UNIT 1 : CONCEPTS OF NOSQL : MONGODB

1.1 Concepts of NoSQL. Advantages and Features

1.1.1 MongoDB Data Types (String, Integer, Boolean, Double, Arrays, Objects)

1.1.2 Database Creation and Dropping Database

1.2 Create and Drop Collection

1.3 CRUD Operations (Insert, Update, Delete, Find, Query and Projection Operators)

1.4 Operators (Projections, Update, Limit(), sort() and Aggregation Commands)

1.1 Concepts of NoSQL. Advantages and Features

1.1.1 MongoDB Data Types (String, Integer, Boolean, Double, Arrays, Objects)

1.1.2 Database Creation and Dropping Database

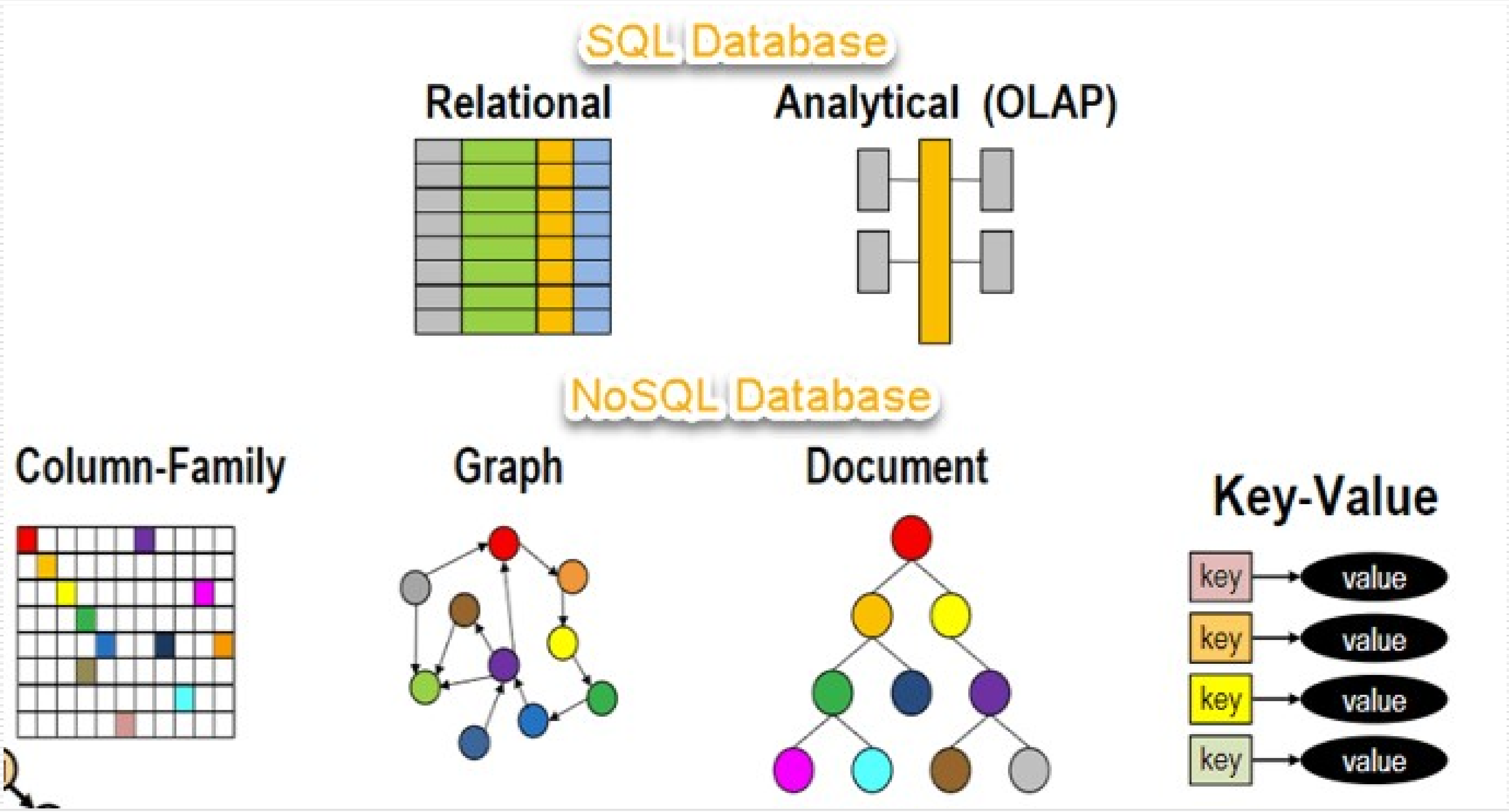
Introduction

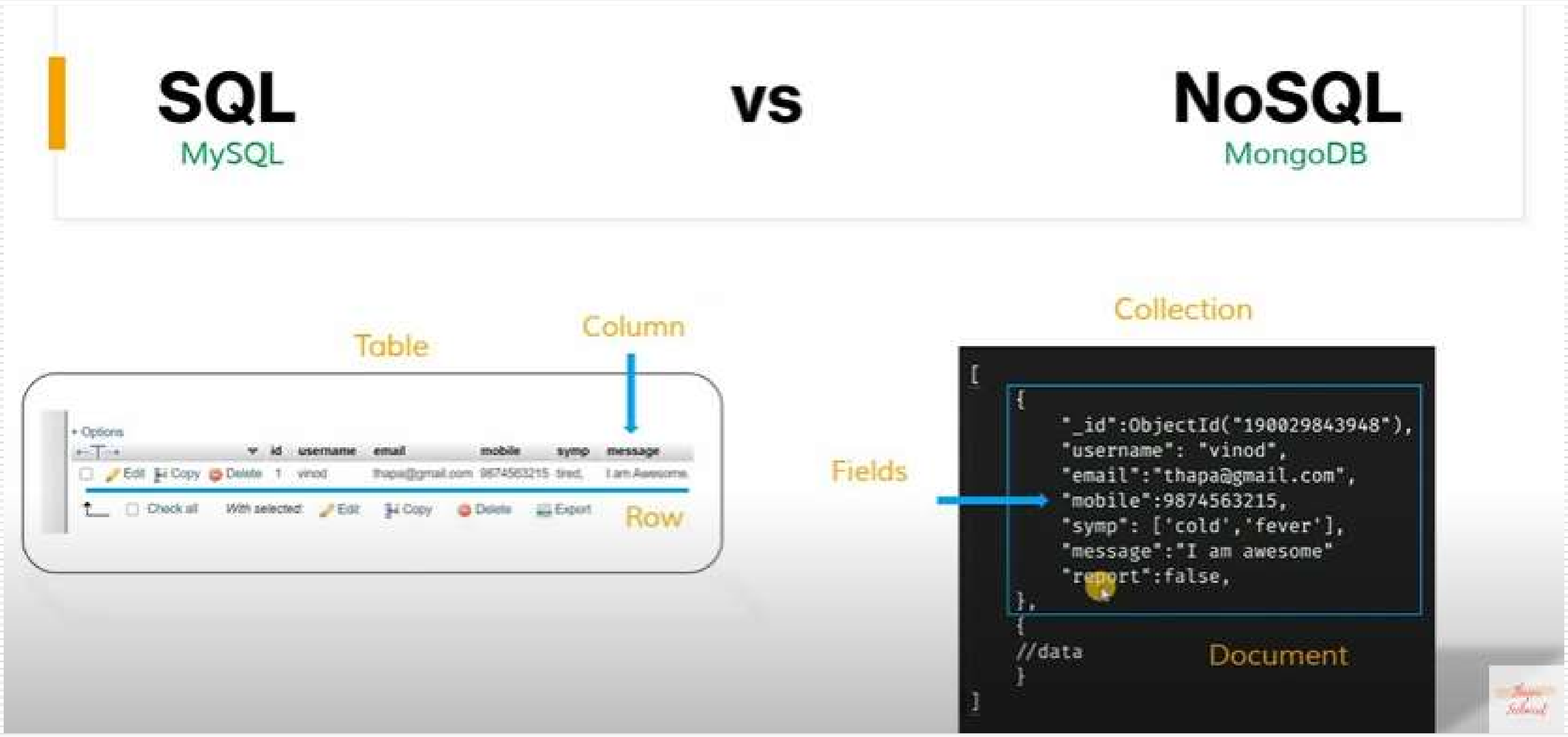
- A **NoSQL (Not Only SQL or Not SQL)** originally referring to **non SQL** or **non relational** is a database that provides a mechanism for storage and retrieval of data.
- This data is **modeled**, it means other than the **tabular relations** used in relational databases.
- **NoSQL are non-tabular databases and store data differently than relational tables**. NoSQL databases come in a variety of types based on their data model. The main types are document, key-value, wide-column, and graph.

- **Carl Strozzi** introduced the NoSQL concept in **1998**.
- Traditional RDBMS uses SQL syntax to store and retrieve data for further insights. Instead, a NoSQL database system encompasses a wide range of database technologies that can store structured, semi-structured, unstructured and polymorphic data.

Models of NOSQL Database

- Key-Value pair-based databases.
- Column-based databases.
- Document-oriented databases.
- Graph databases.


SQL Database
Relational
Analytical (OLAP)
NoSQL Database
Column-Family
Graph
Document
Key-Value
key
value
key
value
key
value
key
value



SQL vs NoSQL

MySQL

MongoDB

	id	username	email	mobile	symp	message
Edit Copy Delete	1	vinod	thapa@gmail.com	9874563215	tired,	I am Awesome

```
[
  {
      "_id":ObjectId("190029843948"),
      "username": "vinod",
      "email":"thapa@gmail.com",
      "mobile":9874563215,
      "symp": ['cold','fever'],
      "message":"I am awesome"
      "report":false,
  },
  {
  //data
  }
]
```

- The concept of NoSQL databases became popular with **Internet giants like Google, Facebook, Amazon,** etc. who deal with **huge volumes of data**. The system response time becomes slow when you use RDBMS for massive volumes of data.
- To resolve this problem, we could "**scale up**" our systems by upgrading our existing hardware. This process is expensive.

History

- 1998- Carlo Strozzi use the term NoSQL for his lightweight, open-source relational database
- 2000- Graph database Neo4j is launched
- 2004- Google BigTable is launched
- 2005- CouchDB is launched
- 2007- The research paper on Amazon Dynamo is released
- 2008- Facebooks open sources the Cassandra project
- 2009- The term NoSQL was reintroduced

Advantages & Features

1. Support for Multiple Data Models
2. Easily Scalable via Peer-to-Peer Architecture
3. Flexibility: Versatile Data Handling
4. Distribution Capabilities
5. Zero Downtime

1. Support for Multiple Data Models

- Where **relational databases require data to be put into tables and columns** to be accessed and analyzed, the **various data model capabilities of NoSQL databases make them extremely flexible** when it comes to handling data.
- They can ingest **structured, semi-structured, and unstructured data** with equal ease, whereas relational databases are **extremely rigid, handling primarily** structured data.
- Developers and architects **choose a NoSQL database to more easily handle different agile application** development requirements.

2. Easily Scalable via Peer-to-Peer Architecture

- It's not that **Relational Database can't scale**, it's that they can't scale EASILY or CHEAPLY, and that's because they're built with a traditional **master-slave architecture**, which means scaling UP via bigger and bigger hardware servers as opposed to OUT or worse via sharding.
- **Sharding means dividing a database into smaller chunks across multiple hardware servers** instead of a single large server, and this leads to operational administration headaches. Instead, look for a NoSQL database with a masterless, peer-to-peer architecture with all nodes being the same.

3. Flexibility: Versatile Data Handling

- Where relational databases require data to be put into **tables and columns** to be accessed and analyzed, the multi-model capabilities of NoSQL databases make them **extremely flexible** when it comes to handling data.
- They can **easily process structured, semi-structured, and unstructured data**, while relational databases, are **designed to handle primarily structured data**.

4. Distribution Capabilities

- Look for a NoSQL database that is **designed to distribute data at global scale**, meaning it can use **multiple locations** involving multiple data centers and/or cloud regions for write and read operations. Relational databases, in contrast, use a centralized application that is location-dependent (e.g. single location), especially for write operations.
- A key advantage of using a distributed database with a masterless architecture is that you can **maintain continuous availability** because data is distributed with multiple copies where it needs to be.

5. Zero Downtime

- The final but certainly no less important key feature to seek in a NoSQL database is zero downtime. This is made possible by a **masterless architecture**, which allows for multiple copies of data to be **maintained across different nodes**.
- **If a node goes down, no problem: another node has a copy of the data for easy, fast access.** When one considers the cost of downtime, this is a big deal.

MongoDB

- MongoDB, the **most popular NoSQL database**, is an **open-source document-oriented database**. The term ‘NoSQL’ means ‘non-relational’. It means that **MongoDB isn’t** based on the **table-like relational database** structure but provides an all together different mechanism for **storage and retrieval** of data. This format of storage is called **BSON** (similar to JSON format).

- SQL databases store data in **tabular format**. This data is stored in a **predefined data model which is not very much flexible for today's real-world highly growing applications**. Modern applications are **more networked**, **social** and **interactive** than ever. Applications are storing more and more data and are accessing it at higher rates.

Download Link for MongoDB

- SHELL
  - https://downloads.mongodb.com/compass/mongosh-1.4.2-x64.msi
- UI
  - https://fastdl.mongodb.org/windows/mongodb-windows-x86_64-5.0.8-signed.msi
  - https://fastdl.mongodb.org/windows/mongodb-windows-x86_64-5.0.9-signed.msi

1.1.1 MongoDB Data Types

- String
- Integer
- Boolean
- Double
- Arrays
- Objects

- **String** – This is the most commonly used datatype to store the data. String in MongoDB must be UTF-8 valid.

- **Integer** – This type is used to store a numerical value. Integer can be 32 bit or 64 bit depending upon your server.
- **Boolean** – This type is used to store a boolean (true/ false) value.
- **Double** – This type is used to store floating point values.

- **Arrays** – This type is used to store arrays or list or multiple values into one key.
- **Object** – This datatype is used for embedded documents.

SQL	**NOSQL**
DATABASE	DATABASE
TABLES	COLLECTION
ROWS	DOCUMENTS
COLUMNS	FIELDS

1.1.2 Database Creation and Dropping Database

- To Show all Databases
  - show dbs;
- To create new Database
  - use db_name;
  - To create a new database. If database does not exist it will create new database, if database exist then it will switch to particular database.

- To Drop Database
  - db.dropDatabase()
  - To drop database first use the database which you want to drop.
- To check current database
  - db

Collection

- A collection is **a grouping of MongoDB documents**.
- Documents within a collection can have **different fields**.
- A collection is the **equivalent** of a **table** in a relational database system.
- A **collection exists within a single** database.

1.2 Create and Drop Collection

- Create Collection
  - db.createCollection(name,option)
- Drop Collection
  - db.collection_name.drop()

Example:

- db.createCollection("emp_mst");
- db.emp_mst.insert({name:"Hardik Pandya",age:28,type:"All Rounder"});
- db.emp_mst.find();
- db.emp_mst.find().pretty();
- db.emp_mst.drop();

1.3 CRUD Operations

- Insert
- Update
- Delete
- Find
- Query
- Projection Operators

insert()

- To insert document in collection use insert() method.

```
db.collection.insert(
{
    Field1 : value1,
    Filed2 : value2
    ....
}
)
```

Example:

- db.stud.insert({"name":“Rohit","type":"All Rounder","age":33});

update()

- MongoDB's **update()** and **save()** methods are used to update document into a collection. The update() method updates the values in the existing document while the save() method replaces the existing document with the document passed in save() method.

```
db.staff.update(
    {‘field’:’value’},
    {$set:{field:’newvalue’}
    }
    )
```

Example:

- db.stud.update({"age":35},{$set:{ age:33}})

delete

- MongoDB's **remove()** method is used to remove a document from the collection. remove() method accepts two parameters. One is deletion criteria and second is justOne flag.
- **deletion criteria** – (Optional) deletion criteria according to documents will be removed.
- **justOne** – (Optional) if set to true or 1, then remove only one document.
- Syntax:
  - db.COLLECTION_NAME.remove(DELLETION_CRITTERIA)

Example:

- db.stud.remove({"age":33})

find()

- Syntax:
  - db.COLLECTION_NAME.find({},{KEY:1})

Query Operators

- **Comparison**
- **Logical**
- **Element**
- **Evaluation**
- **Array**
- **Bitwise**

Comparison

Name	Description
$eq	Matches values that are equal to a specified value.
$gt	Matches values that are greater than a specified value.
$gte	Matches values that are greater than or equal to a specified value.
$in	Matches any of the values specified in an array.
$lt	Matches values that are less than a specified value.
$lte	Matches values that are less than or equal to a specified value.
$ne	Matches all values that are not equal to a specified value.
$nin	Matches none of the values specified in an array.

- db.stud.find({age:{$eq:32}}).pretty();
- db.stud_mst.find({stud_id:{$gte:2}}).pretty();
- db.stud_mst.find({stud_id:{$in:[1 , 2]}}).pretty();
- db.stud_mst.find({stud_id:{$ne:2}}).pretty();

Logical

Name	Description
$and	Joins query clauses with a logical AND returns all documents that match the conditions of both clauses.
$not	Inverts the effect of a query expression and returns documents that do *not* match the query expression.
$nor	Joins query clauses with a logical NOR returns all documents that fail to match both clauses.
$or	Joins query clauses with a logical OR returns all documents that match the conditions of either clause.

Example:

- db.stud_mst.find({$and:[{stud_fname:{$eq:"Akshay"}},{stud_lname:{$eq:"Kumar"}}]}).pretty()
- db.stud_mst.find({$or:[{stud_fname:{$eq:"Akshay"}},{stud_lname:{$eq:"Sharma"}}]}).pretty()
- db.stud_mst.find({$nor:[{stud_fname:{$eq:"Akshay"}},{stud_lname:{$eq:"Sharma"}}]}).pretty()

Element

Name	Description
$exists	Matches documents that have the specified field.
$type	Selects documents if a field is of the specified type.

Evaluation

Name	Description
$expr	Allows use of aggregation expressions within the query language.
$jsonSchema	Validate documents against the given JSON Schema.
$mod	Performs a modulo operation on the value of a field and selects documents with a specified result.
$regex	Selects documents where values match a specified regular expression.
$text	Performs text search.
$where	Matches documents that satisfy a JavaScript expression.

Array

Name	Description
$all	Matches arrays that contain all elements specified in the query.
$elemMatch	Selects documents if element in the array field matches all the specified $elemMatch conditions.
$size	Selects documents if the array field is a specified size.

Bitwise

Name	Description
$bitsAllClear	Matches numeric or binary values in which a set of bit positions *all* have a value of 0.
$bitsAllSet	Matches numeric or binary values in which a set of bit positions *all* have a value of 1.
$bitsAnyClear	Matches numeric or binary values in which *any* bit from a set of bit positions has a value of 0.
$bitsAnySet	Matches numeric or binary values in which *any* bit from a set of bit positions has a value of 1.

1.4 Operators

- Projections
- Update()
- Limit()
- sort()
- Aggregation Commands

Projection

- In MongoDB, projection means selecting only the necessary data rather than selecting whole of the data of a document. If a document has 5 fields and you need to show only 3, then select only 3 fields from them.

Name	Description
$	Projects the first element in an array that matches the query condition.
$elemMatch	Projects the first element in an array that matches the specified $elemMatch condition.
$meta	Projects the document's score assigned during $text operation.
$slice	Limits the number of elements projected from an array. Supports skip and limit slices.

- To Display all documents
  - db.collection.find()
- To display all documents in pretty format
  - db.collection.find().pretty()
- To display specific data
  - db.collection.find({field:’value’})

- To remove id field
  - **db.emp_mst.find({},{_id:0});**

Update

- Same as discussed in CRUD Operation

Limit()

- To limit the records in MongoDB, you need to use **limit()** method. The method accepts one number type argument, which is the number of documents that you want to be displayed.
- Syntax : db.COLLECTION_NAME.find().limit(NUMBER)

Example:

```
db.stud.find().pretty().limit(1);
```

sort()

- To sort documents in MongoDB, you need to use **sort()** method. The method accepts a document containing a list of fields along with their sorting order. To specify sorting order 1 and -1 are used. 1 is used for ascending order while -1 is used for descending order.
- Syntax:

```
db.COLLECTION_NAME.find().sort({KEY:1})
```

Example:

- db.staff.find().sort({name:1}).pretty()

Aggregation Commands

- Aggregations operations process data records and return computed results.
- Aggregation operations group values from multiple documents together, and can perform a variety of operations on the grouped data to return a single result.
- In SQL count(*) and with group by is an equivalent of MongoDB aggregation.
- Syntax: db.COLLECTION_NAME.aggregate(AGGREGATE_OPERATION)

Name	Description
aggregate	Performs aggregation tasks such as $group using an aggregation pipeline.
count	Counts the number of documents in a collection or a view.
distinct	Displays the distinct values found for a specified key in a collection or a view.

Example:

- db.stud.aggregate([{$count:"myCount"}])
- db.runCommand({distinct:"stud",key:"name"});