SQL to MongoDB Crash Course

Dr. Sucheta Ghosh

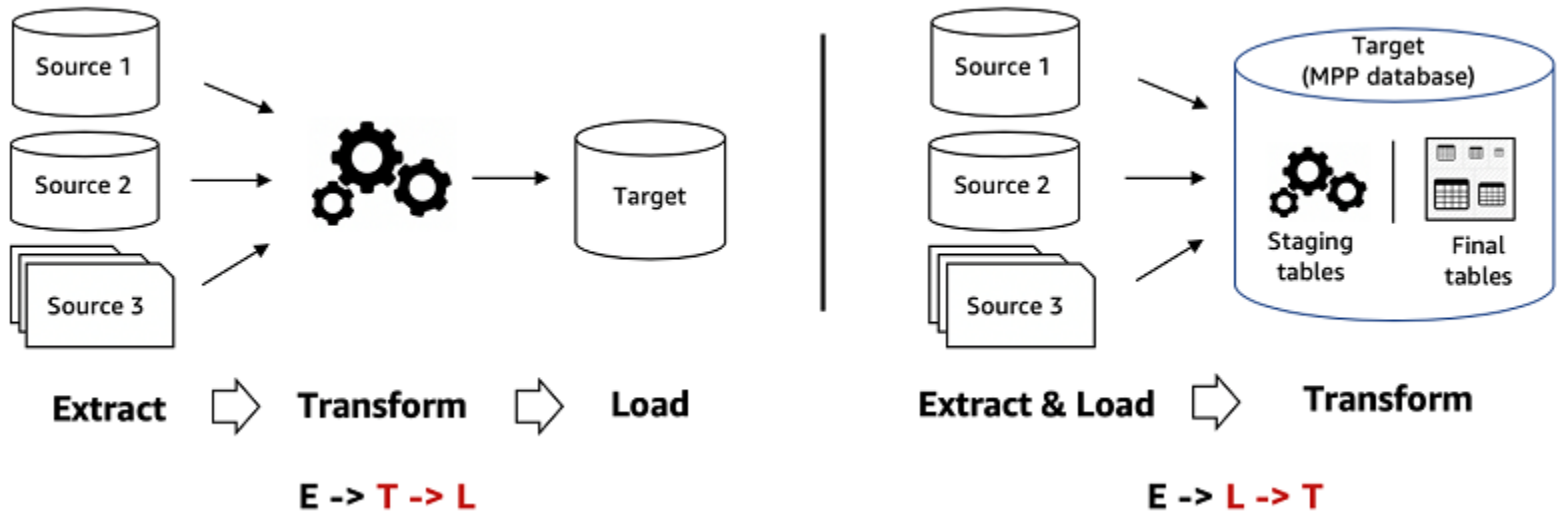
April 10, 2025

Agenda

- Understand the fundamentals of SQL and NoSQL
- Compare SQL and MongoDB
- Explore MongoDB structure and syntax
- Translate SQL queries to MongoDB queries
- Identify real-world simple applications with python SQL/NoSQL

Revision+

- Data can be numeric(continuous or discrete), categorical (grouped into categories), or ordinal (ordered categories).
- Data can be either structured (organized in a defined format like tables) or unstructured (like free text, images, or videos), and semi-structured.
- ETL vs. ELT

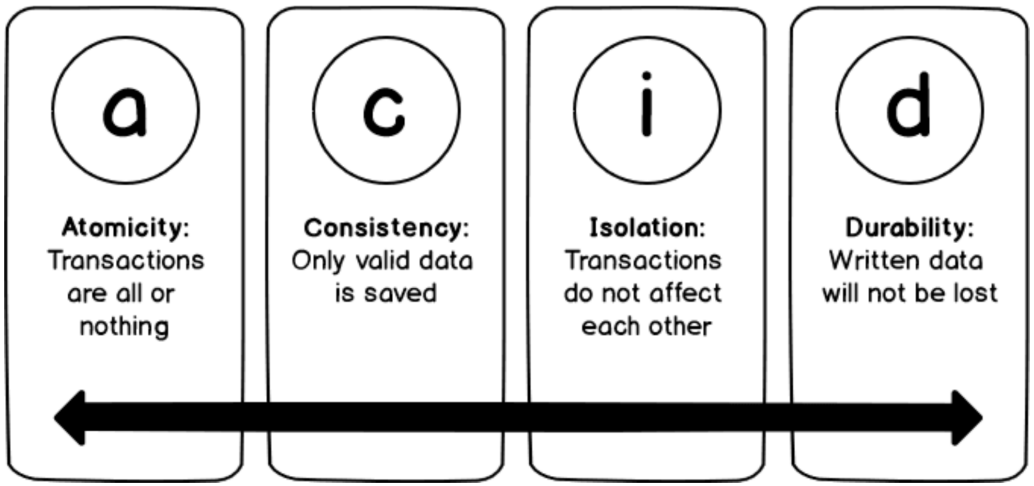


Data & Databases Matter...

- Structured, Semi-Structured, Unstructured data.
- Examples: CSV, JSON, XML, Multimedia.
- Database systems help us organize and query this data.

All started with RDBMS.

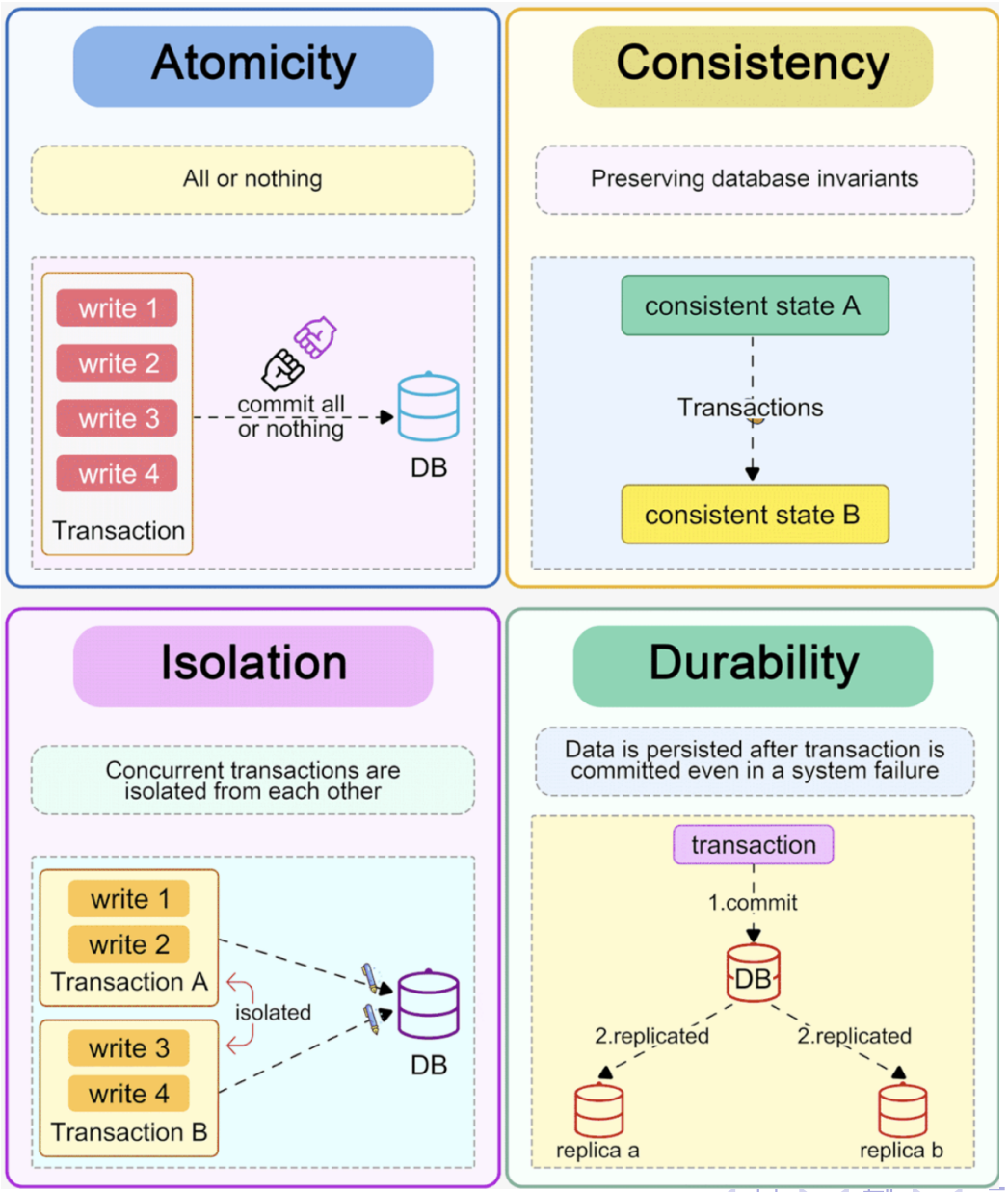
- Data stored in columns and tables
- Relational model with schemas
- Powerful, flexible query language.
- Save Transactions: ACID: Atomicity, Consistency, Isolation, Durability



Note on ACID Properties

- Atomicity: Consider a bank transfer where $100 is moved from Account A to Account B. If the debit from Account A succeeds but the credit to Account B fails, atomicity ensures that the debit is undone, leaving both accounts unchanged.
- Consistency: In the bank transfer example, if the database has a constraint that an account balance cannot be negative, consistency ensures that no transaction will violate this rule.
- Isolation: If one transaction is transferring $100 from Account A to Account B and another transaction is reading the balance of Account A, isolation ensures that the second transaction will only see the updated balance after the transfer is fully complete.
- Durability: Once the bank transfer is successfully completed, the changes to the account balances are saved permanently, even if the system crashes immediately after.

ACID Rules



- ACID in short

What is SQL?

- Structured Query Language.
- Works with relational databases.
- Examples: MySQL, PostgreSQL, SQLite.

SQL Structure.

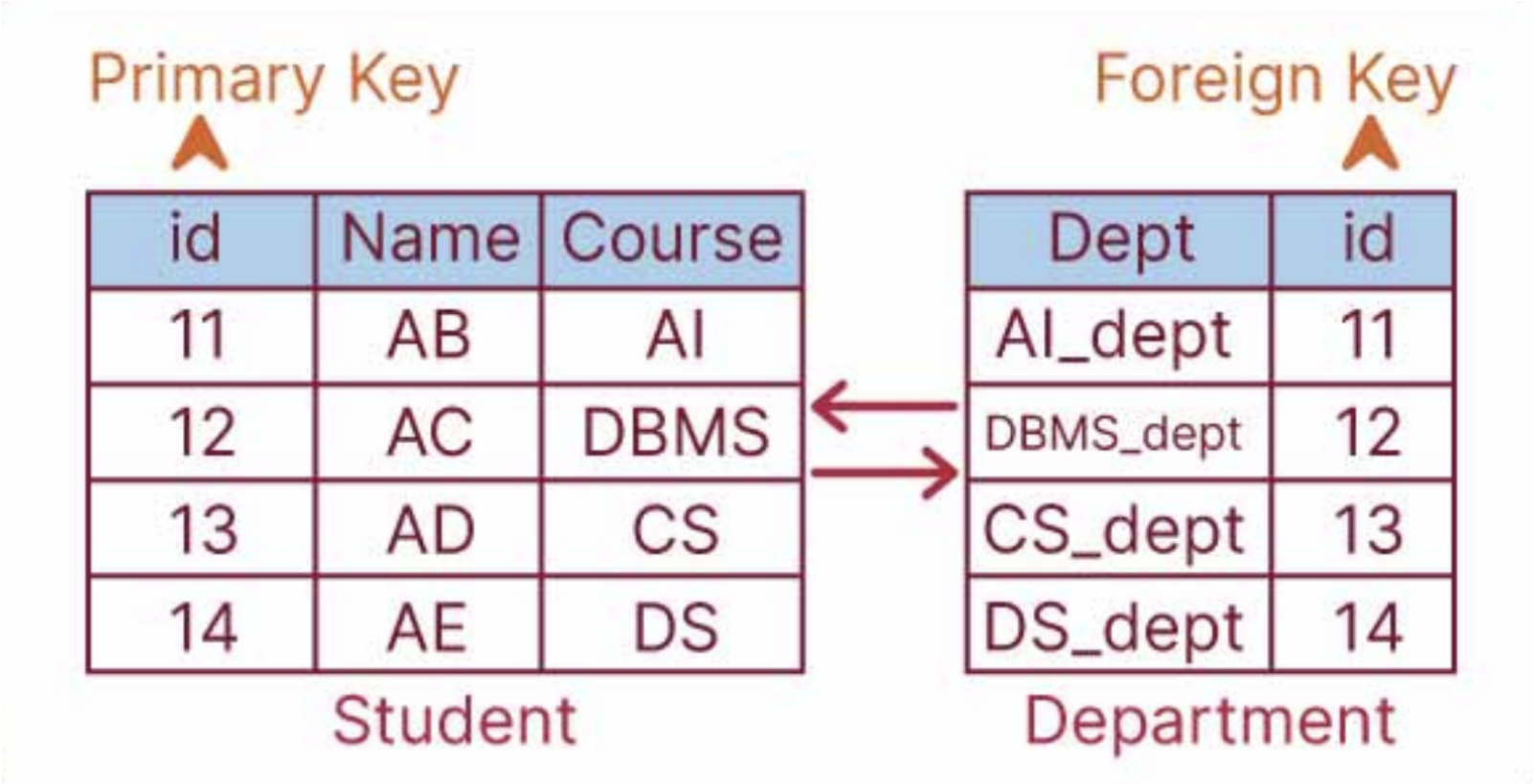
- Tables with rows and columns.
- Fixed schemas.
- Strong consistency.
- Powerful joins

Core SQL Concepts.

- Tables, Primary Keys, Foreign Keys.
- CRUD operations: SELECT, INSERT, UPDATE, DELETE.
- Indexing data on Tables INDEX
- Joins: INNER, LEFT, RIGHT. FULL

SQL: Primary Vs. Secondary Key

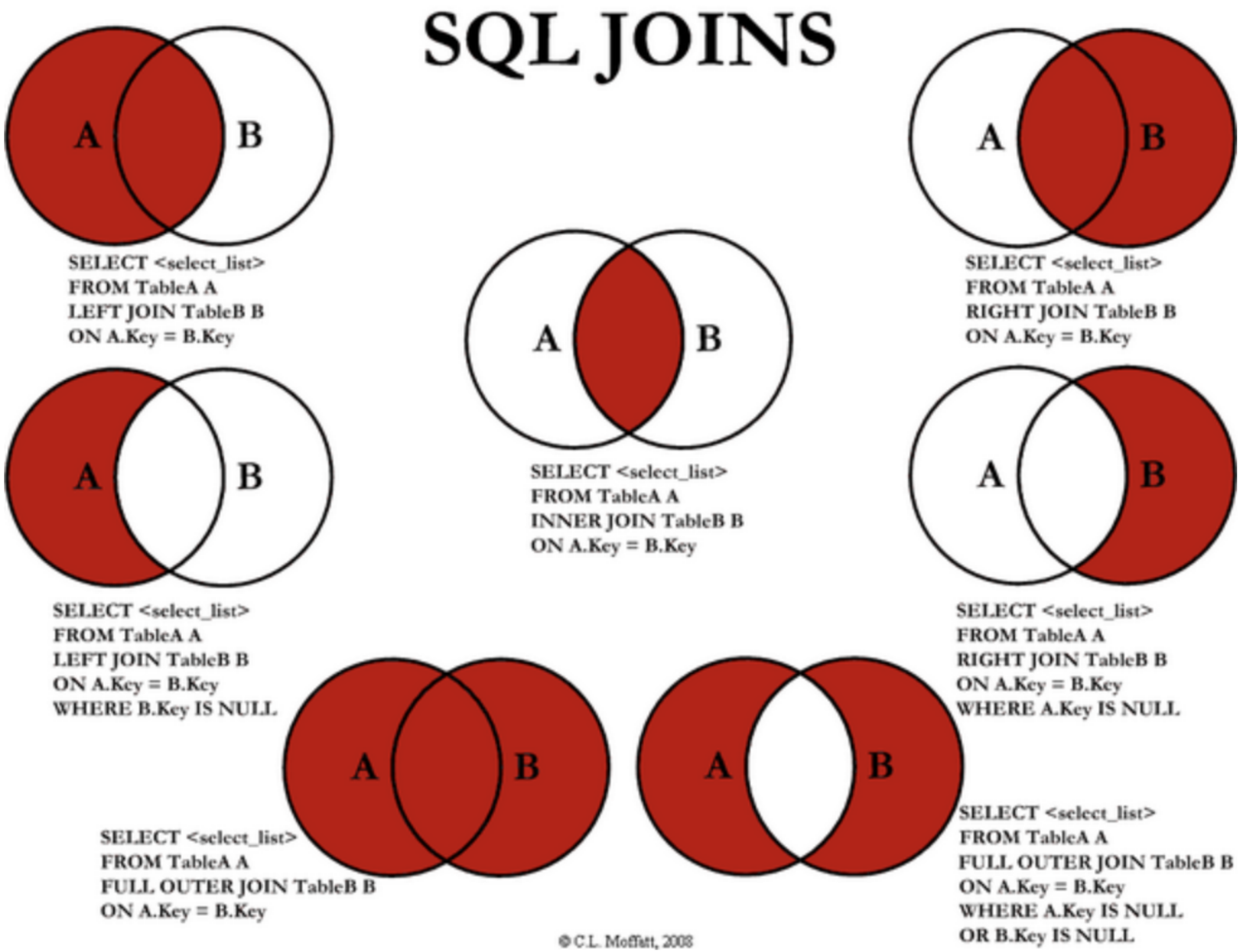
- The primary key is a unique identifier within its table, whereas a foreign key is a reference in one table to a primary key in another.
- An example



JOINS.

- INNER JOIN: Returns only the rows where there is a match in both tables.
- LEFT JOIN (or LEFT OUTER JOIN): Returns all rows from the left table and matching rows from the right table, filling in NULLs where no match is found.
- RIGHT JOIN (or RIGHT OUTER JOIN): Returns all rows from the right table and matching rows from the left table, filling in NULLs where no match is found.
- FULL JOIN (or FULL OUTER JOIN): Returns all rows from both tables, including all rows from the left table and all rows from the right table, filling in NULLs where no match is found.

JOIN: Set Notation



SQL Syntax Examples.

- SELECT * FROM orders WHERE price $>$ 100;
- SELECT u.name, o.date FROM users u JOIN orders o ON u.id = o.user_id;

Example: SQL-1.

```
-- Create database
CREATE DATABASE ShopDB;
USE ShopDB;
-- Create tables
CREATE TABLE Customers (
CustomerID INT PRIMARY KEY,
Name VARCHAR(100)
);
CREATE TABLE Orders (
OrderID INT PRIMARY KEY,
CustomerID INT,
Product VARCHAR(100),
FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)
);
-- Insert sample data
INSERT INTO Customers VALUES (1, 'Alice'), (2, 'Bob');
INSERT INTO Orders VALUES (101, 1, 'Book'), (102, 2, 'Pen');
```

Example: SQL-2.

```
-- Create index on Name
CREATE INDEX idx_name ON Customers(Name);
-- JOIN Query
SELECT Customers.Name, Orders.Product
FROM Customers
JOIN Orders ON Customers.CustomerID = Orders.CustomerID;
```

SQL Pros and Cons.

- Pros: ACID-compliant, strong schema, rich queries
- Cons: Hard to scale horizontally, rigid schema, JOIN overhead

ACID vs. BASE vs. CAP: ACID

- What it is: A model that emphasizes strong consistency and reliability
- What it stands for: Atomicity, Consistency, Isolation, and Durability
- How it works: Uses a synchronization mechanism to ensure that changes are written to all relevant data sets
- Example: SQL databases use the ACID model.

ACID vs. BASE vs. CAP: BASE

- What it is: A model that prioritizes availability and eventual consistency
- What it stands for: Basically Available, Soft State, and Eventual Consistency
- How it works: Doesn't lock data sets, so it can synchronize without a time guarantee.
- Example: NoSQL databases use the BASE architecture.

ACID vs. BASE vs. CAP: CAP

- What it is: A theorem that highlights the trade-offs between consistency, availability, and partition tolerance.
- What it states: It's not possible to guarantee all three of these properties simultaneously in a distributed system.
- How it applies: A database can satisfy two of the three guarantees.
- Example: The CAP theorem states that it's impossible for a distributed database system to simultaneously guarantee all three properties: consistency, availability, and partition tolerance. Examples of databases that prioritize different combinations of these properties include MongoDB (CP), Cassandra (AP), and DynamoDB (AP).

What is NoSQL?.

- Definition: NoSQL databases are non-relational databases designed for big data and real-time web applications.
- Characteristics:
  - Schema-less data model
  - Horizontal scalability
  - High performance and availability
  - Variety of data models (document, key-value, graph, etc.)

Why NoSQL?.

- Schema flexibility
- Easy to scale horizontally
- Suits modern applications: Web apps, IoT, Logs, Analytics
- Non-relational databases (NoSQL), are needed because they offer advantages in handling large, diverse, and rapidly changing datasets, providing flexibility and scalability that traditional relational databases struggle with.

Meet MongoDB.

- Documents = JSON/BSON
- Collections = Groups of documents
- No fixed schema: dynamic and nested data
- MongoDB = Document-oriented

Overview of MongoDB.

- What is MongoDB?
  - A popular NoSQL document database
  - Stores data in BSON (Binary JSON) format
  - Allows for flexible and dynamic schemas
- Use Cases:
  - Content management systems
  - Real-time analytics
  - Internet of Things (IoT) applications

MongoDB in Action

```
{
  "name": "Alice",
  "age": 30,
  "orders": [
    { "id": 1, "product": "Pen" },
    { "id": 2, "product": "Notebook" }
  ]
}
```

MongoDB CRUD - Read

```
db.users.find({ age: { $gt: 25 } });
```

MongoDB CRUD - Insert

```
db.users.insertOne({ name: "John", age: 22 });
```

MongoDB CRUD - Insert-Update

```
db.users.updateOne(
  { name: "John" },
  { $set: { age: 23 } }
);
```

MongoDB CRUD - Delete

```
db.users.deleteOne({ name: "John" });
```

Aggregation Framework

- Similar to SQL GROUP BY
- Use ‘$match‘, ‘$group‘, ‘$sort‘
- Built for analytics queries

Aggregation Example

```
db.sales.aggregate([
  { $match: { price: { $gt: 50 } } },
  { $group: { _id: "$category", total: { $sum: "$price" } } }
]);
```

Indexing in MongoDB

- Improves performance
- Similar to SQL indexes

```
db.users.createIndex( name: 1 );
```

Data Modeling in MongoDB

- Embedded documents vs referenced collections
- Use embedded for 1:1 or 1:many (read-heavy)
- Use reference for normalized (write-heavy)

Embedded Example

```
{ "name": "Alice",
"address": {
"city": "Berlin",
"zip": "10115"
}
}
```

Referenced Example

```
db.users.insertOne( name: "Bob", address_id: ObjectId("...")
);
db.addresses.insertOne( _id: ObjectId("..."), city: "Paris" );
```

$lookup Example (Join)

```
db.orders.aggregate([  $lookup:  from: "products", localField:
"product_id", foreignField: "_id", as: "product_details"   ]);
```

SQL vs MongoDB Terminology

SQL	**MongoDB**
Table	Collection
Row	Document
Column	Field
JOIN	`$lookup` or Embedded
Index	Index

SQL to MongoDB Translation Examples

SQL	MongoDB
SELECT * FROM users	`db.users.find({})`
WHERE age $>$ 30	`{ age: $gt: 30 }`
INSERT INTO ...	`db.collection.insertOne({...})`

CRUD Commands Comparison: SQL vs NoSQL

```
1. CREATE -- Insert Data
-- SQL
INSERT INTO Customers (CustomerID, Name) VALUES (3, 'Charlie');
-- NoSQL
db.customers.insert_one("_id": 3, "name": "Charlie")
```

CRUD Commands Comparison: SQL vs NoSQL

```
2. READ -- Retrieve Data
-- SQL
-----All records
SELECT * FROM Customers;
----- Filtered records
SELECT * FROM Customers WHERE Name = 'Charlie';
-- NoSQL
# All documents
list(db.customers.find())
# Filtered query
db.customers.find_one("name": "Charlie")
```

CRUD Commands Comparison: SQL vs NoSQL

```
3. UPDATE -- Modify Data
-- SQL
UPDATE Customers SET Name = 'Charles' WHERE CustomerID = 3;
-- NoSQL
db.customers.update_one(
"_id": 3,
"$set": "name": "Charles"
)
```

CRUD Commands Comparison: SQL vs NoSQL

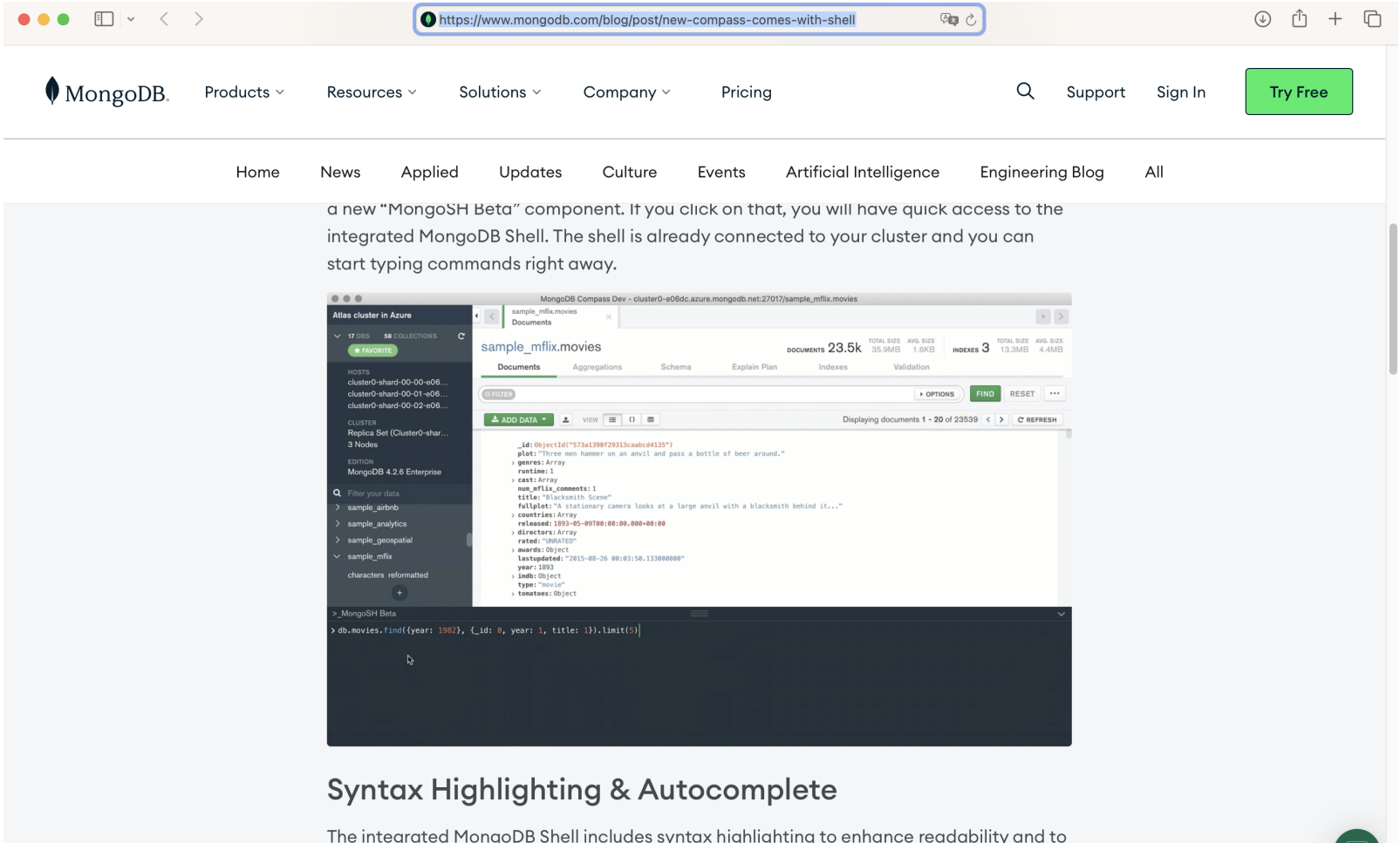
```
4. DELETE -- Remove Data
-- SQL
DELETE FROM Customers WHERE CustomerID = 3;
-- NoSQL
db.customers.delete_one("_id": 3)
```

Tooling for MongoDB

- MongoDB Compass (GUI)
- MongoDB Atlas (Cloud DB)
- Mongo Shell / mongosh

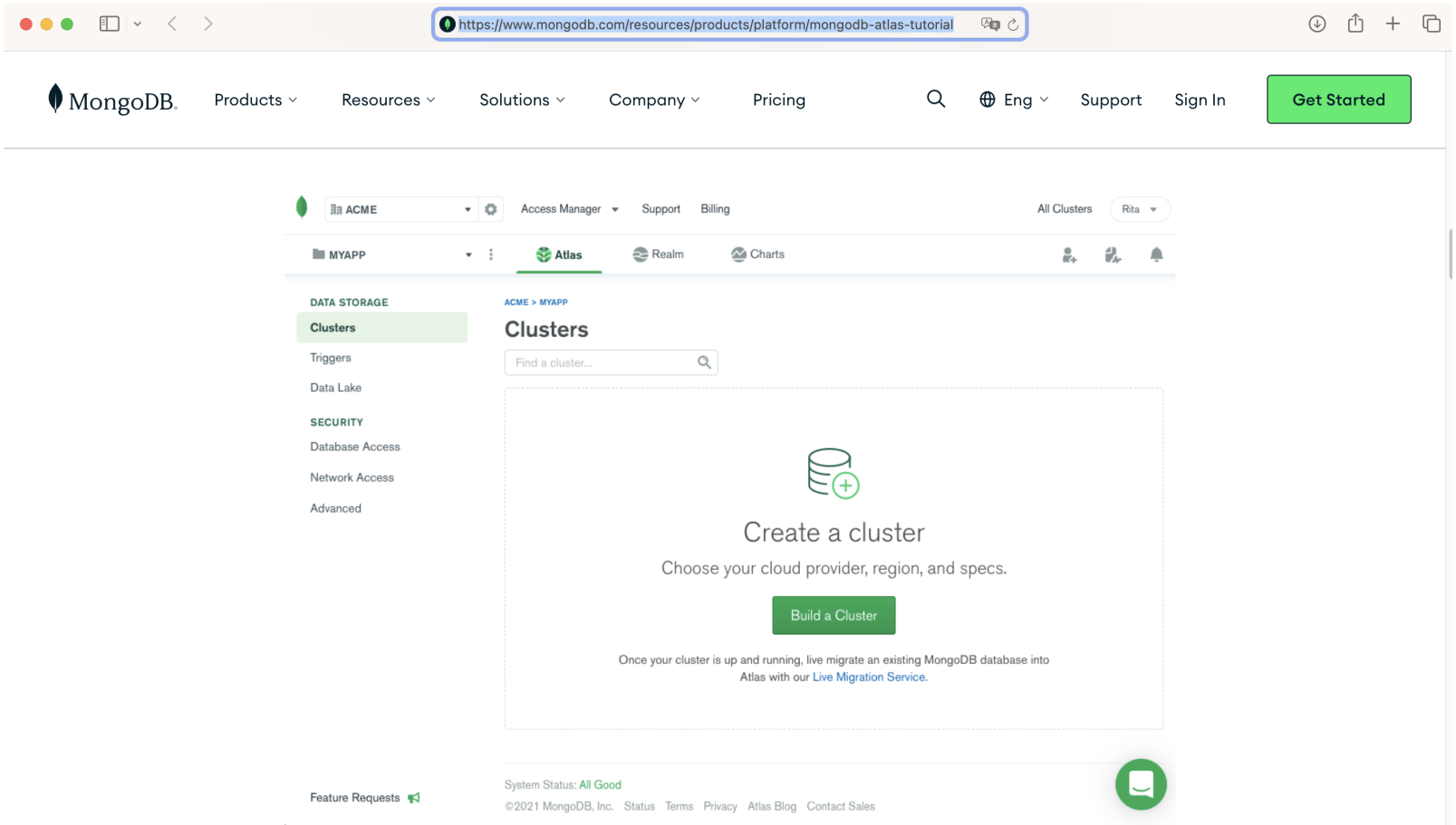
MongoDB Compass (Screenshot)

- Browse collections
- Visualize documents
- Build queries and aggregations
- Compass Screenshot



MongoDB Atlas (Screenshot)

- Free tier cloud cluster
- Secure and scalable
- Good for prototyping and learning
- Atlas Screenshot



Limitations of MongoDB

- Lacks strong relational enforcement
- No cross-collection ACID transactions (except v4.0+)
- Indexing and aggregation need tuning for large datasets

SQL + NoSQL Together

- Microservices may use MongoDB
- Reporting layer may use SQL (PostgreSQL/MySQL)
- Polyglot persistence: choose the best DB for the job

Use Cases Comparison

Use Case	SQL	MongoDB
Banking	✓ ACID-safe	X
Product Catalog	X Complex Joins	✓ Flexible
Log Analytics	X	✓ Fast ingest

Recap

- SQL: reliable for structured data
- MongoDB: flexible, fast for dynamic data
- Learn both and apply based on project needs

Quiz: What Would You Use?

- IoT stream data storage?
- Inventory management system?
- Mobile app user profiles?
- (Discuss: SQL or MongoDB and why?)

Thank You!

Questions? sucheta.ghosh@lehrb.hs-offenburg.de