Fullstack Development

Database Design

Content

- Database ranking
- SQL database
- NoSQL database
- Schema patterns
- Some useful information

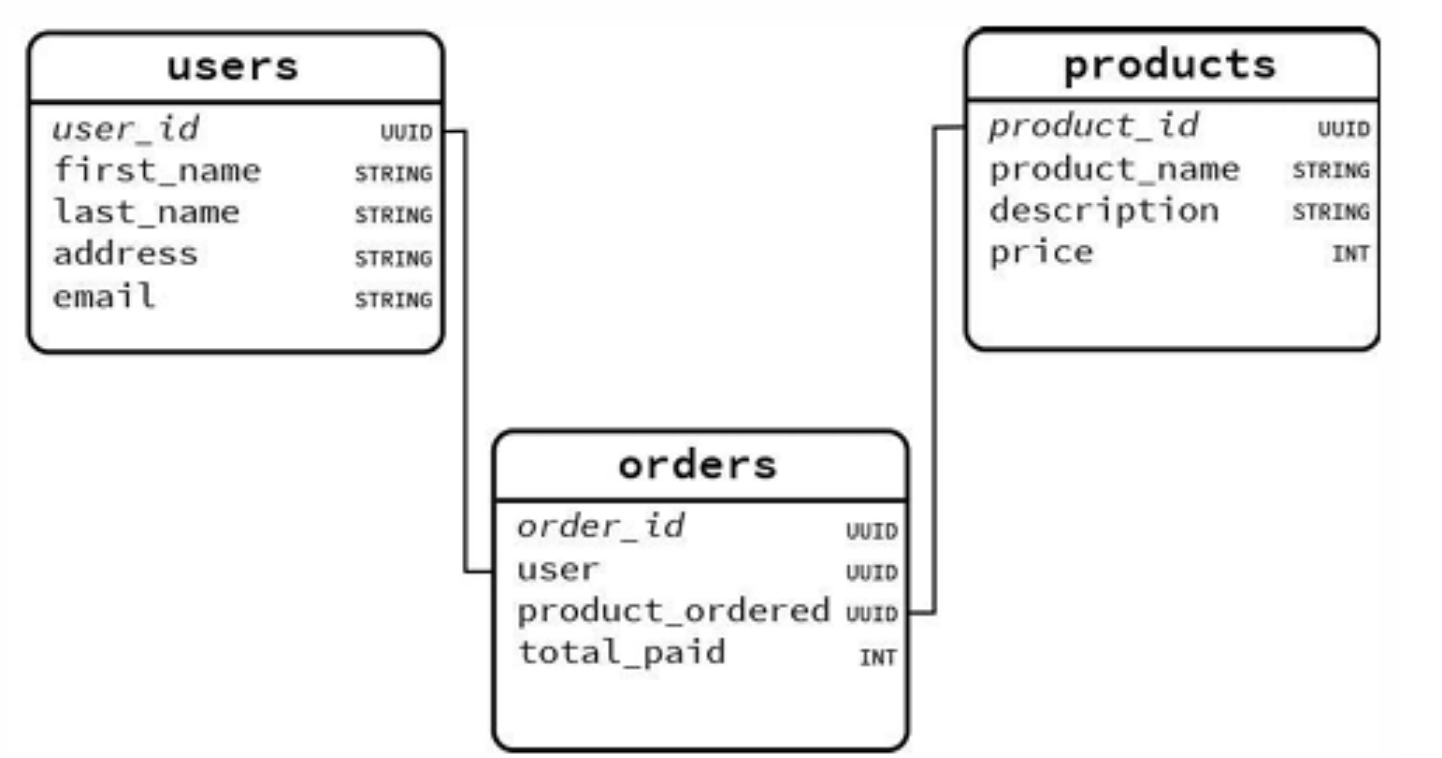
Database Engine Ranking

- Database engine
  - DBMS (Database Management System)
- A brief history of databases
- DB-Engines Ranking

SQL Database

- Relational database
- Organize data into `tables` of related information
- Utilize `Structured Query Language (SQL)` for managing/manipulating data

SQL Database



Popular RDBMS

- Open source: MySQL, PostgreSQL
- Commercial: Oracle Database, Microsoft SQL Server, IBM DB2
- RDBMS Ranking

SQL

The standard language used to interact with SQL databases

- Data Definition Language (DDL)
  - e.g., `CREATE TABLE`, `ALTER TABLE`, `DROP TABLE`
- Data Manipulation Language (DML)
  - e.g., `INSERT`, `UPDATE`, `DELETE`, `SELECT`
- Data Control Language (DCL)
  - e.g., `GRANT`, `REVOKE`

ACID Properties

- An acronym that stands for ...
  - `Atomicity`
  - `Consistency`
  - `Isolation`
  - `Durability`
- Ensure reliable transaction processing and data integrity
- What does ACID Means?

NoSQL

- `non SQL` or `not only SQL`
- Store data in a format other than relational tables

Types of NoSQL Database

- Document-oriented
- Column-oriented
- Graph-based
- Key-Value pair
- Time series

Document Database

- The data is stored in `document`
- Each `document` is typically a `nested structure` of `keys` and `values`
- **Possible to retrieve only parts of a document**
- The most commonly used data format are `JSON`, `BSON`, and `XML`
- e.g., MongoDB, Apache CouchDB

Document Database

(a) Embedded Data Model

```
{
  _id: <ObjectId1>,
  username: "123xyz",
  contact: {
              phone: "123-456-7890",
              email: "xyz@example.com"
            },
  access: {
             level: 5,
             group: "dev"
           }
}
```

Embedded sub-document

Embedded sub-document

(b) Normalized Data Model

user document

```
{
  _id: <ObjectId1>,
  username: "123xyz"
}
```

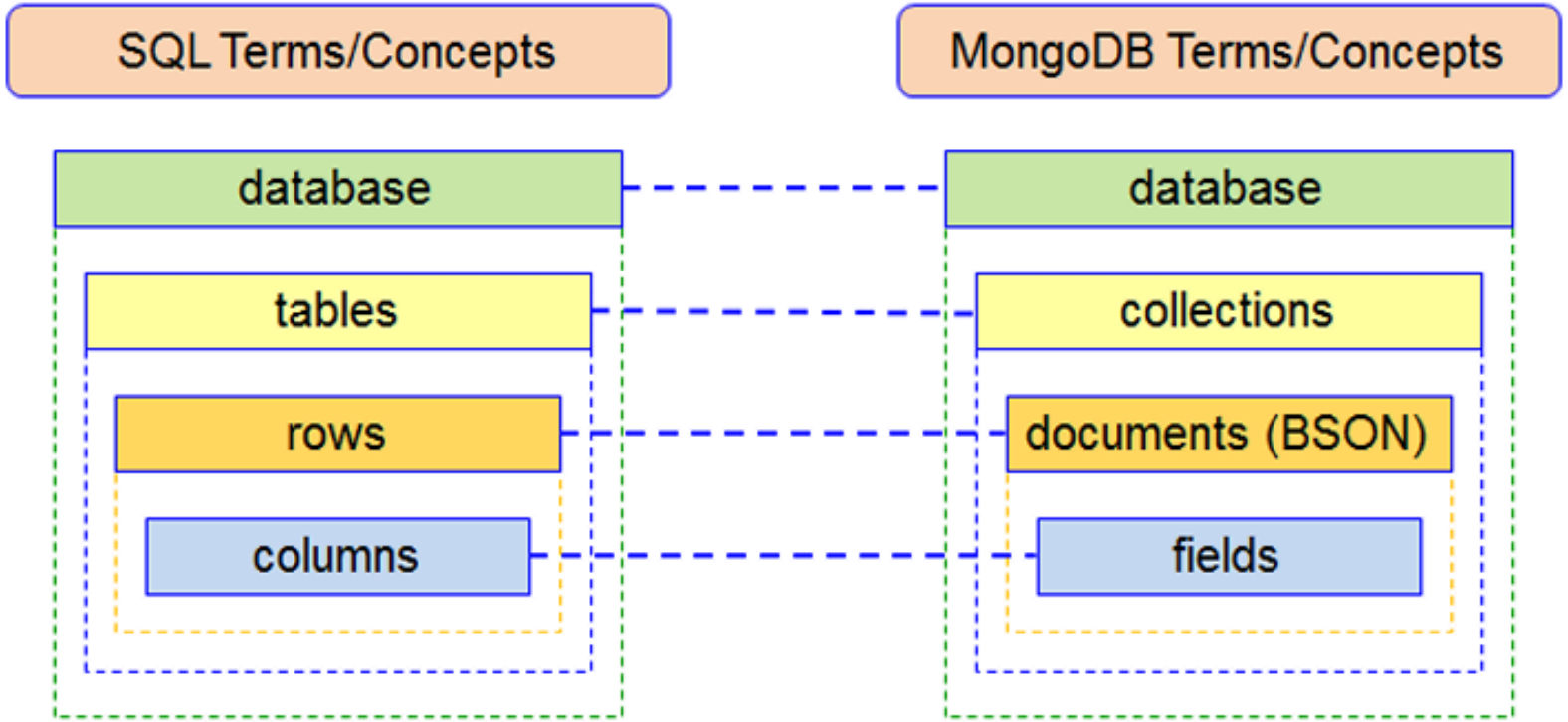
contact document

```
{
  _id: <ObjectId2>,
  user_id: <ObjectId1>,
  phone: "123-456-7890",
  email: "xyz@example.com"
}
```

access document

```
{
  _id: <ObjectId3>,
  user_id: <ObjectId1>,
  level: 5,
  group: "dev"
}
```

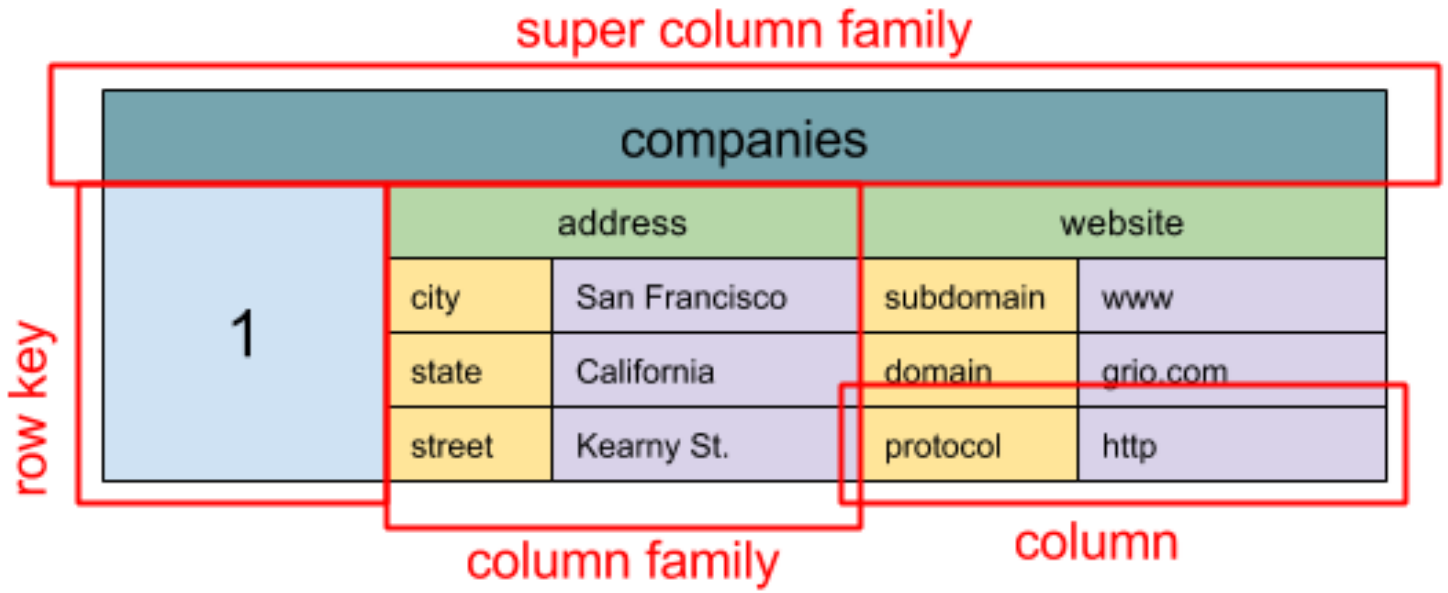
Document Database: Terminology



Wide Column Data Store

- Store data in columns rather than rows
- Able to store large amounts of data in a single column
- Allows to reduce disk resources and the time to retrieve information
- Highly scalable and flexible
- e.g., Apache Cassandra

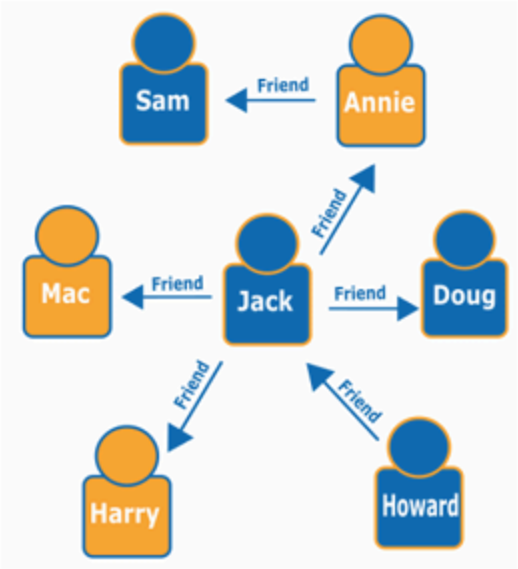
Wide Column Data Store



Graph Database

- Store and query highly connected data
- Data are modeled in the form of **entities** (`nodes`) and **relationships** (`edges`) between them
- Able to traverse from `nodes` or `edges` along defined relationship types until reaching some `defined condition`
  - Results : `lists` , `maps` , or graph `traversal path`
- e.g., Neo4j

Graph Database



Node Node

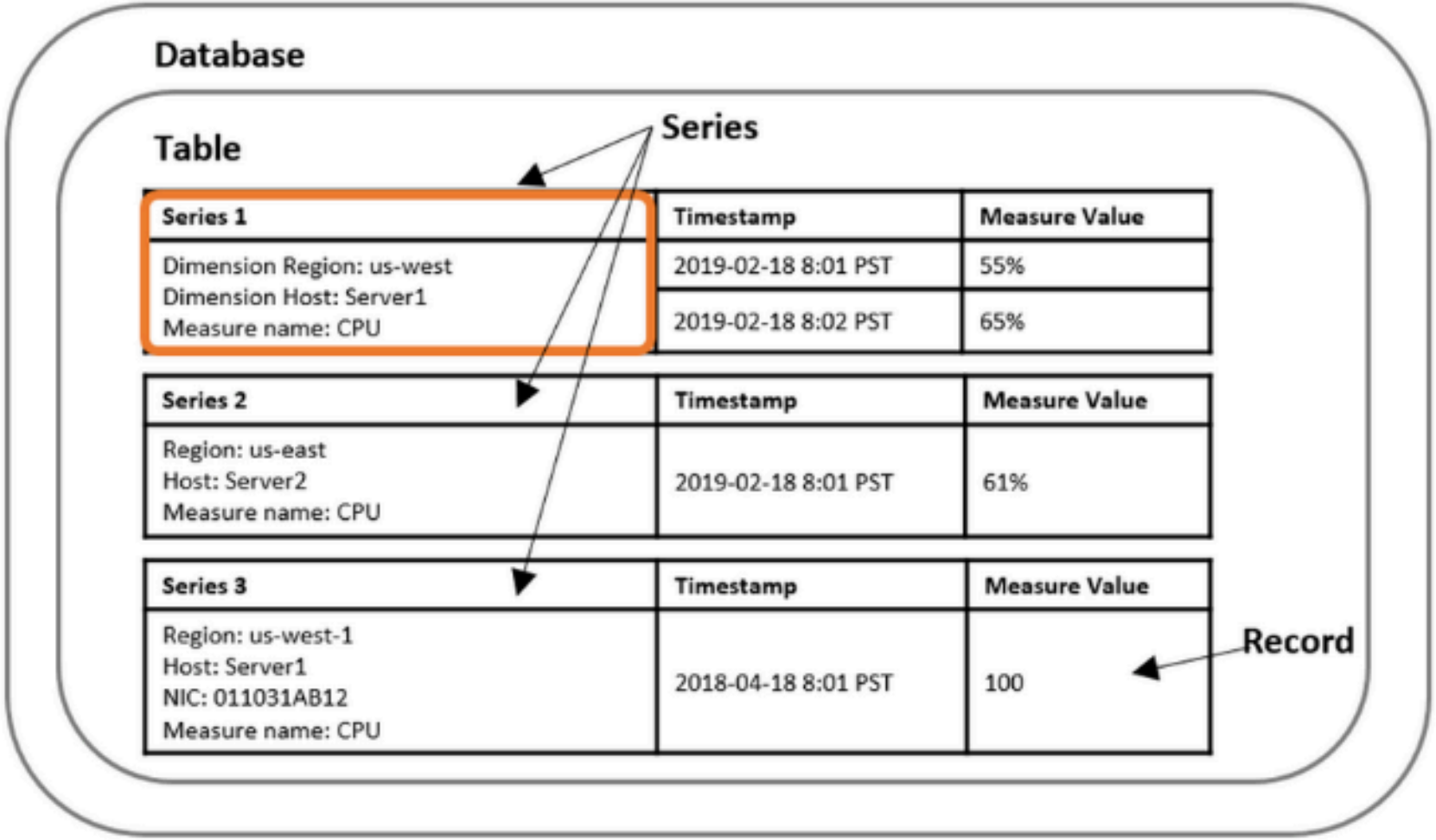
```
MATCH(:Person{name:"Dan"})-[:LOVES]→(:Person{name:"Ann"})
```

LABEL PROPERTY LABEL PROPERTY

Time Series Database

- Store and retrieve data records that are **sequenced by** `time`
  - Sets of data points associated with `timestamps` and stored in time sequence order
- Easy to measure how data change over time (e.g., IoT application)
- e.g., InfluxDB, Prometheus

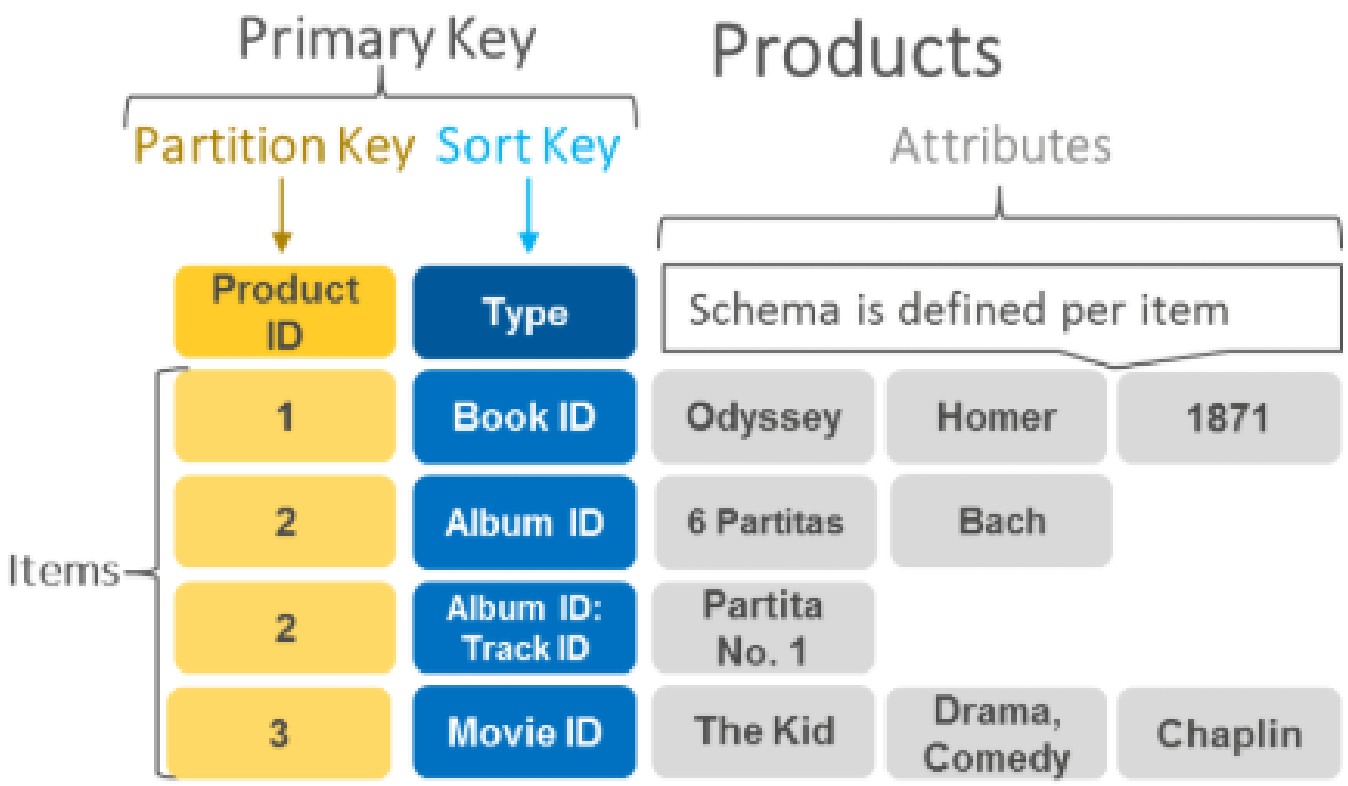
Time Series Database



Key-value Data Store

- Stores data as a collection of `key-value pairs`
- Each data item is identified by a `unique key`
- The `value` can be anything (string, number, object, ...)
- e.g., Redis, Memcached

Key-value Data Store



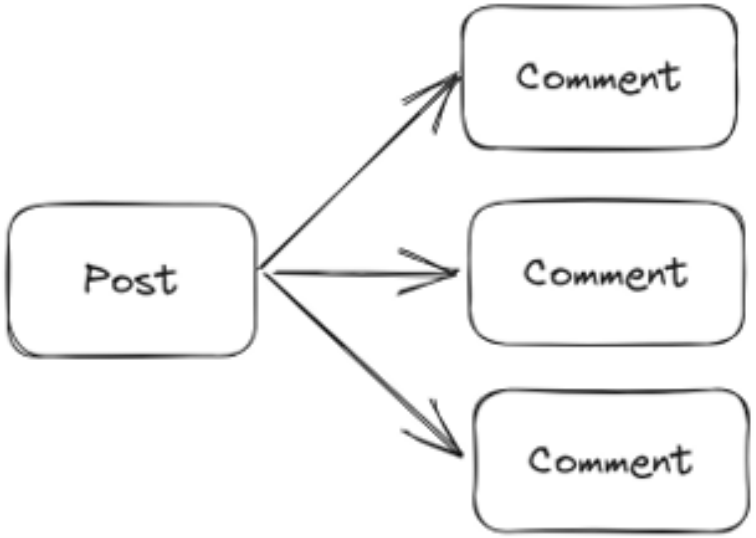
Database Schema

What is Database Schema?

- DB Schema defines how data is organized within the databasae
- Outlining how data is logically stored
- **Key components**:
  - Tables, Columns, Data types, Constraints
  - Primary / Foreign keys
  - Relationships (`one-to-one`, `one-to-many`, `many-to-many`)

Relationship : One-to-Many

e.g., **"Social media status post"**



- A `Post` may have many `comments`
- A `comment` belongs to only one `Post`

SQL Schema : One-to-Many

Post Table

#	Name	Datatype
1	id	INTEGER
2	ownerId	INTEGER
3	postText	TEXT
4	createdAt	TIMESTAMP
5	updatedAt	TIMESTAMP

Comment Table

#	Name	Datatype
1	id	INTEGER
2	postId	INTEGER
3	createdAt	TIMESTAMP
4	updatedAt	TIMESTAMP
5	commentText	TEXT

SQL Query : One-to-Many

e.g., **"Get a `Post` together with its `Comments`"**

Post

Id	postText
...	...
2	Good Evening
...	...

Comment

postId	commentText
...	...
2	Good Evening Too !
2	Hi BRO!
...	...

SELECT * FROM `Post` JOIN `Comment` ON `Post.Id` = `Comment.postId` ;

NoSQL Schema #1 : One-to-Many

Option 1 - Embedding `Comments` as array in `Post` document

- Assuming that a `Post` has <u>less than a hundred</u> `Comments`

```
{
  "_id": {…},
  "postId": 2,
  "postText": "Good evening!",
  "comments": [
    {
      "commentText": "Good evening Too!",
      "username": "bob"
    },
    {
      "commentText": "Good evening Bro!",
      "username": "Alice"
    }
  ]
}
```

NoSQL Schema #2 : One-to-Many

Option 2 - Reference to other collections, avoiding massive array

Post Collection

```
_id: ObjectId('649bd52d16b194f723bae06a')
postId: 2
postText: "Good evening!"
```

Comment Collection

```
_id: ObjectId('649bde5f16b194f723bae06e')
commentText: "Good evening Too!"
username: "bob"
postId: 2
```

```
_id: ObjectId('649bde9716b194f723bae06f')
commentText: "Good evening Bro!"
username: "Alice"
postId: 2
```

- Reference each `Comment` to a single `Post`
- What if a `Post` may have thousands of `Comments`

Summary : One-to-Many

SQL

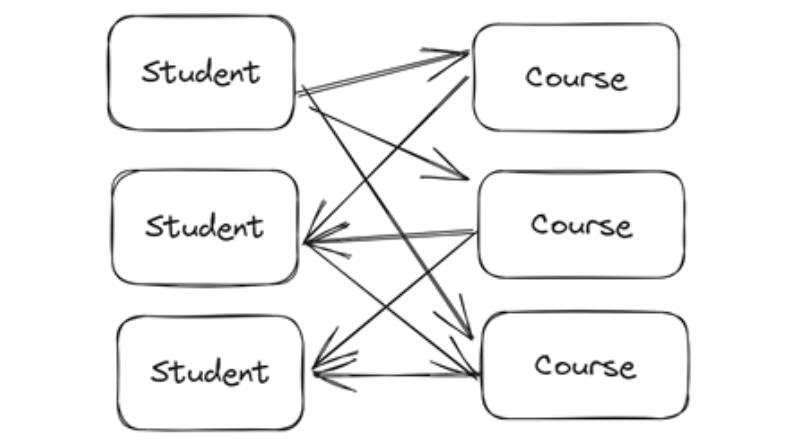
- Create two tables with a `foreign key` (representing a relationship)

NoSQL

- Embedding an `array of objects` in `another type of object`
- References multiple `objects` to `another type of object`

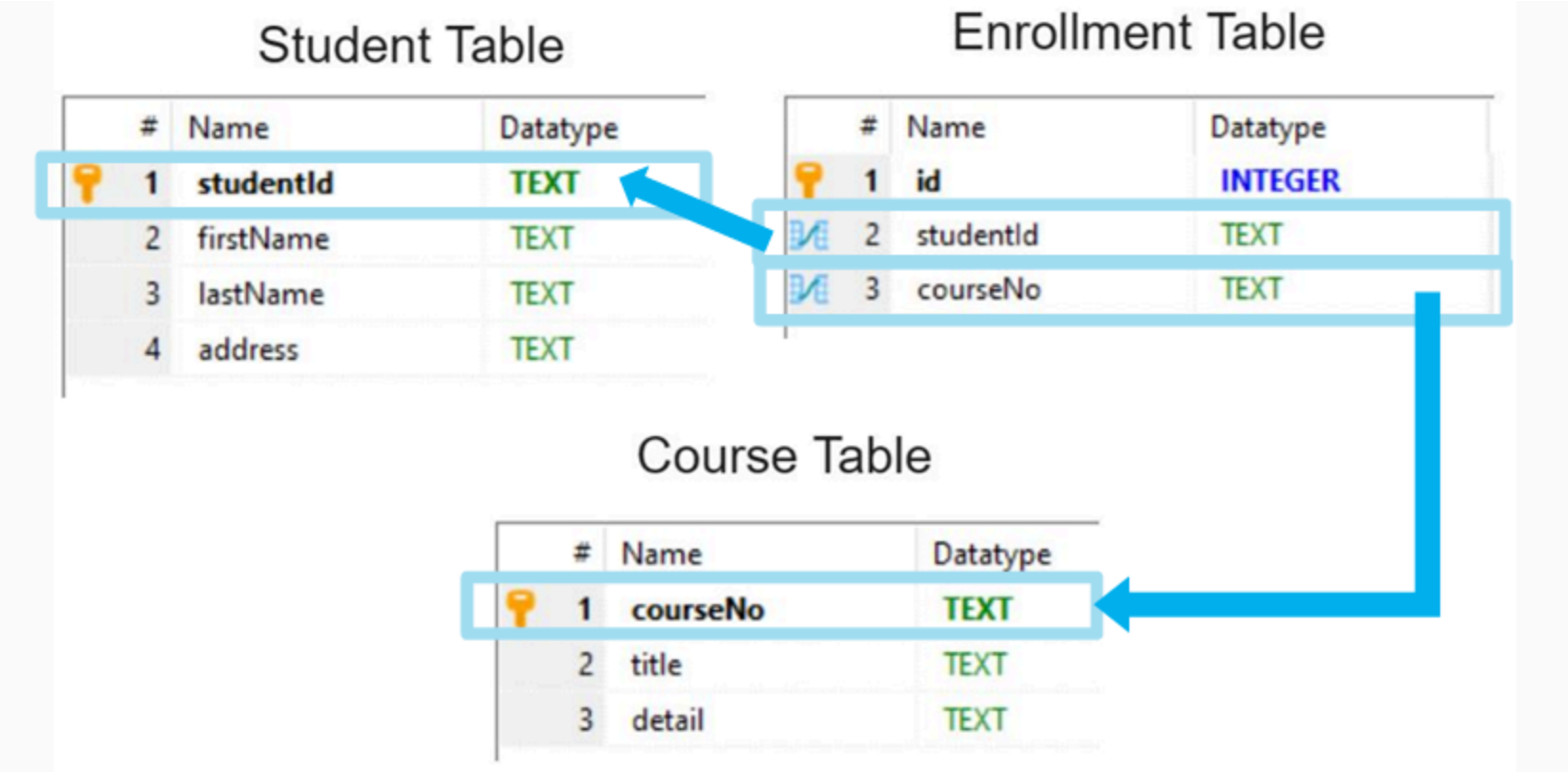
Relationship : Many-to-Many

e.g., **"Students Enrollment"**



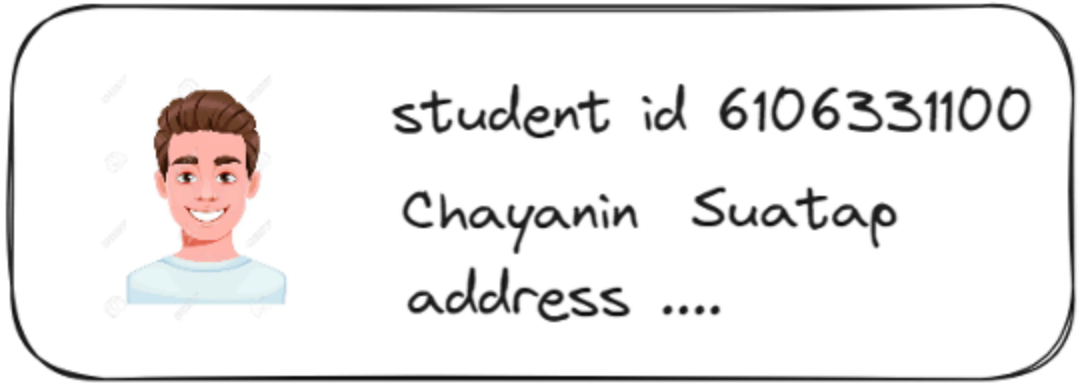
- A `Student` may enroll in multiple `Courses`
- A `Course` is enrolled by many `Students`

SQL Schema : Many-to-Many



SQL Query : Many-to-Many

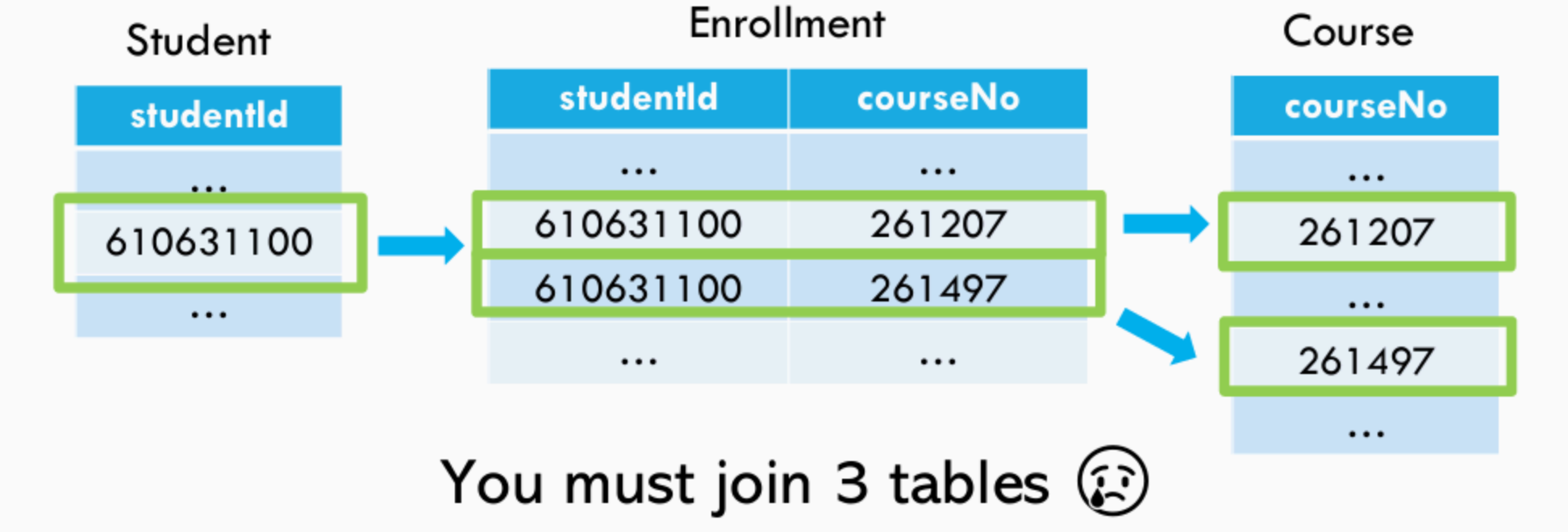
e.g., "**Get all** `Courses` **title enrolled by a** `Student` **with specified** `studentId` "



course no	course title
261207	BASIC COMP LAB
261497	FULL STACK DEV

SQL Query : Many-to-Many

e.g., **"Get all `Courses` title enrolled by a `Student` with specified `studentId`"**



NoSQL Schema #1 : Many-to-Many

Option 1 - Embedding a list of `Courses` in a `Student` document

Student

```
_id: ObjectId('649bfc8916b194f723bae07e')
studentId: "610631100"
firstName: "Chayanin"
lastName: "Suatap"
address: "Some place on earth"
▾ coursesEnrolled: Array
    0: "261207"
    1: "261497"
```

Course

```
_id: ObjectId('649bfd7e16b194f723bae07f')
courseNo: "261207"
title: "BASIC COMPUTER ENGINEERING LAB"
detail: "Teaching web development using React and JavaScript"
```

```
_id: ObjectId('649bfe3716b194f723bae080')
courseNo: "261497"
title: "FULL STACK DEVELOPMENT"
detail: "Teaching advance development and technologies"
```

NoSQL Query #1 : Many-to-Many

e.g., **"Which `Students` enroll in my `Course`"**

NoSQL Schema #2 : Many-to-Many

Option 2 - Embedding a list of `Students` in a `Course` document

Student

```
_id: ObjectId('649c036016b194f723bae082')
studentId: "610631100"
firstName: "Chayanin"
lastName: "Suatap"
address: "Some place on earth"
```

```
_id: ObjectId('649c037c16b194f723bae083')
studentId: "610631101"
firstName: "Po"
lastName: "-"
address: "China"
```

Course

```
_id: ObjectId('649c02ad16b194f723bae081')
courseNo: "261207"
title: "BASIC COMPUTER ENGINEERING LAB"
detail: "Teaching web development using React and JavaScript"
▾ students: Array
    0: "610631100"
    1: "610631102"
```

```
_id: ObjectId('649bfe3716b194f723bae080')
courseNo: "261497"
title: "FULL STACK DEVELOPMENT"
detail: "Teaching advance development and technologies"
▾ students: Array
    0: "610631101"
    1: "610631102"
```

What if we want both?

NoSQL Schema #3 : Many-to-Many

Option 3 - Embedding a list of `References` in both documents

Student

```
_id: ObjectId('649c5446b325ea1ba80bc21a')
studentId: "610631100"
firstName: "Chayanin"
lastName: "Suatap"
address: "Some place on earth"
▾ coursesEnrolled: Array
    0: "261207"
    1: "261497"
```

Course

```
_id: ObjectId('649c02ad16b194f723bae081')
courseNo: "261207"
title: "BASIC COMPUTER ENGINEERING LAB"
detail: "Teaching web development using React and JavaScript"
▾ students: Array
    0: "610631100"
    1: "610631102"
```

- `Pros` : query efficiently from both sides
- `Cons` : duplicate data, need to update on both side