**Graph database based timetabling system**

**Overview of project:**

This is a semester 2 project for Graph Theory module of year 3 Software development course. Aim of this project is to develop a prototype timetabling system for GMIT like the one currently used. As this is a graph theory module, Neo4J database will be used to store the timetable data. As requirement was for the prototype timetable I have decided to make a small-scale timetable loosely based on my 3rd year software development course. As such, lab group count, programme name and subjects present are same as in real timetable for my course. I use small scale prototype as Neo4J is very scalable and as such a small prototype can accurately represent potential of the possible future timetabling system.

**Database used:**

Neo4J is going to be used for the storage of data for this project. It is a graph based database which stores data in nodes, connected by relationships between them thus the database itself is very intuitive to implement. Neo4J uses Cypher query language for its queries.

Neo4J can be downloaded for free from:

<https://neo4j.com/download/>

Once downloaded use the provided installer to install Neo4J on the local device. Once installed and started database can be accessed from Web interface on a URL: <http://localhost:7474>

Advantages of using Neo4J:

1. Schema-less;
2. Very efficient storage of information that is not fully structured;
3. ACID transaction support;
4. It is very easy to represent connected data in graph based database;
5. Navigation of connected data is very easy;
6. Doesn’t require complex Joins to retrieve connected data. It is very easy to get neighbouring nodes and relationship details

Disadvantages of Neo4J:

1. No sharding support. This means that entire database is in one server.

**Data to be stored in Database:**

Looking at the existing database model that is used here in GMIT. I have decided that main data to stored can be separated into these basic units:

1. Programme name;
2. Subject/module name;
3. Lab group ID;
4. Room name/number;
5. Timeslot for lecture to take place at;

Appropriate Neo4J nodes have been created for a small-scale prototype of the timetable. Every node in the timetable has a single property “Name” which allows to differentiate unique nodes among nodes of same type. I am using a single programme type node named Software Dev. as a root node for the prototype graph.

Cypher Query to create programme node:

**CREATE (p:Programme{Name:"Software Dev"});**

Cypher query to create a subject/module node:

**CREATE (s:Subject{Name:"Graph Theory"});**

Cypher query to create LabGroup node:

**CREATE (l:LabGroup{Name:”Group X”});**

Cypher query to create Room node:

**CREATE (r:Room{Name:”Room X”});**

Cypher query to create Timeslot node:

**CREATE (t:Timeslot{Name:”Slot X”});**

For the needs of prototype, I have created 1 Programme node (Software Dev), 2 Subject nodes (Graph Theory and Database Management), 3 LabGroup nodes (Group X, Group Y and Group Z), 3 Room nodes (Room X, Room Y, Room Z) and 4 Timeslot nodes (Slot W, Slot X, Slot Y and Slot Z).

This makes it a total of 13 nodes in small scale prototype database.

**Database structure:**

For the design of the database and how the layout of it should look like I decided to try and follow the logical flow of hierarchy.

1. Programme contains many modules in it;
2. Subject is being taught to 1 or more groups in the programme year;
3. Student group studies the given module in the assigned room;
4. Room is being given to group of students at certain timeslot;

From the above list 4 Relationships can be obtained:

1. Contains;
2. Taught to;
3. In;
4. At.

As such, these are the names for Relationships between the nodes in prototype database.

Cypher queries used to create above relationships:

1. **MATCH (p:Programme{Name:"Software Dev"}), (s:Subject{Name:"Database Mgmt"}), (s2:Subject{Name:"Graph Theory"}) CREATE (p)-[r:Contains]->(s), (p)-[r:Contains]->(s2) return p, s, s2;**
2. Following query is for single relationship only. Other relationships of this type were created by changing LabGroup name accordingly. **MATCH (s:Subject{Name:"Graph Theory"}), (l:LabGroup{Name:"Group X"}) CREATE (s)-[r:Tought\_To]->(l);**

For Database Management to LabGroup relationships I created all relationships in single query:

**MATCH (s:Subject{Name:"Database Mgmt"}),(la:LabGroup{Name:"Group X"}),(lb:LabGroup{Name:"Group Y"}), (lc:LabGroup{Name:"Group Z"}) CREATE (s)-[:Tought\_To]->(la), (s)-[:Tought\_To]->(lb), (s)-[:Tought\_To]->(lc);**

1. Single example of creating relationship between LabGroup X and all the rooms this group studies in: **MATCH (l:LabGroup{Name:"Group X"}), (r:Room) CREATE (l)-[rel:In]->(r);**  Other group to room relationships were created in same way.
2. Single example of relationship creation between multiple timeslots that room can be assigned to and a single room: **MATCH (r:Room{Name:"Room X"}), (t:Timeslot) CREATE (r)-[rel:At]->(t);**

The chain of nodes and relationships in the timetable starts at the programme and looks like this:

Programme X-[Contains]->Subject X-[Taught\_to]->LabGroup X-[In]->Room X-[At]->Timeslot X time.

This layout of nodes and relationships between them is most logical in my opinion.

**Observations:**

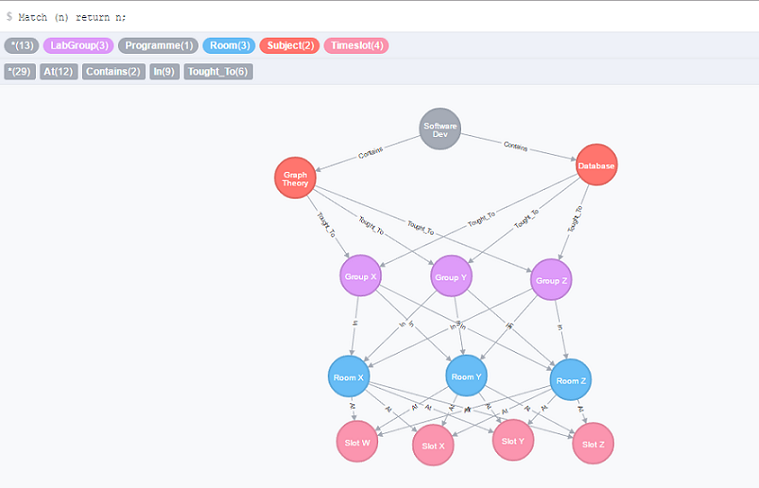
Also, part of database design and relationships coming out of my approach to it there are the several rules for timetable:

1. Same room can only be assigned to single timeslot at a time;
2. Same student group can be in only one room per timeslot;
3. Same student group can be in same room at different timeslots (Example of multi hour lectures/labs or different subjects being taught in same room to same group of students in succession).

**Outcome of my approach:**

Once implemented, the database has main logical relationships in place for easy tracking of what subject is being taught to which group in what room at the given timeslot. My target was to make the timetable easy to navigate and understand.

Final prototype of the database looks like this:



Observations and probable development beyond prototype stage:

* Restrictions are needed for the relationships. Once room has been assigned to timeslot for a specific group, there should be no way for other group to book same room on same timeslot.
* Programme unique lab group ID’s should be introduced once timetable moves beyond single programme. This is due to fact that different programmes have different number of groups (due to difference in student numbers studying given course).
* Some subjects are taught in multiple programmes at the same time. This could mean introducing unique ID for subjects as well.
* For added logic once programmes from different schools (like science, engineering or arts) are added the supervising departments can be added as nodes above programme nodes. Possible relationship between department and programme nodes can be “Supervises” as departments usually supervise programmes that are being taught in their respective schools in college.
* If implemented in full the database timetable would have full data sets of available rooms and programmes in college. Some extra information would still be needed as number of lab groups varies from programme to program. This information would need to be implemented manually. Full set of timeslots would be corresponding with college opening times. With opening times counted as 9am to 10pm that would make it for 13 timeslots per day and 65 timeslots per working week.