Databases

sql || no-sql

what is a database?

- it is a **collection of data**, typically describing the activities of one (or more related) application(s)
- the goal is to organize data in a way that facilitates **efficient *retrieval* and *modification***
- **note:** the data maintained by a system are much more important/valuable than the system itself
- A **database management system** (DBMS) is a software program to assist in maintaining and utilizing large databases

advantages of using a dbms

- data independence
- efficient data access
- data integrity and security
- data administration
- concurrent access and crash recovery
- reduced application development time

more on data independence

- **Idea:** application programs are isolated from changes in the way the data is structured & stored.
  - Indirect access supports:
    - advanced data structures
    - data restructuring
    - distribution and load balancing,
    - …
    - all without changes to applications

  - **Note:** A very important advantage of using a DBMS!

more on data independence

- **Logical**: applications immune from changes in the logical structure of the data.
  - Example:
    - Student (name: string, major: string, DOB: integer)
    - …
    - …
- **Physical**: applications immune from physical storage details.
  - Such as the file structure and the choice of indexes

more on relational model

Idea. *All information is organized in flat relations.*

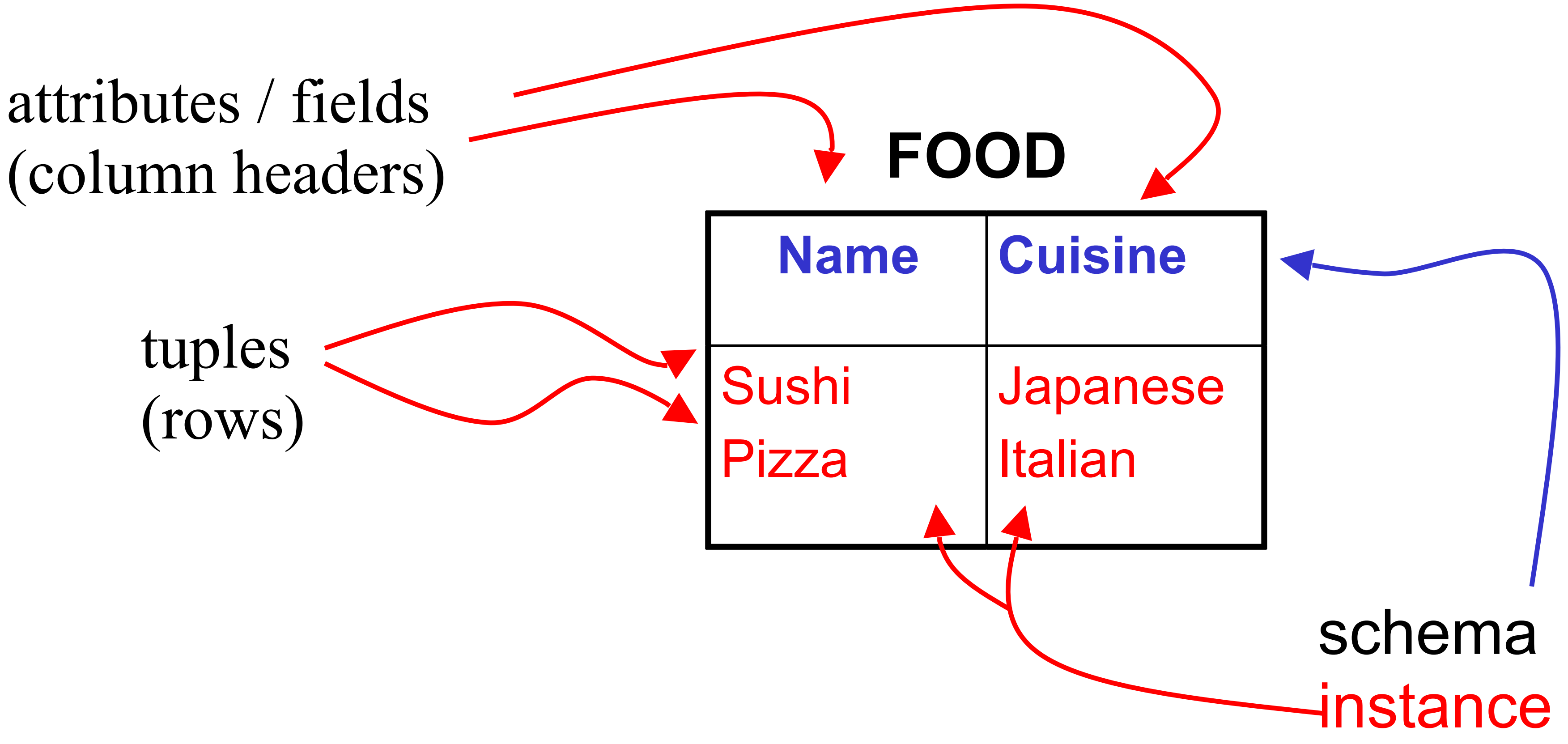
- Features:
  - very simple and clean data model
  - *often* matches how we think about data
  - abstract model that underlies SQL, the most popular database language
  - powerful and *declarative* query/update languages
  - semantic integrity constraints

transaction

A **transaction** is any ***one execution*** of a process in a DBMS, which is seen as a series of **actions**—such as *reads* and *writes*, followed by a *commit* or an *abort*.

- Properties of transactions: **(ACID)**
  - **Atomic:** either all actions or nothing are carried out.
  - **Consistency:** must preserve the DB constraints.
  - **Isolation:** understandable without considering other transactions.
  - **Durability:** once committed, the changes made are permanent.

a relation is a table



FOOD

Name	Cuisine
Sushi	Japanese
Pizza	Italian

more tabular form

FOOD

<u>Name</u>	Cuisine
Pizza	Italian
Stroganoff	Russian
Poutine	Canadian

STUDENT

<u>ID</u>	Name	Major
1022083920	Adam	Math
901183280	Saniya	CS

LIKES

<u>Student</u>	<u>Food</u>
1022083920	Pizza
1022083920	Poutine
901183280	Pizza

that's why relations are often called "tables".

SQL examples

-
  ```
  INSERT INTO food VALUES ( “Pizza”, “Canadian” );
  ```

-
  ```
  UPDATE food SET cuisine = “Italian”
     WHERE name = “Pizza”;
  ```

-
  ```
  SELECT name FROM food
     WHERE cuisine = “Russian”;
  ```

-
  ```
  SELECT cuisine, COUNT(*) AS “count”
     FROM food
     GROUP BY cuisine;
  ```

-
  ```
  SELECT DISTINCT cuisine
     FROM food,
              (SELECT food as name FROM likes,student
               WHERE  major=“CS”) csLikes
     WHERE food.name=csLikes.name;
  ```

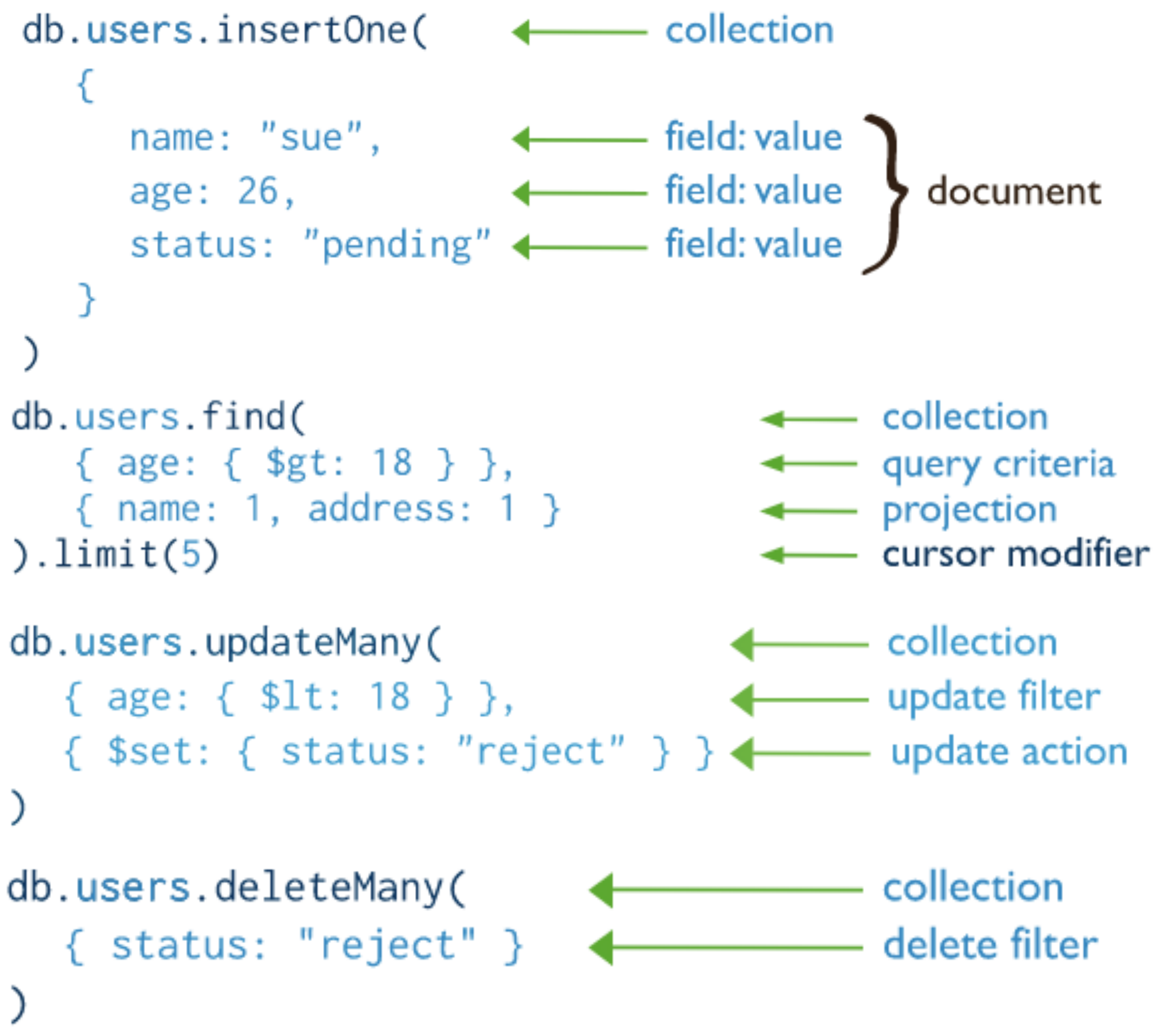
MongoDB

no-sql

what is MongoDB?

- Document-oriented NoSQL database
- Documents are JSON-like
- Documents are stored in Collections. Collection are created upon inserting first Document
- Full-featured queries and advanced MapReduce operations
- Javascript can be used as part of some queries
- Open-Source
- Robust scalability features

CRUD



```
db.users.insertOne(
   {
      name: "sue",
      age: 26,
      status: "pending"
   }
)
```

```
db.users.find(
   { age: { $gt: 18 } },
   { name: 1, address: 1 }
).limit(5)
```

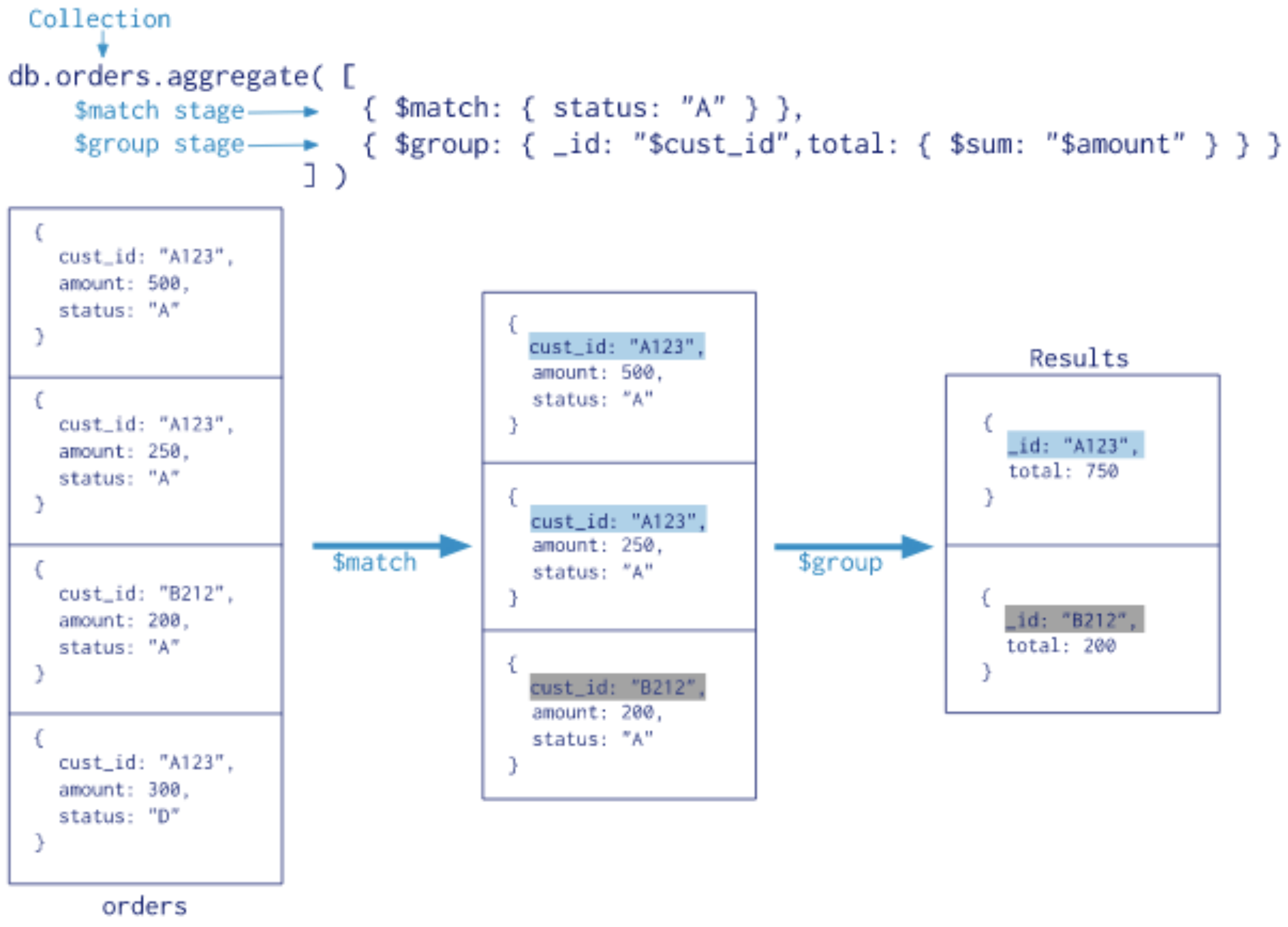
```
db.users.updateMany(
   { age: { $lt: 18 } },
   { $set: { status: "reject" } }
)
```

```
db.users.deleteMany(
   { status: "reject" }
)
```

CRUD

- MongoDB guarantees atomicity at the Document level
- There is a rich set of operators available to work with complex Documents, allowing to query deep values in their schema
- While there is no concept of a transaction, a database can be locked ($isolated)
- Or a Two-Phase Commit approach can be implemented, since updates to a document can be conditional on a value plus atomic.
- MongoDB package provides slightly different syntax than the one supported by the command line tool.

Aggregation



```
db.orders.aggregate( [
      { $match: { status: "A" } },
      { $group: { _id: "$cust_id",total: { $sum: "$amount" } } }
] )
```

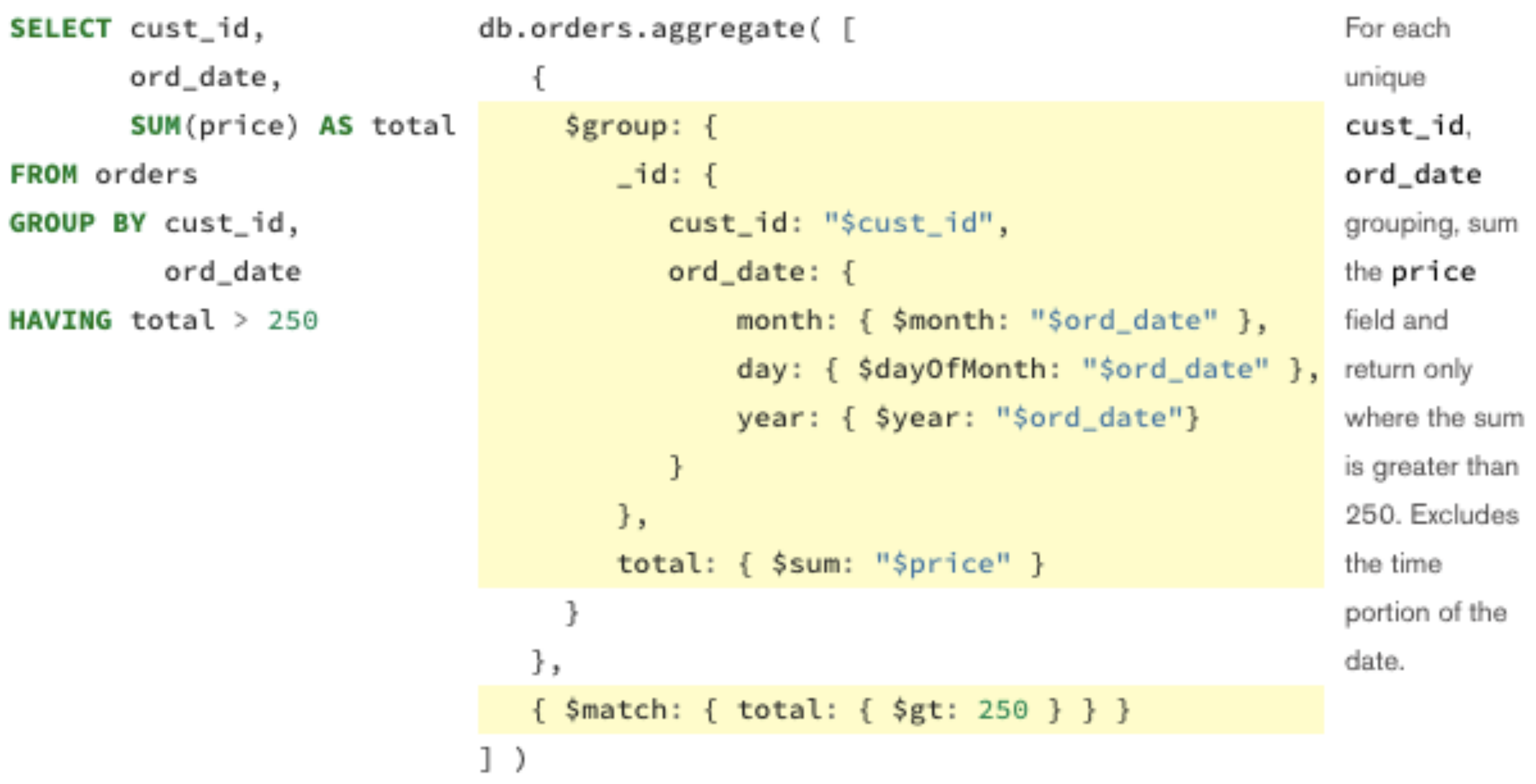
```
{
  cust_id: "A123",
  amount: 500,
  status: "A"
}
{
  cust_id: "A123",
  amount: 250,
  status: "A"
}
{
  cust_id: "B212",
  amount: 200,
  status: "A"
}
{
  cust_id: "A123",
  amount: 300,
  status: "D"
}
```

```
{
  cust_id: "A123",
  amount: 500,
  status: "A"
}
{
  cust_id: "A123",
  amount: 250,
  status: "A"
}
{
  cust_id: "B212",
  amount: 200,
  status: "A"
}
```

```
{
  _id: "A123",
  total: 750
}
{
  _id: "B212",
  total: 200
}
```

Aggregation

- MongoDB offers a very powerful aggregation concept modelled after pipelines.

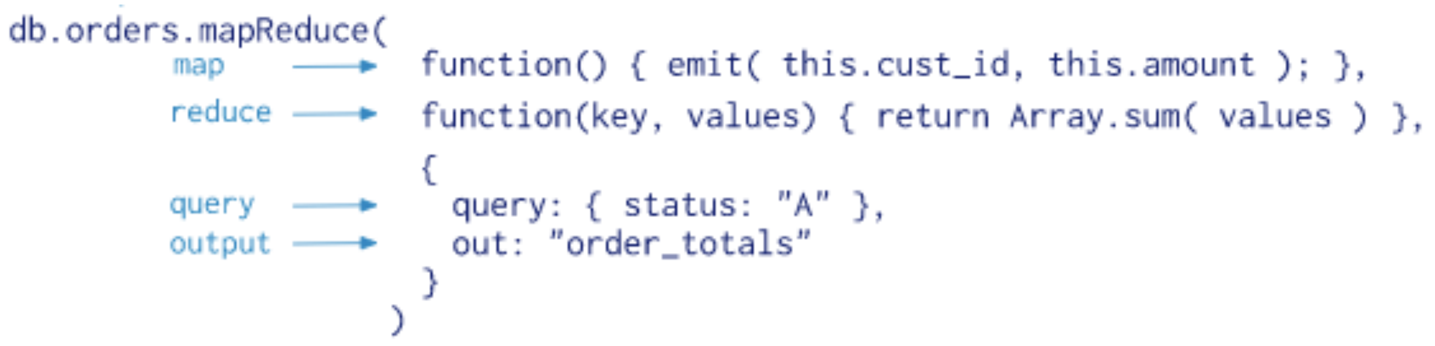


```
SELECT cust_id,
       ord_date,
       SUM(price) AS total
FROM orders
GROUP BY cust_id,
         ord_date
HAVING total > 250
```

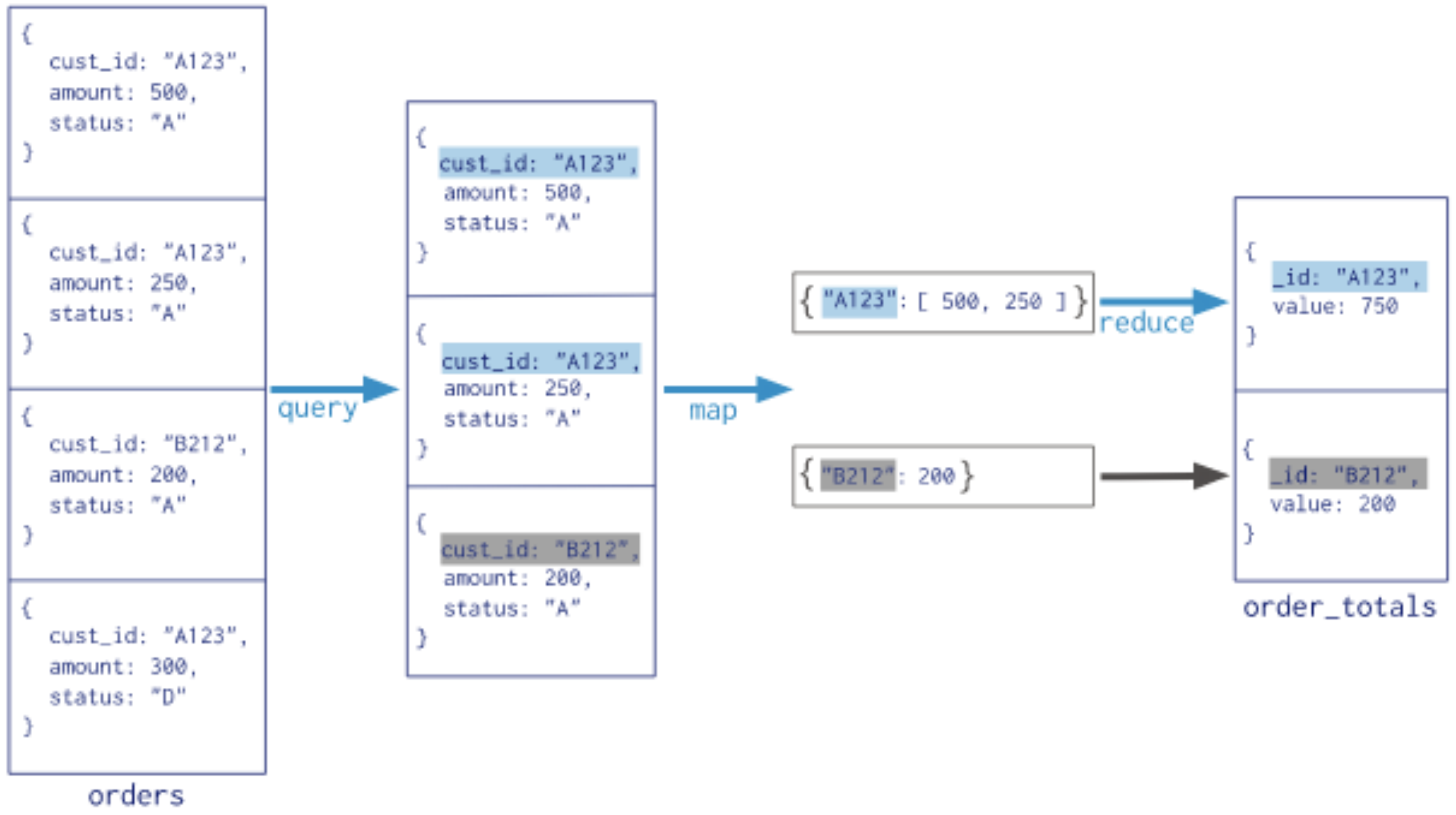
```
db.orders.aggregate( [
   {
     $group: {
        _id: {
           cust_id: "$cust_id",
           ord_date: {
               month: { $month: "$ord_date" },
               day: { $dayOfMonth: "$ord_date" },
               year: { $year: "$ord_date"}
           }
        },
        total: { $sum: "$price" }
     }
   },
   { $match: { total: { $gt: 250 } } }
] )
```

For each unique `cust_id`, `ord_date` grouping, sum the `price` field and return only where the sum is greater than 250. Excludes the time portion of the date.

MapReduce



```
db.orders.mapReduce(
        map    →    function() { emit( this.cust_id, this.amount ); },
        reduce →    function(key, values) { return Array.sum( values ) },
                    {
        query  →      query: { status: "A" },
        output →      out: "order_totals"
                    }
                   )
```



{ cust_id: "A123", amount: 500, status: "A" }
{ cust_id: "A123", amount: 250, status: "A" }
{ cust_id: "B212", amount: 200, status: "A" }
{ cust_id: "A123", amount: 300, status: "D" }
orders
query
{ cust_id: "A123", amount: 500, status: "A" }
{ cust_id: "A123", amount: 250, status: "A" }
{ cust_id: "B212", amount: 200, status: "A" }
map
{ "A123": [500, 250] }
{ "B212": 200 }
reduce
{ _id: "A123", value: 750 }
{ _id: "B212", value: 200 }
order_totals