Introducing MongoDB

Assessable Learning Outcomes

- What is MongoDB?
- Why is it useful?
- How can I
  - Use a database, containing collections
  - Query existing collections
- What are the design principles?
  - Design a 'schema' for use in an application
- Choose a design and a type of database.

Document Database

- A document is a data structure composed of field and value pairs.
- MongoDB documents are similar to JSON objects.
- The values of fields may include other documents, arrays, and arrays of documents.

MongoDB advantages

- MongoDB say the advantages of using documents are:
  - Documents (i.e. objects) correspond to native data types in many programming languages.
  - Embedded documents and arrays reduce need for expensive joins.
  - Dynamic schema supports fluent polymorphism.

Key Features

- High Performance
  - Supports embedded data models (reduces I/O)
  - Allows Indexes
- Rich Query Language
  - Read, write, aggregation, text search, geospatial.
- High Availability
  - Uses replication
- Horizontally scalable through sharding.
- Supports heterogeneous storage engines.

CAP Theorem

- C – Consistency
  - This is a given in a relational database
  - If a RDBMS is not in a consistent state, it is not available.
    - Remember what happens when two sessions are accessing the same data – only the consistent data is available to both.
- A – Availability
  - A MongoDB database will always have data available because it has replicas of every piece of data and it only promises 'eventual consistency'
- P – Partition tolerance (*tolerating network breaks*)
  - In a distributed situation, RDBMS will not allow a transaction to go through unless all nodes involved are in a consistent state.
  - In a distributed situation, MongoDB will go ahead with transactions for available nodes and let the others catch up later.

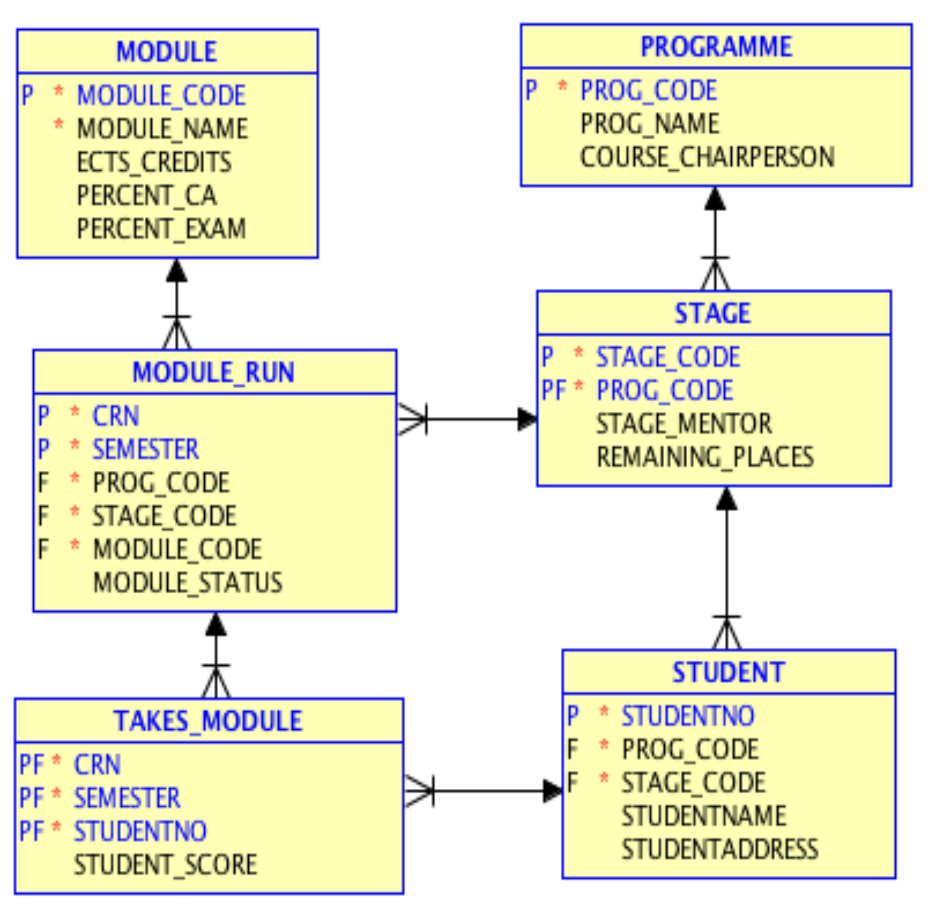
Using MongoDB

- We will experiment on a service that is hosted by DIT
- You will install your own version, on your laptop.

Remember

- There are no joins in a MongoDB query.
  - Queries may be piped, to result in the equivalent of a join.
- Transactions in MongoDB involving more than one instruction are not atomic.

Our student-modules ERD



- Note the ERD
  - There are a lot of weak entities
  - To get a transcript, a lot of joins are required.
  - The structure is not intuitive.

Scenarios in MongoDB

- Add a student collection, embedding the module codes in the student.

- Student
  - Studentno (can be used as _id)
  - StudentName
  - Prog_code
  - Stage_code
  - Modules:
    - Module_code
    - Student_score

Scenarios in MongoDB

- Student
  - Studentno (as _id)
  - StudentName
  - Prog_code
  - Stage_code
  - Modules:
    - Module_code
    - Student_score

```
db.student.insert({ _id :
'C22345678',
StudentName: 'Joe Bloggs',
prog_code: 'DT222',
stage_code: 3,
modules: [
{Module_code: 'CMPU3010',
Student_score: 87},
{Module_code: 'CMPU3048',
Student_score: 62}]})
```

```
db.student.insert([{
_id: 'C12345678',
studentname: 'Joe Bloggs',
prog_code: 'DT228',
stage_code: 3,
modules: [
{module_code: 'CMPU3010',student_score: 66},
{module_code: 'CMPU3047',student_score: 45}
]},
{
_id: 'C12345679',
studentname: 'Jane Bloggs',
prog_code: 'DT228',
stage_code: 4},
{_id: 'C12345670',
studentname: 'Enda Kenny',
prog_code: 'DT222',
prog_year: 1,
societies_joined: ['Young Fine Gael','Rowing']}
])
```

Sample entry

- Note:
  - We don't need to leave space for null attributes
  - We can add extra fields whenever we like.

```
db.student.insert([{
_id: 'C12345678',
studentname: 'Joe Bloggs',
prog_code: 'DT228',
stage_code: 3,
modules: [
{module_code: 'CMPU3010',student_score: 66},
{module_code: 'CMPU3047',student_score: 45}
]},
{
_id: 'C12345679',
studentname: 'Jane Bloggs',
prog_code: 'DT228',
stage_code: 4},
{_id: 'C12345670',
studentname: 'Enda Kenny',
prog_code: 'DT222',
prog_year: 1,
societies_joined: ['Young Fine Gael','Rowing']}
])
```

Sample entry

- Note:
  - We don't need to leave space for null attributes
  - We can add extra fields whenever we like.
  - MongoDB will not stop you from calling an attribute different names in different documents, but this hampers searching.

Or this way…

- Add a module collection, embedding the students who sit it in the module:

- Module
  - Modulecode
  - Title
  - ECTS credits
  - CA percent
  - Exam percent
  - Students:
    - StudentNo

Adding modules

- Module
  - Modulecode
  - Title
  - ECTS credits
  - CA percent
  - Exam percent
  - Students:
    - StudentNo

```
db.module.insert([{
    _id: 'CMPU3010',
    title: 'Databases 2',
    ECTS: 5,
    CAPercent: 40,
    ExamPercent: 60,
    Students: ['C12345678','C12345671']

},
{   _id: 'CMPU1011',
    title: 'Build a PC',
    ECTS: 5,
    CAPercent: 100,
Students:
['C16123456','C16123457','C16123458']}])
```

Or this way

- Module
  - Modulecode
  - Title
  - ECTS credits
  - CA percent
  - Exam percent
  - Students:
    - StudentNo (reference)
- Student
  - Studentno (can be used as _id)
  - StudentName
  - Prog_code
  - Stage_code
  - Modules:
    - Module_code (reference)
    - Student_score

Indexes in MongoDB

- As for relational databases, MongoDB indexes support the efficient execution of queries. Without indexes, MongoDB must perform a *collection scan* (the equivalent of a full-table scan in RDBMS).
- Indexes store a small portion of the collection's data set in an easy to traverse form. The index stores the value of a specific field or set of fields, ordered by the value of the field.

Indexes

- Every document is automatically indexed on the _id attribute at insertion.
- Indexes can be created for other attributes using the db.collection.createIndex () method.

Getting started

CAN WE REUSE THE DATA WE ALREADY HAVE?

There are ways to reuse data

- By hand
  - Type the code into notepad++
- Very simple (no embedding)
  - Export your SQL output to an excel file. Find a converter online and run it.
  - Write a SQL query to generate it
- A little harder (with embedding)
  - Write a PL/SQL program to do it or
  - Use XML SQL commands to generate it.

Write a SQL query:

- Spool redirects output to wherever it is sent, but the output needs a bit of work when it's done.
- `spool "C:\wherever\students.js"`
- `SELECT 'db.student.insert({_id: "'||studentno||'", studentname: "'||studentname||'", prog_code: "'||prog_code||'", stage_code: '||`
- `  stage_code||'})' FROM STUDENT;`
- `spool off;`

PL/SQL generation

- Use the spool feature again but run a program.
- The program needs 2 cursors
- For each student (use a student cursor):
  - Start the db..insert command
  - Use dbms_output.put (not put_line) to add attributes to it.
  - If the student takes modules , add a modules array:
    - For each module the student takes (new cursor)
      - Add the module code
    - Finish off the array
    - Close the cursor
  - Finish the insert command.

Additional material

- The MongoDB tutorial will get you up and running.
- There are some scripts that I have written / generated:
  - Studentxml.sql uses SQLXML to generate an XML file for conversion to JSON.
  - Students.xml is generated using SQL for XML. It can be easily converted to JSON.
  - Createflatstudentsql.sql uses SQL to generate students with no embedding
  - Createflatstudent.sql uses PL/SQL to generate students with no embedding
  - Createmongostudent.sql uses PL/SQL to generate students with embedded array of modules taken.