": 500
    },
    "image": {
        "src": "Images/Sun.png",
        "name": "sun1",
        "hOffset": 250,
        "vOffset": 250,
        "alignment": "center"
    },
    "text": {
        "data": "Click Here",
        "size": 36,
        "style": "bold",
        "name": "text1",
        "hOffset": 250,
        "vOffset": 100,
        "alignment": "center",
        "onMouseUp": "sun1.opacity = (sun1.opacity / 100) * 90;"
    }
}}
```

Structure



MongoDB

Database

Collections

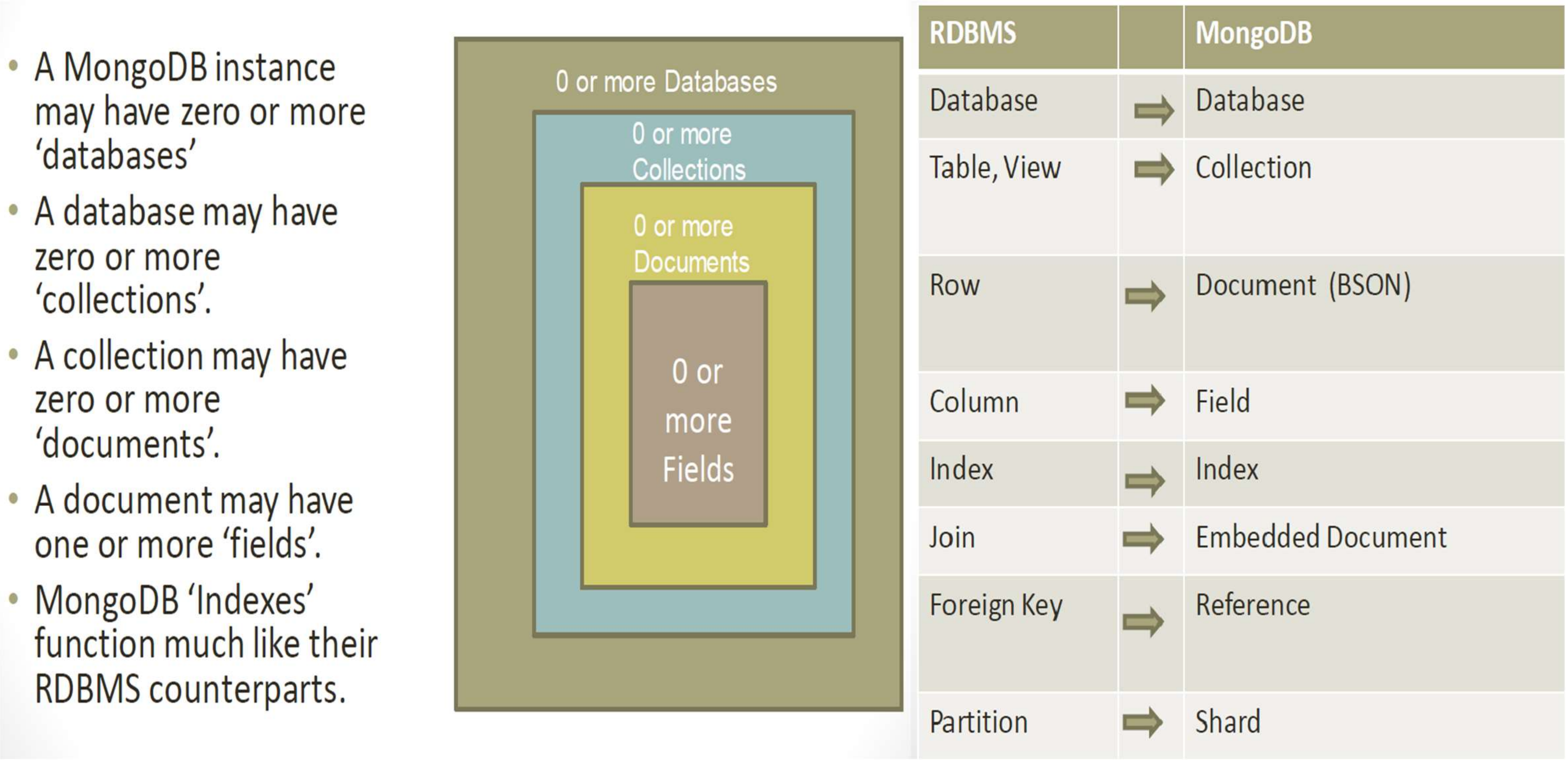
Documents

Fields

Example

```
{
        "_id" : ObjectId("5fa80120e6f1eff7a1c9a6ec"),
        "deptname" : "Marketing",
        "deptmanager" : "Jacob",
        "deptrank" : 3
}
```

Mapping with RDBMS



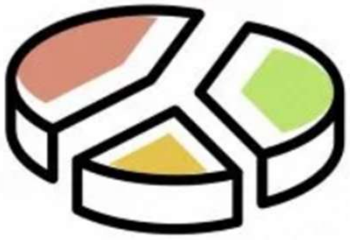
- A MongoDB instance may have zero or more 'databases'
- A database may have zero or more 'collections'.
- A collection may have zero or more 'documents'.
- A document may have one or more 'fields'.
- MongoDB 'Indexes' function much like their RDBMS counterparts.

RDBMS		MongoDB
Database	⇨	Database
Table, View	⇨	Collection
Row	⇨	Document (BSON)
Column	⇨	Field
Index	⇨	Index
Join	⇨	Embedded Document
Foreign Key	⇨	Reference
Partition	⇨	Shard

Replication and Partitioning

Replication

Replication (Copying data)— Keeping a copy of same data on multiple servers that are connected via a network.



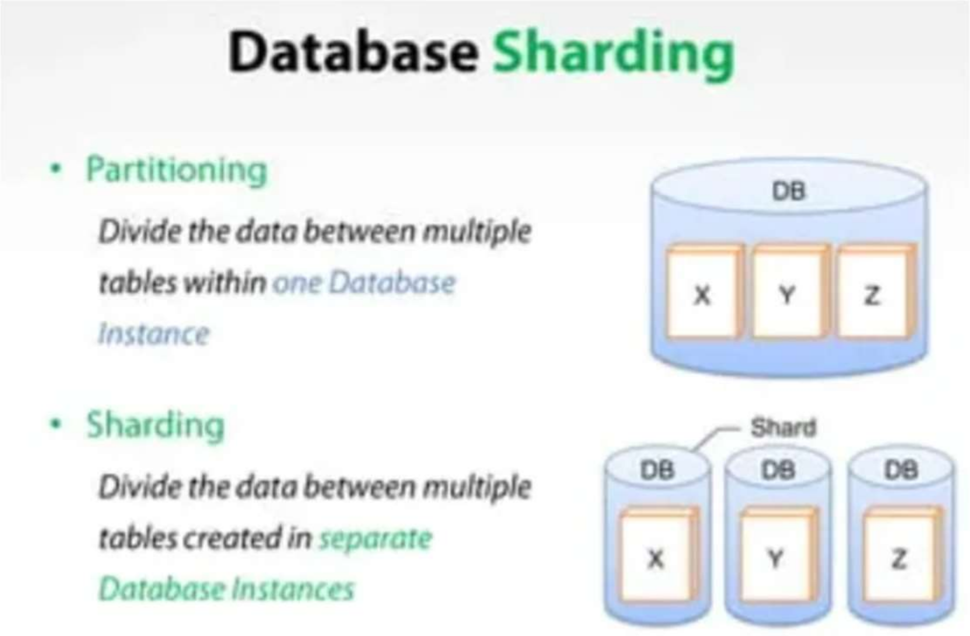
Partitioning

Partitioning — Splitting up a large monolithic database into multiple smaller databases based on data cohesion. e.g. Horizontal (sharding) and Vertical (increase server size) partitioning.

Partitioning and Sharding

All orders placed in January can be stored in one partition, all orders placed in February can be stored in another partition, and so on. Each partition can then be stored on a separate server. This way, when the company needs to retrieve orders from a particular time period, it can query only the relevant partition, which can significantly improve performance.

All orders from customers in North America can be stored on one server, all orders from customers in Europe can be stored on another server, and so on. This way, when the company needs to retrieve orders from a particular region, it can query only the relevant server, which can help improve performance and reduce the load on any one server.



Database Sharding

- Partitioning

  Divide the data between multiple tables within one Database Instance

- Sharding

  Divide the data between multiple tables created in separate Database Instances

Wide Column DB

Table:User

ID	111	
Super Column : Name		
	Column : FName	ABC
	Column : LName	XYZ
Super Column : Address		
	Column : city	BANGALORE
	Column : state	KA
	Column : country	India
Super Column : Orders		
	Orders	[200, 201]

ID	111	
Super Column : Name		
	Column : FName	ABC
	Column : LName	XYZ

Table:Order

ID	200	
Super Column : PRODUCT		
	Column : Product	Laptop
Super Column : Price		
	Column : Amount	5000

ID	201	
Super Column : PRODUCT		
	Column : Product	Keyboard
Super Column : Price		
	Column : Amount	500

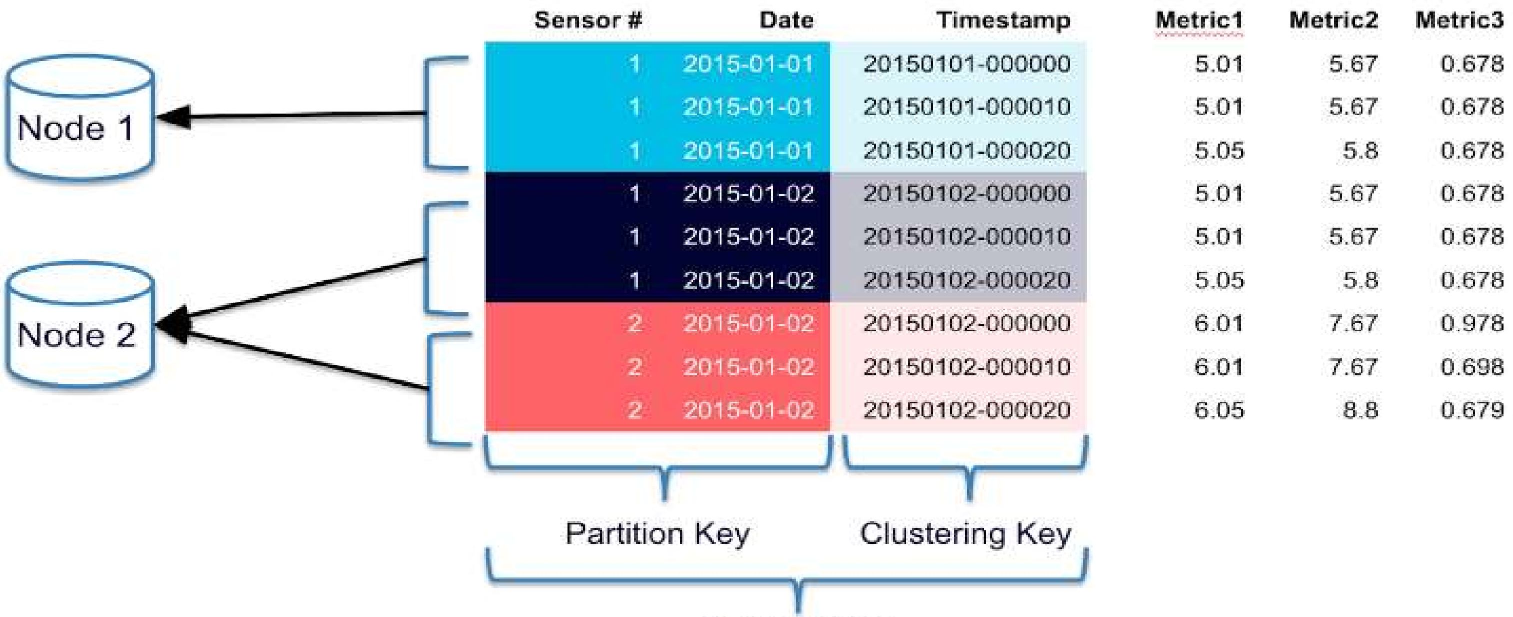
Popular usecases

- IOT
- Inventory management
- Big Data

Example

- Cassandra
- HBase

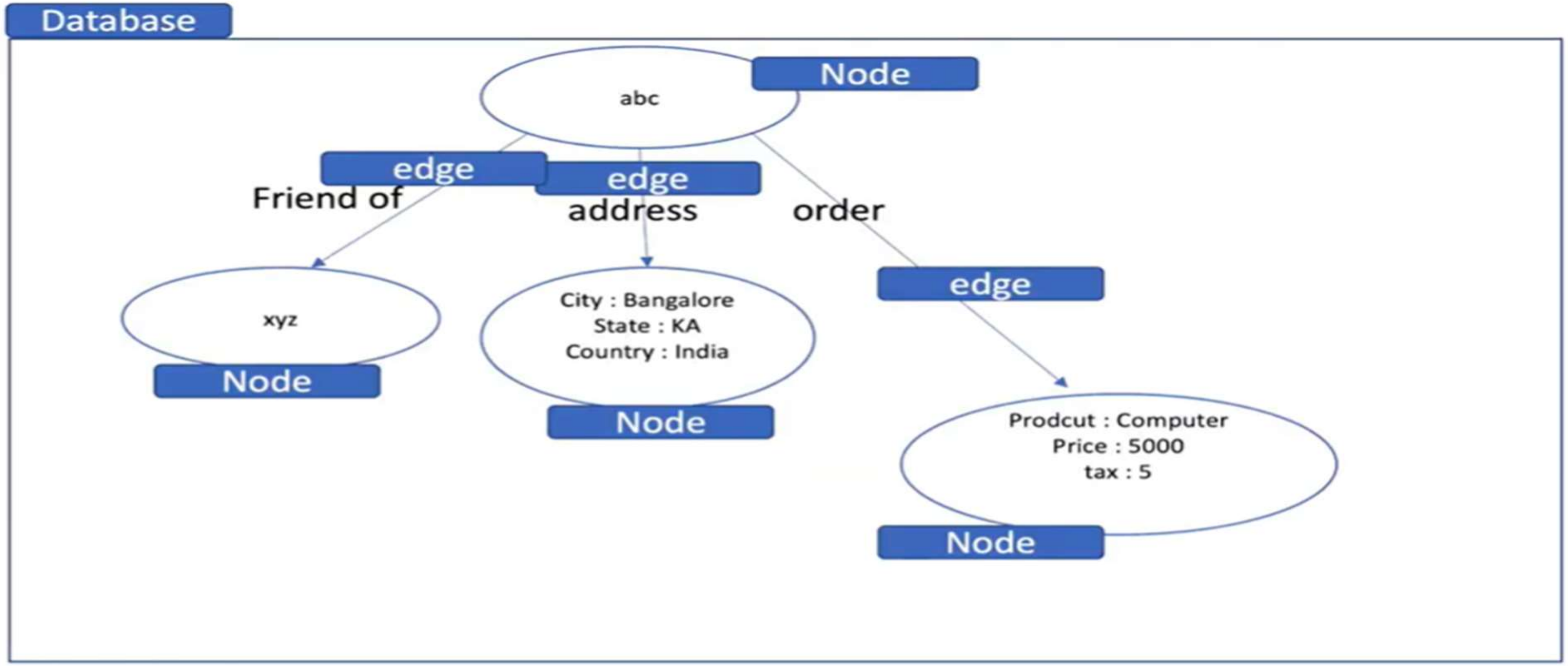
Cassandra Strategy



Sensor #	Date	Timestamp	Metric1	Metric2	Metric3
1	2015-01-01	20150101-000000	5.01	5.67	0.678
1	2015-01-01	20150101-000010	5.01	5.67	0.678
1	2015-01-01	20150101-000020	5.05	5.8	0.678
1	2015-01-02	20150102-000000	5.01	5.67	0.678
1	2015-01-02	20150102-000010	5.01	5.67	0.678
1	2015-01-02	20150102-000020	5.05	5.8	0.678
2	2015-01-02	20150102-000000	6.01	7.67	0.978
2	2015-01-02	20150102-000010	6.01	7.67	0.698
2	2015-01-02	20150102-000020	6.05	8.8	0.679

- Cassandra achieves a compromise between the two partitioning strategies .
- A table in Cassandra can be declared with a *compound primary key* consisting of several columns. Only the first part of that key is hashed to determine the partition, but the other columns are used as a concatenated index for sorting the data in Cassandra's SSTables.

Graph DB



Popular usecases

- Social networking
- IAM
- Recommendation

Example

- Neo4j
- JanusGraph

That is all