SQL - Read from multiple tables

Example: relational model

student

id	name	zip
s1	jack	10027
s2	jill	11211
s3	qing	10003
s4	arun	10012

major

id	name
m1	physics
m2	biology
m3	chemistry

student-major

student_id	major_id
s1	m1
s1	m2
s2	m3
s4	m2

Example: SQL

Gather all the data for Jack

```
select * from student
where name = ‘Jack’
```

```
select major.name
from major, student-major
where student-major.student_id = “s1”
   and major.id = student-major.major_id
```

Example: SQL

Get all the data for Jack in one query (join)

```
select s.name, s.zip, m.name
from student s
  join student-major sm on s.id = sm.student_id
  join major m on sm.major_id = m.id
where s.name = ‘Jack’
```

Example: SQL

Get all data that we can (outer join)

```
select s.name, s.zip, m.name
from student s
left join student-major sm on s.id = sm.student_id
join major m on sm.major_id = m.id
```

MongoDB version

```
{
    _id: s1,
    name: 'jack',
    zip: '10027',
    major: ['physics','biology']
}
{
    _id: s2,
    ......
}
```

MongoDB - Schema-less DB

Structure

- Documents are the smallest unit
  - JSON-like format
- Documents form a collection
  - Each document is a ‘record’ in a collection
  - Documents don’t have to have the same format

MongoDB pros and cons

Pros

- Flexibility with Structure
- Object structure is clear
- No joins!
- Easy to scale
- Maps to application objects
- Easy to replicate

Cons

- No transaction support
- Data integrity?
- Replica lags may be a problem in write-heavy databases
- Not good for multi-user systems (“global” write locks)

Set up

The use of MongoDB is more “manual“ than other database engines

- Installation
  - On Mac – Using homebrew or Manually.

- Post install
  - Look at the config file:
    - Default location – /usr/local/etc/mongod.conf
    - Identify locations of log file and data folder.

- Start the Mongo Server
  - Using config file – mongod ––config *<path-to-config-file>*
  - Pointing to data location – mongod ––dbpath *<path-to-data folder>*

Basics Actions

MongoDB is “forgiving” – Entities will be created if they don’t exist

- ➡ Open Shell
  - ➡ mongo or mongo *<db name>*
  - ➡ Database doesn’t have to pre-exist
- ➡ List databases
  - ➡ show databases
- ➡ Create new database
  - ➡ use *<db name>*
- ➡ Delete a database
  - ➡ db.dropDatabase()

Basics Actions

- List collections (tables)
  - show collections

- Create new collection
  - First insert creates the collection - (See Insert command)

- Define an index
  - db.<*collection name*>.createIndex({<*key*>: *1*}, { unique: true })
  - 1 is ascending order. -1 is descending

- Add a new column
  - Not relevant - no columns in Mongo

- Delete a collection
  - db.<*collection name*>.drop()

CRUD Operations

- CREATE - add documents (records) to a collection:
  - Syntax - db.<*collection name*>.insert(<*document*>)
  - <document> is a valid json - {key1: val1, key2: { key3: val3 } }
    - If the _id field is not provided - documents get an _id field assigned with a unique ObjectId.

insert example

```
db.restaurants.insert(
  {
    "address" : {
      "street" : "2 Avenue",
      "zipcode" : "10075",
      "building" : "1480",
      "coord" : [ -73.9557413, 40.7720266 ]
    },
    "borough" : "Manhattan",
    "cuisine" : "Italian",
    "grades" : [
      {
        "date" : ISODate("2014-10-01T00:00:00Z"),
        "grade" : "A",
        "score" : 11
      },
      {
        "date" : ISODate("2014-01-16T00:00:00Z"),
        "grade" : "B",
        "score" : 17
      }
    ],
    "name" : "Vella",
    "restaurant_id" : "41704620"
  }
)
```

CRUD Operations

- READ - find data in a collection
  - Syntax - db.<*collection name*>.*find*(<*condition*>, <*projection*>)
  - All Data - db.<*collection name*>.*find*()
  - Conditionally list from a collection.
    - db.<*collection name*>.*find*(<*condition*>)
    - <*condition*> *is a json document -*
    - {} - *returns all documents*
    - { key: value }
    - { key: { <*operator*>: value }}
    - Multiple Conditions:
      - AND - { key1: value1, key2: value2 }
      - OR - { $or: [{key1: value1}, {key2: value2}] }
- Keys can be embedded fields, e.g. “grades.grade”.
- Examples of <operator> are $eq, $gt, $in

CRUD Operations

- READ – find data in a collection
  - Conditionally get fields – db.<*collection name*>.*find({}, <projection>)*
  - *<projection> is a valid json listing the fields (_id is returned by default) –*
    - *{key1: 1, key2: 1} – return only key1 and key2 (and _id)*
    - *{_id: 0} – don't return _id*
      - note – true/false can replace 1/0
  - By Default mongo returns 20 documents. If there are more – type it to iterate through rest of the records.
  - To sort documents – db.<*collection name*>.*find().sort({key: 1})*
  - To display as indented json – db.<*collection name*>.*find().pretty()*

CRUD Operations

- READ – find data in a collection
  - Many more “helper” find methods:
    - findAndModify()
    - findOne()
    - findOneAndDelete()
    - findOneAndReplace()
    - findOneAndUpdate

CRUD Operations

- UPDATE - update data in a collection
  - Syntax - db.*<collection name>*.*update(<condition>, <update>, <options>)*
  - *<condition> - we know…*
  - *<update> - a json document with keys and values to update*
    - *{key1: new_val1, key2: new_val2} - replaces the whole document*
    - *{$set: { key1: new_val1}} - updates only the specified keys*
  - <options> - a json document:
    - By Default mongo updates a single document. To update many - {multi: true}
    - Upsert - to add a record if no match for the condition - {upsert: true}

CRUD Operations

- DELETE - delete data from a collection
  - Syntax -
    - db.*<collection name>.deleteOne(<condition>)*
    - db.*<collection name>.deleteMany(<condition>)*
    - db.*<collection name>.remove(<condition>, <options>)*
  - *<condition> - we know…*
  - <options> - justOne
    - Defaults to true - to limit deletion to one record
    - Set to false to allow multiple document deletion

CRUD Operations

- Advanced READ
  - Joining collections
    - Traditionally done in two steps
      - Find documents in collection1
      - Use the values of the “join” field from collection1 to find documents in collection2
    - Use $aggregate and $lookup
      - db.collection2.aggregate([

```
{$match: {
   key1: val1
}},
{$lookup: {
   from: “collection1”,
   localField: “key2”,
   foreignField: “key2”,
   as “key2_values”}
}
])
```

MongoDB in Python