COMP 430
Intro. to Database Systems

MongoDB
What is MongoDB?

“Hu\textit{m}ongous” DB

- NoSQL, no schemas DB
- Lots of similarities with SQL RDBMs, but with more flexibility
- No joins because data is denormalized & pre-joined
- Open source
Collections of documents

Collection = table
Document = record

Every document has unique _id, even if not user-defined.

Documents need not have the same fields.
Relating data via embedding

```json
{
  _id: "S1234",
  first_name: "John",
  last_name: "Smith",
  major: "ECON",
  address: {
    street: "1514 Nowhere St.",
    city: "Houston",
    state: "TX"
  }
}
```

Denormalized. Possible redundancy.

Data all stored together for locality.
Relating data via embedding

```
{
    _id: "S1234",
    first_name: "John",
    last_name: "Smith",
    major: "ECON",
    addresses: [{
        street: "1514 Nowhere St.",
        city: "Houston",
        state: "TX"},
    {street: "7729 Somewhere St.",
        city: "Kansas City",
        state: "KS"}]
}
```
Relating data via references

```json
{
   _id: 34237,
   first_name: "John",
   last_name: "Smith"
}

{ 
   title: "A book about books",
   author: 34237
}

{ 
   title: "A little bookish book",
   author: 34237
}

{ 
   title: "A new book",
   author: 34237
}

_id field = primary key
Reference = foreign key
Relating data via references

_id field = primary key
Reference = foreign key
Relating data via embedding + references

Denormalized. Many-to-many junction info put in one document.

{ 
  first_name: “John”,
  last_name: “Smith”
  books: [103, 279, 541]
}

{ 
  first_name: “Mary”,
  last_name: “Jones”,
  books: [103, 541]
}

{ 
  _id: 103,
  title: “A book about books”
}

{ 
  _id: 279,
  title: “A little bookish book”
}

{ 
  _id: 541,
  title: “A new book”
}
Embedding & relating replaces joins

No need for joins, because data is already explicitly linked.

This gives up SQL’s flexibility of joining on conditions other than PK=FK. Focus on the common case.
DB/application interface

Intention is for DB & application to have equivalent data representation.

• Explicit references
• Built-in lists and associative arrays
• Flexible field types
• Flexible field membership
Storage model

Emphasize data locality:
- Each document stored as one unit, even with embedding
- Each collection stored as one unit.

Uses BSON, a binary-encoded variant of JSON key/value pairs
CRUD operations

SQL-like, but with a library
Filtering

- `db.students.find({major: "COMP"})`
- `db.students.find({major: "COMP", minor: "STAT"})`
- `db.students.find({$or: [{major: "COMP"}, {major: "ELEC"}])})`
- `db.students.find({major: {$in: ["COMP", "ELEC"]})})`
- `db.students.find({gpa: {$gt: 3.8}})`
- `db.students.find({address.street: {$exists: true}})`

Collection name

Syntax structured as key/value pairs with some special $keywords.
Selecting fields

db.students.find({major: "COMP"}, {first_name: true, last_name: true, major: false})
Inserting & updating

```javascript
db.students.insert({
  first_name: "Susan",
  last_name: "Walsh"
})

db.students.update({
  first_name: "John",
  last_name: "Smith"
},
{
  $set: {major: "HIST"},
  multi: true,
  upsert: true
})

db.students.update({
  first_name: "John",
  last_name: "Smith"
},
{
  first_name: "Jonathan",
  last_name: "Smythe"
})

db.students.update({
  major: "COMP"
},
{
  $unset: {minor: ""}
})
```
Deleting

```
db.students.remove({major: "COMP"})
```
Transactions

By default, each write to a *document* is atomic.

```javascript
db.students.update(..., $isolated: true)
```

```javascript
db.students.remove(..., $isolated: true)
```

Make entire collections’ writes atomic.

Not intended for transaction-heavy systems.
Some other features
Grouping & aggregation

`db.students.aggregate({$match: ...},
	{$group: ...},
	{$sort: ...})`

`db.students.group(...)`

`db.students.mapReduce(map_fn, reduce_fn)`

`db.students.distinct("major")`
B-tree indices

Automatically create index on _id.

db.students.ensureIndex({last_name: 1, gpa: -1})

Creates index that first orders last_name ascending, then gpa decending.
Data distribution

Supports replication & *sharding* (partitioning)