Databases for Data Science

Lecture 08 · 2022-09-21

Today

- Assignment 1 review
- SQL vs NoSQL
- JSON
- MongoDB basics
- MongoDB exercises

Assignment 1: Review

Problem 1

```sql
-- homelessness db
SELECT *
FROM bookings
WHERE TO_CHAR(dob, 'YYYY') IN ('1990', '1991', '1992');
```

- *Write your own explanation of this query.*

Problem 2

```sql
-- mini_example db
SELECT name,
    employment.job_title,
    employment.employer,
    employment.salary,
    averages.average_salary
FROM employment
    LEFT JOIN person on person.user_id = employment.user_id
    LEFT JOIN (
        SELECT job_title,
            AVG(salary) AS average_salary
        FROM employment
        GROUP BY job_title
    ) averages ON employment.job_title = averages.job_title;
```

- If we changed the left joins to inner joins, what would be different?

Problem 3

```sql
-- homelessness db
SELECT DISTINCT
    info.dob,
    info.charge,
    info.name
FROM (
    SELECT
        charge,
        MIN(dob) as dob
    FROM
        bookings
        JOIN charges ON charges.bookingnumber=bookings.bookingnumber
    GROUP BY charge
) oldest JOIN (
    SELECT
        charge,
        name,
        dob
    FROM
        booking_dates JOIN charges ON booking_dates.bookingnumber=charges.bookingnumber
) info
ON oldest.charge=info.charge AND oldest.dob=info.dob
ORDER BY charge,name
LIMIT 20;
```

Problem 4

PostgreSQL will not run the following query.

```
-- homelessness db
SELECT
    soid,
    address,
    MIN(dob)
FROM bookings
GROUP BY address;
```

- Why doesn't this work? Explain briefly.
- How does the Problem 3 query avoid this issue?

Problem 5

- Find every row in the `booking_dates` table where the booking date is on or after January 1, 1990. Sort the results by booking date.

Problem 6

Text search

- (a) Find every row in the `arrestees` in which the person's first name is "Mary".
- (b) List every **unique** first name in the `arrestees` table that starts with "Mary".
  - e.g. "Mary Ann", "Mary Lu", etc.
  - Hint: consider using the `SPLIT_PART` function.

Problem 7

- (a) Using `booking_dates`, find the age of each person at the date of their first arrest. Display the age in years, months and days.
- (b) Using `charges` and `booking_dates`, find the average age of the people arrested for each charge at the date of arrest.
- (c) Find the difference in age between the oldest and youngest age arrested for each charge. Display each difference as a number of years, months and days.
- (d) Using `booking_dates`, `COUNT` the number of people living at each street address. Ensure that each person is only counted once; use the SOID to identify each person.

Problem 8

Exploration

- (a) How do we know if someone is homeless? This question is exploratory. Show and explain any exploratory queries you use, and cite any external sources of information (e.g. maps, articles). Develop and state a criterion for inferring homelessness.
- (b) Write a query that returns the columns of `bookings` with an additional `homelessness` column (true or false).

Exercise: Peer Review

Discussion: SQL and NoSQL Databases

Questions relevant to our reading assignment:

- What motivated the development of relational (SQL) databases?
  - What might have changed since then?

- What motivated the development of non-relational (NoSQL) databases?
  - How do they differ?

- Why might *we* prefer one or the other?

JSON

Java**S**cript **O**bject **N**otation

- Widely used data-interchange format
- Native to JavaScript, but used by nearly every language
- Balances flexibility, human-friendliness, machine-friendliness
  - (Compare XML, CSV)

JSON: Collections

JSON specifies a tree structure of *objects* and *arrays*.

Object

Like a dictionary: maps keys to values.

```
{
  "name": "Alex",
  "major": "Data Science",
  "email": "alex@ncf.edu"
}
```

Arrays

Like a list: arbitrarily many values, accessed by index.

```
[
  "Alex",
  "Bryce",
  "Charlie"
]
```

JSON: Value Types

- Objects
- Arrays
- Strings
- Numbers
- Booleans
- `null`

```
{}
[]
"abcde"
123, 3.01, 1.2e6
true, false,
null
```

What *don't* we see in this list?

JSON

You can mix different data types in the same collection.

```
{
  "name": "Alex",
  "is_student": true,
  "gpa": 3.7,
  "major": null,

  // Array property
  "dorms": [

    // Objects inside the array
    { "hall": "Dortstein", "room": 123 },
    { "hall": "Pritzker", "room":234 }
  ]
}
```

JSON

Lots of flexibility; choose a data model that is intuitive but code-friendly.

```
// List of mixed types
// (harder to parse)
{
  "name": "Alex",
  "contact": [
    "alex@ncf.edu",
    "alex@example.com",
    "555-123-4567"
  ]
}
```

```
// List of objects with
// type annotations
{
  "name": "Alex",
  "contact": [
    {
      "type": "email",
      "address": "alex@ncf.edu"
    },
    {
      "type": "email",
      "address": "alex@example.com"
    },
    {
      "type": "phone",
      "number": "555-123-4567"
    }
  ]
}
```

```
// Separate arrays for
// each type
{
  "name": "Alex",
  "email": [
    "alex@ncf.edu",
    "alex@example.com"
  ],
  "phone": [
    "555-123-4567"
  ]
}
```

JSON

```
{
  "name": "Alex",
  "is_student": true,
  "gpa": 3.7,
  "major": null,

  // Array property
  "dorms": [

    // Objects inside the array
    { "hall": "Dortstein", "room": 123 },
    { "hall": "Pritzker", "room":234 }
  ]
}
```

Individual Exercise:

Write a JSON file for your own resume.

CSV vs JSON

Predefined flat tables vs. flexible nested structures.

```
person_id, name
100, Alex
200, Bryce
```

```
enrollment_id, person_id, course
20130, 100, Databases
20139, 100, Algorithms
20140, 200, Databases
```

```
[
  {
    "name": "Alex",
    "enmrollments": [
      "Databses", "Algorithms"
    ]
  },
  {
    "name": "Bryce",
    "enrollments": ["Databases"]
  }
]
```

MongoDB

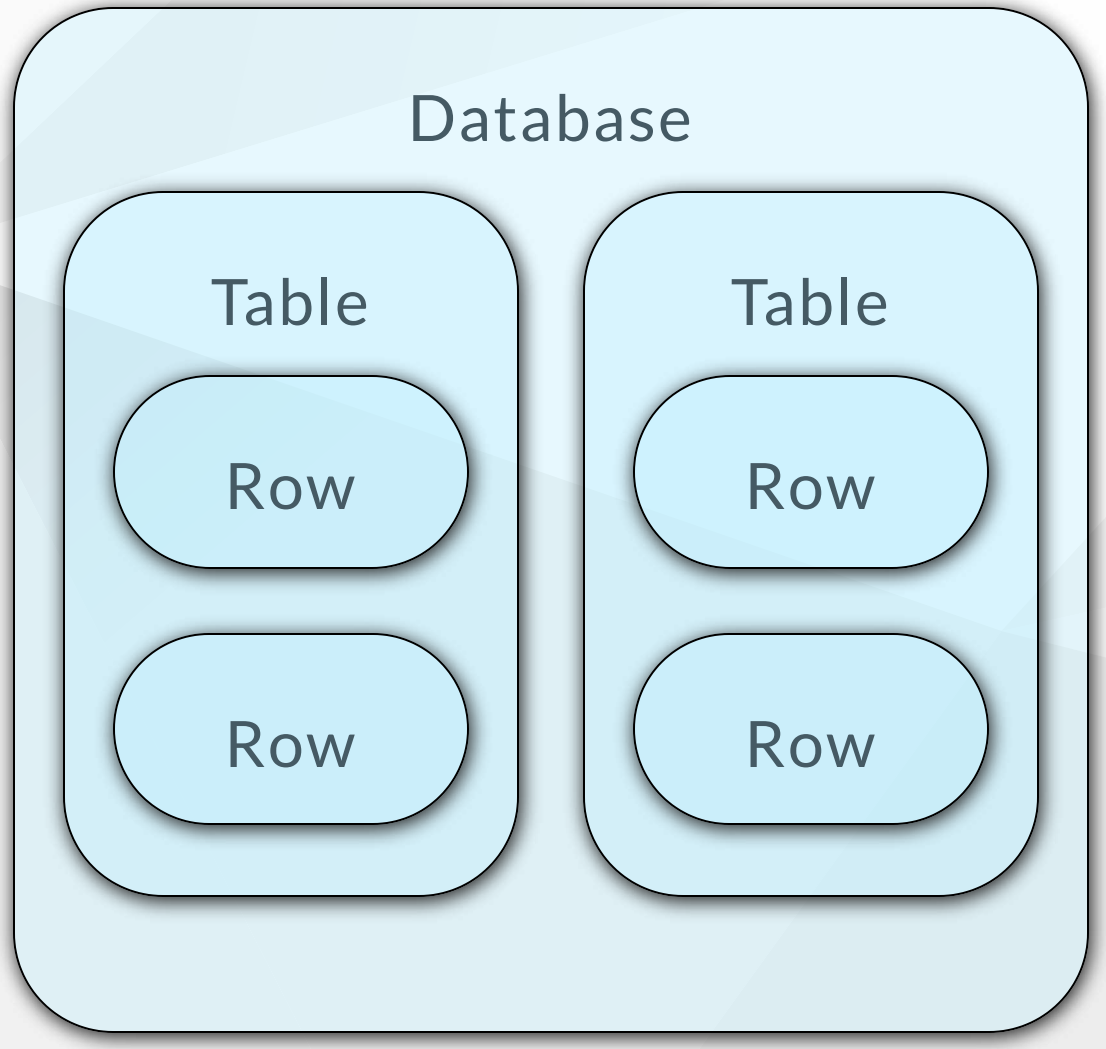
Mongo is a *document store*.

- Definition?

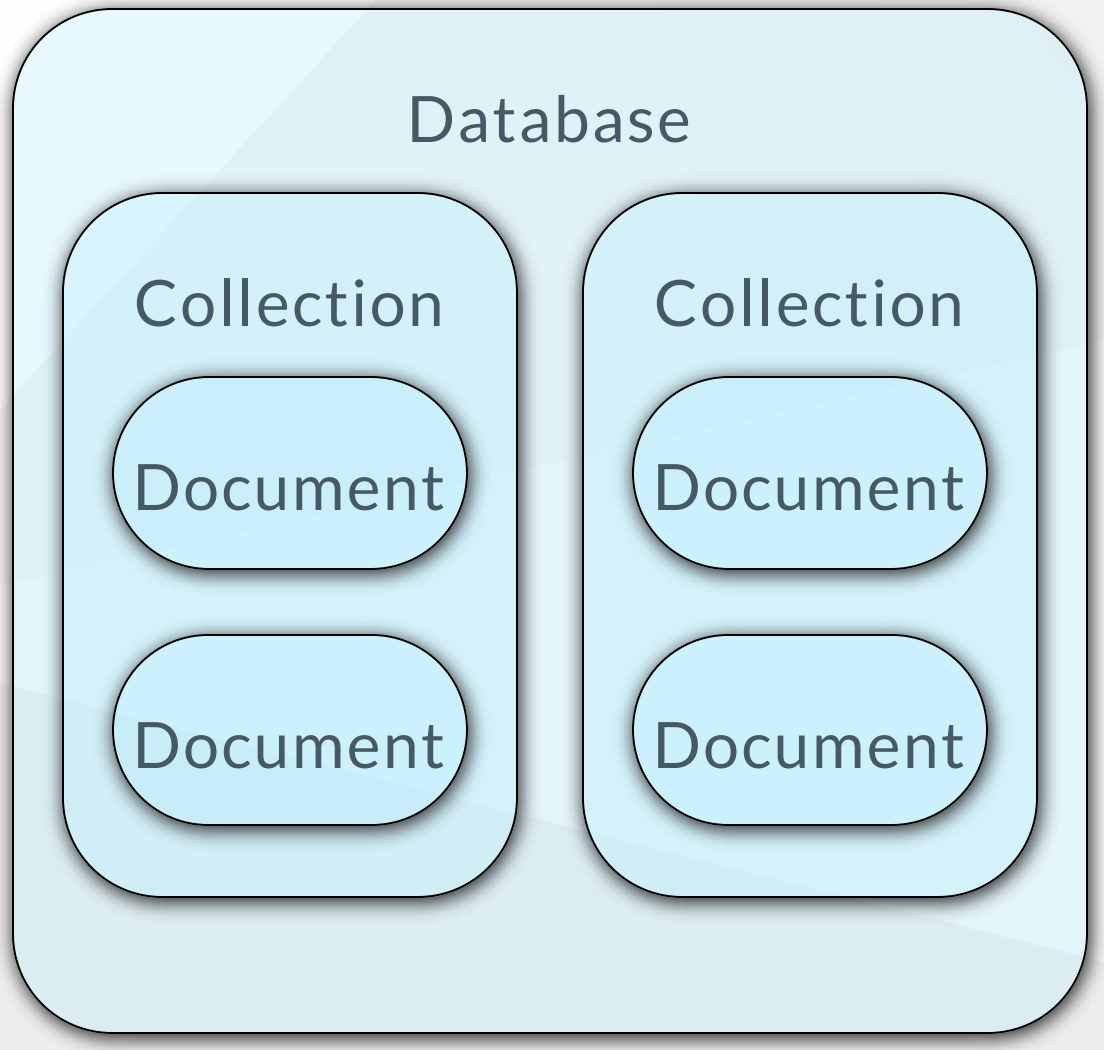
Documents in Mongo are in BSON (Binary JSON) format.

- More efficient for storage
- Some extra data types
- Maximum document size of 16MB

SQL Database



MongoDB



MongoDB | Shell

Entering the Mongo shell (similar to psql):

- `[user@cs1 ~] mongo`

List the existing databases on the server:

- `> show dbs`

Enter a database:

- `> use sample_airbnb`
  - If this DB doesn't exist, it will be created!

MongoDB | Shell

As with PSQL, there are plenty of commands.

To list them:

- `> help`

e.g.: how do we view a list of collections?

MongoDB | Shell

Instead of SQL, we use JavaScript.

`db`	access the `db` object
`db.towns`	access the `towns` property of `db`
`db.towns.help()`	call the `help` method of `db.towns`
`var x = 5;`	declare and define a variable `x`
`x+5`	evaluate `x+5` and print the result

MongoDB | Queries

To query a collection, we call various methods attached to it.

```
// get the number of documents
> db.listingsAndReviews.count()

// query all documents
> db.listingsAndReviews.find()

// query all documents and pretty-print the output
> db.listingsAndReviews.find().pretty()
```

MongoDB | Queries

Calling `[some collection].find()` returns a *cursor*: a stateful iterator over the results.

```
> var cur = db.listingsAndReviews.find()

// Display info from the cursor, as we did before
> cur

// Iterate the cursor
> cur.next()

// Get the size of the result set
> cur.next()

// Show the query plan and other details
> cur.explain()
```

MongoDB | Queries

To get documents matching a condition, add a *filter* to the `find()` call:

```
// Match a top-level property
> db.listingsAndReviews.find({property_type: "House"})

// Match a nested property
> db.listingsAndReviews.find({"address.country_code": "US"})

// Match multiple properties
> db.listingsAndReviews.find({property_type: "House", "address.country_code": "US"})
```

- What does this correspond to in SQL?

Exercise: Find all of the two-bedroom apartments.

MongoDB | Query Operators

We've used this syntax for *equality* filters:

```
// { <field>: <value>}
```

We can also use *operators* to define more general filters.

```
// { <field>: { <operator>: <value> } }

// Comparison
{ accommodates: { $gt: 5 } }

// Inclusion
{ "address.country_code": { $in: ["US", "CA"]}}
```

MongoDB | Query Operators

Full list: https://www.mongodb.com/docs/v4.4/reference/operator/query/

Exercise: Find all of the listings with a rating above least 90.

Exercise: Find all of the listings with more than two bedrooms and a weekly price of less than $1000.

Exercise: Find all of the condos and apartments in the US whose host is named "Alex".

MongoDB | Query Operators

Queries on array contents:

```
// Match one array item
db.listingsAndReviews.find({"amenities": "Internet"})

// Match two array items
db.listingsAndReviews.find({"amenities": {$all: ["Internet", "TV"]}})

// Match *exact* array contents (this will be empty!)
db.listingsAndReviews.find({"amenities": ["Internet", "TV"]});

// Match document within array
db.listingsAndReviews.find({"reviews.reviewer_name": "Xavier"});
```

Exercise: Find all of the apartments with TV *or* internet service.

MongoDB | Projection

We can also choose which parts of each document to return.

```
// Get the street address and bedroom count of every listing
db.listingsAndReviews.find({}, {"address.street": 1, bedrooms: 1});

// Get the amenity list and review score for every listing without internet
db.listingsAndReviews.find(
  {"amenities": {$ne: "Internet"} },
  {"amenities": 1,"review_scores.review_scores_rating": 1}
);
```

- What does this correspond to in SQL?

Exercise: Get the host name of every listing in Brazil.

MongoDB | Map

We can apply a function to every item in the output. (Or in an array, etc.)

```
// Get the first letter of each string
['Alex','Charlie'].map(s=>s.substring(0,1))

// Apply a more complicated function
db.listingsAndReviews.find().map(listing => {
  let count = listing.reviews.length;
  return `${listing.name} has ${count} reviews`;
})
```

Exercise: Write a function to decide whether you would consider staying at a given listing. Use `map` to apply this function to all listings.

MongoDB | Inserting data

Navigate to your own database:

- `> use [YOUR NAME]`

Create a new collection:

- `> db.createCollection(students)`

You can also create a collection *implicitly* by adding data to it!

MongoDB | Inserting data

It's pretty simple:

```
// db.collection.insert( <object> )

> db.students.insert({"name": "Alex");
> db.students.insert({"student_id": "N10234567"});

// Use an array to insert several objects
> db.students.insert([{"name": "Charlie"}, {"name": "Dylan"}])
```

At this point, you can put *anything* in any collection.

- Why might this be helpful?
- Why might this be a problem?

MongoDB | Inserting data

JavaScript can be useful for automating tasks.

```
> var names = ["Alex","Bryce","Charlie","Dylan"];
> var new_students = [];
> for (var name in names) {
  new_students.push({name:"name"});
}
> db.students.insert(new_students);
```

Exercise: Use JS to generate and insert your own set of students. Assign each one a unique email address.

MongoDB | Update

Where we had `.find(filter, projection)`, we now have `.updateMany(filter, command)`.

```
> db.student.updateMany({"name": "Alex"}, { $set: {"gpa": 4.0}});
```

Exercise: Assign random GPAs to all of your students.

MongoDB | Delete

Similarly to a `DELETE FROM tablename WHERE condition`, we have `.deleteMany(filter)`.

```
// Remove matching records
> db.student.deleteMany({"name": "Bryce"});

// Remove all records
> db.student.deleteMany({});
```

Exercise: Delete all records for students with GPA below 1.0.

MongoDB | Documentation

Manual:
https://www.mongodb.com/docs/v6.0/

Cheat sheet:
https://www.mongodb.com/developer/products/mongodb/cheat-sheet/

[more exercises?]

Next time

- MapReduce
- Aggregation in MongoDB
- Problem-solving exercises