Graph Time

A proof-of-concept data engineering project

Petter Lovehagen

2024-08-29

Table of contents

1 Exploring Graph Data Models for Timetabling Insights

A proof-of-concept data engineering project

Supervisor: Xiaodong Li

Programme: MSc Data Science

Institution: University of the West of England, Bristol

Intro Video:

Abstract:

Timetables are central to the daily experience of university students and staff. Timetables also influence resource utilisation and play a key role in institutional efficiency. In short, timetables are critical to the individual experience and the overall success of an institution. But timetables are also contentious - there is no 'perfect' timetable, only a series of compromises that balance competing priorities.

This project explores the use of graph data models to provide deeper insights into timetables. The aim is to investigate the viability of graph-based approaches for enhanced timetabling analytics and reporting. The objectives include designing a graph data model, developing a configurable ETL pipeline, and discussing how graph-based analysis could contribute to quantitatively measuring timetable quality by introducing the concept of a *Timetable Quality Index*.

The project is a proof-of-concept, and the results are intended to inform future research and development in timetabling analytics. The project is implemented primarily in Python, using the Neo4j cloud instance graph database, and the results and documentation are presented in a Quarto website.

Table 1.1: Approximate Word Count Breakdown

Section	Word count
Abstract	170

Section	Word count
Main Sections (excl. tables, code, images, references, footnotes, headers)	7189
Appendices (**inc**. everything)	14082

2 Introduction

University timetabling is the process of scheduling resources within the constraints of an academic institution and calendar. At its core, it involves collecting and combining time slots, rooms, students, and other resources while satisfying a multitude of constraints and preferences to achieve a viable outcome.

However, the reality of timetabling is far more complicated than this simple definition suggests.

Timetablers must juggle numerous hard constraints (e.g., room capacities, pre-assigned times) and soft constraints (e.g., staff preferences, student travel times) to reach a workable solution. The scale of this task, combined with interdependencies between scheduling decisions, makes university timetabling one of the most challenging administrative tasks in higher education (de Werra, 1997).

Timetables can make or break a university - they shape the *daily* experiences of students and staff, influence resource utilisation, and play a significant role in institutional efficiency. The complexity of timetabling stems from various factors:

- **Scale**: Tens of thousands of students and activities, and limited resources create a logistical nightmare.
- **Constraints**: Juggling hard limits (room capacities) and soft preferences (College desires) is a constant balancing act.
- **Interdependencies**: Changes in one part of the schedule can have cascading effects throughout the entire timetable.
- **Diversity of Needs**: Different organisational units (colleges, faculties, schools, departments) have varying requirements and preferences.
- **Optimisation Goals**: Timetablers must balance efficiency, fairness, and quality of education.

While traditional studies on "timetabling" focus heavily on generating or optimising feasible timetables (Bonutti et al., 2012; Ceschia, Di Gaspero and Schaerf, 2023; Rudová, Müller and Murray, 2011) – ensuring no clashes or rule violations – this project explores a different facet: how analysing scheduled timetables can lead to deeper insights and ultimately, improved quality for all stakeholders.

3 Background and Motivation

3.1 Personal

Many years ago, I grappled with the complexities of timetable generation and optimisation, and battled with trying to balance competing, but conflicting demands like maximising room utilisation **and** adhering to staff working patterns **and** producing a 'decent' timetable for the students. It is an *unwinnable* battle.

These experiences and challenges left an indelible mark - highlighting the need for robust tools and metrics to understand and assess timetable quality - a factor which is often overshadowed by the pursuit of mere *feasibility*.

This project is the result of a *deliberate clash* of my professional experiences and data science learning where I aim to deliver a practical solution to a real-world problem.

3.2 Research Gap: Bridging Theory and Practice

Much current research into university timetabling centres on combinatorial optimisation (Chen et al., 2021), that is using various sophisticated techniques designed to efficiently generate feasible solutions given a set of constraints. This computationally-driven optimisation research is often referred to as the university course timetabling problem (UCTTP) and is categorised as NP-hard[1], meaning finding the absolute "best" timetable is exceptionally challenging (Babaei, Karimpour and Hadidi, 2015; Herres and Schmitz, 2021; Wikipedia contributors, 2024).

Consequently, significant effort has been dedicated to developing algorithms like constraint programming (Holm et al., 2022) and local search techniques such as Tabu Search and simulated annealing (Oude Vrielink et al., 2019), aiming to create workable timetables within reasonable timeframes. While crucial for advancing algorithmic development, these idealised scenarios[2] do not fully capture the dynamic complexity of real-world university timetabling.

[1] "In computational complexity theory, a computational problem H is called **NP-hard** if, for every problem L which can be solved in non-deterministic polynomial-time, there is a polynomial-time reduction from L to H." (Wikipedia contributors, 2024)

[2] Research on computational optimisation often makes use of standardised datasets and predefined constraints in order to facilitate comparison, repeatability and evaluation.

Universities grapple with constantly shifting demands: fluctuating student populations, evolving institutional preferences, resource limitations, and the ever-present need to balance diverse stakeholder needs. These complexities extend beyond simply finding a feasible solution – they necessitate tools to understand the trade-offs inherent in any timetable, enabling informed decisions about which "good" outcomes to prioritise (Lindahl, 2017).

3.3 Hypothesis...Enter the Graph

This is where I believe graph data structures *could offer unique potential*.

Timetables are inherently about relationships: curriculum linked to lecturers, students connected through shared modules, rooms associated with specific times and capacities. Graph databases excel in this domain, offering a way to unlock insights hidden within the complex web of a university timetable.

While algorithms generate optimised solutions, there remains a gap in post-generation analysis – e.g. the ability to delve into a timetable's nuanced impacts on student and staff experience. Despite the acknowledged importance and impact of factors like room allocation and teaching period distribution, traditional optimisation-focused approaches lack the tools to explore these relationships in depth (Ceschia, Di Gaspero and Schaerf, 2023; Lindahl, 2017; Rudová, Müller and Murray, 2011), particularly in a real-world scenario.

This potential for deeper analysis motivates the exploration of graph data structures for enhancing timetable understanding and, ultimately, improving timetable quality for all stakeholders.

4 What is a “good” timetable?

One of the most challenging aspects of university timetabling is defining what constitutes a “good” timetable. Despite best efforts, it is virtually impossible to deliver universal satisfaction from a university timetable. The quality of a timetable is inherently subjective and varies among stakeholders depending on their preferences and the demands on their time.

Based on surveys across various institutions, students typically prioritise (Dowland, 2018; Norman, 2022):

- A clash-free timetable
- Early release of timetables
- Clear and accurate information
- Full-year timetable availability
- Minimal changes after publication
- Effective communication of any changes
- Balanced distribution of classes

The above are easy-to-measure deliverables but they do not address what a ‘good’ timetable should look like; individual stakeholders often have conflicting priorities:

- **Students** may prefer learning times that align with their personal commitments (e.g. part-time employment or travel)
- **Academic staff** may prefer teaching times that align with their research or personal commitments.
- **Administrators** might focus on efficient resource utilisation and institutional sustainability.
- **Organisational units** may have specific needs for specialised rooms or equipment.

This divergence in preferences and the complex interplay of constraints make it challenging to define and achieve a universally “good” timetable (Lindahl et al., 2018). It is this complexity that sets the stage for this project.

4.1 Consider these timetables:

- The first timetable is evenly spread over five days.
- The second timetable has two days free of activities.

- The third timetable has activities on five days, with gaps.

Which timetable is better? Is any of them 'good'? The answer is *it depends!* or *none of them!*

(Click to enlarge)

Week 13 - w/c 13/11/2023 or 11/11/2024

	8:00	8:30	9:00	9:30	10:00	10:30	11:00	11:30	12:00	12:30	13:00	13:30	14:00	14:30	15:00	15:30	16:00	16:30	17:00	17:30	18:00	18:30	19:00	19:30
Mon																					UPHPKC-30-3 PreRec/01 Pre-recorded Material Applied Historical Research 7-17, 24-32, 35-36			
Tue			UPHPJY-30-3 L01_oc/01 Lecture (On Campus) Stalin and Stalinism 3S703 FR 7-13, 15-17, 24-32, 35-36		UPHPJY-30-3 S01_oc/01 Seminar (On Campus) Stalin and Stalinism 4D007 FR 7-13, 15-17, 24-32, 35-36					UPHPL5-30-3 TB1 L01_oc/01 Lecture (On Campus) Mafias, Mythologies and Criminal Networks: the United States and the Globalization of Crime 2S609 FR 7-17							UPHPL5-30-3 W01_oc/02 Workshop (On Campus) Mafias, Mythologies and Criminal Networks: the United States and the Globalization of Crime 2S610 FR 7-17, 24-32, 35-36							
Wed																								
Thu																								
Fri									UPHPLH-30-3 L01_oc/01 Lecture (On Campus) International Politics in North Africa and the Middle East 3B064 FR 7-17, 24-31, 35-36		UPHPLH-30-3 W01_oc/01 Workshop (On Campus) International Politics in North Africa and the Middle East 3S511 FR 7-17													

Week 13 - w/c 13/11/2023 or 11/11/2024

	8:00	8:30	9:00	9:30	10:00	10:30	11:00	11:30	12:00	12:30	13:00	13:30	14:00	14:30	15:00	15:30	16:00	16:30	17:00	17:30	18:00	18:30
Mon																	UMADQ8-15-1 D_ol/01 Drop In (Online - Live) Essential Information Skills 7-17					
Tue			PAL Accounting TB1_oc/07 JT PAL (On Campus) Accounting PAL Leader(s): PAL 2N24 FR 7-17										UMADQ8-15-1 CP_oc/08 <12-13> Computer Practical (On Campus) Essential Information Skills 3X229 FR 12-13		UMED8D-15-1 L1_oc/01 Lecture (On Campus) Economic Principles in a Contemporary Context 2D67 FR 7-17		UMADMY-15-1 GT_oc/08 Group Tutorial (On Campus) Foundations of Financial Accounting 3X107 FR 7-17					
Wed			UMADQ7-15-1 Le1_oc/01 Lectorial (On Campus) Introduction to Management Accounting 2B025 FR 7-17		UMADMY-15-1 D_ol/01 Drop In (Online - Live) Foundations of Financial Accounting 7-17				UMADQ7-15-1 Le2_oc/01 Lectorial (On Campus) Introduction to Management Accounting 2X112 FR 7-17										CBL CH TB1 AF1 W02_ol/01 Support Session (Online - Live) Academic Support Hour - Time Management 13			
Thu							UMADMY-15-1 Le_oc/01 Lectorial (On Campus) Foundations of Financial Accounting 2X112 FR 7-17		UMADQ8-15-1 Le_oc/01 Lectorial (On Campus) <7-13, 17> Essential Information Skills 2X112 FR 7-13, 17													
Fri					UMED8D-15-1 GT_oc/08 Group Tutorial (On Campus) Economic Principles in a Contemporary Context 3C002 FR 7-17		UMADQ7-15-1 GT_oc/08 Group Tutorial (On Campus) Introduction to Management Accounting 4X103 FR 7-17															

5 Project Aims and Scope

The primary aim of this project is to **investigate the viability of using graph data structures** for enhanced timetabling analytics and reporting.

5.0.1 Objectives

1. Designing an **extensible, system-agnostic graph data model** for university timetables
2. Developing a **configurable ETL** (extract, transform, load) pipeline to transition from relational to graph database representations of timetables
3. Discussing how **graph-based approaches to timetabling analysis** could contribute to measuring and improving timetable quality.

To achieve these objectives, I implement and evaluate a set of proof-of-concept analytical metrics which leverage the graph data model whilst discussing performance capabilities (and limitations) against traditional relational approaches.

It is important to note that this project is positioned as a proof-of-concept and exploratory study.

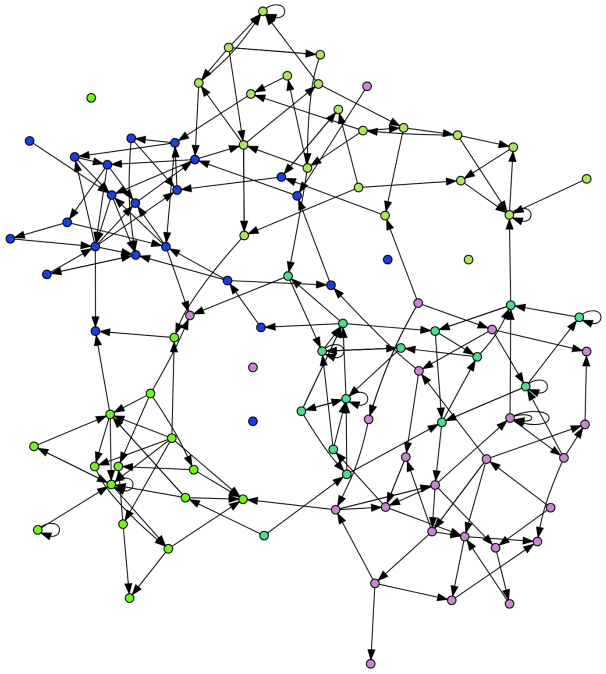
I will **not**:

- reinvent a full-scale timetabling system
- attempt to optimise real-time timetable generation

I will:

- focus on the data engineering aspects of transitioning from relational to graph data models
- demonstrate the potential of graph-based approaches in the analysis of university timetables
- provide a foundation for future analytical work.

Let's graph!



: A randomly generated graph for illustrative purposes.

See Appendix for graph generator code

6 Graph vs Relational Data Models

As outlined in the project aims, my hypothesis is that graph-based approaches have the *potential* to offer new insights and efficiencies in timetable analysis. This section will briefly explore the theoretical underpinnings of graph data structures and their application to the domain of university timetabling.

6.1 Relational Models

6.1.1 Tables, Joins and the Limits of Interconnectedness

Relational databases, using SQL[1] as their query language, have long been the go-to for managing data, including timetabling information. They structure data into tables, where rows represent instances of entities (e.g. individual rooms, staff, or students) and columns represent entity attributes (name, capacity, email, etc.).

Relationships between these entity tables are established through foreign keys, forming links between tables. This often involves intermediary "relationship" tables to handle the many-to-many nature of timetabling data (e.g., a student attends many activities, and an activity has many students) (Khan et al., 2023;Sokolova, Gómez and Borisoglebskaya, 2020).

While robust and well-understood, relational databases start to show their limitations when dealing with the highly interconnected nature of timetables:

- **Join Complexity**: Even seemingly simple queries, like *"find students attending a specific lecturer's class in a particular building,"* require joining multiple tables. As queries become more nuanced, the number of joins increases, often impacting performance, especially with large datasets.
- **Rigidity**: Relational databases rely on a predefined schema, making them less adaptable to evolving needs. Adding new entities or relationships is not possible without disrupting existing queries and applications.

[1] Structured Query Language (Wikipedia contributors, 2024)

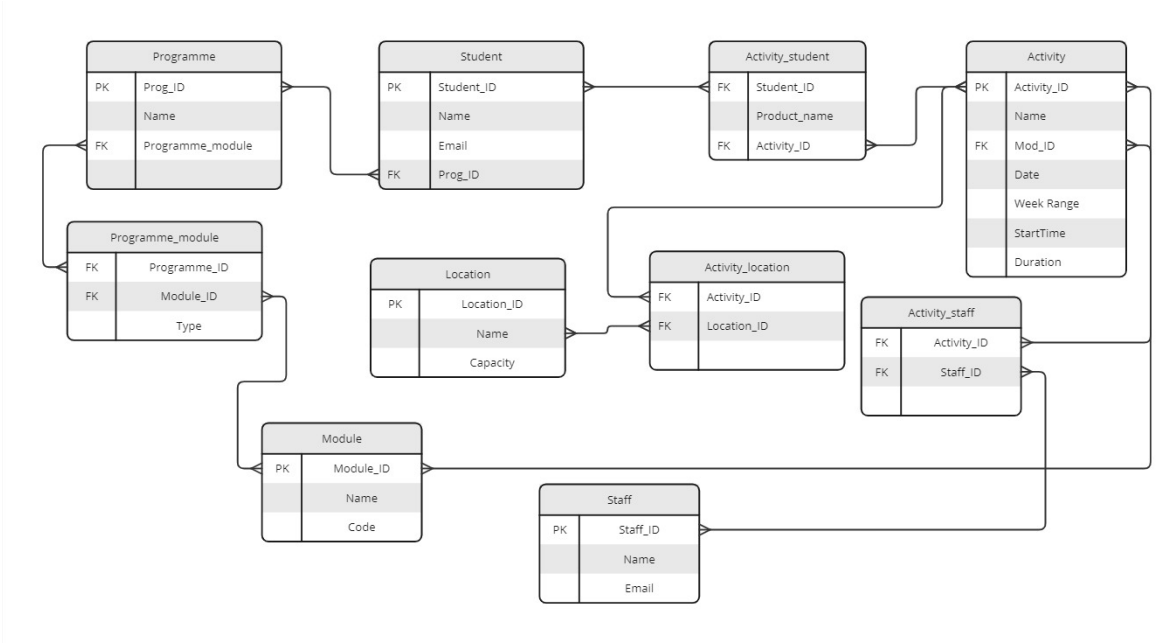


Figure 6.1: Example Simple Entity Relationship Diagram

6.2 Graph Models

6.2.1 Embracing interconnectedness

In contrast to the rigid table structure of relational databases, graph databases offer a more intuitive and flexible approach for representing interconnected data like timetables. They utilise:

- **Nodes**: Represent entities. These are often the "nouns" like activity, room, staff, student.
- **Edges**: Represent relationships between nodes. These are often the "verbs" like TAUGHT_BY, ENROLLED_IN, SCHEDULED_AT, OWNED_BY.

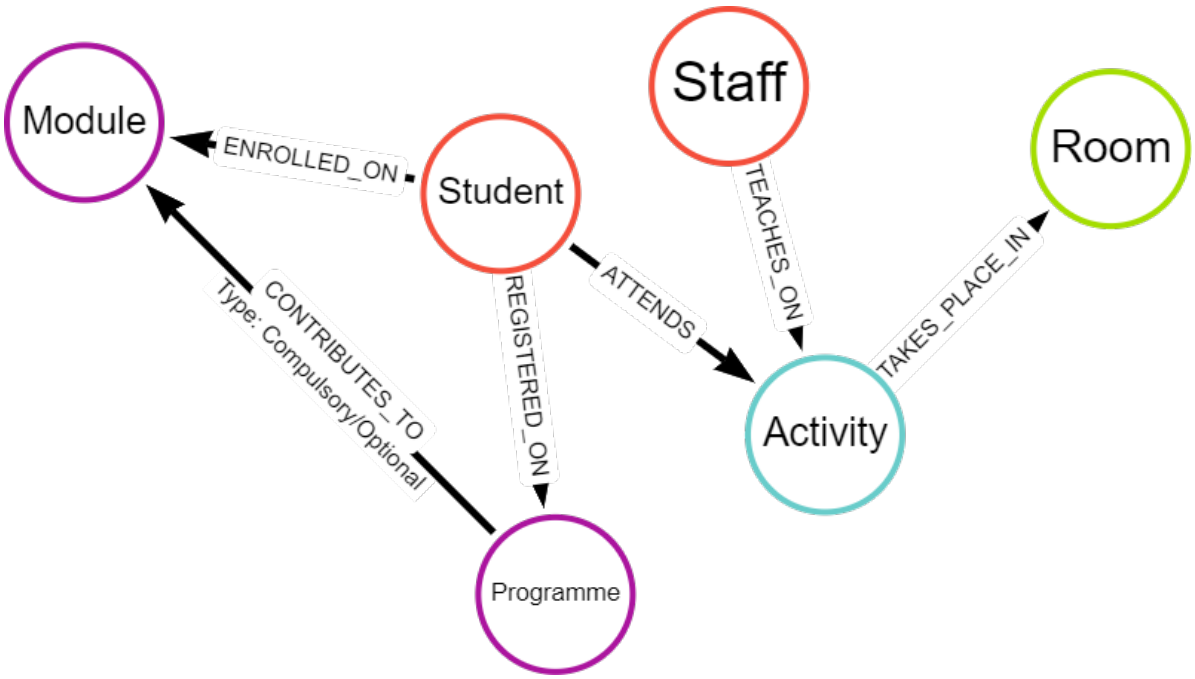


Figure 6.2: Simple Graph Data Model

This node-and-edge structure inherently reflects how timetabling elements connect. Instead of relying on cumbersome joins, relationships are directly encoded in the data model itself. This results in several advantages:

- **Natural Representation**: Graph databases visually and conceptually mirror the relationships inherent in timetables, making them easier to understand and query.
- **Relationship-Centric Queries**: Graph databases are optimised for traversing and analysing relationships. Queries that would require multiple joins in a relational database often become significantly simpler and faster in a graph database.
- **Flexibility**: The schema-less or schema-optional nature of most graph databases allows for greater flexibility in data modeling. New entities or relationships can be added effortlessly without impacting existing structures or queries (Nan and Bai, 2019; Webber, Eifrem and Robinson, 2013).

6.3 Comparing queries

Example Insight: Find all students attending a specific lecturer's class in a particular building

Representative queries have been written in SQL and Cypher to find this insight. The SQL query is much longer and requires six joins, each coming at a computational cost.

6.3.1 SQL Query

```
SELECT DISTINCT ss.[FirstName], ss.[LastName], ss.[Email]
FROM [RDB_MAIN2223].[rdowner].[V_STUDENTSET] ss
INNER JOIN [RDB_MAIN2223].[rdowner].[V_ACTIVITY_STUDENTSET] acts ON ss.[Id] = acts.[StudentSe
INNER JOIN [RDB_MAIN2223].[rdowner].[V_ACTIVITY] a ON acts.[ActivityId] = a.[Id]
INNER JOIN [RDB_MAIN2223].[rdowner].[V_ACTIVITY_LOCATION] al ON a.[Id] = al.[ActivityId]
INNER JOIN [RDB_MAIN2223].[rdowner].[V_LOCATION] l ON al.[LocationId] = l.[Id]
INNER JOIN [RDB_MAIN2223].[rdowner].[V_BUILDING] b ON l.[BuildingId] = b.[Id]
INNER JOIN [RDB_MAIN2223].[rdowner].[V_ACTIVITY_STAFF] ast ON a.[Id] = ast.[ActivityId]
WHERE ast.[StaffId] = 'StaffID'
  AND b.[Name] = 'BuildingName';
```

6.3.2 Cypher Query

In contrast, the Cypher query pattern is much simpler - written in one line (MATCH pattern). The query is more intuitive and easier to understand, especially for those unfamiliar with the database schema.

```
MATCH (s:Student)-[:ATTENDS]->(a:Activity)<-[:TEACHES_ON]-(st:Staff),
      (a:Activity)-[:TAKES_PLACE_IN]->(r:Room)
WHERE st.last_name = "LecturerLastName" AND r.building = "BuildingName"
RETURN s.first_name, s.last_name, s.email
```

6.4 Key Differences and Implications

Feature	Relational Model	Graph Model
Data Structure	Tables with rows and columns	Nodes and edges
Schema	Rigid, predefined	Flexible, schema-less or schema-optional
Relationship Handling	Foreign keys, joins	Direct connections (edges)

Feature	Relational Model	Graph Model
Query Performance	Can be slow for relationship-heavy queries	Optimised for traversing relationships, potentially faster
Data Modeling	Less intuitive for interconnected data	Naturally represents complex relationships
Adaptability	Less adaptable to schema changes	More flexible, accommodates evolving data needs

These advantages position graph databases as a powerful tool for uncovering insights hidden within complex, interconnected datasets like university timetables.

7 Graph Data Model for Timetabling

Having discussed advantages of graph databases for representing interconnected data, this section delves into the specifics of a proposed graph data model tailored for university timetabling.

7.1 An Iterative Approach

Due to flexibility, creating graph data models is an iterative process: design -> build -> test -> review -> revise -> …and repeat.

My first model was small in scope, incorporating *minimal* nodes and properties in an MVP[1] approach. Eventually, my expanded model was created in a cloud-instance of Neo4j Aura.

7.2 Core Nodes - Building Blocks

At its core, the timetable model revolves around four key entities represented as nodes:

Node	Property	Description	Data Type
Student	firstName	Legal first name	string
	lastName	Legal last name	string
	studentID	University identifier	integer
	splusID	Timetable URN	string
Lecturer	firstName	First name	string
	lastName	Last name	string
	staffID	University identifier	integer
	splusID	Timetable URN	string
Room	name	Room name	string
	splusID	Timetable URN	integer
Activity	name	Activity name	string

[1] Minimum Viable Product: "First, a definition: the minimum viable product is that version of a new product which allows a team to collect the maximum amount of validated learning about customers with the least effort or in other words building the most minimum version of their product that will still allow them to learn." (Ries, 2024)

Node	Property	Description	Data Type
	description	Activity description	string
	startTime	Scheduled start time	datetime
	endTime	Scheduled end time	datetime
	date	Date of activity	date

7.3 Relationships - Connecting the Dots

The core nodes are interconnected through relationships that reflect the dynamics of a timetable:

- (Student)-[**IS_ALLOCATED_TO**]->(Activity)
- (Staff)-[**TEACHES_ON**]->(Activity)
- (Activity)-[**TAKES_PLACE_IN**]->(Room)

7.4 MVP model

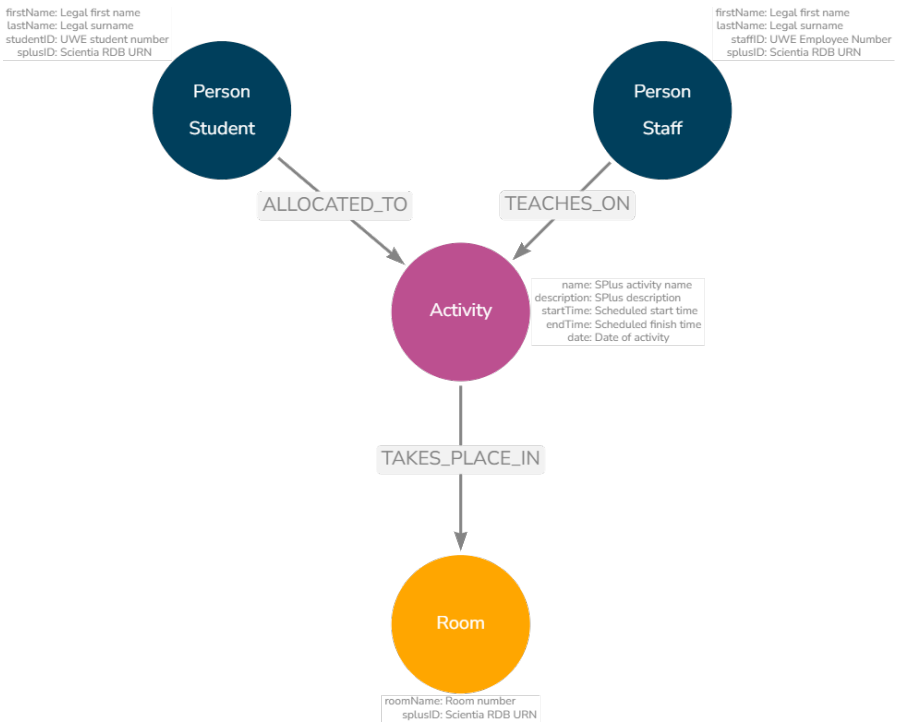


Figure 7.1: Core Nodes and Properties

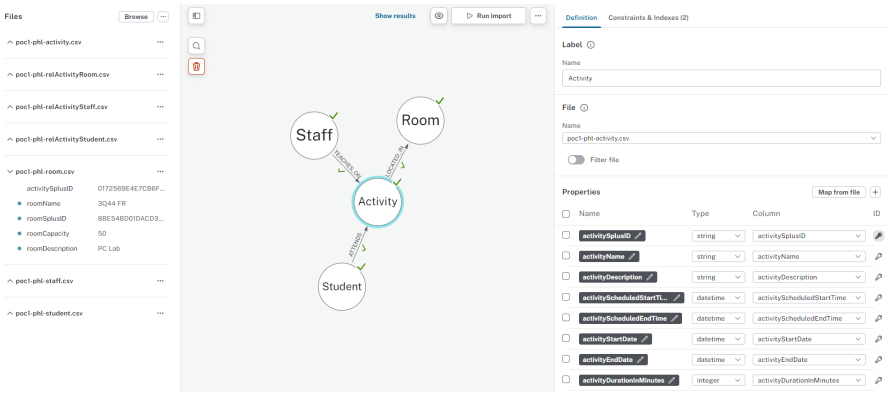


Figure 7.2: Neo4j Interface showing basic nodes and properties

8 Early Insights

8.1 Unveiling Basic Patterns

Even with this basic model, we can easily extract valuable insights, for example:

- **Activity Load**: Identify staff with the highest number of teaching activities or total teaching hours.
- **Student Timetable Profiles**: Calculate average hours per student or per programme to understand workload distribution.
- **Resource utilisation**: Determine the busiest teaching locations or times on campus.
- **Anomaly detection**: Identify students who have unexpected profiles or unusual combinations.

8.1.1 Example code

Busiest locations overall

```
MATCH (r:room)<-[:OCCUPIES]-(a:activity)
WITH r, sum(a.actDuration)/60 AS totalDurationInHours
RETURN r.roomName AS Room, r.roomCapacity AS Capacity, r.roomType AS Type, totalDurationInHou
ORDER BY totalDurationInHours
DESC LIMIT 3
```

Room	Capacity	Type	totalDurationInHours
2Q12 FR	25	PC LAB	21
4Q69 FR	36	PC LAB	19
3E11 FR	48	TEACHING	18

Busiest location for a specific time

```
MATCH (r:room)<-[:OCCUPIES]-(a:activity)
WHERE a.actStartTime = localtime({hour:9, minute:0, second:0})
WITH r, count(a) AS Count, a.actStartTime AS StartTime
RETURN r.roomName AS Room, Count, StartTime
ORDER BY Count DESC
LIMIT 3
```

Room	Count	StartTime
2Q12 FR	86	09:00:00
3E28 FR	50	09:00:00
3E11 FR	49	09:00:00

Students with below/above average hours

This query returns students and whether they have more or less scheduled time on their timetable compared to the programme cohort average. [1]

```
MATCH (s:student)-[:ATTENDS]->(a:activity)
WITH s.stuProgName AS progName, s.stuID_anon AS studentID, SUM(a.actDurationInMinutes) AS stu
WITH progName,
     AVG(studentTotalDuration) / 60 AS progAverageHoursPerStudent,
     collect({studentID: studentID, studentTotalHours: studentTotalDuration / 60}) AS student
UNWIND studentsData AS studentData
RETURN progName,
       progAvgHrsPerStudent,
       studentData.studentID AS studentID,
       studentData.studentTotalHours AS studentTotalHours,
       CASE
           WHEN studentData.studentTotalHours < (progAverageHoursPerStudent * 0.9) THEN 'Belo
           WHEN studentData.studentTotalHours > (progAverageHoursPerStudent * 1.1) THEN 'Abov
           ELSE 'Average'
       END AS compare
```

[1] The average is calculated based on the total duration of activities attended by each student. The below/above average classification is based on a 10% deviation from the average. Alternative approaches have been used to define the average and the deviation threshold including median values and standard deviations.

Table 8.3: Using Percentage

prog Name	progAvgHours PerStudent	studentID	student TotalHours	compare
“Maths NS”	274.7692307692308	“stu-23442558”	361	“Above Average”
“Maths NS”	274.7692307692308	“stu-91911371”	126	“Below Average”
“Maths NS”	274.7692307692308	“stu-75224499”	251	“Average”

When using standard deviation (1SD) the same three students have a different outcome. This is primarily due to the exceptionally large standard deviation. This is because students on this version of the programme could be trailing modules and either attending significantly more or less activity, making the range very large.

Table 8.4: Using Standard Deviation

prog Name	progAvgHours PerStudent	progStDevHours PerStudent	studentID	student TotalHours	compare
“Maths NS”	274.7692307692308	108.81231781219205	“stu-23442558”	361	“Average”
“Maths NS”	274.7692307692308	108.81231781219205	“stu-91911371”	126	“Below Average”
“Maths NS”	274.7692307692308	108.81231781219205	“stu-42997469”	251	“Below Average”

9 Model Expansion

The true power of the graph model lies in its extensibility. Introducing additional nodes and properties allows for a more comprehensive representation and enables more sophisticated analysis. The resulting graph model will depend on desired use cases and performance requirements, but the following are some potential expansions to the basic model:

9.0.1 Potential Expansions:

- **Organisational Units**: Include departments, colleges, or schools to analyse timetabling within organisational structures.
- **Curriculum Data**: Incorporate modules and programmes to understand the interconnectedness of courses and student enrolment patterns.
- **Activity Types**: Differentiate between lectures, seminars, labs, etc., for a more granular analysis of teaching and learning activities.
- **Activity Delivery**: Understand teaching delivery (virtual, in-person, hybrid, drop-in).
- **Student Attributes**: Add properties like "international student", "reasonable adjustment flag", "first-year student".

The below image (click to enlarge) shows a graph model augmented with additional data contained within the timetable database. It is much richer and therefore more complex, but this allows for richer analysis.

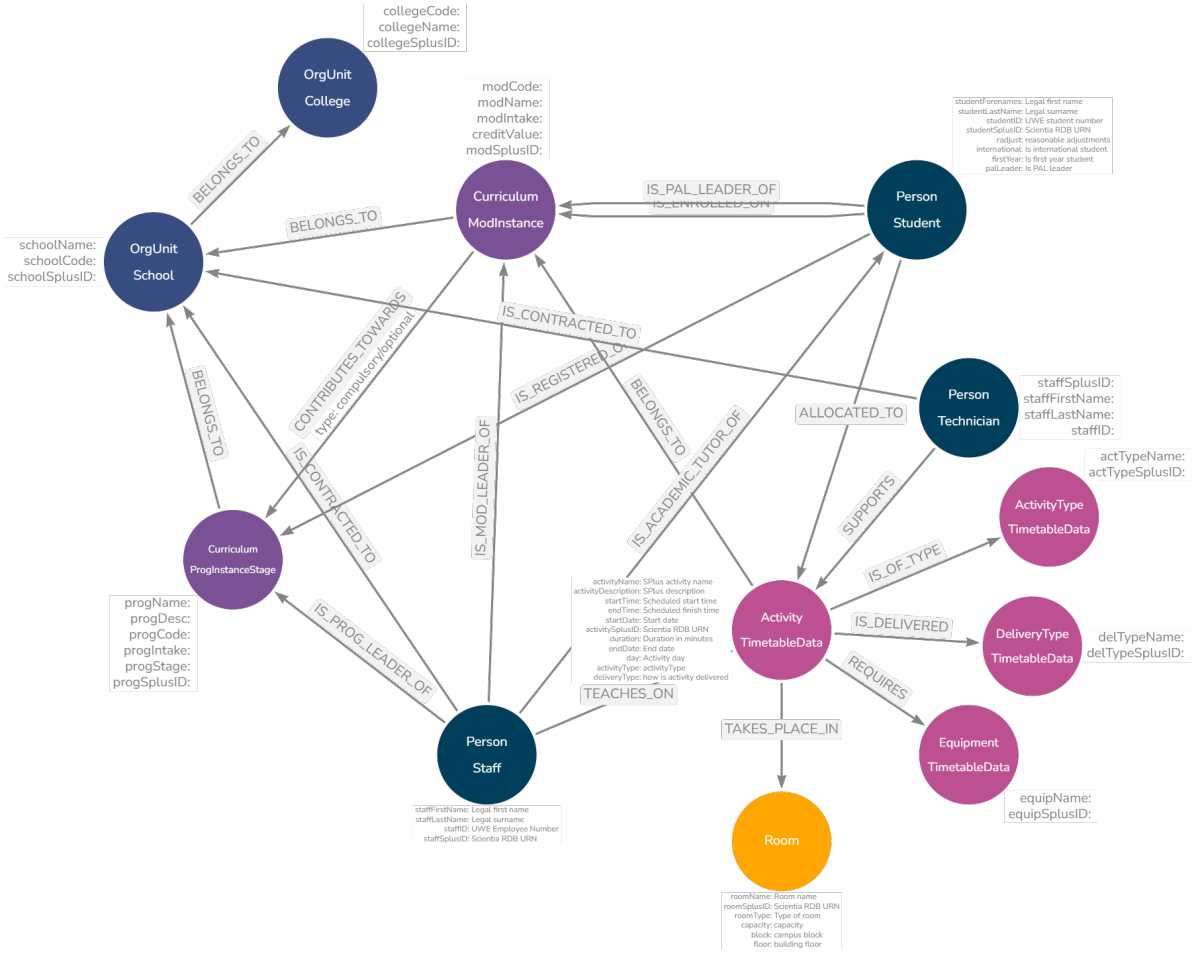


Figure 9.1: Example of Expanded Timetable Graph Model

10 Graphing Time

The biggest challenge I encountered when translating timetables into graph data involved temporal elements - that is, `start` and `end times`, `dates`, `weeks`, `recurrences`, `durations`, etc. While the basic model successfully captured the core entities and relationships, it lacked the necessary detail to perform meaningful time-based analysis.

The flexibility of graph databases is appealing but finding the optimal balance between efficient representation, query performance, and data redundancy requires careful consideration. This section details some challenges encountered and the approach taken for the proof-of-concept.

10.1 The Problem of Normalised Time

Traditional relational databases often store timetable information in a highly normalised[1] format, condensing recurring events into single rows with date ranges, week patterns, or lists of occurrences. While efficient for storage and basic display, this approach severely hinders analysis, especially when aiming to:

- **Identify Time-Based Patterns**: Determining if students lack lunch breaks or experience excessive gaps between classes becomes difficult when time is fragmented across multiple fields.
- **Perform Aggregations**: Calculating total teaching hours for a lecturer across specific weeks or days requires complex queries and data transformations.
- **Model Temporal Relationships**: Representing relationships between activities based on their temporal proximity, such as students attending consecutive classes, becomes convoluted.

10.2 Exploring Potential Solutions

Several time modelling approaches were considered, each with its own trade-offs.

[1] "Normalization is the process of organizing data in a database. It includes creating tables and establishing relationships between those tables according to rules designed both to protect the data and to make the database more flexible by eliminating redundancy and inconsistent dependency." (helenclu, 2024)

To illustrate this, let's explore using a fictional example - `Introduction to Graph Databases` - focusing on the `lecture`:

Table 10.1: Example Source Data (Relational)

Name	Activity Type	Day	StartTime	EndTime	Weeks
ITGD	Lecture	Wednesday	09:00	11:00	1-3, 5-7, 9-13
ITGD	Seminar	Wednesday	10:00	13:00	4, 7-8, 15
ITGD	Seminar	Monday	13:00	16:00	4, 7-8, 15

10.2.1 Option 0: Proof-of-concept activity

The basic model creates nodes for each activity *exactly* as they exist in the relational database. This simple approach is perfectly acceptable but makes any time-based calculations difficult because each activity node can represent a different number of occurrences due to the week ranges.

This in turn means you cannot simply COUNT each activity "equally" - for example, the lecture has 11 instances, each of two hours. The seminars have four instances, each of three hours. Calculating aggregations, finding clashes and similar is very challenging.

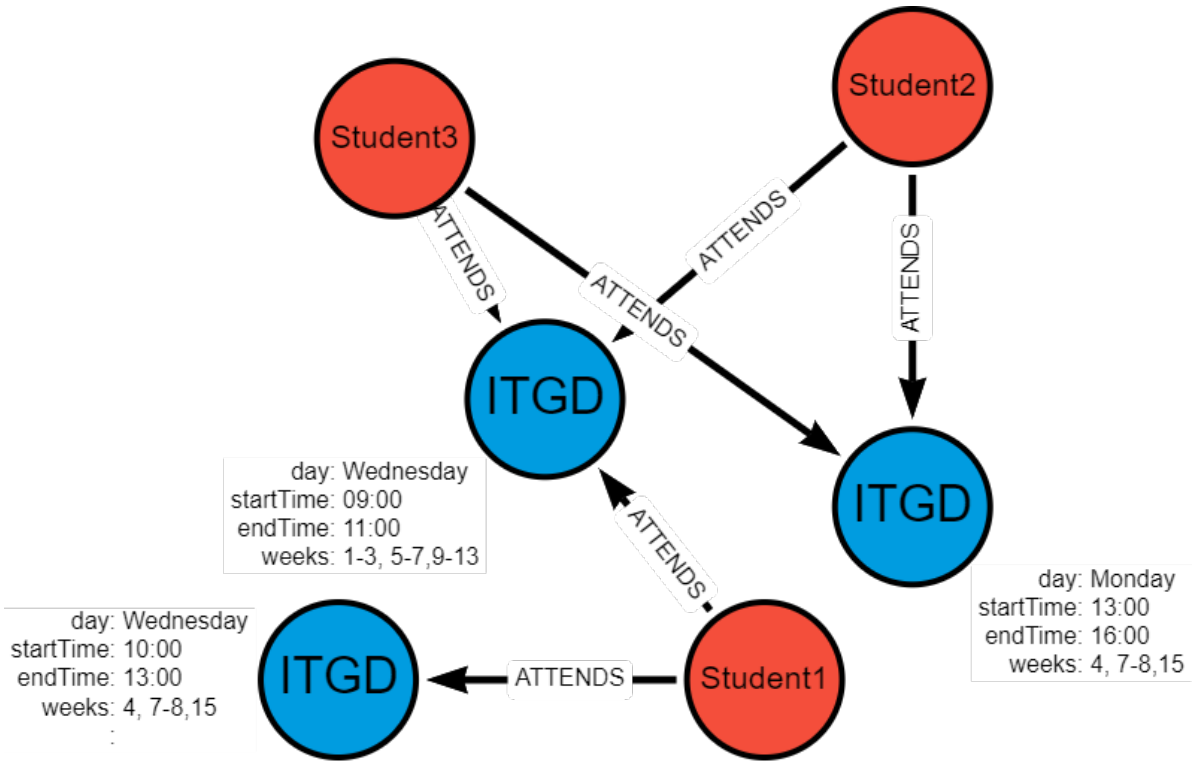


Figure 10.1: Basic example of Graphing Normalised Activities

If we assume that each student attends the lecture and one of the seminars, some students have a clash in week 7 (Wednesday 10:00-11:00) - this is very difficult to identify and isolate in a highly normalised dataset.

10.2.2 Option 1: Unique Activity Nodes

Option 1 addresses this by creating nodes for each unique combination of `name`, `startTime`, `endTime` and `date` - this means de-normalising the relational data and deliberately introducing duplication.

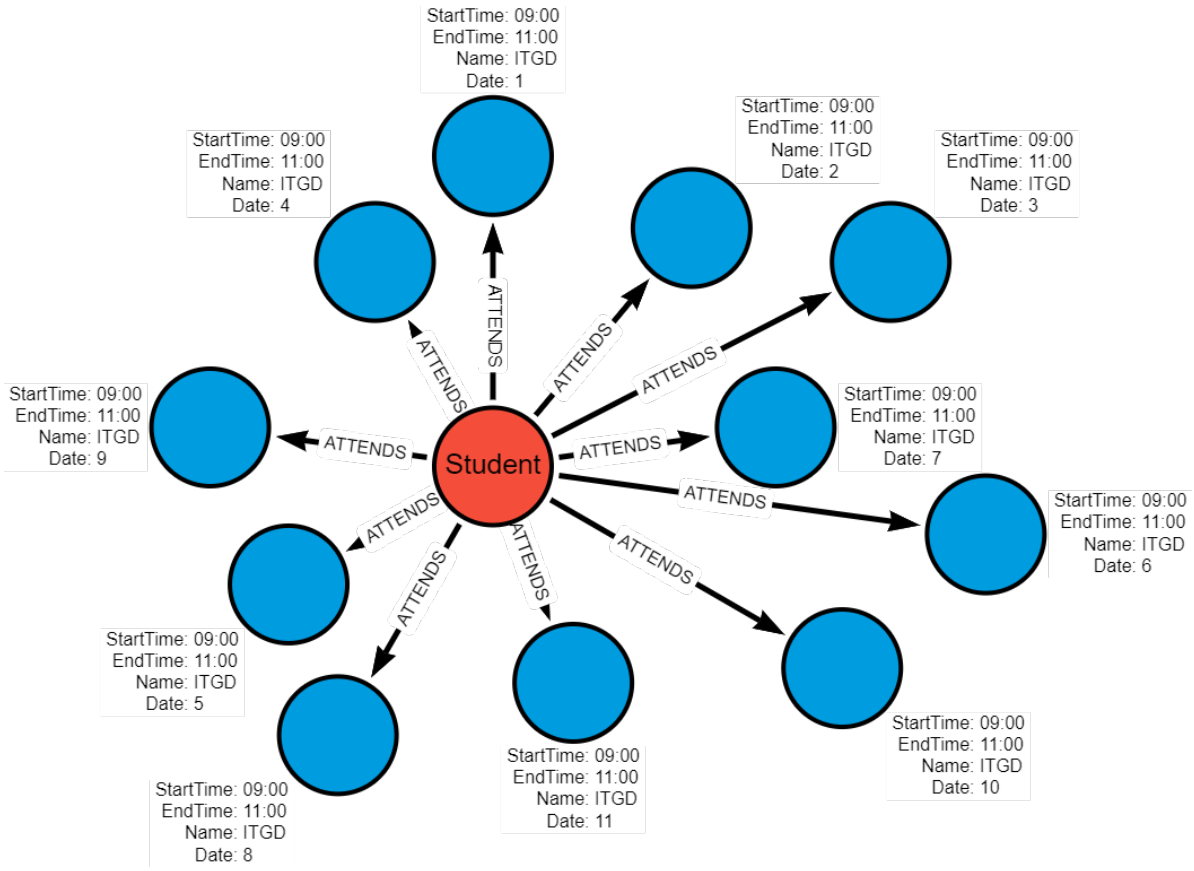


Figure 10.2: Unique Activity Nodes Graph

Graph Structure:

- 11 separate `Activity` nodes one for each occurrence (date)
- Each node has `date`, `startTime`, `endTime` properties
- Only `date` is different between each node.

```
// cypher structure
(Activity {Name: "ITGD", Date: "2024-01-03", StartTime: "09:00", EndTime: "11:00"})
(Activity {Name: "ITGD", Date: "2024-01-10", StartTime: "09:00", EndTime: "11:00"})
...
(Activity {Name: "ITGD", Date: "2024-03-20", StartTime: "09:00", EndTime: "11:00"})
```

Pros:

- *Conceptual Simplicity*: Easy to understand and implement.
- *Direct Time Representation*: Time is directly associated with each activity instance.

Cons:

- *Node Proliferation*: Leads to a high volume of nodes, potentially impacting performance with large datasets.

Use Case dependent

- *Time-Based Queries*: Answering questions about time patterns or conflicts requires traversing numerous nodes and relationships. Some queries will benefit - e.g. identifying clashes which may only occur in a specific week, others will become more complex as de-normalised data needs to be re-aggregated.

10.2.3 Option 2: Date and Time Nodes

Option 2 creates a *single activity* node but also additional `date` and `time` nodes, as required, thus not proliferating activities.

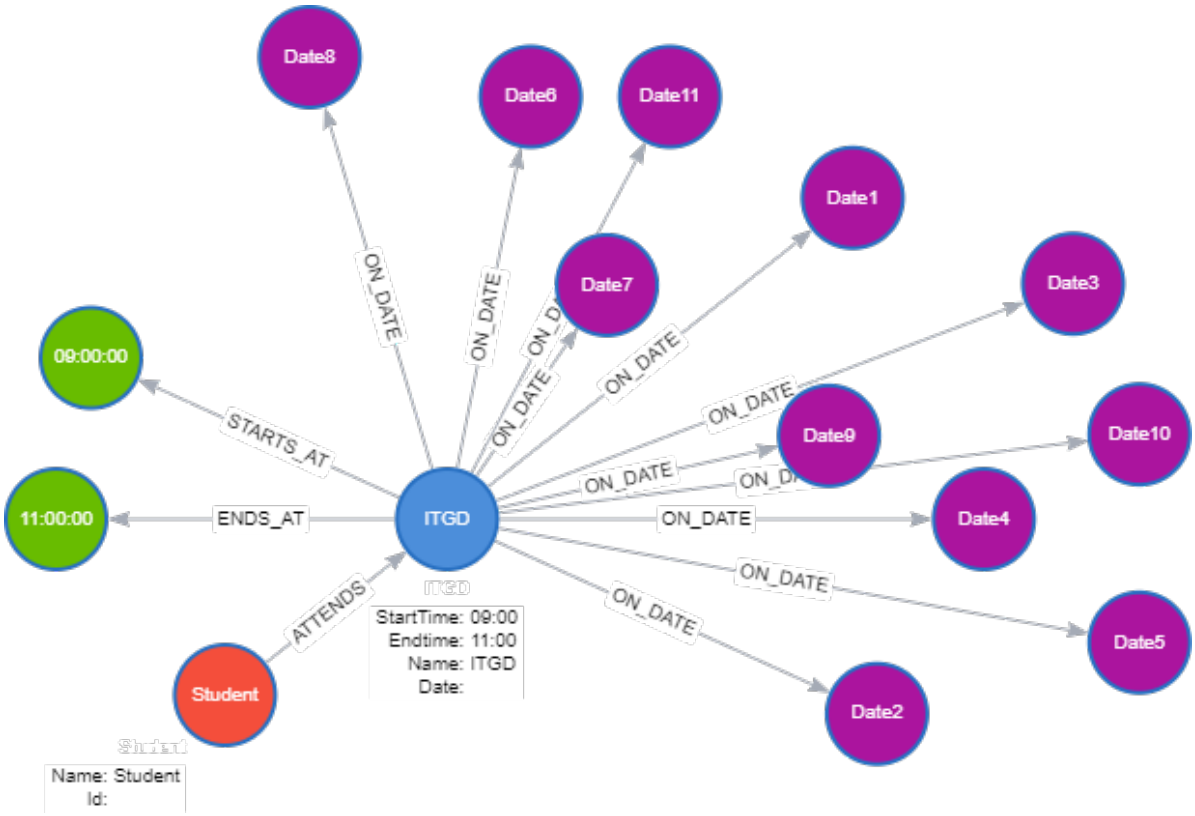


Figure 10.3: Time and Date Nodes

Graph Structure:

- 1 Activity node
- 11 Date nodes - shared by ALL activities on those dates.
- 2 Time nodes (09:00 and 11:00) - shared by ALL activities on those times!
- *Additional Relationships*
  - Activity -[:SCHEDULED_ON]-> Date (11 relationships)
  - Activity -[:STARTS_AT]-> Time (11 relationships to 09:00)
  - Activity -[:ENDS_AT]-> Time (11 relationships to 11:00)

```
// cypher structure
(Activity {Name: "ITGD"})
    -[:SCHEDULED_ON]-> (Date {date: "2024-01-03"})
    -[:SCHEDULED_ON]-> (Date {date: "2024-01-10"})
    ...
    -[:STARTS_AT]-> (Time {time: "09:00"})
    -[:ENDS_AT]-> (Time {time: "11:00"})
```

Key point: Relationships encode which activity happens when.

Pros:

- *Increased Flexibility*: Facilitates queries across time ranges and aggregations across time slots.
- *Reduced Redundancy*: Avoids replicating time information for activities occurring on the same date and time.
- *Lower Node Count*: Potentially fewer nodes overall compared to Option 1 as `date` and `time` nodes are shared with all activities in the database.

Cons:

- *Increased Model Complexity*: Requires managing relationships between Activity, Date, and Time nodes.
- *Potential Performance Overhead*: Querying might involve traversing multiple relationships, impacting efficiency.

10.2.4 Option 3: Date and Time Block Nodes

Option 3 creates a single activity but instead of individual start and end time nodes, we use predetermined `timeBlocks` encompassing both. For example, if using 30-minute blocks, we would have a node for "09:00-09:30" and another for "09:30-10:00", etc.

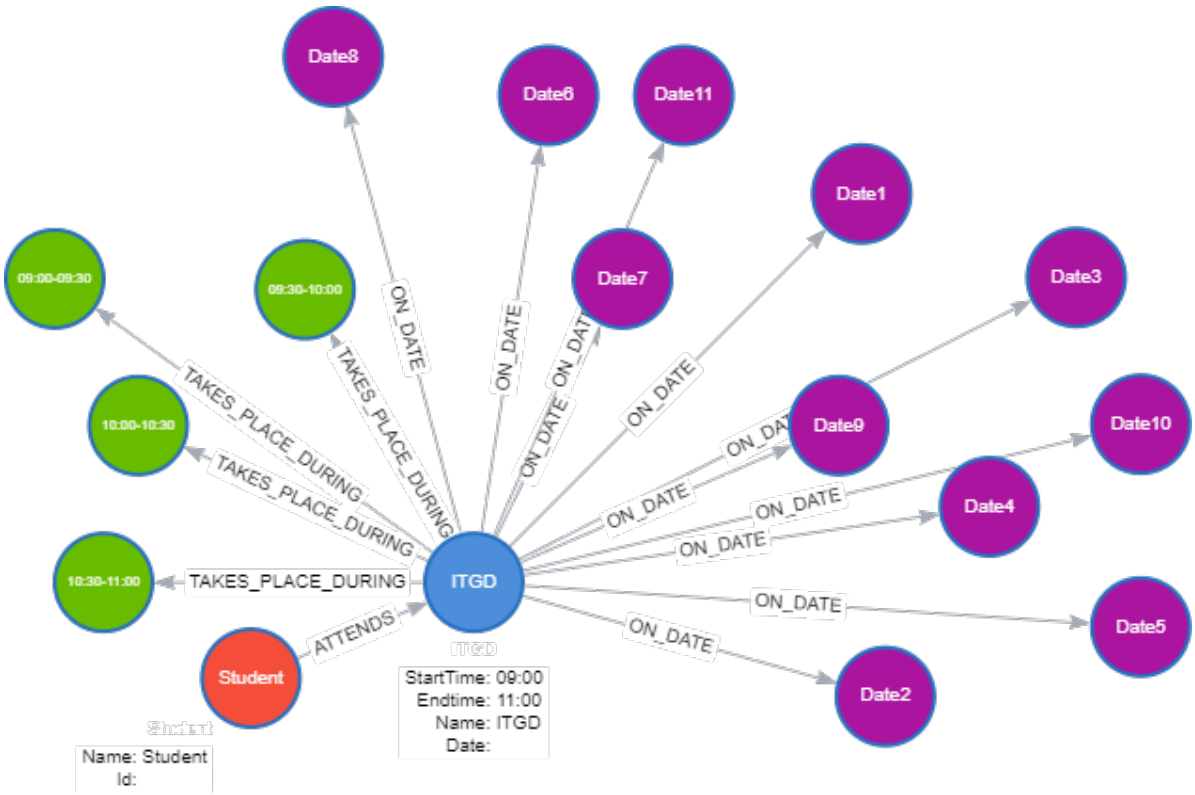


Figure 10.4: TimeBlock and Date Nodes

Graph Structure:

- 1 Activity node
- 11 Date nodes
- 4 Timeblock nodes (09:00-09:30, etc.) - shared by ALL activities on those times!
- *Additional Relationships*
  - `Activity` -[:SCHEDULED_ON]-> `Date` (11 relationships)
  - `Activity` -[:TAKES_PLACE_DURING]-> `timeBlock 09:00-09:30` (11 relationships)
  - `Activity` -[:TAKES_PLACE_DURING]-> `timeBlock 09:30-10:00` (11 relationships)
  - ...

```
// cypher structure
(Activity {Name: "ITGD"})
    -[:SCHEDULED_ON]-> (Date {date: "2024-01-03"})
    -[:SCHEDULED_ON]-> (Date {date: "2024-01-10"})
```

```
    ...
    -[:TAKES_PLACE_DURING]-> (TimeBlock {timeBlock: "09:00-09:30"})
    -[:TAKES_PLACE_DURING]-> (TimeBlock {timeBlock: "09:30-10:00"})
    -[:TAKES_PLACE_DURING]-> (Timeblock {timeBlock: "10:00-10:30"})
    -[:TAKES_PLACE_DURING]-> (Timeblock {timeBlock: "10:30-11:00"})
```

Pros:

- *Granular Time Representation*: Enables analysis at specific time intervals
- *Easier Time Calculations*: Duration is encoded and allows for easy calculations.

Cons:

- *Potential for Data Sparsity*: Some time blocks might be sparsely populated, leading to storage inefficiencies.
- *Potential for High Node Codes*: Lots of `TimeBlocks` if using small intervals
- *Less flexible*: Timeblocks are not dynamic.

10.2.5 Variations

StartTime and Duration: This option simplifies the model by representing time using only `StartTime` and `DurationInMinutes` properties on the `Activity` node, omitting explicit `EndTime` nodes. This approach is suitable for duration-based queries but it is limiting in that it is more difficult to query events occurring at specific times, overlapping time ranges or on end-times.

```
// cypher structure
(Timeblock {name: "09:00-11:00", start: 09:00, end: 11:00, duration:120})
(Timeblock {name: "10:30-11:30", start: 10:30, end: 11:30, duration:60})
(Timeblock {name: "11:00-12:00", start: 11:00, end: 12:00, duration:60})
```

Time Chains: This option retains date and time nodes, but instead of having relationships from activity, the nodes are chained: `activity` -> `startTime` -> `endTime`.

Time as Relationship Property: This option stores time information as properties on the relationship between `Activity` and `Date` nodes. This approach is more compact but can be less intuitive and may limit the ability to query based on time.

Dynamic TimeBlocks: This variation does not pre-create timeblocks based on a set interval (e.g. 30 minutes). They are created dynamically as required by the data and what already exists. For example, activities at 09:00-11:00, 10:30-11:30 and 11:00-12:00 would require these TimeBlocks:

10.2.6 Summary

Table 10.2: Model comparison

#	Option	Pros	Cons
0	**Direct transfer (Normalised)**	Simple	Minimal benefits (for time calculations)
1	**Unique Occurrences**	Simple, direct	High node count, complex time pattern queries
2	**Date & Time Nodes**	Lower activity node count, good for time-based queries	More complex relationships
3	**Date & TimeBlock Nodes**	Granular, easier duration calculations	Potentially high node count, sparsity if blocks are fine-grained

Given the proof-of-concept scope of this project, I chose to proceed with Option 1. While this approach can lead to node proliferation, it offers the most straightforward implementation for exploring fundamental time-based queries and insights. It also acts as an easy jumping off point for exploring any of the other options.[2]

[2]Some of the above variations are explored in Appendix-Student Clashes

11 Data Engineering Overview

A main objective of my project is the development of a data pipeline which efficiently and securely transfers selected university timetabling data from a relational database (MS SQL) to a graph database (Neo4j).

This section provides an overview of the pipeline architecture, fundamental design principles, implementation approach and key learning takeaways.

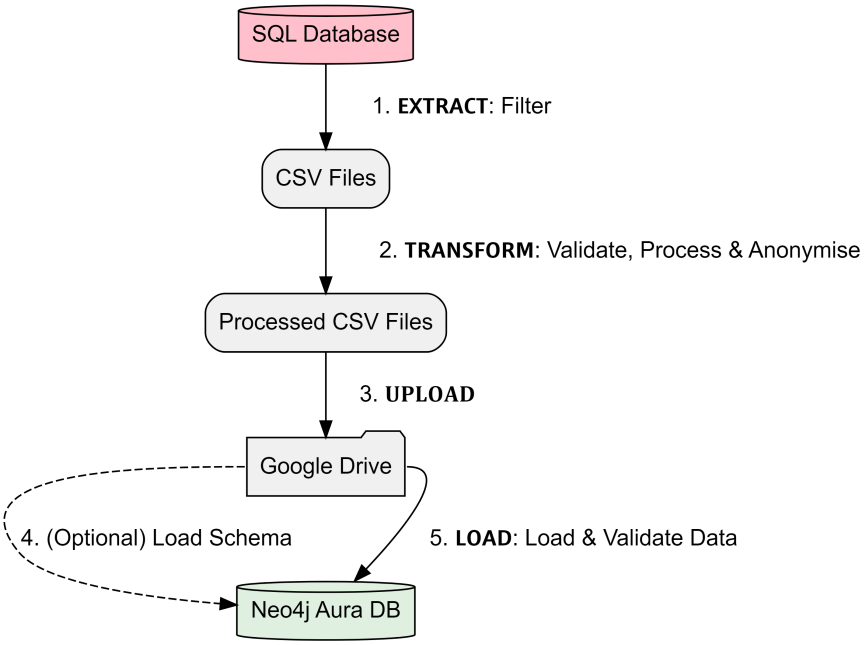
11.1 High-level Architecture

The data pipeline consists of these core stages:

1. **Extraction**: Data is extracted from the SQL database and saved into CSV files.
2. **Transformation**: CSV files are processed, cleaned, transformed, merged, and anonymised using Python.
3. **Intermediate Storage**: Processed CSVs are uploaded to Google Drive (required for Neo4j Aura free instance).
4. **Loading**: Clean data is processed and loaded into Neo4j.

11.2 Design Principles

Several "best practices" in data handling, processing, and database management were incorporated in developing this ETL. The data pipeline is built on several core design principles:



Data Pipeline Overview

Figure 11.1

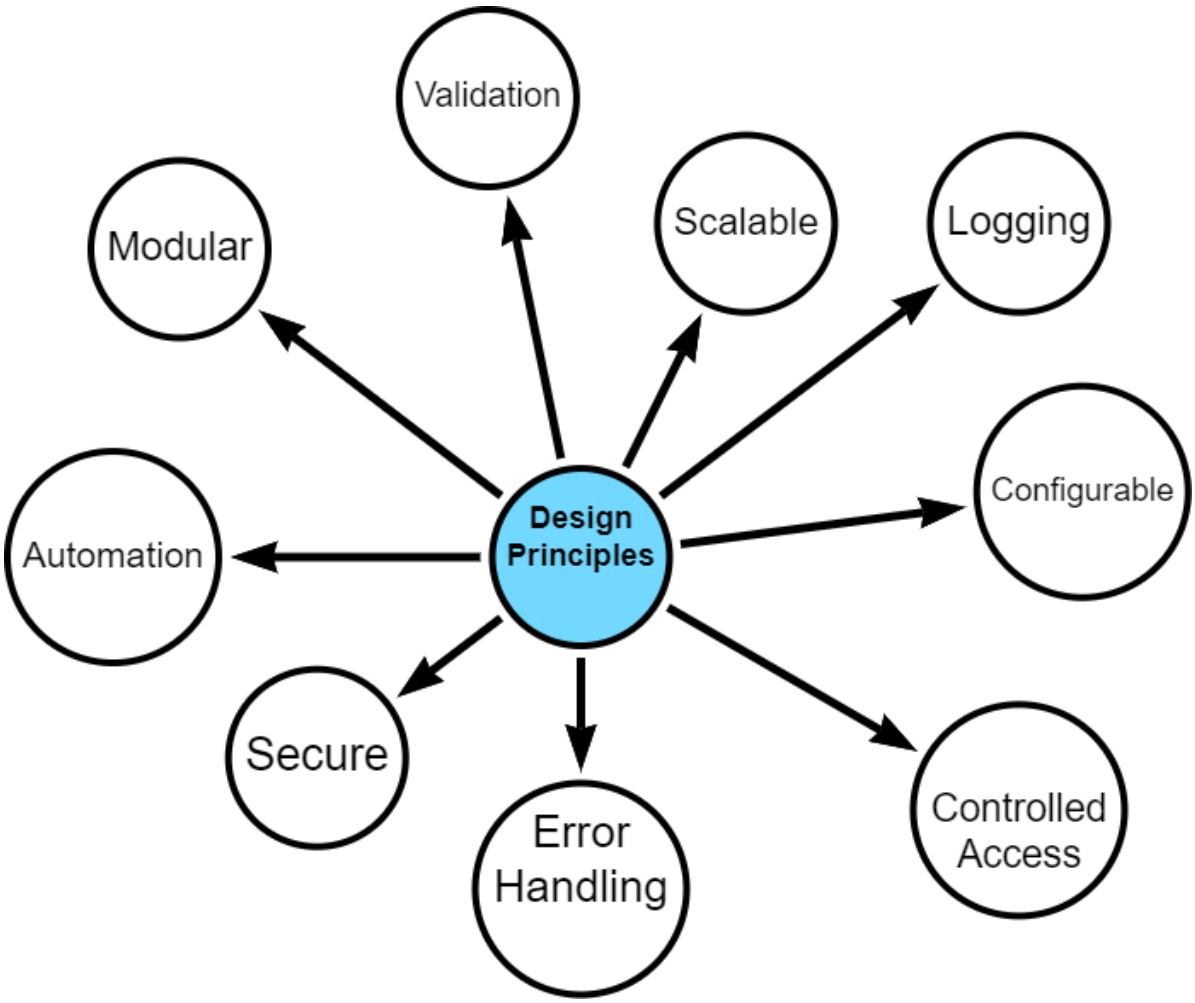
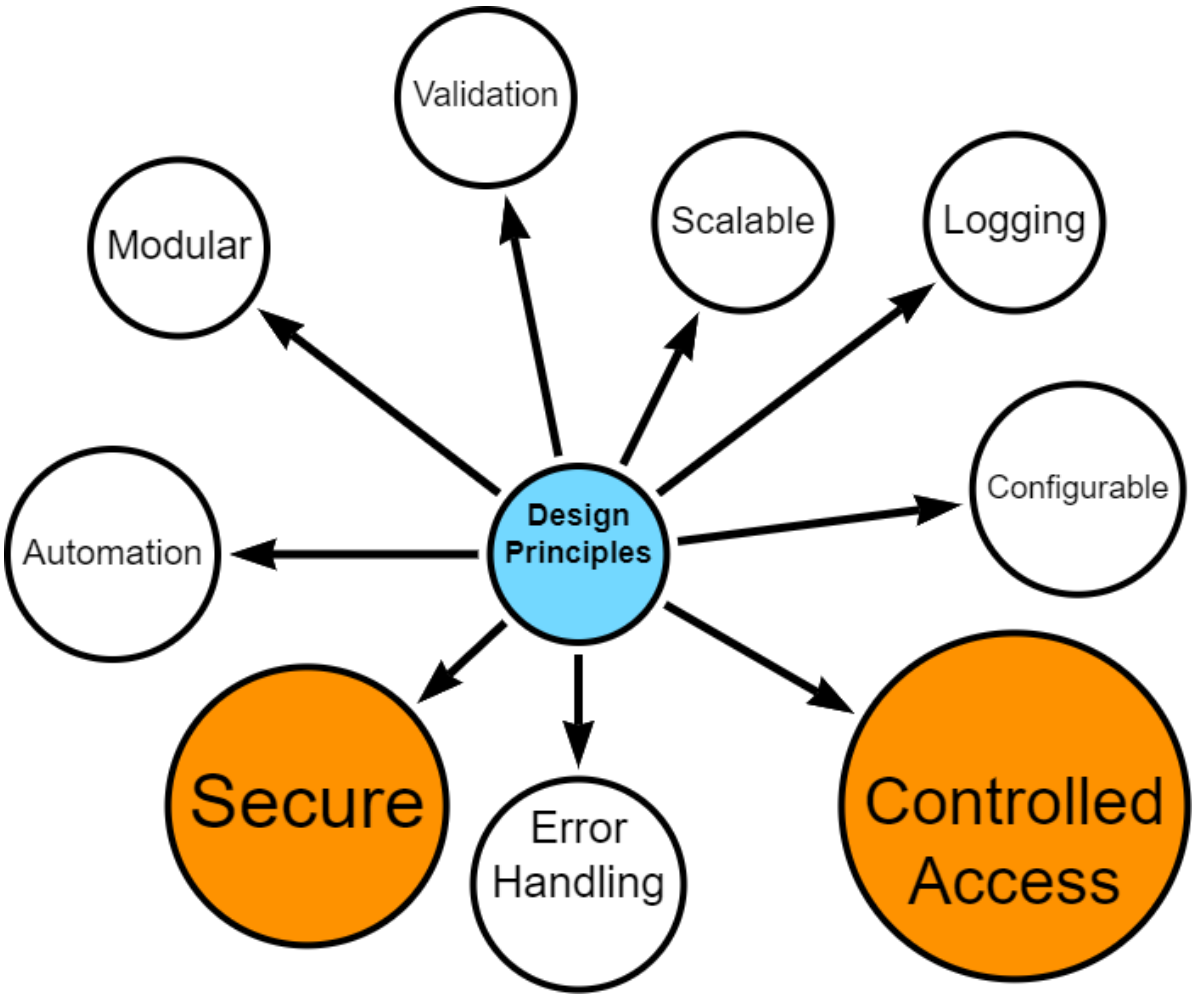


Figure 11.2: Design Principles

I started with a strong sense of what I wanted to build - a modular, scalable, secure and configurable design - however, what *exactly* this meant was only discovered during the development process.

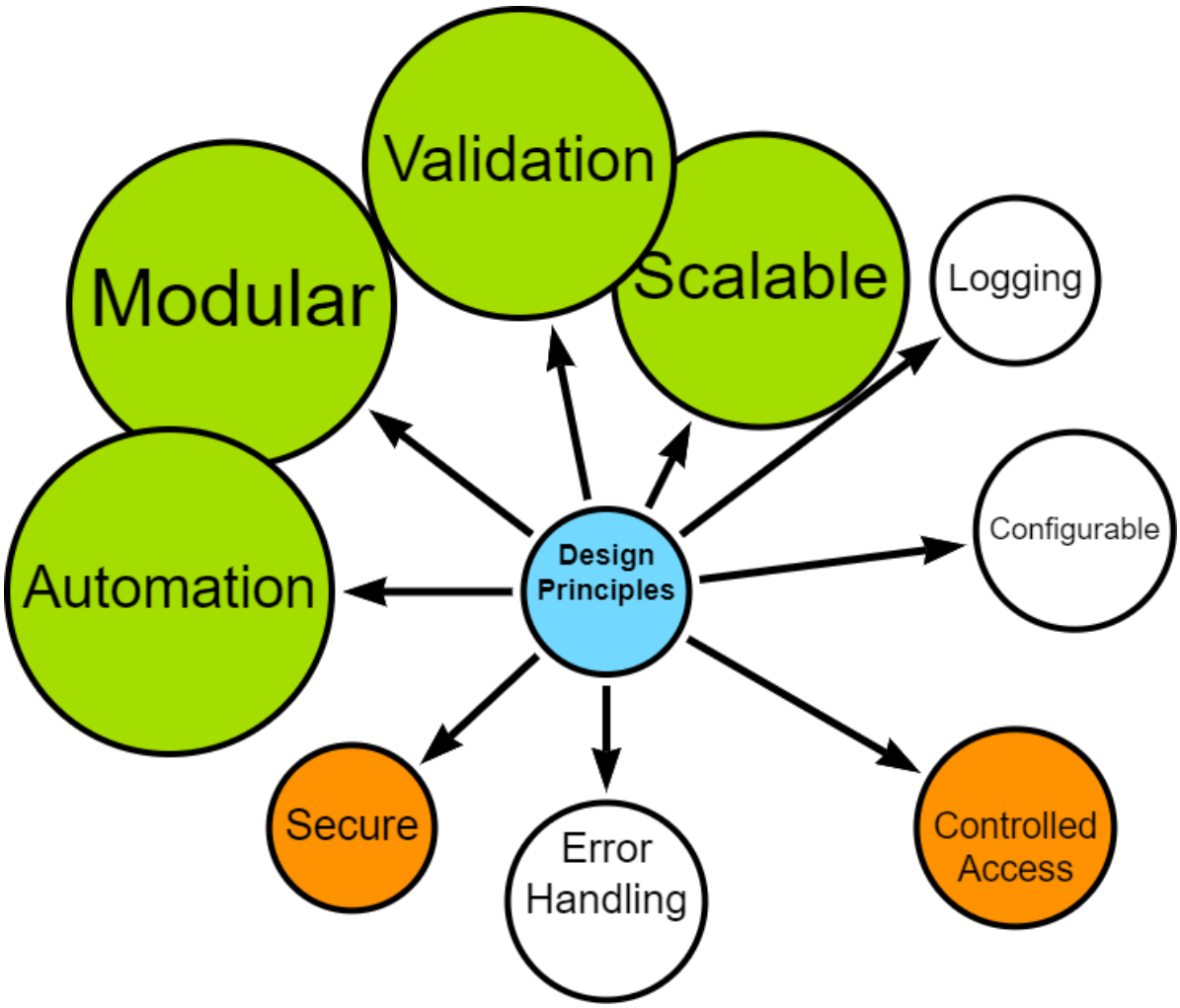
Given project constraints - deadline, word-limits, resources, data, technology - it is fair to say that compromises were made. That said, it was important that the final artefact is one that can be developed further for specific business use-cases.

11.2.1 Security and Data Protection



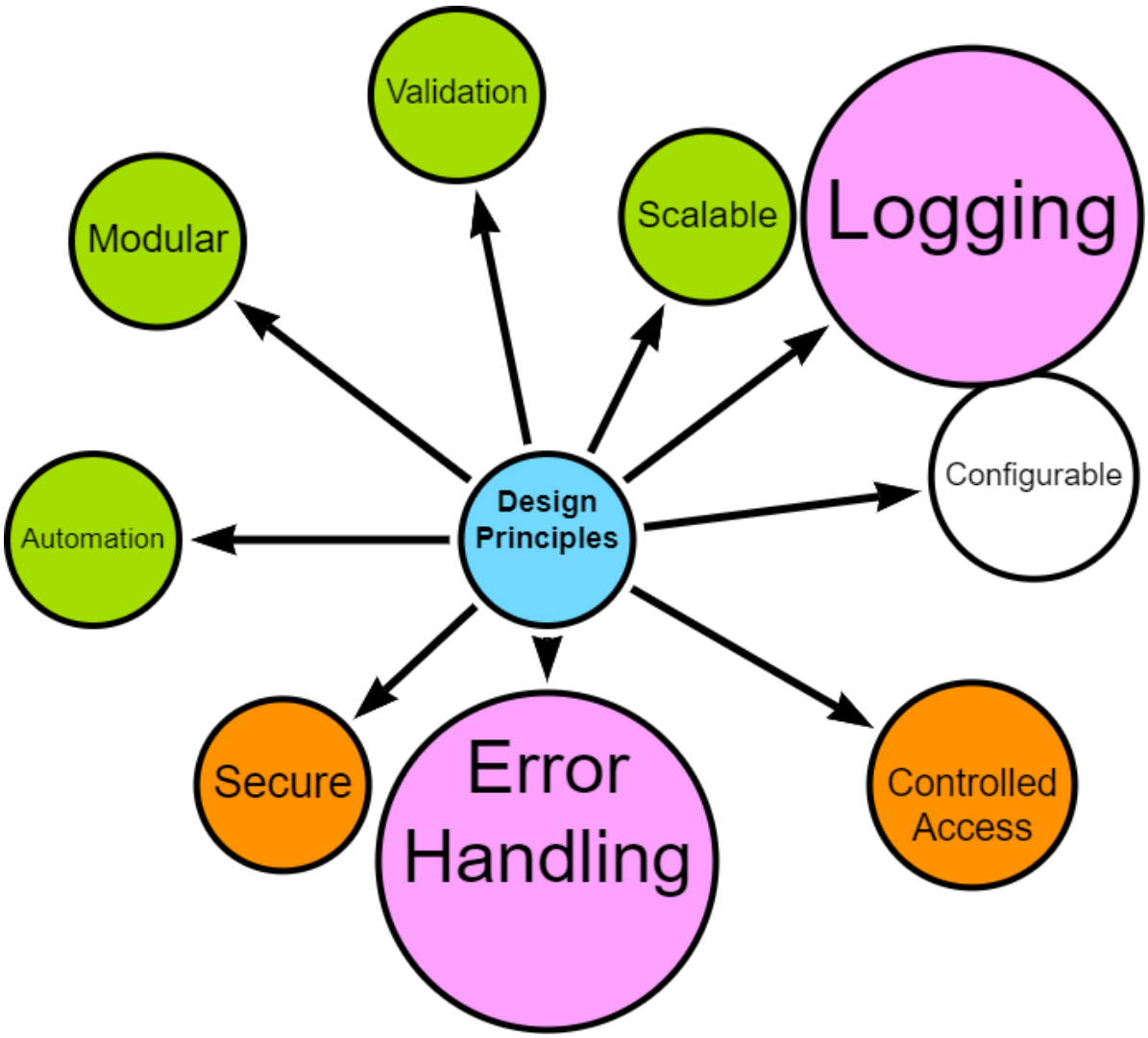
- Secure access controls
- Data anonymisation
- Controlled handling of personally identifiable information

11.2.2 Modularity, Scalability and Automation



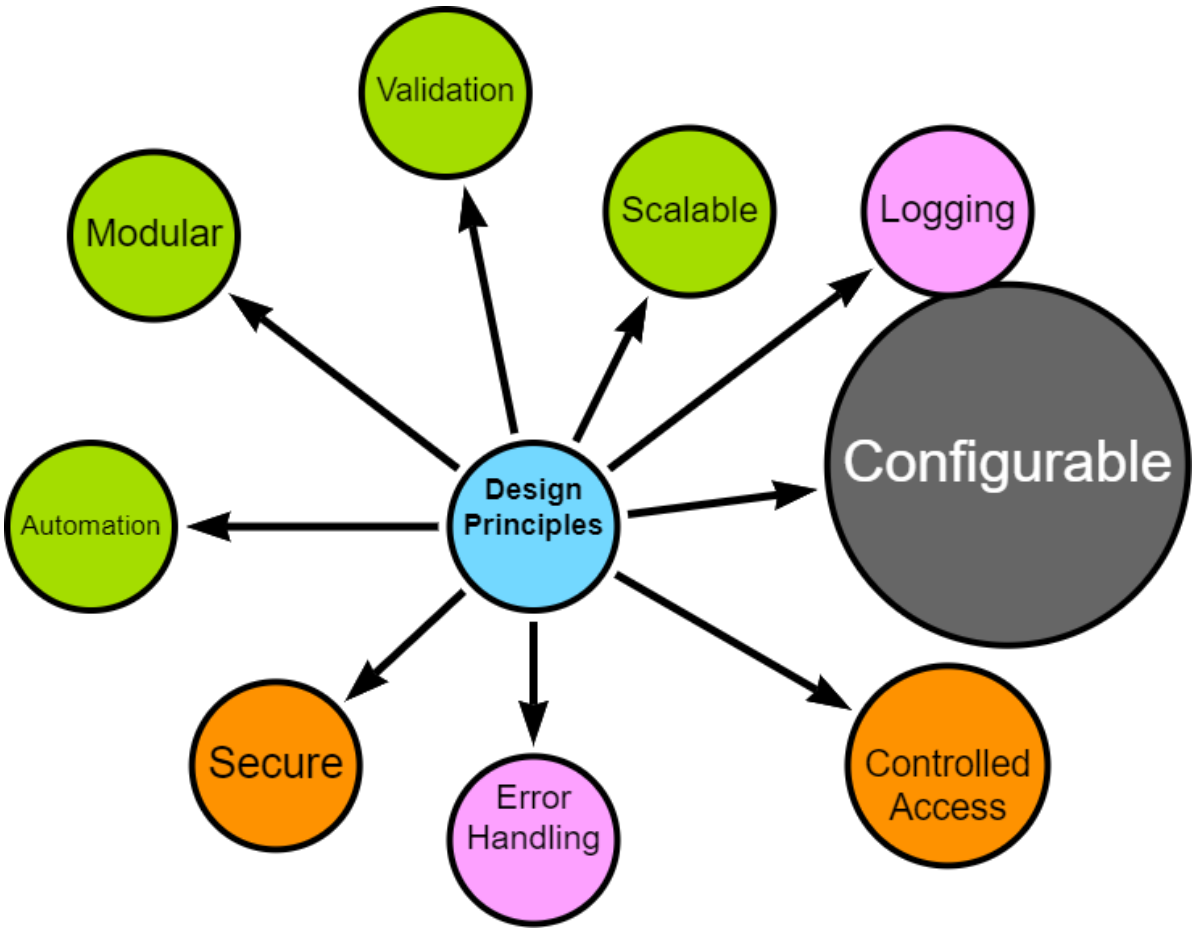
- Distinct, interoperable modules (extract, process, upload, load)
- Ability to handle increased data volume and complexity
- Automation, where possible
- Configurable data processing options (e.g., data chunking, row processing)
- Optimised, where possible

11.2.3 Error Handling and Logging



- Robust error handling
- Comprehensive logging for troubleshooting and auditing

11.2.4 User configurable



- Flexible configuration options for data filtering, directory controls, and schema handling

11.3 Implementation Approach

The pipeline was developed using an iterative approach, allowing for continuous discovery, refinement and improvement.

Crucial aspects of the implementation include:

- **Technology Stack**: Python for data processing, MS SQL for source data, Neo4j for the target graph database. See Technology Stack for more details.
- **Cloud Integration**: Utilisation of Google Drive for intermediate storage, compatible with Neo4j Aura.

- **Validation**: Implemented at various stages to ensure data integrity and fitness for processing.
- **Testing**: Continuous simulated unit testing to ensure that components are behaving as expected.

11.4 Upcoming Sections

The following sections will delve into specific implementation details of each stage, demonstrating how these principles are put into practice, before reflecting on lessons learned and potential future enhancements.

12 Data Engineering Approach

I followed an iterative, agile-inspired approach despite being a team of one. This method allowed for flexibility, continuous improvement and the opportunity to adapt to new insights during the process (Beck, K., et al. 2001).

The bulk of my effort was spent *prototyping*, *testing* and *reviewing* with each iteration resulting in a new challenge, issue, or opportunity.

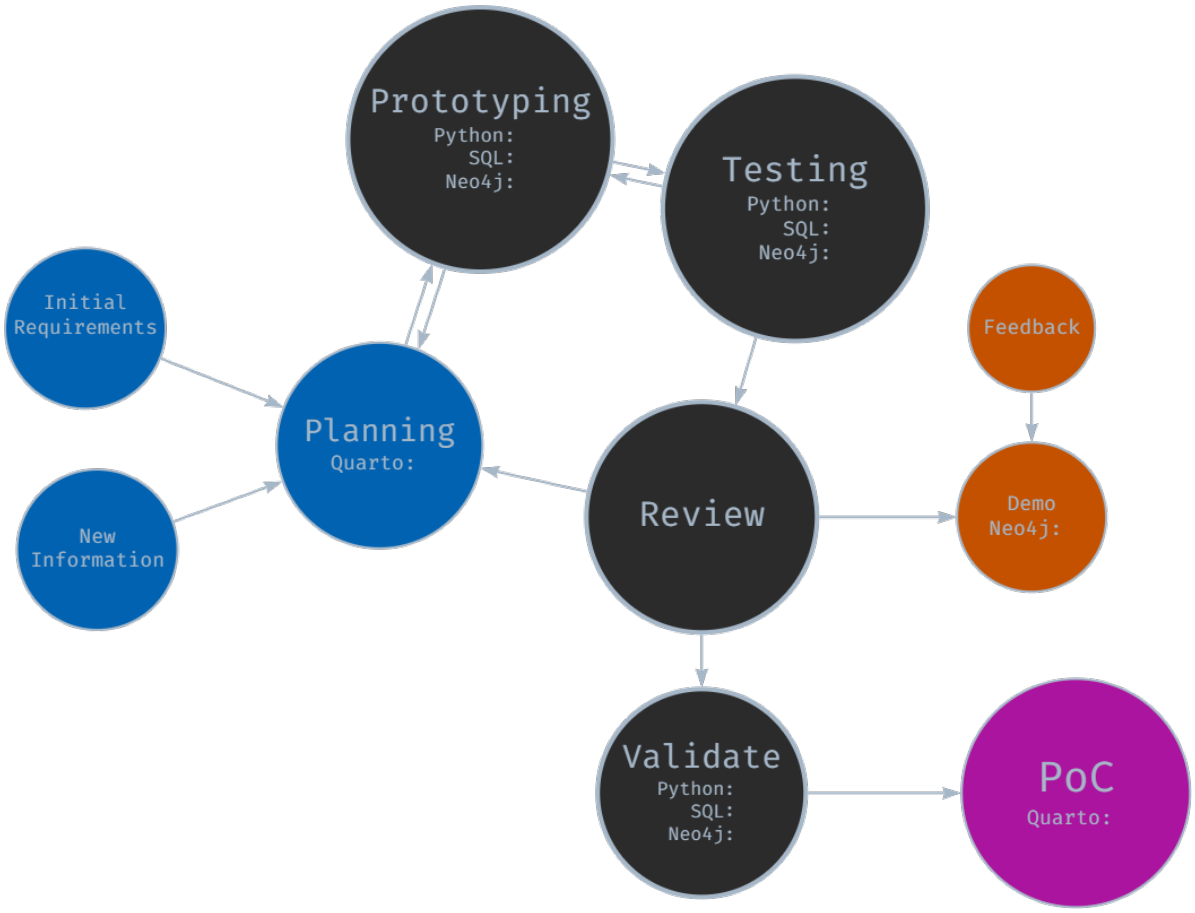


Figure 12.1: Iterative Development Approach

12.0.1 Initial Planning and Requirements Gathering

The development cycle began with initial high-level planning and requirements gathering, where I imagined how each stage *should* work, trying to bear in mind future-proofing and repeatability principles.

I defined core functionality for each module (extraction, transformation, loading) and outlined initial technical requirements and constraints. The planning documentation was maintained in Quarto and markdown files in a centralised repository for project information.

12.0.2 Prototyping

Following initial planning, rapid prototyping was undertaken for each module:

- SQL prototyping for data extraction queries
- Python prototyping for data transformation and processing logic
- Neo4j prototyping for graph database schema and loading procedures

This stage allowed for quick exploration of different approaches and early identification of potential challenges as well as giving me the confidence to continue with my exploration.

12.0.3 Component-Based Development and Testing

- Each module (extraction, transformation, loading) was developed separately with a view to distinct "handovers"
- An iterative, component-based testing approach was employed
- While formal unit tests were not always created, each component was thoroughly tested for functionality

This approach allowed for continuous progress while maintaining a focus on component-level quality. It was during this phase that I started expanding configuration, logging and error-handling options - and I am glad I did!

12.0.4 Integration -> Review -> Demo -> Feedback -> Repeat

As components reached a (more) stable state, they were integrated and reviewed:

- Components were combined to form larger functional units
- Integrated functionality was occasionally demonstrated to subject matter experts (e.g. data manager)
- Feedback was gathered on functionality, usability, and alignment with requirements

Insights gained from reviews, demonstrations and ongoing development were continuously fed back into the process. New requirements or modifications were documented, for example updates to SQL SELECT statements and data model interpretations.

Each change required decisions - but I did not always make the right ones!

12.0.5 Version Validation and Documentation

At pivotal junctures, e.g., when a stable version was achieved:

- End-to-end validation of the entire pipeline was performed.
- Results were documented in notebooks, including opportunities for improvement.
- Bugs and opportunities were logged for future iterations.

12.0.6 Continuous Learning and Adaptation

Learning and adaptation became central to the project. Each iteration brought new insights, often through trial and error and certainly through unintended consequences or unforeseen complications. Early challenges included the need to modularise components before they became unmanageable and resisting the temptation to make overly ambitious changes. With practice, I became better at recognising when refactoring was necessary.

Developing the ETL was not a linear journey. There were many moments of frustration, periods of seemingly endless, painstaking troubleshooting, and a constant battle against the urge to over-deliver. Yet, with each stumble, the process itself became more refined, transforming into a powerful tool for identifying and resolving issues.

While core MVP (minimum viable product) requirements remained relatively stable (*I set them after all!*), iterating allowed me to seize opportunities for enhancement. Each chance to modularise, parameterise, or fine-tune sparked an almost compulsive drive for improvement, pushing the pipeline beyond its initial scope.

Ultimately it all resulted in a robust, flexible solution that can adapt (relatively) gracefully to unforeseen challenges and serve as the starting point for future opportunities.

13 Configuration and Logging

Configuration and logging are essential components of the ETL pipeline. Config allows the user to manage different aspects of the ETL pipeline, while logging provides a record of the pipeline's execution. They emerged from initial design and from discovering during development.

13.0.1 Main Configuration options

- Configuration parameters are centralised in Python scripts.
- The design primarily aims for automatic and dynamic operation with well-structured data, but includes override options.
- A YAML file (Appendix-config) holds configuration options, including general settings for filtering data extraction, dynamic folder/filepath creation, and secure credential storage.
- Config also controls options for validation, data augmentation and graph structures (nodes, relationships) to be created.

13.0.2 Logging

- Each module has its own log file with customisable log level (DEBUG, INFO, WARNING, ERROR, CRITICAL).
- Timing function which tracks and stores various execution and elapsed times with a view to optimising performance or identifying bottlenecks.

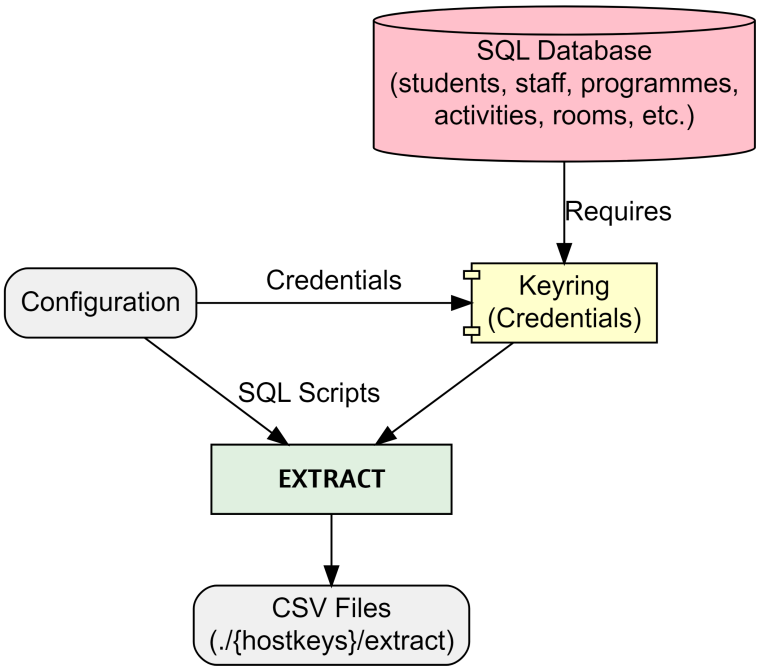
13.0.2.1 Example Extract Log

13.0.2.2 Example Google Drive Log

13.0.2.3 Example Process Log

13.0.2.4 Example Load Log

14 Extraction



Extract

EXTRACT starts by securely connecting to the specified SQL database using encrypted credentials stored with keyring. The combination of `configuration` and `SQL scripts` determine which data will be extracted by filtering based on programme(s) of study and specifying which nodes, relationships and properties to extract. Additional options include specifying `chunk size` if extracting significant amounts of data, for example.

The process performs basic validation at every step ensuring secure connection before running SQL SELECT statements and storing extracted data as local csv files.

14.0.1 SQL example

```
SELECT DISTINCT a.[Id] AS actSplusID,
     CONCAT(a.[Id], '-', adt.[Week], '-', adt.[Day]) AS actGraphID,
     a.[Name] AS actName,
     a.[Description] AS actDescription,
     a.[DepartmentId] AS actDeptSPlusID,
     adt.[StartDateTime] AS actStartDateTime,
     adt.[EndDateTime] AS actEndDateTime,
     adt.[Week] AS actWeekNum,
     adt.[Occurrence] AS actOccurrence,
     a.[ModuleId] AS actModSplusID,
     a.[ScheduledDay] AS actScheduledDay,
     a.[StartDate] AS actFirstActivityDate,
     a.[EndDate] AS actLastActivityDate,
     a.[PlannedSize] AS actPlannedSize,
     a.[RealSize] AS actRealSize,
     a.[Duration] AS actDuration,
     a.[DurationInMinutes] AS actDurationInMinutes,
     a.[NumberOfOccurrences] AS actNumberOfOccurrences,
     a.[WeekPattern] AS actWeekPattern,
     a.[ActivityTypeId] AS actActivityTypeSplusID,
     a.[WhenScheduled] AS actWhenScheduled,
     a.[IsJtaParent],
     a.[IsJtaChild],
     a.[IsVariantParent],
     a.[IsVariantChild]
FROM ##TempActivity a
INNER JOIN ##TempActivityDateTime adt ON a.[Id] = adt.[ActivityID];
```

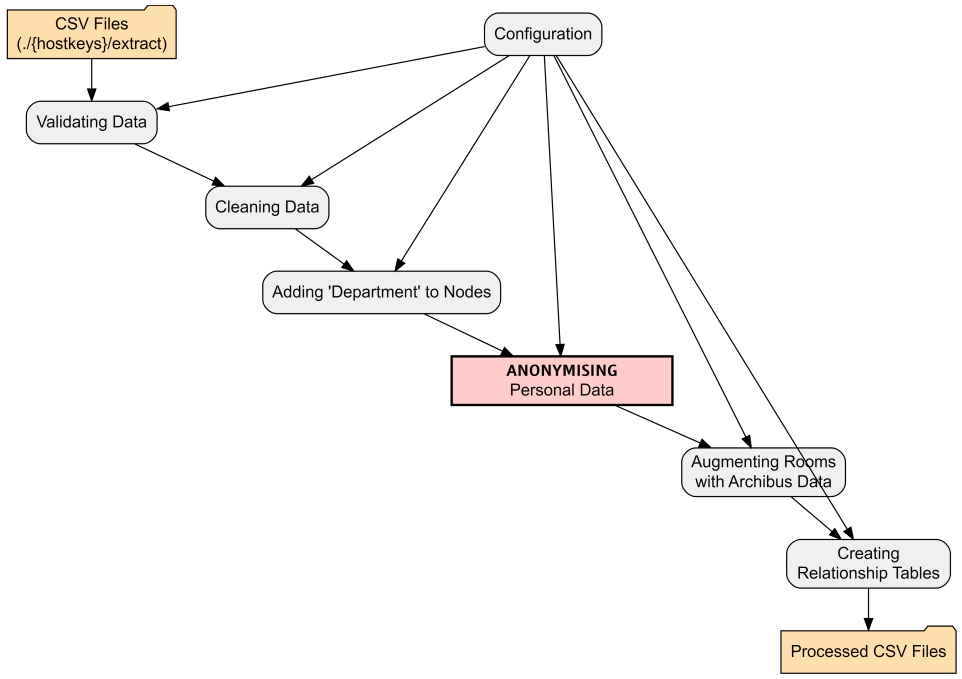
14.0.2 Snippet: extract_data.py

```
# extract_main.py
from logger_config import extract_logger
from extract_data import main as extract_main
from config import EXTRACT_DIR, HOSTKEYS, CHUNK_SIZE
from utils import execution_times

def run_extraction():
    extract_logger.info("Starting data extraction process")
    extract_logger.info(f"Output Directory: {EXTRACT_DIR}")
    extract_logger.info(f"Hostkeys: {HOSTKEYS}")
```

```
    extract_logger.info(f"Chunksize: {CHUNK_SIZE}")

    try:
        extract_main()
    except Exception as e:
        extract_logger.exception("An error occurred during data extraction:")
    finally:
        extract_logger.info("Data extraction completed.")


    # Log the execution times
    extract_logger.info("Extraction Time Summary:")
    for func_name, exec_time in execution_times.items():
        extract_logger.info(f"Function {func_name} took {exec_time:.2f} seconds")


if __name__ == "__main__":
    run_extraction()
```

15 Transformation

TRANSFORM picks up where EXTRACT finished by using the extracted csv files as the source.



Configuration allows the user to control which nodes and relationships are included and how they are processed. There are options to specify validation, cleaning, data linking, anonymisation and relationship details.

It is also possible to specify datatypes. Neo4j assumes `string` datatype unless it is well-formatted or pre-determined. Config allows the user to specify specific datatypes like dates, times, point, Boolean, etc.

15.1 All data

1. **Validation** - basic validation of the data is performed. Validation is extensible and can be expanded, as requirements are identified.

2. **Cleaned** - basic cleaning of all data is performed by stripping empty space and removing non-printable characters, etc. using regex. The cleaning functionality is expandable.

With clean data, the transformation proper starts:

15.2 Nodes and relationships

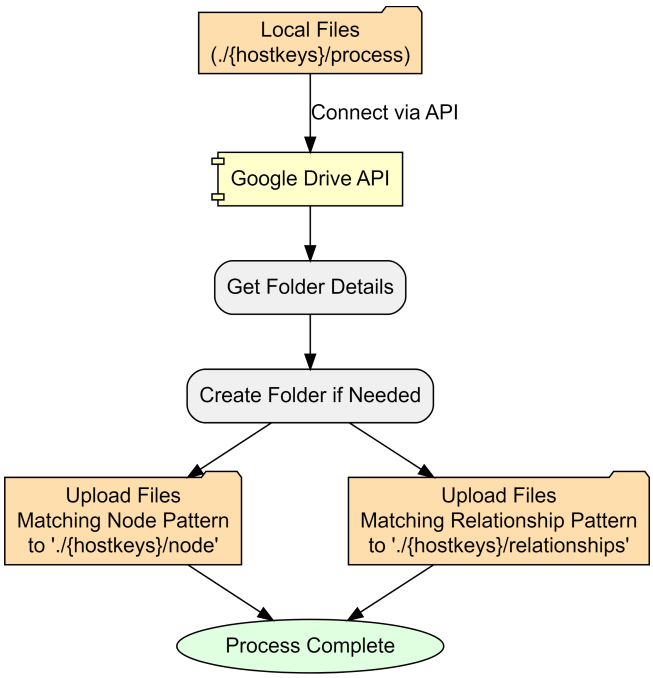
1. **Add Organisational Unit** - where appropriate, the University Organisational Unit (e.g. College, School, Department) is added to the node. This will be picked up as a property during load.

2. **Data Augmentation** - Room data is augmented with additional properties from the location master database, including latitude, longitude, square meterage, etc. Data augmentation is extensible.
3. **Anonymisation** - Personal data is anonymised. An anonymisation function was developed to remove and replace any personally identifiable information (PII). The pipeline extracts minimal PII but this is safely anonymised. The functional also adds fake emails. See Appendix for additional details
4. **Relationships** - Based on requirements in the configuration, relationships are extracted including optional relationship properties.

16 Google Load

The free cloud instance of Neo4j (Aura) requires that csv files are stored in public cloud storage like Google Drive or Dropbox.

Therefore, my project requires an intermediary step.



File storage and directories are controlled via Config. I created a publicly shared folder in Google drive which contains all project csvs:

```
- root: Google Drive folder
  - hostkeys (automatically created, unless override)
    - nodes
    - relationships
```

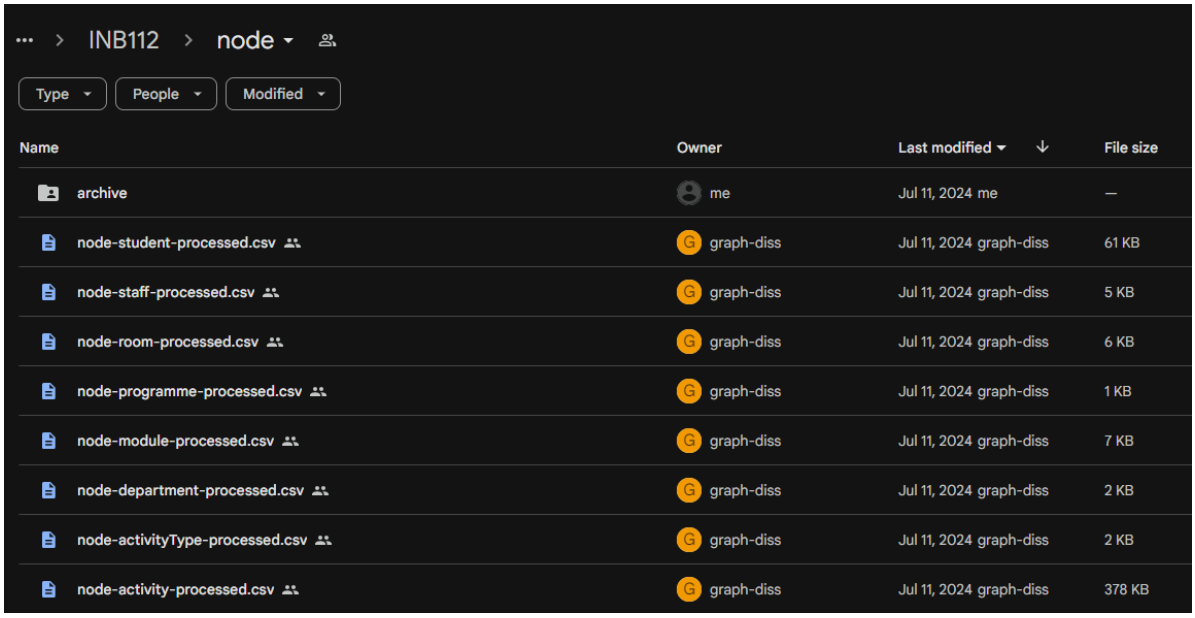
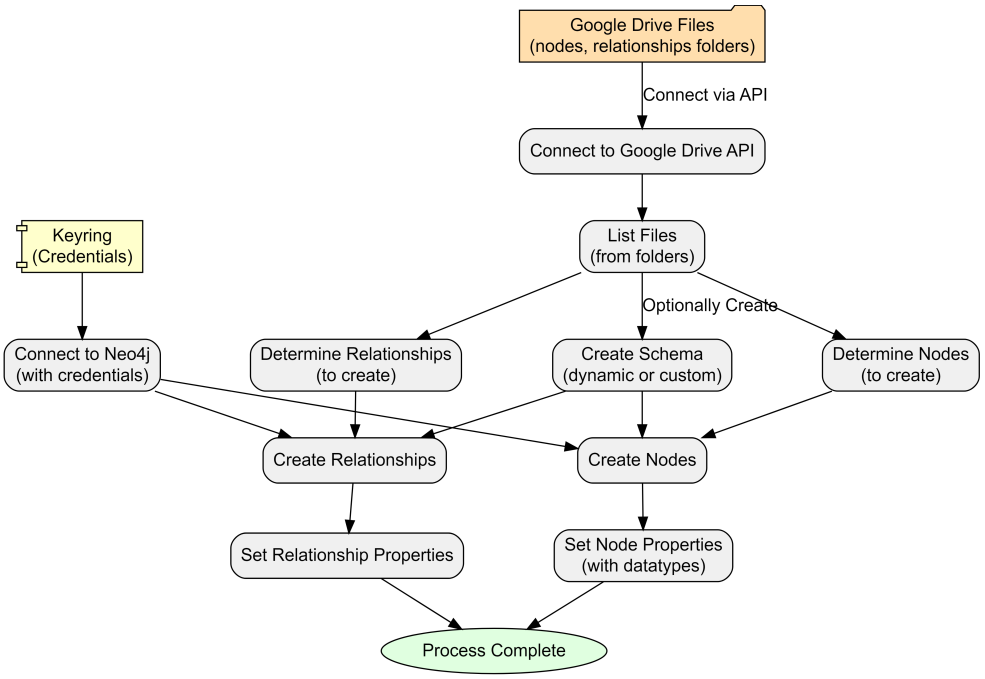


Figure 16.1: Screenshot of Google Drive folder and files for MSc Data Science (INB112): note that files were created by Google API

17 Neo4j Load

With accessible csv files, the final module of the ETL pipeline creates (or updates) nodes and relationships in the Neo4j instance.



There are two authentication requirements:

1. **Google Drive** to get node and relationship files and data.

2. **Neo4j Aura instance** is connected to with `Keyring` encrypted credentials.

Nodes and relationships are dynamically processed by using a file-pattern matching approach. However, this can be overridden within configuration, if needed.

Also in configuration is the option to create a database schema. There are three options:

1. **No schema**
2. **Dynamic** (default) - creates unique constraints based on nodes
3. **Custom** - allows the user to specify specific constraints prior to loading.

At this point, the ETL loads data on a row-by-row basis, reading the public csv files. Columns become properties with data types cross-referenced from a data-mapping dictionary in the configuration.

If there have been no errors - we should have data in our Neo4j Aura instance!

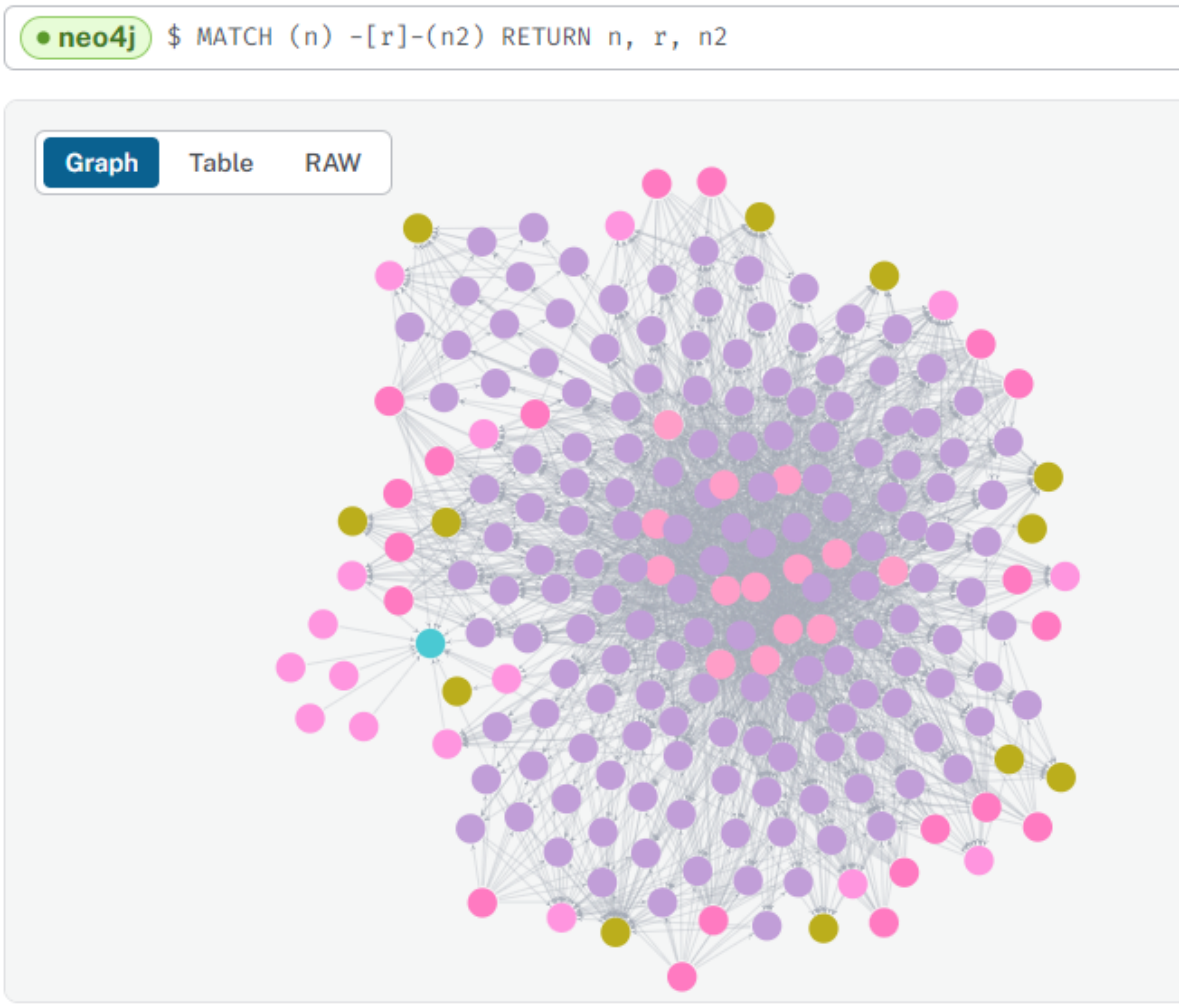


Figure 17.1: Data successfully loaded for “MSc Artificial Intelligence (I400)”

18 Reflections

I was always wary that the data engineering portion of my project might be too ambitious in both scale and scope. However, the reality of its magnitude became increasingly apparent.

Yet, despite my initial awareness, I found myself continually expanding the project's boundaries, often pushing for a "gold-plated" solution rather than acknowledging when certain aspects were "good enough." [1] This tendency towards scope creep, while driven by a desire for excellence, has significantly increased the project's complexity and time requirements.

The learning curve has been exceptionally steep. I've had to rapidly acquire proficiency in a diverse range of technologies and tools: Python, Neo4j, Google APIs, Quarto, and GraphViz. This intensive learning process, while challenging, has also been incredibly rewarding. My technical toolkit has expanded far beyond my initial expectations - but this also contributed to the continuously expanding scope, as each new skill opened possibilities for further enhancement and the necessity for on-the-fly troubleshooting.

Unexpected challenges have been a constant companion. From deleted servers and databases to access issues to discrepancies between development environments (such as missing certificates), I've encountered a wide array of unforeseen obstacles. These issues have necessitated the development of strong troubleshooting skills and a flexible approach to problem-solving.

While often frustrating, these challenges have also provided valuable learning opportunities, pushing me to deepen my understanding of the systems and technologies I'm working with.

18.1 Lessons Learned

1. **Scope management is crucial**: Work on recognising when a solution is "good enough" and resist the urge to continually expand scope. Set clear boundaries at the start and be prepared to reassess and adjust plans when necessary.
2. **Embrace modularisation from the beginning**: Avoid the temptation to create oversized code blocks. Maintain a list of "future enhancements" to prevent immediate implementation of every idea.

[1] As my manager often needs to remind me…

3. **Balance documentation with development**: Document sufficiently during the development process but save comprehensive documentation for appropriate milestones. This approach maintains progress while ensuring proper record-keeping.
4. **View obstacles as learning opportunities**: Embrace continuous learning and see challenges as chances to grow. Invest time in understanding the right technologies and approaches, particularly focusing on modularisation.
5. **Celebrate incremental progress**: Recognise and appreciate small achievements throughout the development process. This helps maintain motivation and provides a clearer sense of overall progress.

The next section will start looking at the newly transferred data in the graph database.

19 Timetable Metrics

So far, we have explored the complexities and challenges of university timetabling, the pros and cons of graph databases and investigated different graph data models. We have also built and implemented a data engineering solution which extracts, transforms and loads data from SQL RDMS to a graph database.

In this section, we delve deeper into the concept of timetable quality metrics and explore how graph databases can help us quantify and measure the quality of timetables.

19.0.1 Defining Timetable Quality

As discussed in the introduction, the inherent complexity of timetabling, with its competing objectives and subjective evaluations, makes it difficult to objectively assess the quality of a timetable. There is no universally agreed-upon definition of a "good" timetable, as it is often a balancing act between satisfying hard constraints (e.g., avoiding clashes) and optimising for softer constraints (e.g., minimising travel time).

How can we move beyond anecdotal evidence and subjective opinions to a more data-driven understanding of timetable quality?

I propose the development of a *timetable quality index*.

19.0.2 Towards a Quantifiable Measure

A **Timetable Quality Index** (TQI)is a quantifiable and measurable *score* reflecting the overall "goodness" of a timetable, both at individual and aggregate levels. This score would be based on a flexible and adaptable system of penalties and rewards tied to specific metrics, allowing institutions to tailor the index to their unique needs and priorities.

Use cases for TQI include:

19.0.2.1 Benchmarking and comparison

The index allows institutions to compare timetable quality across different programmes, departments, or even years, facilitating the identification of best practices and areas for improvement. It would aid in benchmarking timetable models and approaches.

19.0.2.2 Resource optimisation

Insights from the index can help institutions allocate resources, such as lecture rooms and teaching staff, more effectively by identifying underutilised or overbooked facilities.

19.0.2.3 Student experience enhancement

By prioritising metrics related to student well-being, such as travel time and consecutive teaching hours, institutions can enhance the overall student experience and satisfaction.

19.0.2.4 Data-driven decision making

Historical timetable quality data can inform future planning and course scheduling, allowing institutions to anticipate and address potential issues proactively.

19.0.2.5 Stakeholder communication

The TQI can serve as a transparent and data-driven tool for communicating the performance and challenges of the timetabling process to various stakeholders, including faculty, students, and administrative staff. It takes the guesswork out of the discussion.

19.0.3 Implemented Metrics

The foundation of this quality index lies in defining and calculating specific metrics that capture various aspects of timetable quality.

19.0.3.1 Constraint or preference violations

Individual timetables have certain measurable qualities - the shape and feel of the timetable. Often, these can be summarised into rules or constraints - either desirable qualities to strive for or undesirable qualities to avoid. They can be 'hard' - **must** not be violated, or 'soft' - *should* not be violated.

The presence or absence of these qualities on an individual's timetable can be measured in the form of a reward or penalty. The flexibility of the graph database allows for experimentation in terms of which metrics are most relevant, how they should be calculated, where they should be stored, and how they should be weighted in the overall quality score.

Examples include:

- maximum hours per day, e.g., no more than 6 hours of teaching per day

- maximum consecutive hours, e.g., no more than 3 hours of teaching without a break
- minimum hours per day, e.g., at least 2 hours of teaching per day
- lunch break, e.g., should have a break between 12-2pm
- minimal idle time, e.g., no more than a 4-hour gap between activities on a day
- preferred timeblocks, e.g., bonus points for activities scheduled in the core of the day

19.0.3.2 Distance-based metrics

By incorporating room location data, we can calculate travel distances and times between activities. Long travel times or back-to-back activities in distant locations would incur penalties. See Rooms and Spaces for more details.

19.0.3.3 Resource Utilisation

Metrics related to room utilisation, such as occupancy rates and frequency of use, can provide insights into the efficient allocation of space. Low utilisation rates could be penalised, encouraging more effective use of resources.

19.0.3.4 Activity Characteristics

Factors like activity clashes, oversubscription rates, and room suitability (size, type) can impact student experience. Penalties could be based on the severity of these characteristics.

20 Metric Aggregations

With the individual metrics calculated, the next step is to aggregate these into meaningful scores at different levels. This could be at the student level, programme level, department level, or even at the room or building object.

The metrics used and their weightings will depend on the use-case and the priorities of the institution. For example, a student-level score could be used to identify students with particularly poor timetables, while a programme-level score could be used to compare the quality of timetables across different programmes, and a room-level score could be used to identify rooms that are underutilised or overbooked or are otherwise unsuitable.

This allows for a more nuanced understanding of timetable quality and can help identify areas for improvement.

20.0.0.1 Student-level

Each student node can have a quality score reflecting their individual timetable experience based on assigned activities and associated penalties.

20.0.0.2 Programme-level

By aggregating student scores within a programme, we gain insights into the overall quality experienced by students in that programme.

20.0.0.3 Other groupings

Scores can be aggregated at various levels, such as by department, room type, or time slot, to identify potential areas for improvement.

21 Implementing TQI

Prototype queries have been identified to identify constraint violations. Several of these queries are quite complex but their final form will be dependent on the use-case as well as the graph data model.

As a simple example, the following query identifies students with back-to-back activities in different buildings, highlighting a potential travel time issue:

```
// Identify students with back-to-back activities in different buildings
MATCH (s:Student)-[:ATTENDS]->(a1:Activity)-[:NEXT]->(a2:Activity)
WHERE a1.endTime = a2.startTime AND a1.building <> a2.building
RETURN s.name, a1.name, a2.name, a1.building, a2.building
```

Or we could craft a query to calculate the travel time between back-to-back activities:

```
// Calculate travel time between consecutive activities for a student on a specific date
MATCH (s:student {stuFullName_anon: "David Johnson"})-[:ATTENDS]->(a1:activity)-[:OCCUPIES]->
      (s)-[:ATTENDS]->(a2:activity)-[:OCCUPIES]->(r2:room)
WHERE a1.actEndTime = a2.actStartTime AND a1.actStartDate = a2.actStartDate AND a1 <> a2 AND
      a1.actStartDate IN [date("2023-01-11"), date("2022-09-27"), date("2023-03-14")]
RETURN DISTINCT
    s.stuFullName_anon,
    a1.actName AS act1, a1.actStartDate AS date, a1.actStartTime+"-"+a1.actEndTime AS act1Ti
    point.distance(r1.location, r2.location) AS distance,
    round(point.distance(r1.location, r2.location) / 1.4) AS walkingTimeSeconds // Calculate
```

	s.stuFullName_a	act1	date	act1Times	act2Times	act2	distance	walkingTimeSec
1	"David Johnson"	"UFCEM1-60-M SB L_oc/01"	2022-09-27	"10:00:00-11:00:00"	"11:00:00-12:00:00"	"UFCEM1-60-M SB GT_oc/01"	0.0	0.0
2	"David Johnson"	"UFCEP1-30-M CP_oc/01"	2023-01-11	"14:00:00-17:00:00"	"17:00:00-19:00:00"	"UFMFHR-15-M TB1 CP_oc/01"	26.878321926	19.0
3	"David Johnson"	"UFCEM1-60-M W_oc/01"	2023-03-14	"09:00:00-12:00:00"	"12:00:00-14:00:00"	"UFCEN1-15-M W_oc Wk 34/01"	143.77014457	103.0

Figure 21.1: Travel time between activities

See Cypher Queries - Hard Constraints and Cypher Queries - Soft Constraints for more examples and details.

21.0.1 Penalty and Reward System

One way of implementing this is to store the quality score as a property on the relevant node (student, programme, room, etc.). Starting with a baseline score, the quality score is dynamically updated by subtracting penalties and adding rewards based on the specific metrics calculated. The weighting of these penalties and rewards can be adjusted to reflect institutional priorities.

Using the back-to-back activities example above, we can imagine using either `distance` or `walkingTimeSeconds` and a sliding scale to calculate a penalty. For example, if the walking time is greater than 5 minutes, a penalty of -3 points could be applied with the penalty increasing as the walking time increases.

Further examples:

No lunch break: -5 points

Back-to-back activities 5+ minutes apart: -3 points per instance

Activity clash: -10 points

Room at full capacity: -2 points

High room utilisation rate: +2 points

22 TQI Summary

22.0.1 Visualisation of Results

The calculated scores and underlying metrics can be effectively visualised using various techniques:

Bloom visualisations in Neo4j: These can provide an intuitive overview of timetable quality across different programmes, time slots, or other groupings. They enable users to explore hierarchical relationships, identify patterns and outliers, and drill down into specific data points. See Perspectives for initial ideas.

Charts and dashboards: Bar charts, line graphs, and heatmaps can be used to display and compare scores, identify trends, and track changes over time. Interactive dashboards can be built to provide a various views of timetable quality metrics and enable stakeholders to explore the data, identify trends, and make informed decisions.

22.0.2 Potential Challenges

While the concept of a timetable quality index offers many benefits, there are several challenges to acknowledge:

Data quality and availability: As with any analysis, accuracy and completeness of timetable data is crucial to calculate reliable quality scores. Inconsistent or missing data can lead to inaccurate results and skewed conclusions.

Complexity of metrics: Defining and calculating meaningful metrics that capture the nuances of timetable quality can be challenging and time-consuming.

Metric Definition and Weighting: Ironically, the very attempt to quantify quality is based on subjective judgements - which metrics to include, how to calculate them and how to weight them.

22.0.3 Benefits and Future Development

A timetable quality index is a potentially a powerful mechanism which can help universities gain a more quantifiable and data-driven understanding of their timetabling function.

The flexibility of graph allows for rapid prototyping, experimentation and deployment of new metrics and scoring systems. This can help institutions to identify areas for improvement, allocate resources more effectively, and enhance the overall student experience.

Future developments could include:

- Identifying additional metrics that capture the quality of timetables more comprehensively.
- Refining the weighting system for penalties and rewards based on stakeholder feedback and institutional priorities.
- Incorporating additional datasets, such as student preferences or transportation schedules, to enhance the accuracy and granularity of the quality index.
- Incorporating additional constraints and preferences, such as room suitability, staff availability, student preferences and any reasonable adjustments.
- Developing interactive dashboards that allow users to explore timetable data, simulate changes, and assess their impact on the quality score.

23 Final Thoughts

As I reflect on this project, I am struck by my progress in realising the original objectives I set out to achieve and what it took to get here. Through a combination of perseverance, problem-solving, and a relentless drive to deliver a tangible solution, I am proud to say that I have successfully:

1. Designed an **extensible, system-agnostic graph data model** for university timetables, providing a flexible and adaptable foundation for capturing the complex relationships inherent in timetabling data.
2. Developed a **configurable ETL (extract, transform, load) pipeline** to seamlessly transition from relational database representations to a graph database, unlocking new possibilities for timetable analysis and optimisation.
3. Discussed how **graph-based approaches to timetabling analysis** can contribute to measuring and improving the overall quality of university timetables, a critical aspect of enhancing the student experience and institutional efficiency.

But I could not have done it without others. I am grateful for the support, guidance and encouragement I have received from many people along the way.

23.1 Reflections on Journey

This journey has had a little bit of everything: challenges, setbacks, breakthroughs, and moments of clarity. It has tested my limits, pushed me to grow, and allowed me to create something that I believe can make a meaningful impact. I have learned a lot about myself, my capabilities, and the power of perseverance; I have gained new skills and insights; I have developed a deeper understanding of where I want to go from here.

Admittedly, the project was ambitious, and I found myself struggling with to navigate the ever-expanding scope, not knowing when to 'stop.' But, I am proud of what I have achieved and learned, as well as where I am right now - when I can comfortably draw a line under this project as a proof-of-concept, knowing that it provides a foundation for future work and exploration.

In short, it has been fulfilling and I did what I wanted to do - something exploratory, practical, new, challenging, and impactful.

But it is especially rewarding to receive feedback from subject matter experts, such as the timetabling data manager at UWE[1], who said:

- `Opens new reporting and analytics opportunities`
- `Very useful graphical representation of the relations within the database`
- `Huge time saving comparing to current SQL methods`
- `Easily adjustable and scalable`
- `Date and time represented in much better way`
- `Makes reporting of timetable clashes such an easy task`

23.2 Looking Ahead: The Future of Graph at Universities

Looking ahead, I am confident that the potential of graph databases in the realm of timetabling analysis has only begun to be explored. My project has only scratched the surface but there is a vast reserve of untapped opportunity. By continuing to explore and refine the concepts introduced in this project, Higher Education Institutions can unlock new levels of insight to improve efficiency, agility, and student satisfaction.

And it does not only apply to timetabling datasets.

Universities hold a significant amount of interconnected data that can be leveraged to improve the student experience, make more informed decisions, and drive positive change. The possibilities are endless - to illustrate, I have included some blue-skies thinking in the Appendix.

[1]Personal communication with Wojciech Lewicki, 19 August 2024 - when reviewing and discussion the project.

24 Appendix: Table of Contents

This project contains a significant amount of information in appendices. It was always my intention to provide well-documented code and detailed explanations of the data engineering process, the Cypher queries, etc.

The appendices are listed below:

General

Data Engineering

Cypher Queries

R. Hard (timetabling) Constraints

S. Student Clashes

T. Soft Constraints

U. Rooms and Spaces

V. Perspectives

Supervision

W. Supervision

X. Notes Example 1

Y. Notes Example 2

Z. Notes Example 3

25 A: Random Graph Generator

The function below generates a random graph (dot file) using Graphviz.

To render, ensure that graphviz is installed or save to file and render within documents using Quarto or similar.

```
import graphviz
import random
import string
from collections import defaultdict

def generate_random_graph(num_nodes=50, num_edges=100, num_clusters=5, colors=None):
    """Generates a random Graphviz graph with clusters and random colours.

    Args:
        num_nodes: Number of nodes in the graph.
        num_edges: Number of edges in the graph.
        num_clusters: Number of clusters to create.
        colors: List of colours to use for clusters (optional). If not provided, random colou
    """

    dot = graphviz.Digraph("G")
    dot.attr(fontname="Helvetica,Arial,sans-serif")
    dot.attr(layout="neato")
    dot.attr(start="random")
    dot.attr(overlap="false")
    dot.attr(splines="true")
    dot.attr(size="8,8")
    #dot.attr(dpi="300")

    # nodes to clusters, random colours if not provided
    cluster_assignments = {}
    if colors is None:
        colors = ["#%06x" % random.randint(0, 0xFFFFFF) for _ in range(num_clusters)]

    for i in range(num_nodes):
```

```
        cluster_assignments[i] = random.randint(0, num_clusters - 1)

    # random node names, colouur assignment
    nodes = []
    for i in range(num_nodes):
        node_name = ''.join(random.choices(string.ascii_lowercase + string.digits, k=8))
        nodes.append(node_name)
        cluster_id = cluster_assignments[i]
        color = colors[cluster_id]
        dot.node(node_name, label="", shape="circle", height="0.12", width="0.12", fontsize="


    # random edges (with a higher probability of staying within clusters)
    edges = []
    for _ in range(num_edges):
        src_cluster = random.randint(0, num_clusters - 1)
        dst_cluster = src_cluster if random.random() < 0.8 else random.randint(0, num_cluste
        src_node = random.choice([node for i, node in enumerate(nodes) if cluster_assignments
        dst_node = random.choice([node for i, node in enumerate(nodes) if cluster_assignments
        edges.append((src_node, dst_node))

    #  edges to the graph
    for edge in edges:
        dot.edge(*edge)

    return dot
```

26 B: Technology Stack

This project used a variety of tools, applications, programming languages, and so on. Below is a high-level record of the 'tech stack' - the `what` and `why`:

26.0.1 Programming

1. `Python` - Main programming language.
2. `SQL` - SELECT queries to extract source data from relational database.
3. `Cypher` - Querying language for Neo4j Graph Databases.
4. `Batch` - Windows command language to handle yaml files and multi-format rendering.
5. `VSCode` - Main IDE (Integrated Development Environment).

26.0.2 Documentation

- `Quarto` - Open-source technical publishing system.
- `Jupyter` - Open-source application for interactive notebooks.
- `Zotero` - Open-source reference management system.

26.0.3 Visualisation

- `Graphviz` - Open-source graph visualisation application.
- `Mermaid` - Open-source JavaScript diagramming tool.
- `Arrows` - Neo4j Labs diagramming tool.

26.0.4 Versioning

- `Github` - Web-based platform for version control and collaboration using Git.

26.0.5 Python Libraries

Several Python libraries were explored in the development of this prototype. The below libraries are the ones used in the current implementation.

26.0.5.1 Directory/File Handling

- `os` - Interacting with the operating system for tasks like creating, deleting, and navigating directories and files.
- `glob` - Finding files and directories based on pattern matching.
- `io` - Working with input/output streams for reading and writing data.

26.0.5.2 Data Handling

- `pandas` - Handling tabular data for analysis and manipulation.
- `json` - Encoding and decoding JSON (JavaScript Object Notation) data.

26.0.5.3 Typing and Logging

- `typing` - Adding type hints to code for better code readability, maintainability, and static type checking.
- `logging` - Configuring and managing logging for application.
- `time` - Working with time-related functions, potentially used for logging timestamps.

26.0.5.4 Database Connectivity

- `keyring` - Securely storing and retrieving passwords and other sensitive information.
- `pyodbc` - Connecting to and interacting with SQL databases using the Open Database Connectivity (ODBC) standard.
- `neo4j` - Interacting with Neo4j graph databases.

26.0.5.5 Google API Integration

- `googleapiclient` - Interacting with various Google APIs.
- `google.oauth2` - Handling OAuth 2.0 authentication for accessing Google services securely.

26.0.5.6 Anonymisation

- `random` - Generating random numbers and making random choices.
- `hashlib` - Implementing various secure hash and message digest algorithms.
- `Faker` - Generating fake data for testing and development purposes.

27 C: Configuration YAML

The below is an example of configuration options configured in more human readable YAML format.

```yaml
# ETL Pipeline Configuration

general:
  hostkeys:
    - INB112
    # - N420
  folder_name: '' # default to hostkey if empty

file_paths:
  root_dir: '.'  # default to current working directory
  nodes_folder_url: # (Optional) override for dynamic lookup) eg "https://drive.google.com/d
  relationships_folder_url: # (Optional) override for dynamic lookup) eg."https://drive.googl
  gdrive_root_folder_url: "1iWkeTubJ0xZ6I728emoj9BkqZm7dL2fq"
  gdrive_folder_name: # Leave commented out to use default (hostkey)
  google_credentials_path: 'credentials/graph-diss-dbbdbb5e5d00.json'
  department_source: 'node-dept-all.csv'
  archibus_source: 'archibus.csv'

data_processing:
  chunk_size: 20000
  temp_tables_sql_file: "create_temp_tables.sql"
  node_output_filename_template: "node-{node}-processed.csv"
  rel_output_filename_template: "rel-{relationship}-processed.csv"

neo4j:
  #max_connection_retries: 5
  #max_transaction_retry_time: 30
  schema:
    apply: True
    type: 'dynamic' # Options: 'dynamic', 'custom'
    custom_path: ''
```

```yaml
  batch_size: 1000

logging:
  log_level: "INFO" # Options: DEBUG, INFO, WARNING, ERROR, CRITICAL

nodes:
  department:
    filename_pattern: "node-dept-all*.csv"
    dept_join_col: null
    node_suffix: 'dept'
    node_id: "deptSplusID"
  module:
    filename_pattern: "node-module-by-pos-temp*.csv"
    dept_join_col: "modSplusDeptID"
    node_suffix: "mod"
    node_id: "modSplusID"
  room:
    filename_pattern: "node-room-by-pos-temp*.csv"
    dept_join_col: null
    node_suffix: 'room'
    node_id: "roomSplusID"
  programme:
    filename_pattern: "node-pos-by-pos-temp*.csv"
    dept_join_col: "posSplusDeptID"
    node_suffix: "pos"
    node_id: "posSplusID"
  activityType:
    filename_pattern: "node-activitytype-by-pos-temp*.csv"
    dept_join_col: 'actTypeDeptSplusID'
    node_suffix: 'actType'
    node_id: 'actTypeSplusID'
  staff:
    filename_pattern: "node-staff-by-pos-temp*.csv"
    dept_join_col: "staffDeptSplusID"
    node_suffix: "staff"
    dtype:
      staffSplusID: str
      staffID: str
    node_id: "staffSplusID"
  student:
    filename_pattern: "node-student-by-pos-temp*.csv"
    dept_join_col: "stuDeptSplusID"
```

```
    node_suffix: "stu"
    dtype:
      stuSplusID: str
      studentID: str
    node_id: "stuSplusID"
  activity:
    filename_pattern: "node-activity-by-pos-temp*.csv"
    dept_join_col: null
    node_suffix: null
    dtype:
      actSplusID: str
      actTypeSplusID: str
      actRoomSplusID: str
      actStaffSplusID: str
      actStuSplusID: str
      actStartDateTime: str
      actEndDateTime: str
      actFirstActivityDate: str
      actLastActivityDate: str
      actWhenScheduled: str
    node_id: "actGraphID"

relationships:
  activity_module:
    filename_pattern: "rel-activity-module-by-pos-temp*.csv"
    node1_col: "actSplusID"
    node2_col: "modSplusID"
    relationship: "BELONGS_TO"
  activity_room:
    filename_pattern: "rel-activity-room-by-pos-temp*.csv"
    node1_col: "actSplusID"
    node2_col: "roomSplusID"
    relationship: "OCCUPIES"
  activity_staff:
    filename_pattern: "rel-activity-staff-by-pos-temp*.csv"
    node1_col: "staffSplusID"
    node2_col: "actSplusID"
    relationship: "TEACHES"
  activity_student:
    filename_pattern: "rel-activity-student-by-pos-temp*.csv"
    node1_col: "stuSplusID"
    node2_col: "actSplusID"
```

```
    relationship: "ATTENDS"
  activity_activityType:
    filename_pattern: "relActivityActType*.csv"
    node1_col: "actSplusID"
    node2_col: "actActivityTypeSplusID"
    relationship: "HAS_TYPE"
  module_programme:
    filename_pattern: "rel-mod-pos-by-pos-temp*.csv"
    node1_col: "modSplusID"
    node2_col: "posSplusID"
    relationship: "BELONGS_TO"
    properties:
      - "modType"

data_type_mapping:
  activity:
    actStartDateTime: ['datetime', '%Y-%m-%d %H:%M:%S']
    actEndDateTime: ['datetime', '%Y-%m-%d %H:%M:%S']
    actFirstActivityDate: ['date2', '%Y-%m-%d']
    actLastActivityDate: ['date2', '%Y-%m-%d']
    actPlannedSize: 'int'
    actRealSize: 'int'
    actDuration: 'int'
    actDurationInMinutes: 'int'
    actNumberOfOccurrences: 'int'
    actWhenScheduled: ['datetime', '%Y-%m-%d %H:%M:%S']
    actStartDate: ['date', '%Y-%m-%d']
    actEndDate: ['date', '%Y-%m-%d']
    actStartTime: 'time'
    actEndTime: 'time'
    actScheduledDay: 'int'
  room:
    roomCapacity: 'int'

display_name_mapping:
  activity: "actName"
```

28 D: Anonymisation

The following code snippet is shows how I anonymised personal data in a DataFrame using the `Faker` library.

The code generates fake names, emails, and IDs for staff or student data based on the unique IDs in the extract DataFrame. The anonymised data is then merged back with the original DataFrame, and the original columns are removed.

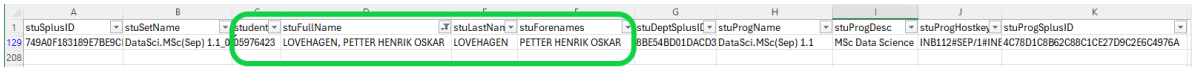


Figure 28.1: Pre-anonymisation Extract

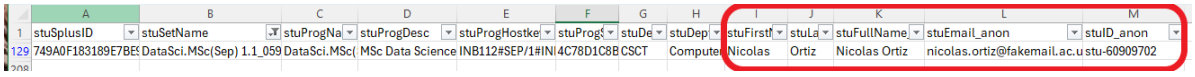


Figure 28.2: Post-anonymisation Extract

```python
import random
import hashlib
from faker import Faker
import pandas as pd


def anonymise_data(df):
    """
    anonymises cols in df by generating fake names, emails, and IDs.
    """
    process_logger.info("Starting anonymisation")
    process_logger.info(f"Columns in dataframe: {df.columns.tolist()}")

    # staff or student data
    if 'staffSplusID' in df.columns:
        process_logger.info("Processing staff data")
        id_col = 'staffID'
        prefix = 'staff'
```

```
        columns_to_remove = ['staffFullName', 'staffLastName', 'staffForenames', 'staffID']
    elif 'stuSplusID' in df.columns:
        process_logger.info("Processing student data")
        id_col = 'studentID'
        prefix = 'stu'
        columns_to_remove = ['stuFullName', 'stuLastName', 'stuForenames', 'studentID']
    else:
        process_logger.error("Neither 'staffSplusID' nor 'stuSplusID' found in columns.")
        return df  # Return original dataframe if required columns are missing

    # dictionary to store anonymised data
    anon_data = {}

    # generate anonymised data for each unique ID
    for unique_id in df[id_col].unique():
        # create a seed based on the unique_id
        seed = int(hashlib.md5(str(unique_id).encode()).hexdigest(), 16) & 0xFFFFFFFF
        fake = Faker()
        fake.seed_instance(seed)
        random.seed(seed)

        first_name = fake.first_name()
        last_name = fake.last_name()
        full_name = f"{first_name} {last_name}"
        email = f"{first_name.lower()}.{last_name.lower()}@fakemail.ac.uk"
        anon_id = f"{prefix}-{random.randint(10000000, 99999999):08d}"

        anon_data[unique_id] = {
            f'{prefix}FirstName_anon': first_name,
            f'{prefix}LastName_anon': last_name,
            f'{prefix}FullName_anon': full_name,
            f'{prefix}Email_anon': email,
            f'{prefix}ID_anon': anon_id
        }

    # create a new df with anonymised data
    df_anon = pd.DataFrame.from_dict(anon_data, orient='index')

    # reset the index and rename it to match the original ID column
    df_anon = df_anon.reset_index().rename(columns={'index': id_col})

    try:
```

```
        # Merge anonymised data with the original DataFrame
        df_result = pd.merge(df, df_anon, on=id_col)

        # Rmove columns that should be anonymised
        columns_to_remove = [col for col in columns_to_remove if col in df_result.columns]
        df_result = df_result.drop(columns=columns_to_remove)

        process_logger.info("Anonymisation completed successfully")
        return df_result

    except Exception as e:
        process_logger.error(f"Error during anonymisation: {str(e)}")
        return df  # return original df if error
```

29 E: ETL Summary

For the proof-of-concept, the following programmes and associated data (students, staff, modules, activities, rooms, etc.) were extracted from the source system and transformed before being loaded into a Neo4j cloud instance.

The table below summarises the time taken for each programme.

pos	level	hostkey	count	extract & process	gdrive	neo4j
Artificial Intelligence	PG	I400	16	26.8s	26.1s	2m 50.6s
Data Science	PG	INB112	206	57.6s	26.5s	1m 35.1s
Mathematics	UG	G90D	103	38.8s	25.5s	6m 12.9s
Computer Science	UG	I10J	431	1m 9.8s	24.1s	11m 16.1s
Computer Science	UG	G500	45	30s	24.4s	2m 9.5s
Cyber Security and Digital Forensics	UG	G4H4	271	57.4s	51.9s	6m 36.3s
Cyber Security	PG	I900	216	45.7s	25.9s	3m 7.5s
Information Management	PG	P110	42	17.2s	25.8s	1m 35.1s
Information Technology	PG	G56A12	174	28.3s	25.8s	2m 22.3s

The largest programme (Computer Science) took just over 1 minute to extract and process and 11 minutes to load into Neo4j. The Google Load consistently took ~25 seconds regardless of file sizes.

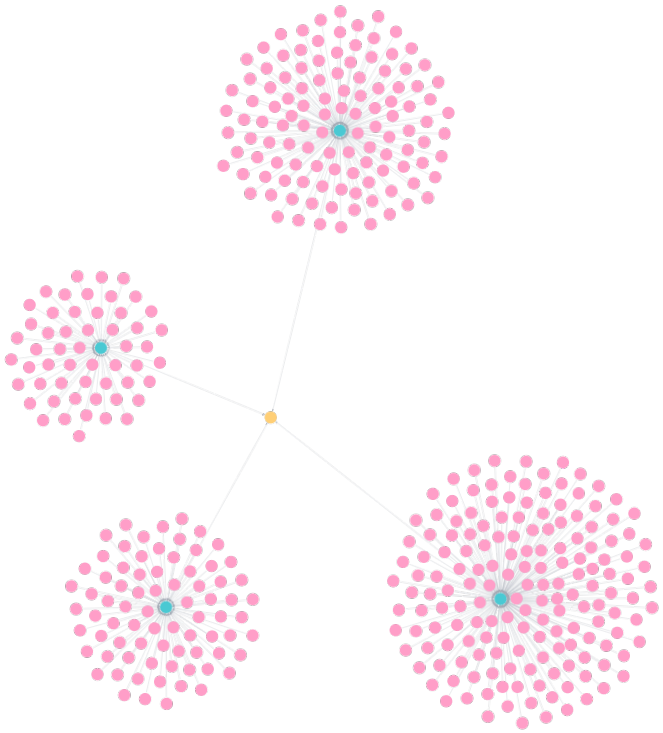


Figure 29.1: Computer Science (I10J) - Department-Programme-Students

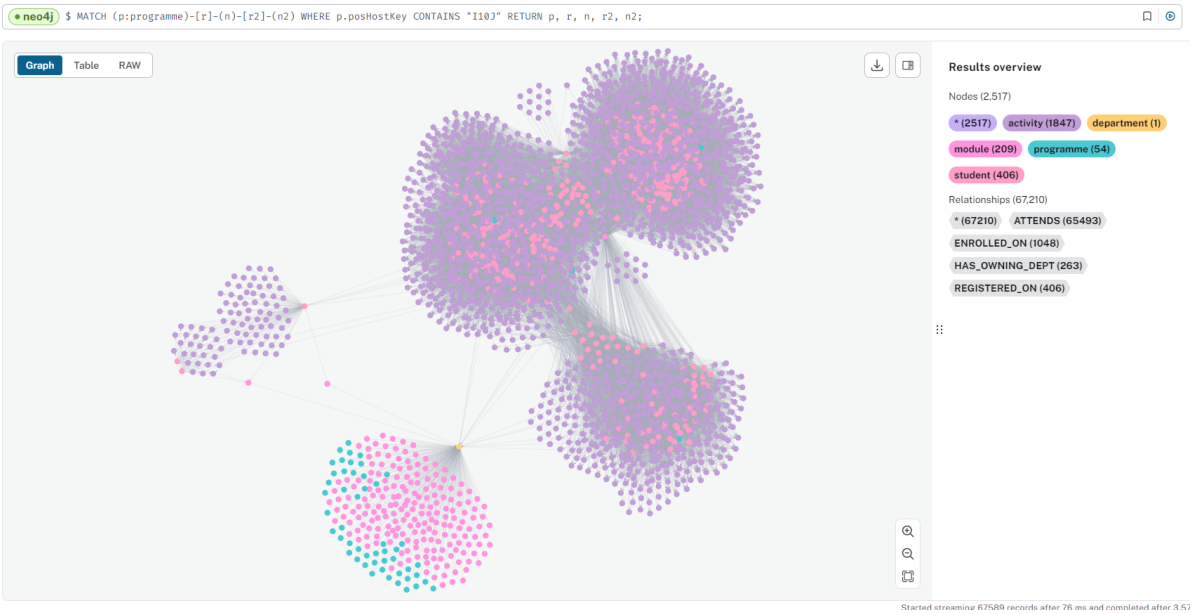


Figure 29.2: Computer Science (I10J) - Department-Programme-Modules

However, the current graph model creates a **significant** amount of relationships between nodes:

node/relationship	count
programme (n)	4
department (n)	1
hasOwningDept (r)	4
student (n)	413
registeredOn (r)	413
module (n)	11
enrolledOn (r)	1048
activity (n)	1847
attends (r)	65493
staff (n)	23
teaches (r)	538
room (n)	32
occupies (r)	462
activityType (n)	15
hasType (r)	1847

30 F: ETL Code Gists

Complete code for the Data Pipeline can be found in these Github gists or on the following pages.

GitHub Gists:

Graph Timetable - Quarto - Config and Misc

Graph Timetable - Quarto - SQL Queries

Graph Timetable - Quarto - Extract

Graph Timetable - Quarto - Transform

Graph Timetable - Quarto - Upload to Google Drive

Graph Timetable - Quarto - Load

Graph Timetable - Quarto - Config and Misc

Graph Timetable - Quarto - SQL Queries

Graph Timetable - Quarto - Extract

Graph Timetable - Quarto - Transform

Graph Timetable - Quarto - Upload to Google Drive

Graph Timetable - Quarto - Load

31 G: Config and Misc

Graph Timetable - Quarto - Config and Misc

1. config.py
2. logger_config.py
3. neo4j_schema.py
4. utils.py
5. connect_to_neo4j.py
6. connect_to_rdb.py

32 H: Extract-SQL

Graph Timetable - Quarto - SQL Queries

1. create_temp_tables.sql
2. node-activity-by-pos-temp.sql
3. node-activityType-by-pos-temp.sql
4. node-dept-all.sql
5. node-module-by-pos-temp.sql
6. node-pos-by-pos-temp.sql
7. node-room-by-pos-temp.sql
8. node-staff-by-pos-temp.sql
9. node-student-by-pos-temp.sql
10. rel-activity-module-by-pos-temp.sql
11. rel-activity-room-by-pos-temp.sql
12. rel-module-programme-by-pos-temp.sql
13. rel-staff-activity-by-pos-temp.sql
14. rel-student-activity-by-pos-temp.sql

33 I: Extract

Graph Timetable - Quarto - Extract

1. temp_table_loader
2. extract_sql_file
3. extract_data
4. extract_main

34 J: Google Drive Load

Graph Timetable - Quarto - Upload to Google Drive

1. gdrive_upload
2. google_drive_utils

35 K: Process

Graph Timetable - Quarto - Process

1. process_utils
2. process_node
3. process_main

36 L: Neo4j Load

Graph Timetable - Quarto - Load

1. load_schema
2. load_utils
3. load_nodes
4. load_relationships
5. convert_properties
6. set_display_properties
7. load_main

37 M: Cypher Queries

These pages collate Cypher[1] queries used in the development of this project. The queries are grouped by the type of operation they perform, such as creating nodes, relationships, or querying the graph. The queries are presented in a format that can be copied and pasted directly into a Neo4j browser or other Cypher-compatible interface.

They represent a *starting point* for further exploration and development of the graph database. The queries are not exhaustive, and there are many more possibilities for querying and analysing the data.

37.1 Constraints

Constraints in Neo4j are used to enforce rules on the graph data, such as ensuring that certain properties are unique or that nodes have specific properties. Constraints can be used to maintain data integrity and prevent duplicate or inconsistent data. For example, we would want to enforce uniqueness constraints on most nodes and relationships, so that we do not duplicate students or allocations, etc.

	id	name	type	entityType	labelsOrTypes	properties
1	20	"graphid_dem	"UNIQUENESS"	"NODE"	["demoRoom"]	["graphid"]
2	14	"rm_id_demoR	"UNIQUENESS"	"NODE"	["demoRoom"]	["rm_id"]

Figure 37.1: Constraints

37.2 Indexes

Indexes in Neo4j are used to speed up queries by allowing the database to quickly locate nodes or relationships based on a property value. They will depend heavily on graph structure and specific use-cases. For example, if performing regular searches on `(activity{actStartTime})` it is probably beneficial to create an index on this property.

[1] "**Cypher** is a declarative graph query language that allows for expressive and efficient data querying in a property graph" (Wikipedia contributors, 2024)

Search performance indexes include `RANGE`, `FULLTEXT`, and `POINT`. Range indexes are the default and support most queries. Text indexes, like `FULLTEXT`, are used for `string`-based searches and optimised for queries containing operators like `CONTAINS` or `STARTS WITH`. Point indexes are used for spatial queries and are optimised for `latitude` and `longitude` properties.

Indexes in Graph

```
from connect_to_neo4j_db import connect_to_neo4j
from neo4j import GraphDatabase

# connect to Neo4j
driver = connect_to_neo4j()

# session
session = driver.session()

# run query
query = """
SHOW INDEXES;
"""
print("Running query...\n")
result = session.run(query)
for record in result:
    print(record)
```

```
Connecting to Neo4j database....
Connected to Neo4j database successfully! Driver: <neo4j._sync.driver.Neo4jDriver object at
Running query...

<Record id=7 name='activity_clash_index' state='ONLINE' populationPercent=100.0 type='RANGE'
<Record id=19 name='graphid_demoRoom_uniq' state='ONLINE' populationPercent=100.0 type='RANGE
<Record id=4 name='index_12b79beb' state='ONLINE' populationPercent=100.0 type='RANGE' entity
<Record id=6 name='index_207d313' state='ONLINE' populationPercent=100.0 type='RANGE' entity
<Record id=8 name='index_241bd22f' state='ONLINE' populationPercent=100.0 type='RANGE' entity
<Record id=9 name='index_2d375d70' state='ONLINE' populationPercent=100.0 type='RANGE' entity
<Record id=0 name='index_343aff4e' state='ONLINE' populationPercent=100.0 type='LOOKUP' entit
<Record id=10 name='index_43c5c824' state='ONLINE' populationPercent=100.0 type='RANGE' entit
<Record id=2 name='index_6acb9a84' state='ONLINE' populationPercent=100.0 type='RANGE' entity
<Record id=5 name='index_7ade165f' state='ONLINE' populationPercent=100.0 type='RANGE' entity
<Record id=1 name='index_f7700477' state='ONLINE' populationPercent=100.0 type='LOOKUP' entit
<Record id=3 name='index_f86013e' state='ONLINE' populationPercent=100.0 type='RANGE' entity
<Record id=18 name='lat_demoRoom' state='ONLINE' populationPercent=100.0 type='RANGE' entity
<Record id=17 name='lon_demoRoom' state='ONLINE' populationPercent=100.0 type='RANGE' entity
```

```
<Record id=16 name='rm_cat_demoRoom' state='ONLINE' populationPercent=100.0 type='RANGE' enti
<Record id=15 name='rm_type_demoRoom' state='ONLINE' populationPercent=100.0 type='RANGE' ent
<Record id=11 name='room_fulltext_index' state='ONLINE' populationPercent=100.0 type='FULLTE
<Record id=12 name='room_geo_index' state='ONLINE' populationPercent=100.0 type='FULLTEXT' en
```

37.2.1 Creating Indexes

This is the general syntax for creating an INDEX and some examples.

```
// General syntax
CREATE INDEX FOR (n:NodeLabel) ON (n.propertyName)

// Examples
CREATE INDEX FOR (a:activity) ON (a.actStartDate);
CREATE INDEX FOR (a:activity) ON (a.actStartTime);
CREATE INDEX FOR (a:activity) ON (a.actEndTime);
CREATE INDEX FOR (a:activity) ON (a.actModSplusID);
CREATE INDEX FOR (a:activity) ON (a.id);
// composite index for clashes
CREATE INDEX activity_clash_index FOR (a:activity) ON (a.actStartDate, a.actStartTime, a.actE

CREATE INDEX FOR (r:room) ON (r.roomName);
CREATE INDEX FOR (r:room) ON (r.roomLatitude);
CREATE INDEX FOR (r:room) ON (r.roomLongitude);

CREATE SPATIAL INDEX room_location_index FOR (r:demoRoom) ON (r.location)
CREATE SPATIAL INDEX demoroom_location_index FOR (r:room) ON (r.location)
```

38 N: Creating Nodes and Relationships

Creating nodes and relationships is the first step in building a graph database. Nodes represent entities in the graph, such as students, rooms, or activities. Relationships connect nodes and represent the connections between them.

38.1 Creating Nodes

Nodes are created using the `CREATE` clause in Cypher. The general syntax for creating a node is:

```
CREATE (n:NodeLabel {propertyName: propertyValue, ...})
```

Where:

- `n` is the node variable
- `NodeLabel` is the label assigned to the node
- `propertyName` is the property name
- `propertyValue` is the value assigned to the property
- `...` represents additional properties

38.1.1 Example: Creating a Student Node

```
CREATE (s:Student {studentID: '123456', studentName: 'Alice'...})
```

38.2 Creating Relationships

Relationships are created using the `CREATE` clause in Cypher. The general syntax for creating a relationship is:

```
MATCH (n1:NodeLabel1), (n2:NodeLabel2)
WHERE n1.propertyName = propertyValue1 AND n2.propertyName = propertyValue2
CREATE (n1)-[r:RELATIONSHIP_TYPE]->(n2)
```

Where:

- `n1` and `n2` are the node variables
- `NodeLabel1` and `NodeLabel2` are the labels assigned to the nodes
- `propertyName` is the property name
- `propertyValue` is the value assigned to the property
- `r` is the relationship variable
- `RELATIONSHIP_TYPE` is the type of relationship
- `->` represents the direction of the relationship
- `MATCH` is used to find the nodes to connect, optional
- `WHERE` is used to filter the nodes, optional
- `CREATE` is used to create the relationship
- `...` represents additional properties

38.2.1 Example: Creating a Relationship Between a Student and an Activity

```
MATCH (s:Student {studentID: '123456'}), (a:Activity {activityID: '789'})
CREATE (s)-[r:ATTENDS]->(a)
```

38.3 Relationships created after ETL

The following relationships were applied to my proof-of-concept graph after loading data using the ETL, using node properties. Delete code also supplied below.

> **Caution**
>
> Use code with caution, especially DELETE code. Run pattern matching and investigate results before committing to delete.

38.3.1 (student)-[REGISTERED_ON]->(programme)

```
// Create REGISTERED_ON relationship between student and programme nodes

// Match student and programme nodes based on matching properties
MATCH (s:student), (p:programme)
WHERE s.stuProgSplusID = p.posSplusID

// Create REGISTERED_ON relationship
MERGE (s)-[:REGISTERED_ON]->(p)
```

```
// Delete REGISTERED_ON relationships
MATCH (:student)-[r:REGISTERED_ON]->(:programme)
DELETE r
```

38.3.2 (student)-[ENROLLED_ON]->(module)

```
// Create ENROLLED_ON relationship between students and modules

// Match student, activity, and module nodes based on ATTEND and BELONGS_TO relationships
MATCH (s:student)-[:ATTENDS]->(a:activity)-[:BELONGS_TO]->(m:module)

// Create ENROLLED_ON relationship
MERGE (s)-[:ENROLLED_ON]->(m)
```

```
// Delete ENROLLED_ON relationships
MATCH (:student)-[r:ENROLLED_ON]->(:module)
DELETE r
```

38.3.3 (activity)-[HAS_TYPE]->(activity_type)

```
// Create HAS_TYPE relationship between activity and activityType nodes

// Match activity and activityType nodes based on matching properties
MATCH (a:activity), (at:activityType)
WHERE a.actTypeName = at.actTypeDescription

// Create HAS_TYPE relationship
MERGE (a)-[:HAS_TYPE]->(at)
```

```
// Find activities without an activityType

MATCH (a:activity)
WHERE NOT EXISTS ((a)-[:HAS_TYPE]->(:activityType))
RETURN a
```

```
// Delete HAS_TYPE relationships between activity and activityType nodes

MATCH (a:activity)-[r:HAS_TYPE]->(at:activityType)
DELETE r
```

38.3.4 (module)-[HAS_OWNING_DEPT]->(department)

```
// Create HAS_OWNING_DEPT relationship between module and department nodes

// Match module and department nodes based on matching properties
MATCH (m:module), (d:department)
WHERE m.modDepartment = d.deptName

// Create HAS_OWNING_DEPT relationship
MERGE (m)-[:HAS_OWNING_DEPT]->(d)
```

```
// Delete HAS_OWNING_DEPT relationships between module and department nodes

MATCH (m:module)-[r:HAS_OWNING_DEPT]->(d:department)
DELETE r
```

38.3.5 (programme)-[HAS_OWNING_DEPT]->(department)

```
// Create HAS_OWNING_DEPT relationship between programme and department nodes

// Match programme and department nodes based on matching properties
MATCH (p:programme), (d:department)
WHERE p.posDepartment = d.deptName

// Create HAS_OWNING_DEPT relationship
MERGE (p)-[:HAS_OWNING_DEPT]->(d)
```

```
// Delete HAS_OWNING_DEPT relationships between programme and department nodes

MATCH (p:programme)-[r:HAS_OWNING_DEPT]->(d:department)
DELETE r
```

39 O: Deleting Nodes and Relationships

Deleting nodes and relationships, using the `DELETE` clause in Cypher, is an important operation in managing the graph database.

39.1 Deleting Nodes

The general syntax for deleting a node is:

```
MATCH (n:NodeLabel {propertyName: propertyValue})
DELETE n
```

Where:

- `n` is the node variable
- `NodeLabel` is the label assigned to the node
- `propertyName` is the property name
- `propertyValue` is the value assigned to the property
- `DELETE` is used to delete the node
- `MATCH` is used to find the node to delete
- `...` represents additional properties

39.1.1 Example: Deleting a Student Node

```
MATCH (s:Student {studentID: '123456'})
DELETE s
```

39.2 Deleting Relationships

The general syntax for deleting a relationship is:

```
MATCH (n1:NodeLabel1)-[r:RELATIONSHIP_TYPE]->(n2:NodeLabel2)
DELETE r
```

Where:

- `n1` and `n2` are the node variables
- `NodeLabel1` and `NodeLabel2` are the labels assigned to the nodes
- `r` is the relationship variable
- `RELATIONSHIP_TYPE` is the type of relationship
- `DELETE` is used to delete the relationship
- `MATCH` is used to find the relationship to delete
- `...` represents additional properties

39.2.1 Example: Deleting a Relationship Between a Student and an Activity

```
MATCH (s:Student {studentID: '123456'})-[r:ATTENDS]->(a:Activity {activityID: '789'})
DELETE r
```

40 P: General Queries

This page contains a *selection* of general queries which can be used to explore the graph database. The queries are designed to provide insights into the data and relationships between nodes.

40.0.1 List all nodes

The following query lists all nodes in the graph.

> **Caution**
>
> Consider size of graph and limits in settings before running this query.

```
MATCH (n)
RETURN n
```

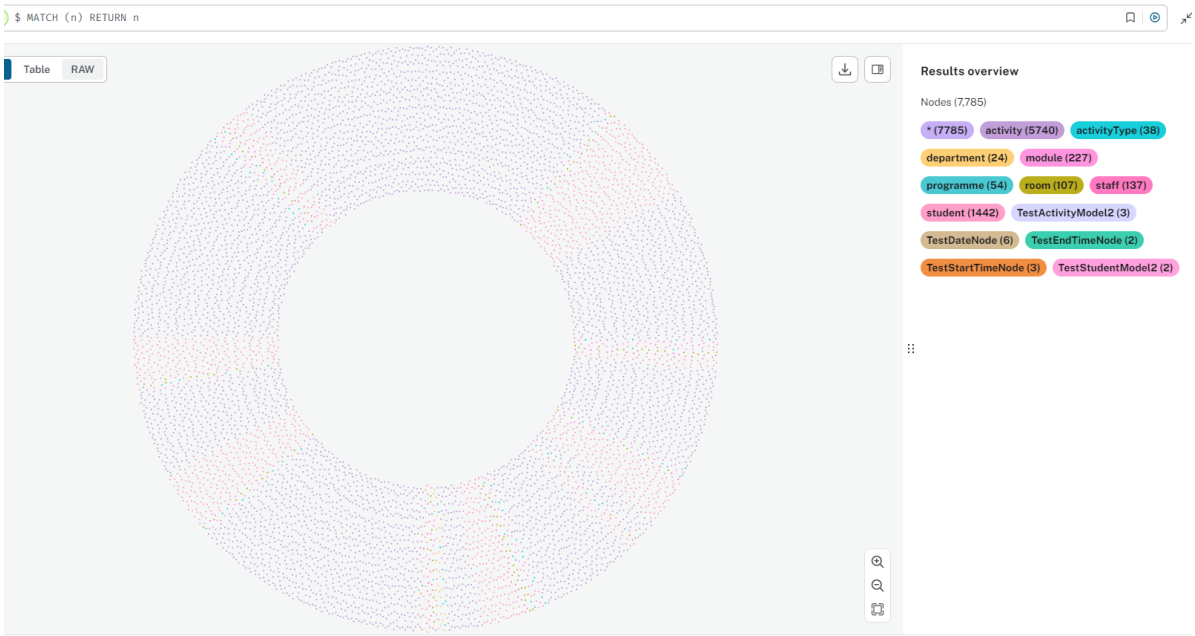


Figure 40.1: All Nodes

40.0.2 datatype of property

Properties in a graph have `datatypes` which will enable different operations and there for insights. The query below returns the datatype of a node property.

```
/* return datatype of actStartTime on activity node */

MATCH (a:activity)
RETURN DISTINCT apoc.meta.cypher.type(a.actStartTime) as actStartTimeType
```

Figure 40.2: Datatype of Property

40.0.3 unique properties

A node or relationship can potentially have many properties. The query below lists the properties of a node - in this case, `activity`.

```
// List unique properties for a Node

MATCH (a:activity)
UNWIND keys(a) AS propertyKey
RETURN COLLECT(DISTINCT propertyKey) AS propertyKeys
//RETURN DISTINCT propertyKey as propertyKeys
```

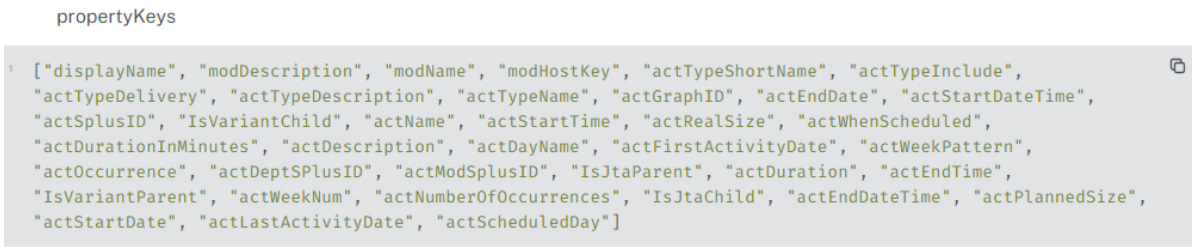


Figure 40.3: Unique Property Values

40.0.4 node labels without relationships

Graph databases are all about the relationships between nodes. It can be useful identifying nodes without relationships as they could indicate a problem with the data, data loading mechanism or be the outliers you want to identify.

For example, in a timetabling scenario, we would expect all nodes to be related to another node. However, we can see that several node labels are orphans. In the proof-of-concept, these results are expected or deliberate, due to the source data.

```
// Find node labels without relationships and their count

MATCH (n)
WHERE NOT EXISTS(()-[]-(n)) AND NOT EXISTS((n)-[]-())
RETURN DISTINCT labels(n) AS nodeLabels, count(n) AS nodeCount
```

	nodeLabels	nodeCount
1	["department"]	17
2	["activityType"]	4
3	["staff"]	12
4	["room"]	15

Figure 40.4: Node labels without Relationships

40.0.5 nodes without relationships - aka orphans

Instead of returning a count of nodes without relationships per node label we can return the nodes as a graph or a table:

```
// Find nodes without relationships

MATCH (n)
WHERE NOT EXISTS(()-[]-(n)) AND NOT EXISTS((n)-[]-())
RETURN n
```

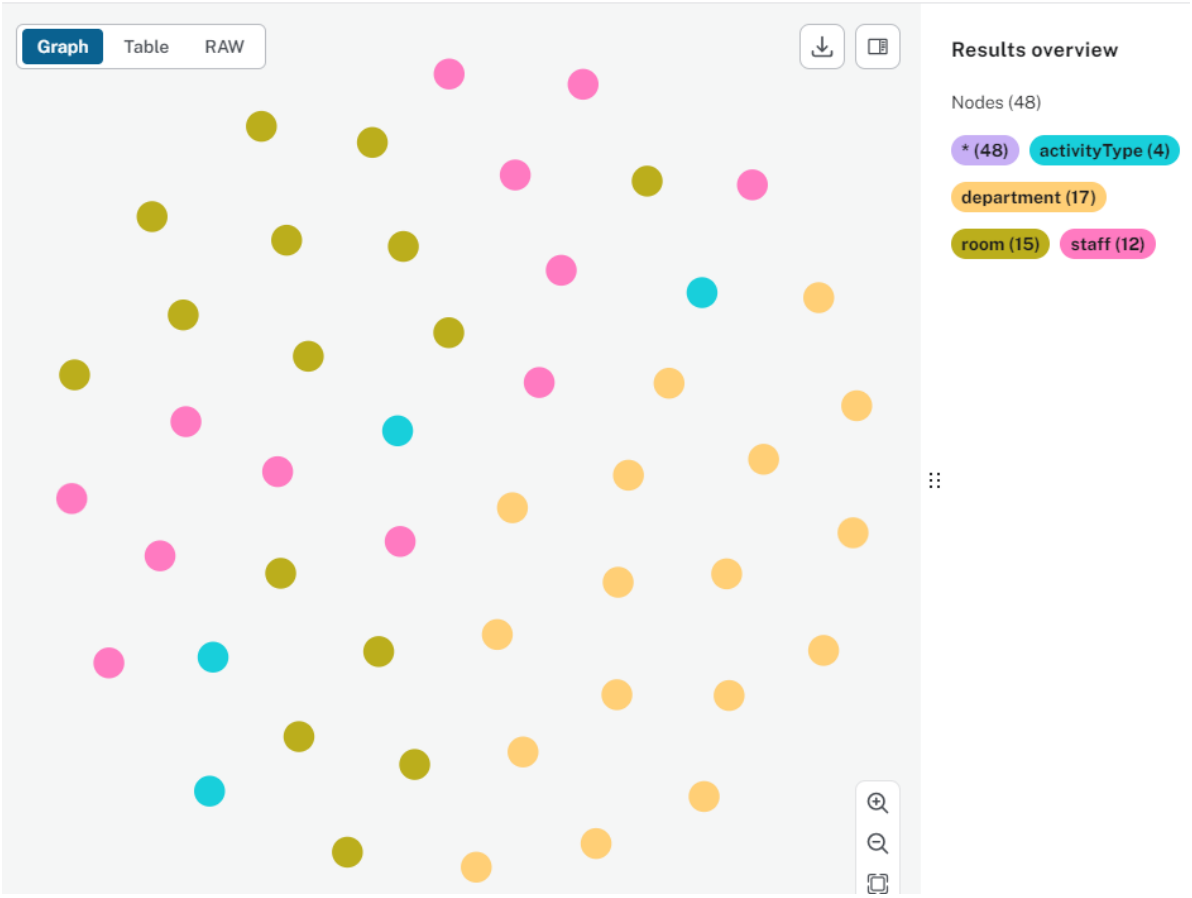


Figure 40.5: Nodes Without Relationships

40.0.6 students without activities

In the timetabling context, we would expect students to be allocated to activities. It turns out that we have 219 students without activities. A bit more investigation indidates that they are all from a particular programme of study run.

```
// Students without Activities
MATCH (s:student)
WHERE NOT (s)-[:ATTENDS]->()
RETURN s
```

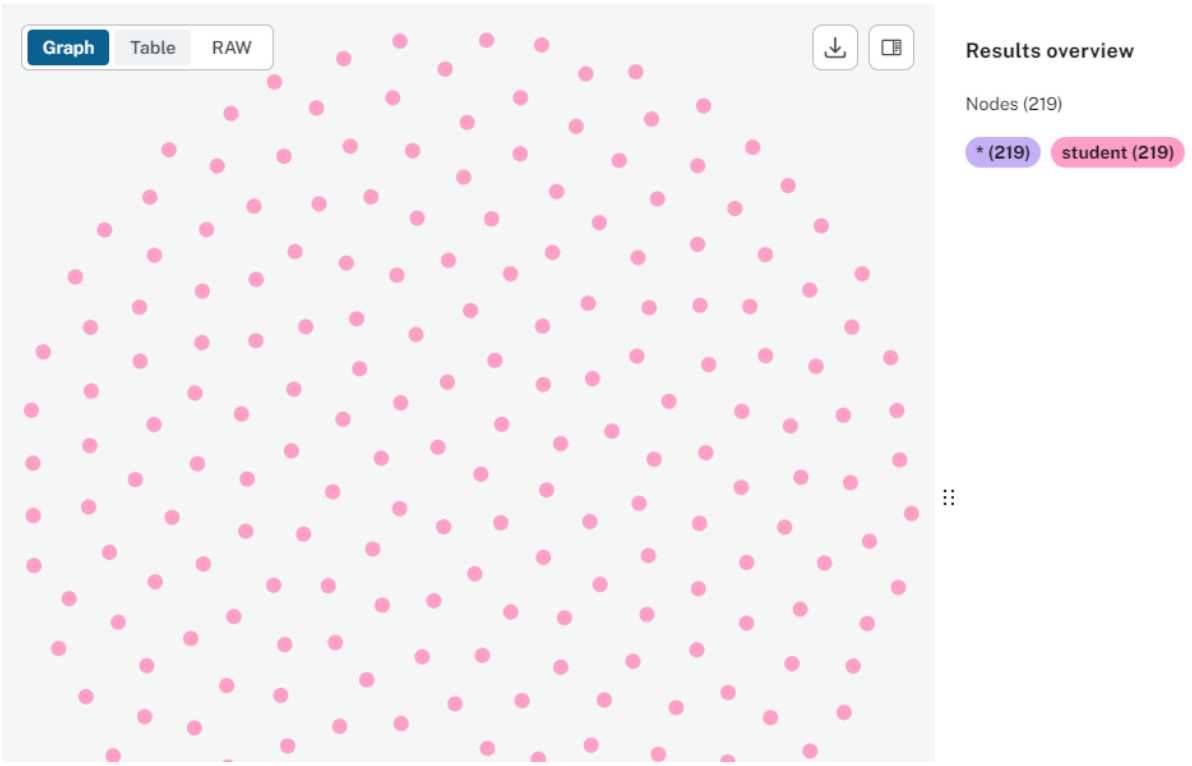


Figure 40.6: Students Without Activities

40.0.7 activityType without activity

Activities can have a `activity type` - the graph model could have `activity type` as a property or as a relationship. The query below finds `activity type`without `activity`. The decision may be to delete these orphaned nodes as they may cause problems with some calculations. They would be created if they become required in the future.

```
// Activities without Rooms

MATCH (at:activityType)
WHERE NOT (at)<-[:HAS_TYPE]-()
RETURN at;
```

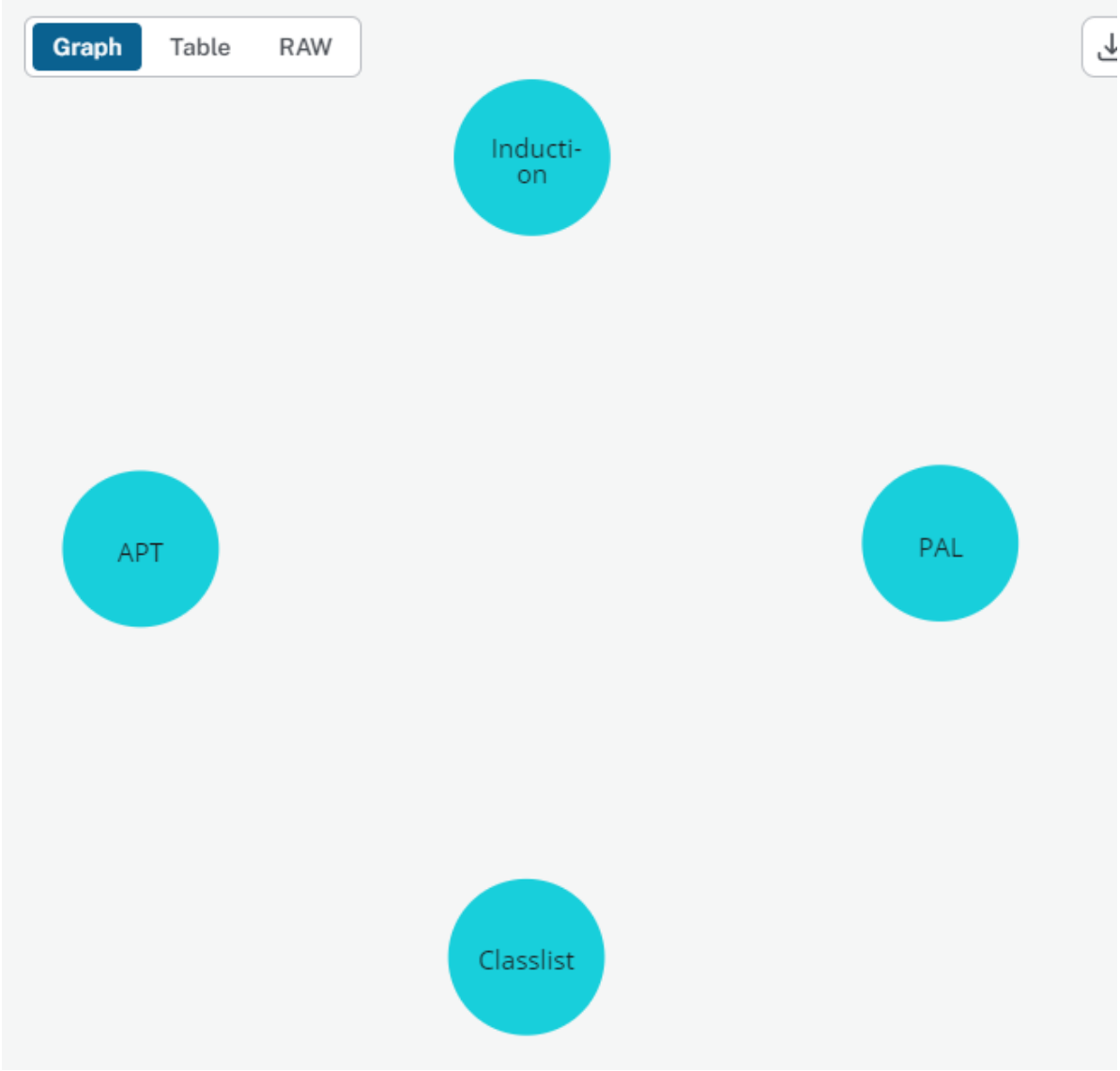


Figure 40.7: ActivityType Without Room

40.0.8 activities without rooms

The graph has over 1500 activity instances without rooms. Most of these will be deliberate - online, virtual sessions - but we may want to query the graph to identify those where a room is expected.

```
// Activities without Rooms

MATCH (a:activity)
WHERE NOT (a)-[]->(:room)
RETURN a;
```

Figure 40.8: Activities Without Rooms

41 Q: Count Queries

This page contains a selection of count queries used to explore the graph database. The queries are designed to provide insights into the data and relationships between nodes. They can be considered starter queries which can be amended depending on the requirements.

41.0.1 Count all nodes - by label

Below are two queries returning the same results - counts of nodes by node label.

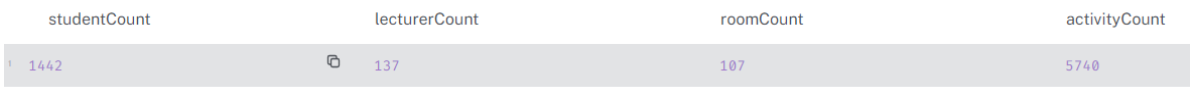
```
// Count of nodes - row per node

UNWIND ["student", "staff", "room", "activity"] AS label
MATCH (n)
WHERE label IN labels(n)
RETURN label, count(n) AS count
```

	label	count
1	"student"	1442
2	"staff"	137
3	"room"	107
4	"activity"	5740

Figure 41.1: Count All Nodes

```
// Count of nodes - single row

MATCH (n:student)
WITH count(n) AS studentCount
MATCH (n:staff)
WITH studentCount, count(n) AS lecturerCount
```

```
MATCH (n:room)
WITH studentCount, lecturerCount, count(n) AS roomCount
MATCH (n:activity)
RETURN studentCount, lecturerCount, roomCount, count(n) AS activityCount
```



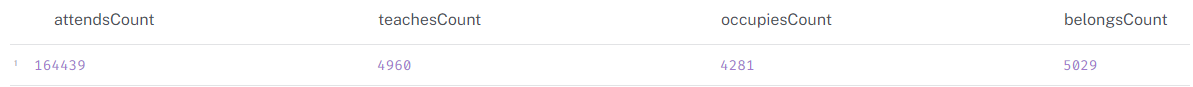
studentCount	lecturerCount	roomCount	activityCount
1442	137	107	5740

Figure 41.2: Count All Nodes

41.0.2 Count all relationships - by type

The query below returns counts of relationships. We can see that there are a significant amount of (student)-[]->(activity) relationships due to how we structured `activity` in the graph - that is, a separate node for each instance.

```
// Count of relationships

MATCH ()-[r:ATTENDS]->()
WITH count(r) AS attendsCount
MATCH ()-[r:TEACHES]->()
WITH attendsCount, count(r) AS teachesCount
MATCH ()-[r:OCCUPIES]->()
WITH attendsCount, teachesCount, count(r) AS occupiesCount
MATCH ()-[r:BELONGS_TO]->()
RETURN attendsCount, teachesCount, occupiesCount, count(r) AS belongsCount
```



attendsCount	teachesCount	occupiesCount	belongsCount
164439	4960	4281	5029

Figure 41.3: Count All Relationships

41.0.3 Activity counts

In this graph, an `activity` is an instance of an activity, that is, a unique combination of `name`, `date`, `start`, `end`, `location`, `staff`. It means *a lot* of activities!

```
// Count of activities

MATCH (a:activity)
RETURN count(a) AS totalActivities;
```

```
// Count of activities on a day

MATCH (a:activity)
WHERE a.actDayName = "Wednesday"
RETURN DISTINCT count(a) AS wednesdayActivities
```

```
MATCH (a:activity)
RETURN DISTINCT a.actDayName AS dayName, count(a) AS activityCount
```

	totalActivities
1	5740

	wednesdayActivities
1	960

	dayName	activityCount
1	"Wednesday"	960
2	"Tuesday"	1446
3	"Friday"	1013
4	"Thursday"	1186
5	"Monday"	1133
6	"Saturday"	1
7	"Sunday"	1

41.0.4 Activity counts by time

The query below connects to the graph via python and returns the result - that is, the number of activities which start at 17:00.

```
from connect_to_neo4j_db import connect_to_neo4j
from neo4j import GraphDatabase

# connect to Neo4j
driver = connect_to_neo4j()

# session
session = driver.session()

# run query
query = """
// Activity count by time (start)

MATCH (a:activity)
WHERE a.actStartTime = localtime("17:00:00")
//AND a.actDayName = "Wednesday"
RETURN count(a) AS activitiesStartingAt5pm
"""
print("Running query...\n")
result = session.run(query)
for record in result:
    print(record)

# close the session and driver
session.close()
driver.close()
```

```
Connecting to Neo4j database....
Connected to Neo4j database successfully! Driver: <neo4j._sync.driver.Neo4jDriver object at (
Running query...

<Record activitiesStartingAt5pm=172>
```

41.0.5 Staff activity count

This query returns the first 5 rows of the query which counts activities by member of staff.

```python
from connect_to_neo4j_db import connect_to_neo4j
from neo4j import GraphDatabase
import pandas as pd

# connect to Neo4j
driver = connect_to_neo4j()

# session
session = driver.session()

# run query
query = """
// Staff activity count

MATCH (st:staff)-[r:TEACHES]->(a:activity)
RETURN st.staffFullName_anon AS staffName, count(a) AS activityCount
ORDER BY activityCount DESC
"""
print("Running query...\n")
result = session.run(query)

# list to hold records
records = []
for record in result:
    records.append(record)

# df of first 5 records
df = pd.DataFrame(records[:5], columns=["staffName", "activityCount"])

# print
print(df)

# close the session and driver
session.close()
driver.close()
```

```
Connecting to Neo4j database....
Connected to Neo4j database successfully! Driver: <neo4j._sync.driver.Neo4jDriver object at 
Running query...

        staffName  activityCount
0  Debbie Nichols            145
```

```
1  Marc Hernandez           127
2    Eileen Allen           126
3    Steven Perez           124
4  Justin Alvarez           112
```

42 R: Hard Constraints

"Hard constraints" in a timetabling context are generally rules or conditions which cannot be violated. Violation would indicate non-viable timetable, e.g. a lecturer being scheduled to teach in two places simultaneously. In reality, hard constraints appear in timetables and are accepted with real-world workarounds.

This page contains cypher queries that can be used to identify where a timetabling hard constraint has been violated.

Example hard constraints include:

- **All Activities Scheduled**: Every lecture, tutorial, lab, etc., must have a designated time and place.
- **No Room Conflicts (aka room clash)**: Two activities cannot be scheduled in the same room at the same time.
- **Room Capacity Sufficient**: The room assigned to an activity must accommodate the expected number of students
- **Person clashes**: People, that is staff and students, cannot be allocated to two or more activities occurring at the same time.
  - **No Staff Conflicts (aka staff clash)**
  - **No Student Conflicts (aka student clash)**
- **Staff Availability Respected**: Activities cannot be scheduled during a staff member's unavailable times (e.g., research days, meetings, unavailability pattern).
- **Curriculum Requirements Met**: Required courses must be offered at times when students can take them

42.0.1 Unscheduled activities

Unscheduled activities can be identified as follows. This query can be tweaked to also search for matches where the property equals '' - that is, a blank.

```
MATCH (a:activity)
WHERE a.actStartDate IS NULL
OR a.actStartTime IS NULL
OR a.actEndTime IS NULL
RETURN a
```

42.0.2 Room clashes

Room or location clashes are where two or more activities are scheduled at the same datetime in the same space and this is not deliberate. These can be identified with the starter query below. The image clearly shows pairs of activities sharing one location. In reality, I suspect that these are deliberate clashes.

```
MATCH (a1:activity)-[r1:OCCUPIES]->(r:room)<-[r2:OCCUPIES]-(a2:activity)
WHERE a1.actStartDate = a2.actStartDate AND a1 <> a2
    AND (
        (a1.actStartTime <= a2.actStartTime AND a1.actEndTime > a2.actStartTime)
        OR
        (a2.actStartTime <= a1.actStartTime AND a2.actEndTime > a1.actStartTime)
    )
RETURN a1, a2, r, r1, r2
```

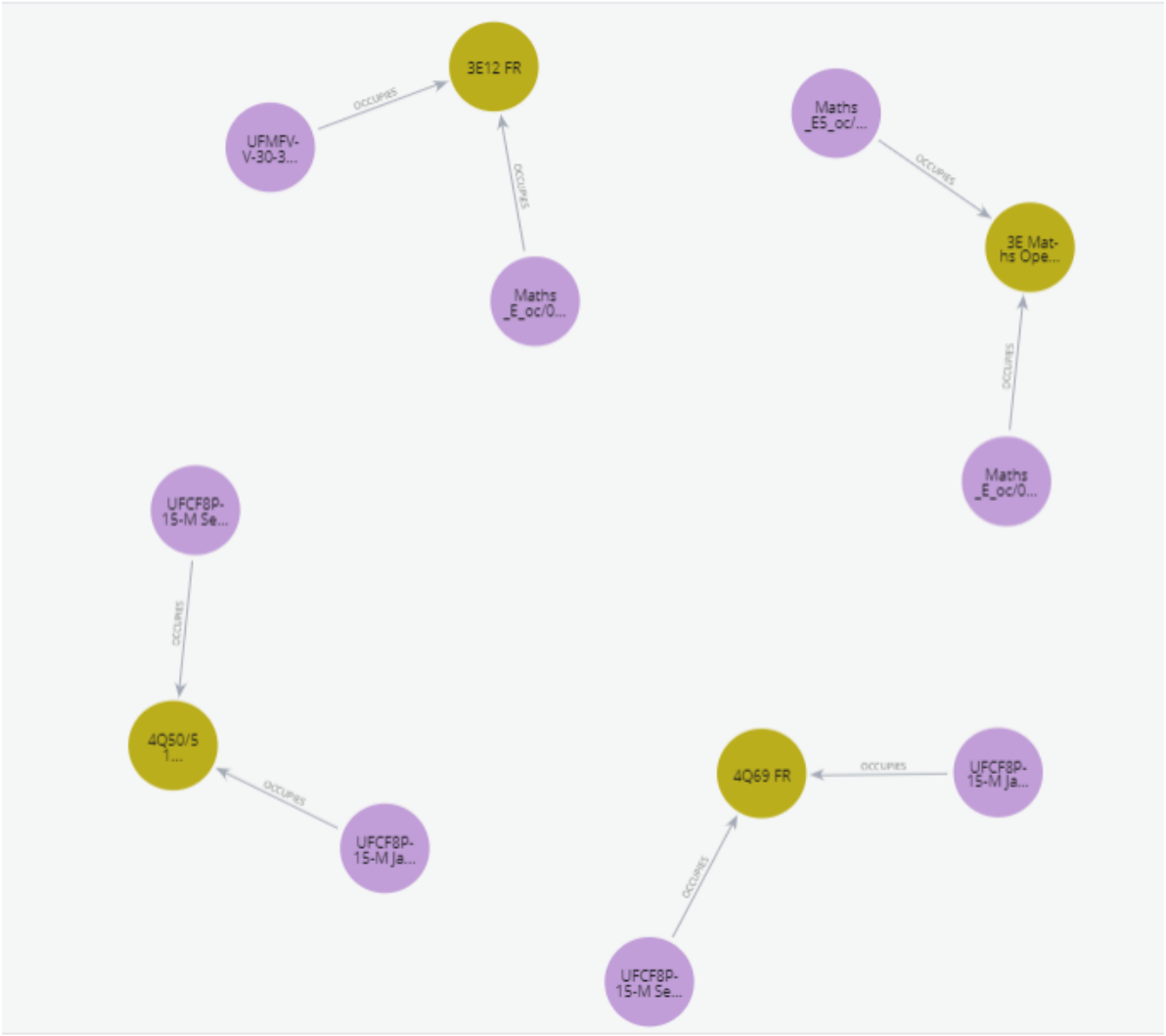


Figure 42.1: Room Clashes

The same results have been returned as a table.

```
from connect_to_neo4j_db import connect_to_neo4j
from neo4j import GraphDatabase
import pandas as pd

# connect to Neo4j
driver = connect_to_neo4j()

# session
```

```
session = driver.session()

# run query
query = """
MATCH (a1:activity)-[r1:OCCUPIES]->(r:room)<-[r2:OCCUPIES]-(a2:activity)
WHERE r.roomName IN ["4Q50/51 FR", "4Q69 FR", "3E Maths Open Zone A", "3E12 FR"]
  AND a1.actStartDate = a2.actStartDate
  AND a1 <> a2
  AND (
        (a1.actStartTime <= a2.actStartTime AND a1.actEndTime > a2.actStartTime)
        OR
        (a2.actStartTime <= a1.actStartTime AND a2.actEndTime > a1.actStartTime)
      )
RETURN a1.actName AS activity1, a2.actName AS activity2,
       r.roomName AS room, a1.actStartDate AS date,
       a1.actStartTime AS activity1_start, a1.actEndTime AS activity1_end,
       a2.actStartTime AS activity2_start, a2.actEndTime AS activity2_end
"""

print("Running query...\n")
result = session.run(query)

# list to hold records
records = []
for record in result:
    records.append(record)

# df
df = pd.DataFrame(records, columns=["activity1", "activity2", "room", "date",
                                    "activity1_start", "activity1_end",
                                    "activity2_start", "activity2_end"])

# print
print(df)

# close session and driver
session.close()
driver.close()
```

```
Connecting to Neo4j database....
Connected to Neo4j database successfully! Driver: <neo4j._sync.driver.Neo4jDriver object at (
Running query...
```

```
                          activity1                          activity2  \
0          UFCF8P-15-M Sep W2_oc/01  UFCF8P-15-M Jan Cont W2_oc jt/01
1  UFCF8P-15-M Jan Cont W2_oc jt/01          UFCF8P-15-M Sep W2_oc/01
2          UFCF8P-15-M Sep W1_oc/01  UFCF8P-15-M Jan Cont W1_oc jt/01
3  UFCF8P-15-M Jan Cont W1_oc jt/01          UFCF8P-15-M Sep W1_oc/01
4                     Maths_E_oc/01                    Maths_E5_oc/01
5                    Maths_E5_oc/01                     Maths_E_oc/01
6                     Maths_E_oc/01              UFMFVV-30-3 DI_oc/01
7              UFMFVV-30-3 DI_oc/01                     Maths_E_oc/01

                   room        date     activity1_start       activity1_end  \
0            4Q50/51 FR  2022-12-13  16:00:00.000000000  17:30:00.000000000
1            4Q50/51 FR  2022-12-13  16:00:00.000000000  17:30:00.000000000
2               4Q69 FR  2022-12-13  14:00:00.000000000  15:30:00.000000000
3               4Q69 FR  2022-12-13  14:00:00.000000000  15:30:00.000000000
4  3E Maths Open Zone A  2022-11-09  09:00:00.000000000  17:00:00.000000000
5  3E Maths Open Zone A  2022-11-09  16:00:00.000000000  17:00:00.000000000
6               3E12 FR  2022-11-08  09:00:00.000000000  17:00:00.000000000
7               3E12 FR  2022-11-08  15:00:00.000000000  16:00:00.000000000

      activity2_start       activity2_end
0  16:00:00.000000000  17:30:00.000000000
1  16:00:00.000000000  17:30:00.000000000
2  14:00:00.000000000  15:30:00.000000000
3  14:00:00.000000000  15:30:00.000000000
4  16:00:00.000000000  17:00:00.000000000
5  09:00:00.000000000  17:00:00.000000000
6  15:00:00.000000000  16:00:00.000000000
7  09:00:00.000000000  17:00:00.000000000
```

42.0.3 Room capacity exceeded

This query identifies activities where the number of students exceeds the room capacity. It includes an optional `WHERE` clause if looking at a specific date range.

```
MATCH (r:room)<-[r1:OCCUPIES]-(a:activity)<-[:ATTENDS]-(s:student)
//WHERE a.Date >= date("2022-01-01") AND a.Date <= date("2022-06-30")
WITH r, a, count(s) as numStudents
WHERE numStudents > r.roomCapacity
RETURN r, a.actStartDate, a.actName AS Activity, r.roomCapacity, numStudents - r.roomCapacity
ORDER BY extraNeeded DESC
```

These are the results in a dataframe:

```
from connect_to_neo4j_db import connect_to_neo4j
from neo4j import GraphDatabase
import pandas as pd

# connect to Neo4j
driver = connect_to_neo4j()

# session
session = driver.session()

# run query
query = """
MATCH (r:room)<-[r1:OCCUPIES]-(a:activity)<-[:ATTENDS]-(s:student)
//WHERE a.Date >= date("2022-01-01") AND a.Date <= date("2022-06-30")
WITH r, a, count(s) as numStudents
WHERE numStudents > r.roomCapacity
RETURN DISTINCT r.roomName, r.roomType,  a.actName AS Activity,
       r.roomCapacity, numStudents - r.roomCapacity AS extraNeeded
ORDER BY extraNeeded DESC
"""

print("Running query...\n")
result = session.run(query)

# list to hold records
records = []
for record in result:
    records.append(record)

# df
df = pd.DataFrame(records, columns=["roomName", "roomType", "Activity",
                                    "roomCapacity", "extraNeeded"])

# print
print("Printing first 5 records...\n")
print(df.head())

# close the session and driver
session.close()
driver.close()
```

```
Connecting to Neo4j database....
Connected to Neo4j database successfully! Driver: <neo4j._sync.driver.Neo4jDriver object at
Running query...

Printing first 5 records...
```

	roomName	roomType	Activity	roomCapacity	extraNeeded
0	3E006 FR	FAC STUD	Maths_E_oc/01	4	88
1	3E007 FR	FAC STUD	Maths_E_oc/01	4	88
2	3E Maths Open Zone B	FAC STUD	Maths_E_oc/01	12	80
3	3E28 FR	TEACHING	Maths_E_oc/01	24	68
4	3E12 FR	PC LAB	Maths_E_oc/01	30	62

42.0.4 Student clashes

This query identifies students who are scheduled to attend two or more activities at the same time. Identifying clashes is a complex undertaking and it is one where the graph structure in terms of nodes, properties and relationships could potentially make a significant different to performance.

The reason for the complexity is that you need to look for overlapping times between two activities for each date, for each student. The query to achieve this and the ensuing calculations will vary significantly depending on the structure and syntax.

Because of this, I explored the student clash scenario in more detail here: Student Clashes

```
MATCH (s:student)-[:ATTENDS]->(a1:activity)
WITH s, a1
MATCH (s)-[:ATTENDS]->(a2:activity)
WHERE a1 <> a2
  AND a1.actStartDate = a2.actStartDate
  AND (a1.actStartTime < a2.actEndTime AND a1.actEndTime > a2.actStartTime)
  AND NOT (a1.actStartTime = a2.actEndTime OR a1.actEndTime = a2.actStartTime)
  AND a1.actName < a2.actName  // Ensure only one direction of the pair is returned
RETURN s.stuFirstName_anon AS Student,
       a1.actStartDate AS ClashDate,
       a1.actName AS Activity1,
       a1.actStartTime + "-" + a1.actEndTime AS Timeslot1,
       a2.actName AS Activity2,
       a2.actStartTime + "-" + a2.actEndTime AS Timeslot2
ORDER BY Student, ClashDate;
```

	Student	ClashDate	Activity1	Timeslot1	Activity2	Timeslot2
1	"Aaron"	2022-11-08	"UFMFLV-30-1 SS1_oc/01"	"10:00:00-11:30:00"	"Maths_E_oc/01"	"09:00:00-17:00:00"
2	"Aaron"	2022-11-08	"Maths_E1_oc/01"	"11:30:00-12:00:00"	"Maths_E_oc/01"	"09:00:00-17:00:00"
3	"Aaron"	2022-11-08	"Maths_E2_oc/01"	"14:00:00-15:00:00"	"Maths_E_oc/01"	"09:00:00-17:00:00"
4	"Aaron"	2022-11-08	"UFMFNV-30-2 DI 1_oc/01"	"13:00:00-14:00:00"	"Maths_E_oc/01"	"09:00:00-17:00:00"
5	"Aaron"	2022-11-08	"UFMFPV-30-2 DI_oc/01"	"15:00:00-16:00:00"	"Maths_E_oc/01"	"09:00:00-17:00:00"

Figure 42.2: Student Clashes

43 S: Student Clashes - Deeper Dive

This page explore different graph data structures and queries for the purposes of identifying student clashes. It illustrates the inherent flexibility of graph databases and that thorough modelling and profiling of the data can lead to more efficient and effective queries.

Use-case is king when it comes to optimised databases and performant queries.

43.1 Scenario

Each model below will use the same basic scenario:

- Two students - Alice and Bob
- Three activities:
  - `ITGD` - Introduction to Graph Databases
  - `Neo4j` - Neo4j for Beginners
  - `TigerDB` - TigerGraph for Data Scientists
  - Each activity has a start and end time
  - Each activity is scheduled for several weeks
- There are deliberate clashes between the activities to illustrate the concept of a student clash

43.2 Model 1 - Activity Occurrence

Each 'occurrence' of an activity is a separate node. This model is simple and easy to understand, but proliferates nodes which lead to inefficient and complex queries.

43.2.1 Create data

```
// Create unique activity nodes (TestActivityModel1)
CREATE (:TestActivityModel1 { actName: "ITGD", date: date("2024-08-06"), startTime: localtime
CREATE (:TestActivityModel1 { actName: "ITGD", date: date("2024-08-13"), startTime: localtime
CREATE (:TestActivityModel1 { actName: "ITGD", date: date("2024-08-20"), startTime: localtime
CREATE (:TestActivityModel1 { actName: "ITGD", date: date("2024-08-27"), startTime: localtime
CREATE (:TestActivityModel1 { actName: "ITGD", date: date("2024-09-03"), startTime: localtime
CREATE (:TestActivityModel1 { actName: "Neo4j", date: date("2024-07-30"), startTime: localtim
CREATE (:TestActivityModel1 { actName: "Neo4j", date: date("2024-08-13"), startTime: localtim
CREATE (:TestActivityModel1 { actName: "Neo4j", date: date("2024-08-27"), startTime: localtim
CREATE (:TestActivityModel1 { actName: "TigerDb", date: date("2024-08-06"), startTime: localt
CREATE (:TestActivityModel1 { actName: "TigerDb", date: date("2024-08-13"), startTime: localt

// Create unique student nodes (TestStudentModel1)
CREATE (:TestStudentModel1 { stuFirstName_anon: "Alice", stuID_anon: "test-student-1" })
CREATE (:TestStudentModel1 { stuFirstName_anon: "Bob", stuID_anon: "test-student-2" });

// Create ATTENDS relationships (one student attends all TigerDb and Neo4j)
MATCH (s:TestStudentModel1 { stuID_anon: "test-student-1" })
MATCH (a:TestActivityModel1) WHERE a.actName IN ["TigerDb", "Neo4j"]
CREATE (s)-[:ATTENDS]->(a);

MATCH (s:TestStudentModel1 { stuID_anon: "test-student-2" })
MATCH (a:TestActivityModel1) WHERE a.actName IN ["ITGD", "Neo4j"]
MERGE (s)-[:ATTENDS]->(a) ;
```

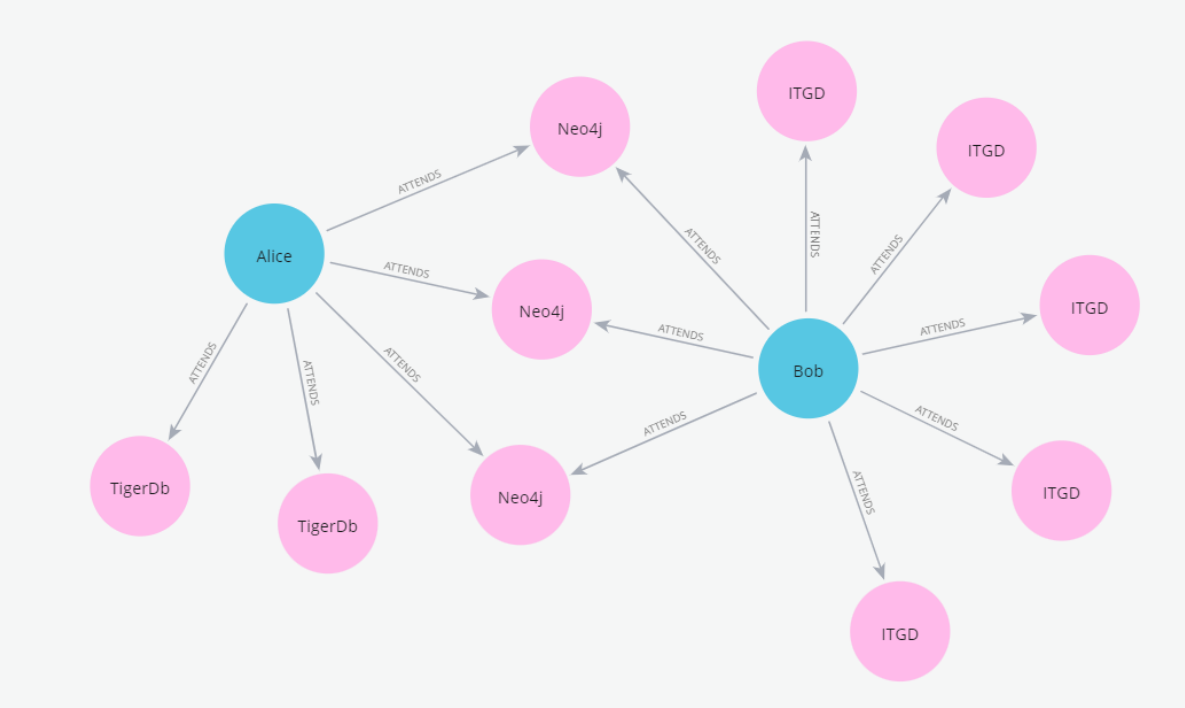


Figure 43.1: Model 1

To identify the clashes, this query can be run:

```
MATCH (s:TestStudentModel1)-[:ATTENDS]->(a1:TestActivityModel1)
WITH s, a1
MATCH (s)-[:ATTENDS]->(a2:TestActivityModel1)
WHERE a1 <> a2
  AND a1.date = a2.date
  AND (a1.startTime < a2.endTime AND a1.endTime > a2.startTime)  //  overlap condition
  AND NOT (a1.startTime = a2.endTime OR a1.endTime = a2.startTime) // xxclude "touching" case
  AND a1.actName < a2.actName  // ensures only one direction of the pair is returned
RETURN s.stuFirstName_anon AS Student,
       a1.date AS ClashDate,
       a1.actName AS Activity1,
       a1.startTime + "-" + a1.endTime AS Timeslot1,
       a2.actName AS Activity2,
       a2.startTime + "-" + a2.endTime AS Timeslot2
ORDER BY Student, ClashDate;
```

Which correctly identifies Bob's clash:

	Student	ClashDate	Activity1	Timeslot1	Activity2	Timeslot2
1	"Bob"	2024-08-13	"ITGD"	"09:00:00-11:00:00"	"Neo4j"	"10:00:00-11:00:00"
2	"Bob"	2024-08-27	"ITGD"	"09:00:00-11:00:00"	"Neo4j"	"10:00:00-11:00:00"

Figure 43.2: Model 1 results

One way of measuring and comparing query performance is to look at the `PROFILE` and `dbhits`. In this instance there are 278 database accesses.

+ProduceResults	0	Student, ClashDate, Activity1, Timeslot1, Activity2, Timeslot2	0	2	0
+Projection	1	(cache[a2.startTime] + $autostring_1) + cache[a2.endTime] AS Timeslot2, (cache[a1.startTime] + $autostring_0) + cache[a1.endTime] AS Timeslot1, cache[a2.actName] AS Activity2, cache[a1.actName] AS Activity1	0	2	0
+Sort	2	Student ASC, ClashDate ASC	0	2	0
+Projection	3	s.stuFirstName_anon AS Student, cache[a1.date] AS ClashDate	0	2	2
+Filter	4	NOT a1 = a2 AND cache[a1.date] = cache[a2.date] AND (cache[a1.endTime] > cache[a2.startTime] AND cache[a1.actName] < cache[a2.actName] AND cache[a1.startTime] < cache[a2.endTime]) AND (NOT cache[a1.startTime] = cache[a2.endTime] AND NOT cache[a1.endTime] = cache[a2.startTime]) AND a1:TestActivityModel1	0	2	91
+Expand(All)	5	(s)-[anon_0:ATTENDS]→(a1)	17	89	102
+CacheProperties	6	cache[a2.startTime], cache[a2.date], cache[a2.endTime], cache[a2.actName]	13	13	52
+Filter	7	a2:TestActivityModel1	13	13	13
+Expand(All)	8	(s)-[anon_1:ATTENDS]→(a2)	13	13	15
+NodeByLabelScan	9	s:TestStudentModel1	10	2	3

Total database accesses: 278, total allocated memory: 2456

Figure 43.3: Model 1 profile

43.3 Model 2 - Date and Time Nodes

Model 2 uses a single node for each activity but has date and time nodes. This model is more complex in that there are more node labels, but can be more efficient for certain queries.

43.3.1 Create data

```
// Create unique time nodes
CREATE (:TestStartTimeNode { time: localtime("09:00:00") })
CREATE (:TestStartTimeNode { time: localtime("10:00:00") })
CREATE (:TestStartTimeNode { time: localtime("11:00:00") })
CREATE (:TestEndTimeNode { time: localtime("11:00:00") })
CREATE (:TestEndTimeNode { time: localtime("12:00:00") })

// Create unique date nodes
```

```
CREATE (:TestDateNode { date: date("2024-07-30") })
CREATE (:TestDateNode { date: date("2024-08-06") })
CREATE (:TestDateNode { date: date("2024-08-13") })
CREATE (:TestDateNode { date: date("2024-08-20") })
CREATE (:TestDateNode { date: date("2024-08-27") })
CREATE (:TestDateNode { date: date("2024-09-03") })

// Create activity nodes
CREATE (:TestActivityModel2 { actName: "ITGD" })
CREATE (:TestActivityModel2 { actName: "Neo4j" })
CREATE (:TestActivityModel2 { actName: "TigerDb" });

// Connect ITGD to dates and times (using MERGE)
MATCH (a:TestActivityModel2 { actName: "ITGD" })
MATCH (d:TestDateNode) WHERE d.date IN [date("2024-08-06"), date("2024-08-13"), date("2024-08
MERGE (a)-[:SCHEDULED_ON]->(d)
WITH a
MATCH (st:TestStartTimeNode { time: localtime("09:00:00") })
MATCH (et:TestEndTimeNode { time: localtime("11:00:00") })
MERGE (a)-[:STARTS_AT]->(st)
MERGE (st)-[:ENDS_AT]->(et);

// Connect Neo4j to dates and times (adjust dates/times and use MERGE)
MATCH (a:TestActivityModel2 { actName: "Neo4j" })
MATCH (d:TestDateNode) WHERE d.date IN [date("2024-07-30"), date("2024-08-13"), date("2024-08
MERGE (a)-[:SCHEDULED_ON]->(d)
WITH a
MATCH (st:TestStartTimeNode { time: localtime("10:00:00") })
MATCH (et:TestEndTimeNode { time: localtime("11:00:00") })
MERGE (a)-[:STARTS_AT]->(st)
MERGE (st)-[:ENDS_AT]->(et);

// Connect TigerDb to dates and times (adjust dates/times and use MERGE)
MATCH (a:TestActivityModel2 { actName: "TigerDb" })
MATCH (d:TestDateNode) WHERE d.date IN [date("2024-08-06"), date("2024-08-13")]
MERGE (a)-[:SCHEDULED_ON]->(d)
WITH a
MATCH (st:TestStartTimeNode { time: localtime("11:00:00") })
MATCH (et:TestEndTimeNode { time: localtime("12:00:00") })
MERGE (a)-[:STARTS_AT]->(st)
MERGE (st)-[:ENDS_AT]->(et);
```

```
// Create Students and ATTENDS relationships (same as Model 1)
CREATE (:TestStudentModel2 { stuFirstName_anon: "Alice", stuID_anon: "test-student-1" })
CREATE (:TestStudentModel2 { stuFirstName_anon: "Bob", stuID_anon: "test-student-2" });

MATCH (s:TestStudentModel2 { stuID_anon: "test-student-1" })
MATCH (a:TestActivityModel2) WHERE a.actName IN ["TigerDb", "Neo4j"]
CREATE (s)-[:ATTENDS]->(a);

MATCH (s:TestStudentModel2 { stuID_anon: "test-student-2" })
MATCH (a:TestActivityModel2) WHERE a.actName IN ["ITGD", "Neo4j"]
MERGE (s)-[:ATTENDS]->(a) ;
```

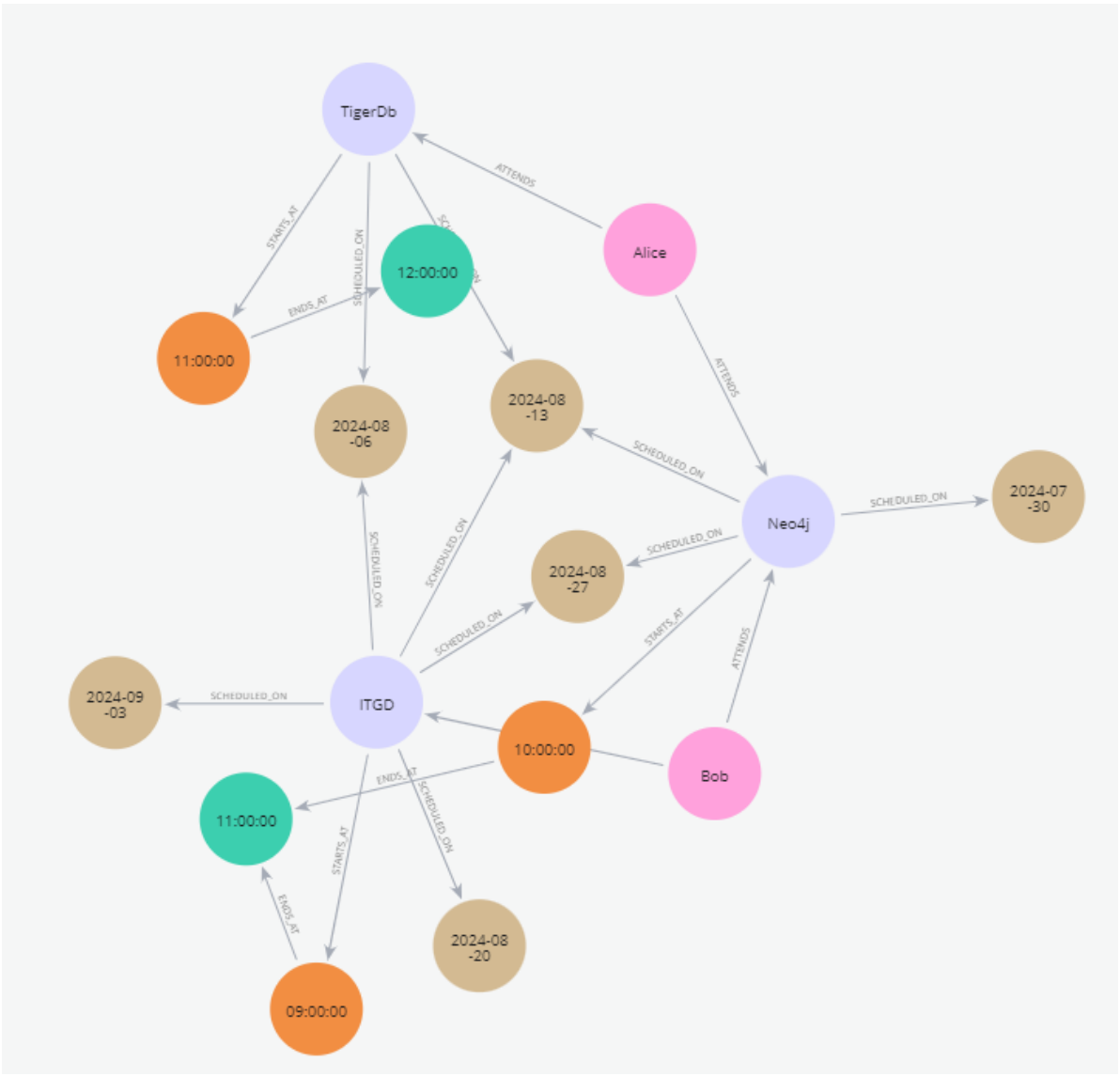


Figure 43.4: Model 2

To identify student clashes, this query can be run.

```
MATCH (s:TestStudentModel2)-[:ATTENDS]->(a1:TestActivityModel2)-[:SCHEDULED_ON]->(d:TestDatel
WITH s, a1, d
MATCH (s)-[:ATTENDS]->(a2:TestActivityModel2)-[:SCHEDULED_ON]->(d) // Same date
MATCH (a1)-[:STARTS_AT]->(st1:TestStartTimeNode)-[:ENDS_AT]->(et1:TestEndTimeNode)
MATCH (a2)-[:STARTS_AT]->(st2:TestStartTimeNode)-[:ENDS_AT]->(et2:TestEndTimeNode)
WHERE a1 <> a2
```

```
  AND (st1.time < et2.time AND et1.time > st2.time)  //  overlap condition
  AND NOT (st1.time = et2.time OR et1.time = st2.time) // xxclude "touching" cases
  AND a1.actName < a2.actName  // ensures only one direction of the pair is returned
RETURN  s.stuFirstName_anon AS Student,
        d.date AS ClashDate,
        a1.actName AS Activity1,
        st1.time + "-" + et1.time AS Timeslot1,
        a2.actName AS Activity2,
        st2.time + "-" + et2.time AS Timeslot2
ORDER BY Student, ClashDate;
```

The results are the same as Model 1, but the profile is different, with fewer `database accesses` (143):

```
| +Filter                    |  7 | cache[et1.time] > cache[st2.time] AND NOT cache[et1.time] = cache[st2.time] AND st2:TestStartTimeNod |            0 |    5 |       21 |
| |                          |    | e                                                                                                      |              |      |          |
| +Expand(All)               |  8 | (a2)-[anon_6:STARTS_AT]→(st2)                                                                          |            0 |    8 |       16 |
| |                          |    |                                                                                                        |              |      |          |
| +Filter                    |  9 | d:TestDateNode                                                                                         |            0 |    8 |        8 |
| |                          |    |                                                                                                        |              |      |          |
| +Expand(All)               | 10 | (a1)-[anon_1:SCHEDULED_ON]→(d)                                                                         |            0 |    8 |       10 |
| |                          |    |                                                                                                        |              |      |          |
| +Filter                    | 11 | NOT a1 = a2 AND cache[a1.actName] < cache[a2.actName] AND a2:TestActivityModel2                        |            0 |    2 |       10 |
| |                          |    |                                                                                                        |              |      |          |
| +Expand(All)               | 12 | (s)-[anon_2:ATTENDS]→(a2)                                                                              |            0 |    8 |       12 |
| |                          |    |                                                                                                        |              |      |          |
| +Filter                    | 13 | s:TestStudentModel2                                                                                    |            0 |    4 |        4 |
| |                          |    |                                                                                                        |              |      |          |
| +Expand(All)               | 14 | (a1)←[anon_0:ATTENDS]-(s)                                                                              |            0 |    4 |        7 |
| |                          |    |                                                                                                        |              |      |          |
| +Filter                    | 15 | a1:TestActivityModel2                                                                                  |            1 |    3 |        3 |
| |                          |    |                                                                                                        |              |      |          |
| +Expand(All)               | 16 | (st1)←[anon_4:STARTS_AT]-(a1)                                                                          |            1 |    3 |        6 |
| |                          |    |                                                                                                        |              |      |          |
| +Filter                    | 17 | (st1:TestStartTimeNode AND et1:TestEndTimeNode)                                                        |            3 |    3 |        6 |
| |                          |    |                                                                                                        |              |      |          |
| +DirectedRelationshipTypeScan | 18 | (st1)-[anon_5:ENDS_AT]→(et1)                                                                        |            3 |    3 |        4 |

Total database accesses: 143, total allocated memory: 2920
```

Figure 43.5: Model 2 profile

43.4 Model 3 - Date Nodes

Model 3 uses a single node for each activity as well as date nodes - start and end times are properties of the activity.

```
// Create unique date nodes
CREATE (:TestDateNodeModel3 { date: date("2024-07-30") })
CREATE (:TestDateNodeModel3 { date: date("2024-08-06") })
CREATE (:TestDateNodeModel3 { date: date("2024-08-13") })
CREATE (:TestDateNodeModel3 { date: date("2024-08-20") })
CREATE (:TestDateNodeModel3 { date: date("2024-08-27") })
CREATE (:TestDateNodeModel3 { date: date("2024-09-03") })
```

```
// Create activity nodes with start/end times as properties
CREATE (:TestActivityModel3 { actName: "ITGD", startTime: localtime("09:00:00"), endTime: loc
CREATE (:TestActivityModel3 { actName: "Neo4j", startTime: localtime("10:00:00"), endTime: lo
CREATE (:TestActivityModel3 { actName: "TigerDb", startTime: localtime("11:00:00"), endTime:

// Connect Activities to Dates (using MERGE)
MATCH (a:TestActivityModel3 { actName: "ITGD" })
MATCH (d:TestDateNodeModel3) WHERE d.date IN [date("2024-08-06"), date("2024-08-13"), date("2
MERGE (a)-[:SCHEDULED_ON]->(d);

MATCH (a:TestActivityModel3 { actName: "Neo4j" })
MATCH (d:TestDateNodeModel3) WHERE d.date IN [date("2024-07-30"), date("2024-08-13"), date("2
MERGE (a)-[:SCHEDULED_ON]->(d);

MATCH (a:TestActivityModel3 { actName: "TigerDb" })
MATCH (d:TestDateNodeModel3) WHERE d.date IN [date("2024-08-06"), date("2024-08-13")]
MERGE (a)-[:SCHEDULED_ON]->(d);

// Create Students and ATTENDS relationships
CREATE (:TestStudentModel3 { stuFirstName_anon: "Alice", stuID_anon: "test-student-1" })
CREATE (:TestStudentModel3 { stuFirstName_anon: "Bob", stuID_anon: "test-student-2" });

MATCH (s:TestStudentModel3 { stuID_anon: "test-student-1" })
MATCH (a:TestActivityModel3) WHERE a.actName IN ["TigerDb", "Neo4j"]
CREATE (s)-[:ATTENDS]->(a) ;

MATCH (s:TestStudentModel3 { stuID_anon: "test-student-2" })
MATCH (a:TestActivityModel3) WHERE a.actName IN ["ITGD", "Neo4j"]
CREATE (s)-[:ATTENDS]->(a) ;
```

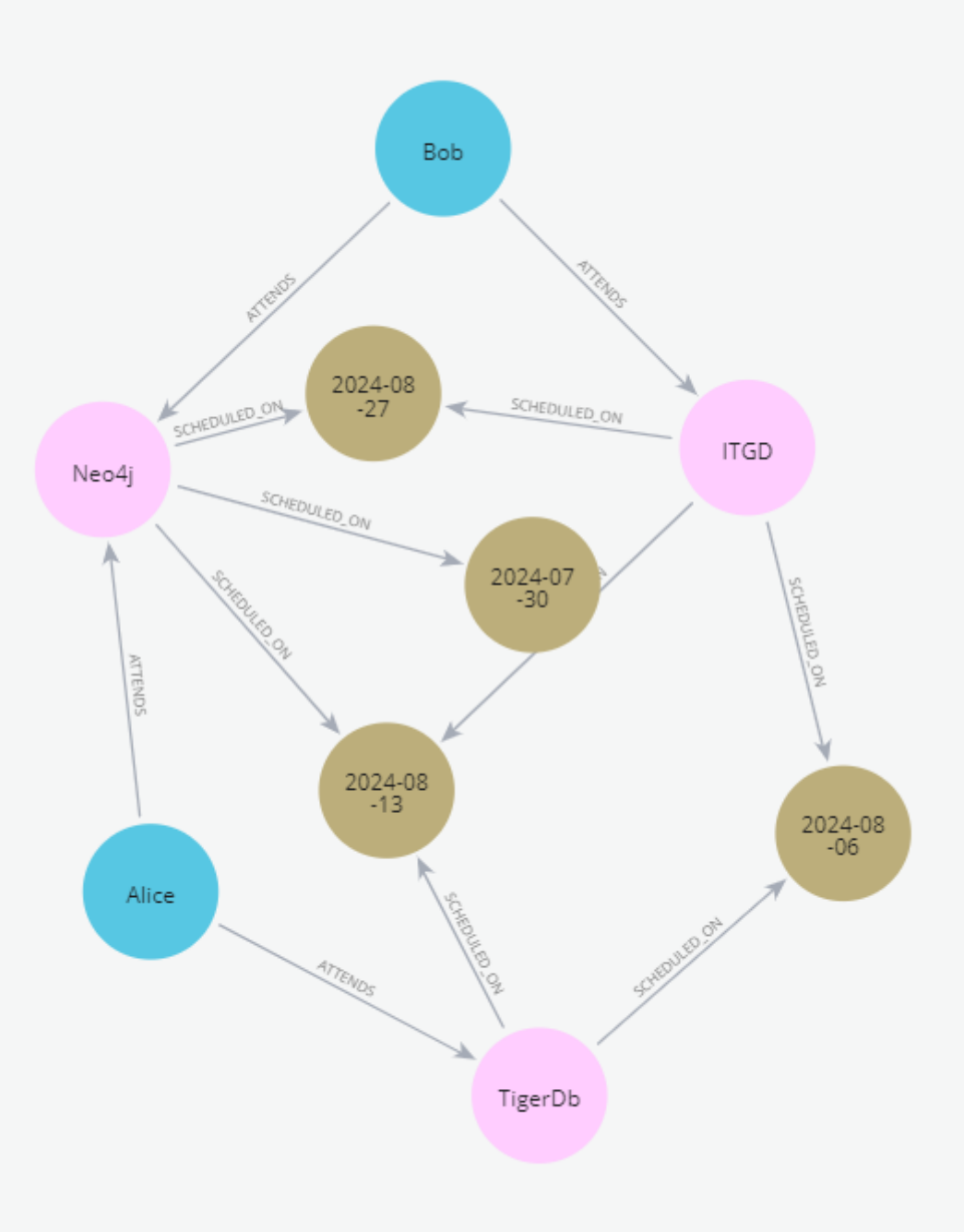
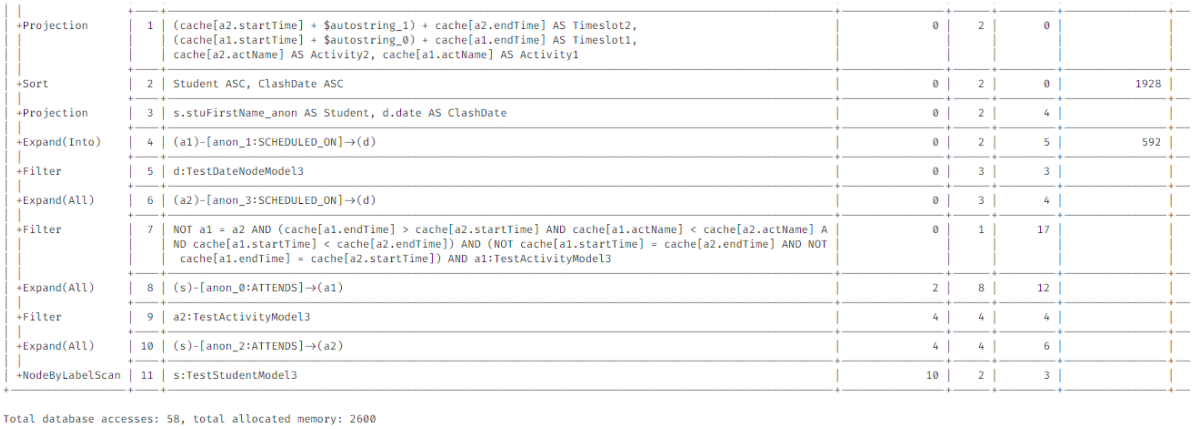


Figure 43.6: Model 3

The results are the same as Model 1 and Model 2, but the profile is different again with even fewer `database accesses` (58):



+Projection	1	(cache[a2.startTime] + $autostring_1) + cache[a2.endTime] AS Timeslot2, (cache[a1.startTime] + $autostring_0) + cache[a1.endTime] AS Timeslot1, cache[a2.actName] AS Activity2, cache[a1.actName] AS Activity1	0	2	0	
+Sort	2	Student ASC, ClashDate ASC	0	2	0	1928
+Projection	3	s.stuFirstName_anon AS Student, d.date AS ClashDate	0	2	4	
+Expand(Into)	4	(a1)-[anon_1:SCHEDULED_ON]→(d)	0	2	5	592
+Filter	5	d:TestDateNodeModel3	0	3	3	
+Expand(All)	6	(a2)-[anon_3:SCHEDULED_ON]→(d)	0	3	4	
+Filter	7	NOT a1 = a2 AND (cache[a1.endTime] > cache[a2.startTime] AND cache[a1.actName] < cache[a2.actName] AND cache[a1.startTime] < cache[a2.endTime]) AND (NOT cache[a1.startTime] = cache[a2.endTime] AND NOT cache[a1.endTime] = cache[a2.startTime]) AND a1:TestActivityModel3	0	1	17	
+Expand(All)	8	(s)-[anon_0:ATTENDS]→(a1)	2	8	12	
+Filter	9	a2:TestActivityModel3	4	4	4	
+Expand(All)	10	(s)-[anon_2:ATTENDS]→(a2)	4	4	6	
+NodeByLabelScan	11	s:TestStudentModel3	10	2	3	

Total database accesses: 58, total allocated memory: 2600

Figure 43.7: Model 3 profile

43.4.1 Model 4 - Times on Relationships

Model 4 uses a single node for each activity and date - start and end times are now properties of the *relationship* between the activity and the date.

```
// Create unique date nodes
MERGE (:TestDateNodeModel4 { date: date("2024-07-30") })
MERGE (:TestDateNodeModel4 { date: date("2024-08-06") })
MERGE (:TestDateNodeModel4 { date: date("2024-08-13") })
MERGE (:TestDateNodeModel4 { date: date("2024-08-20") })
MERGE (:TestDateNodeModel4 { date: date("2024-08-27") })
MERGE (:TestDateNodeModel4 { date: date("2024-09-03") })

// Create activity nodes
MERGE (:TestActivityModel4 { actName: "ITGD" })
MERGE (:TestActivityModel4 { actName: "Neo4j" })
MERGE (:TestActivityModel4 { actName: "TigerDb" });

// Connect ITGD to Dates with START and END relationships
MATCH (a:TestActivityModel4 { actName: "ITGD" })
MATCH (d:TestDateNodeModel4) WHERE d.date IN [date("2024-08-06"), date("2024-08-13"), date("
MERGE (a)-[:STARTS { time: localtime("09:00:00") }]->(d)
MERGE (a)-[:ENDS { time: localtime("11:00:00") }]->(d);

// Connect Neo4j to Dates (adjust dates and times)
MATCH (a:TestActivityModel4 { actName: "Neo4j" })
MATCH (d:TestDateNodeModel4) WHERE d.date IN [date("2024-07-30"), date("2024-08-13"), date("
```

```
MERGE (a)-[:STARTS { time: localtime("10:00:00") }]->(d)
MERGE (a)-[:ENDS { time: localtime("11:00:00") }]->(d);

// Connect TigerDb to Dates (adjust dates and times)
MATCH (a:TestActivityModel4 { actName: "TigerDb" })
MATCH (d:TestDateNodeModel4) WHERE d.date IN [date("2024-08-06"), date("2024-08-13")]
MERGE (a)-[:STARTS { time: localtime("11:00:00") }]->(d)
MERGE (a)-[:ENDS { time: localtime("12:00:00") }]->(d);

// Create Students and ATTENDS relationships
MERGE (:TestStudentModel4 { stuFirstName_anon: "Alice", stuID_anon: "test-student-1" })
MERGE (:TestStudentModel4 { stuFirstName_anon: "Bob", stuID_anon: "test-student-2" });

MATCH (s:TestStudentModel4 { stuID_anon: "test-student-1" })
MATCH (a:TestActivityModel4) WHERE a.actName IN ["TigerDb", "Neo4j"]
MERGE (s)-[:ATTENDS]->(a) ;

MATCH (s:TestStudentModel4 { stuID_anon: "test-student-2" })
MATCH (a:TestActivityModel4) WHERE a.actName IN ["ITGD", "Neo4j"]
MERGE (s)-[:ATTENDS]->(a) ;
```

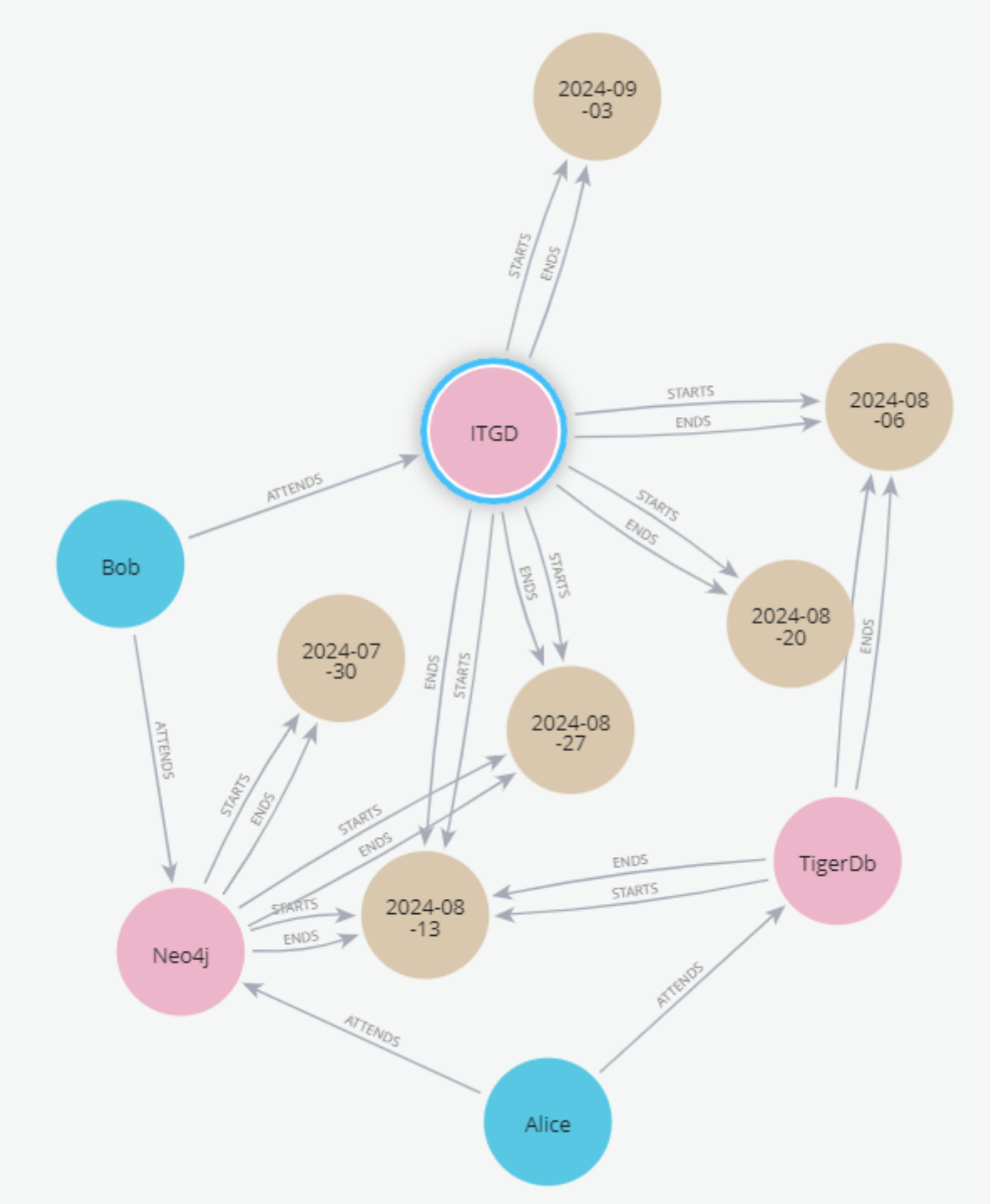


Figure 43.8: Model 4

The results are again the same as Model 1, Model 2, and Model 3, but the profile is different with the most `database accesses` (293):

```
| |                  +----+-----------------------------------------
| +Expand(Into)      |  8 | (a2)-[r2:STARTS]→(d)
| |                  +----+-----------------------------------------
| +Filter            |  9 | a2:TestActivityModel4
| |                  +----+-----------------------------------------
| +Expand(All)       | 10 | (s)-[anon_1:ATTENDS]→(a2)
| |                  +----+-----------------------------------------
| +Projection        | 11 | e1.time AS end1
| |                  +----+-----------------------------------------
| +Expand(Into)      | 12 | (a1)-[e1:ENDS]→(d)
| |                  +----+-----------------------------------------
| +Projection        | 13 | r1.time AS start1
| |                  +----+-----------------------------------------
| +Filter            | 14 | d:TestDateNodeModel4
| |                  +----+-----------------------------------------
| +Expand(All)       | 15 | (a1)-[r1:STARTS]→(d)
| |                  +----+-----------------------------------------
| +Filter            | 16 | a1:TestActivityModel4
| |                  +----+-----------------------------------------
| +Expand(All)       | 17 | (s)-[anon_0:ATTENDS]→(a1)
| |                  +----+-----------------------------------------
| +NodeByLabelScan   | 18 | s:TestStudentModel4
+--------------------+----+-----------------------------------------

Total database accesses: 293, total allocated memory: 4128
```

Figure 43.9: Model 4 profile

43.5 Conclusion

Each model has its own strengths and weaknesses. The choice of model will depend on the specific requirements. The more complex models can be more efficient for certain queries, but can also be more difficult to understand and maintain. The simpler models are easier to understand and maintain, but can be less efficient for certain queries.

Of the four tested, Model 3 was the most efficient in terms of database hits on the *very* small test dataset used. However, this may not be the case with larger datasets. It is important to profile the queries and the data to determine the best model for the specific requirements.

43.6 Delete Data

The cypher below deletes all test data.

43.6.1 Model 1

```
// Delete all TestActivityModel1 nodes
MATCH (a:TestActivityModel1)
DETACH DELETE a;

// Delete all TestStudentModel1 nodes
MATCH (s:TestStudentModel1)
DETACH DELETE s;
```

43.6.2 Model 2

```
// Delete test data for Model 2
MATCH (n)
WHERE n:TestStudentModel2 OR n:TestActivityModel2 OR n:TestDateNode OR n:TestStartTimeNode OR
DETACH DELETE n
```

43.6.3 Model 3

```
// Delete test data for Model 3
MATCH (n)
WHERE n:TestStudentModel3 OR n:TestActivityModel3 OR n:TestDateNodeModel3
DETACH DELETE n
```

43.6.4 Model 4

```
MATCH (n)
WHERE n:TestStudentModel4 OR n:TestActivityModel4 OR n:TestDateNodeModel4
DETACH DELETE n
```

44 T: Soft Constraints

Soft constraints in a timetabling context are (strong) preferences. They should be generally met and only violated when absolutely necessary, although there is an argument for a sliding scale of soft constraint adherence.

For example, a member of staff may be unavailable on Fridays, generally, but at a push can be available. Other examples might include ensuring that students have an opportunity to eat lunch by ensuring at least 30 minutes free time between 12:00-14:00 or minimising travel between activities.

This page contains cypher queries that can be used to identify where a timetabling soft constraint has been violated.

Example soft constraints include:

- **Minimal Idle Time (aka no large gaps):** Minimise gaps in staff and student schedules (within reason).
- **Spread Activities (aka maximum consecutive hours):** Avoid clumping all activities for a student or staff member on one day.
- **Preferred Times:** Consider staff and student preferences for morning, afternoon, or evening classes
- **Travel Time:** Minimise the time students need to travel between consecutive classes (especially on large campuses), e.g. between building blocks or by lat/long
- **Lunch Breaks:** Ensure students have sufficient time for lunch breaks.

44.0.1 Minimal idle time

Identifying time gaps between scheduled activities is very complex and requires several steps, clauses and comprehensions within a single query:

- *grouping and sorting* - activities are grouped by student and date, and sorted within groups to establish the sequence
- *gap calculation* - time difference between the end of one activity and the start of the next is calculated for consecutive pairs of activities within a day
- *filtering and aggregation* - gaps are filtered based on threshold (e.g. 6 hours) and then the maximum gap for each day is identified
- *data restructuring* - output is restructured.

44.0.1.1 Cypher logic for identifying gaps

The below is the logic for identifying gaps between activities, using an example student:

```
MATCH (s:student)-[:ATTENDS]->(a:activity)
WHERE s.stuID_anon = "stu-10085720"
AND a.actStartDate = date("2022-10-03")
WITH s, a
ORDER BY a.actStartTime

// Collecting the start and end times of the activities
WITH s, collect({start: a.actStartTime, end: a.actEndTime}) AS times

// Calculating the gaps in minutes between consecutive activities
WITH s, times,
     [i IN range(0, size(times)-2) |
      duration.between(times[i].end, times[i+1].start).minutes / 60.0] AS gaps

// Finding the maximum gap
RETURN s.stuID_anon AS student, times, gaps, reduce(maxGap = 0.0, gap IN gaps | CASE WHEN ga
```

student	times	gaps	maxGap

"stu-10085720"	[{
	start:08:30:00,
	end:09:00:00
	}, {
	start:11:00:00,
	end:12:30:00
	}, {
	start:16:00:00,
	end:17:00:00
	}, {
	start:17:00:00,
	end:18:30:00
	}]
		[2.0, 3.5, 0.0]	3.5

Figure 44.1: Identifying blocks and time gaps

44.0.1.2 Python code on graph

The below code cell returns the first 5 rows where a 6-hour maximum gap has been violated.

```
from connect_to_neo4j_db import connect_to_neo4j
from neo4j import GraphDatabase
import pandas as pd
```

```
# connect to Neo4j
driver = connect_to_neo4j()

# session
session = driver.session()

# run query (modified RETURN clause)
query = """
// students with gaps between activities
MATCH (s:student)-[:ATTENDS]->(a:activity)
WITH s, a
ORDER BY s.stuFirstName_anon, a.actStartDate, a.actStartTime
// Group activities by student and date
WITH s, a.actStartDate AS date, collect({start: a.actStartTime, end: a.actEndTime, activity:
// calculating the gaps in hours between consecutive activities
WITH s, date, times,
     [i IN range(0, size(times)-2) |
      {gap: duration.between(times[i].end, times[i+1].start).minutes / 60.0,
       firstActivity: times[i].activity,
       secondActivity: times[i+1].activity}] AS gaps
// filtering gaps based on a threshold of 6 hours
WITH s, date, gaps
WHERE any(gapRecord IN gaps WHERE gapRecord.gap > 6.0)
// Finding the maximum gap that exceeds the threshold
WITH s, date, reduce(maxGap = {gap: 0.0, firstActivity: null, secondActivity: null}, gapReco
    CASE WHEN gapRecord.gap > maxGap.gap THEN gapRecord ELSE maxGap END) AS maxGapRecord
// group by student to remove duplications
WITH s.stuID_anon AS student,
     collect({date: date,
              activity1: maxGapRecord.firstActivity.actName,
              activity1_time: maxGapRecord.firstActivity.actStartTime + "-" + maxGapRecord.fi
              activity2: maxGapRecord.secondActivity.actName,
              activity2_time: maxGapRecord.secondActivity.actStartTime + "-" + maxGapRecord.s
              maxGapInHours: maxGapRecord.gap}) AS gapRecords
// Unwind the collected records
UNWIND gapRecords AS record
// Returning the result
RETURN student,
       record.date AS date,
       record.activity1 AS activity1,
       record.activity1_time AS activity1_time,
       record.activity2 AS activity2,
```

```
       record.activity2_time AS activity2_time,
       record.maxGapInHours AS maxGapInHours
ORDER BY student, date
"""

print("Running query...\n")
result = session.run(query)

#  list to hold records
records = []
for record in result:
    records.append(record)

# df
df = pd.DataFrame(records, columns=["student", "date", "activity1", "activity1_time",
                                    "activity2", "activity2_time", "maxGapInHours"])

# print
print(df.head(5))

# close session and driver
session.close()
driver.close()
```

```
Connecting to Neo4j database....
Connected to Neo4j database successfully! Driver: <neo4j._sync.driver.Neo4jDriver object at
Running query...

        student        date                  activity1     activity1_time  \
0  stu-10270089  2023-01-30  UFCFGS-15-1 L_oc/01 <29>  09:00:00-10:00:00
1  stu-10270089  2023-02-06  UFCFGS-15-1 L_oc/01 <30>  09:00:00-10:00:00
2  stu-10270089  2023-02-13  UFCFGS-15-1 L_oc/01 <31>  09:00:00-10:00:00
3  stu-10270089  2023-02-20  UFCFGS-15-1 L_oc/01 <32>  09:00:00-10:00:00
4  stu-10270089  2023-02-27  UFCFGS-15-1 L_oc/01 <33>  09:00:00-10:00:00

                   activity2     activity2_time  maxGapInHours
0  UFCFES-30-1 L2_oc/01 <29>  17:30:00-19:00:00            7.5
1  UFCFES-30-1 L2_oc/01 <30>  17:30:00-19:00:00            7.5
2  UFCFES-30-1 L2_oc/01 <31>  17:30:00-19:00:00            7.5
3  UFCFES-30-1 L2_oc/01 <32>  17:30:00-19:00:00            7.5
4  UFCFES-30-1 L2_oc/01 <33>  17:30:00-19:00:00            7.5
```

44.0.1.3 Python to return total hours and block hours

Alternatively, we can amend the query to return, for each date and student combination, their total scheduled hours, maximum consecutive block hours and the number of activities within the continuous block.

```
from connect_to_neo4j_db import connect_to_neo4j
from neo4j import GraphDatabase
import pandas as pd

# connect to Neo4j
driver = connect_to_neo4j()

# session
session = driver.session()

# run query (modified RETURN clause)
query = """
// calculates - total hours, max block hours, max block activities per day
// to be used for max block hours and max block activities per day
// logic - example

// matches specified student and all attended activities
MATCH (s:student {stuID_anon:"stu-10085720"})-[:ATTENDS]->(a:activity)

// sorts activities by start date and then by start time within each date
WITH s, a ORDER BY a.actStartDate, a.actStartTime

// calculates total hours spent on activities for each date
WITH s, a.actStartDate AS date,
     SUM(a.actDurationInMinutes) / 60.0 AS totalHours,
     // groups activities into blocks based on time overlaps
     REDUCE(
        blockInfo = [],
        activity IN COLLECT(a)
        | CASE
            WHEN blockInfo = [] THEN [[activity]]
            ELSE CASE
                   WHEN head(last(blockInfo)).actEndTime >= activity.actStartTime
                     THEN blockInfo[..-1] + [last(blockInfo) + activity]
                   ELSE blockInfo + [[activity]]
                 END
          END
```

```
     ) AS blocks

// unwinds the list of blocks, processing each block individually
UNWIND blocks AS block

// calculates the total duration in hours for each block
WITH s, date, totalHours, blocks,
     REDUCE(blockHours = 0.0, activity IN block | blockHours + activity.actDurationInMinutes)
     SIZE(block) AS blockActivities

// returns aggregated results
RETURN s.stuFullName_anon AS student, date, totalHours,
       MAX(blockHours) AS blockHours,
       MAX(blockActivities) AS blockActivities
ORDER BY date;
"""

print("Running query...\n")
result = session.run(query)

#  list to hold records
records = []
for record in result:
    records.append(record)

# df
df = pd.DataFrame(records, columns=["student", "date", "totalHours", "blockHours", "blockActi

# print
print(df.head(5))

# close session and driver
session.close()
driver.close()
```

```
Connecting to Neo4j database....
Connected to Neo4j database successfully! Driver: <neo4j._sync.driver.Neo4jDriver object at 0
Running query...

       student        date  totalHours  blockHours  blockActivities
0  Aaron Evans  2022-09-22         1.0         1.0                1
1  Aaron Evans  2022-09-23         2.0         2.0                1
```

```
2  Aaron Evans  2022-09-27         2.0         2.0               2
3  Aaron Evans  2022-09-30         4.0         2.0               2
4  Aaron Evans  2022-10-03         4.5         2.5               2
```

44.0.1.4 Example Use case - identifying students with 5+ hours in a single block

This query returns the first five rows where a student has more than 5 consecutive scheduled hours on a date.

```
from connect_to_neo4j_db import connect_to_neo4j
from neo4j import GraphDatabase
import pandas as pd

# connect to Neo4j
driver = connect_to_neo4j()

# session
session = driver.session()

# run query (modified RETURN clause)
query = """
MATCH (s:student)-[:ATTENDS]->(a:activity)<-[:TEACHES]-(:staff) // Filter for teaching activi
WITH s, a ORDER BY a.actStartDate, a.actStartTime
WITH s, a.actStartDate AS date,
     SUM(a.actDurationInMinutes) / 60.0 AS totalHours,
     REDUCE(
         blockInfo = [],
         activity IN COLLECT(a)
         | CASE
             WHEN blockInfo = [] THEN [[activity]]
             ELSE CASE
                     WHEN head(last(blockInfo)).actEndTime >= activity.actStartTime
                         THEN blockInfo[..-1] + [last(blockInfo) + activity]
                     ELSE blockInfo + [[activity]]
                 END
         END
     ) AS blocks
UNWIND blocks AS block
WITH s, date, totalHours, blocks,
     REDUCE(blockHours = 0.0, activity IN block | blockHours + activity.actDurationInMinutes)
     SIZE(block) AS blockActivities
WHERE blockHours > 5 // Filter for blocks with more than 5 hours
```

```
RETURN s.stuFullName_anon AS student, date, totalHours,
       blockHours,
       blockActivities
ORDER BY date;
"""

print("Running query...\n")
result = session.run(query)

#  list to hold records
records = []
for record in result:
    records.append(record)

# df
df = pd.DataFrame(records, columns=["student", "date", "totalHours", "blockHours", "blockActi

# print
print(df.head(5))

# close session and driver
session.close()
driver.close()
```

```
Connecting to Neo4j database....
Connected to Neo4j database successfully! Driver: <neo4j._sync.driver.Neo4jDriver object at
Running query...

             student        date  totalHours  blockHours  blockActivities
0        Jacob Jones  2022-07-19         5.5         5.5                2
1      Rachael Moore  2022-07-19         5.5         5.5                2
2        Kayla Sharp  2022-07-19         5.5         5.5                2
3         David Rose  2022-07-19         5.5         5.5                2
4  Francisco Holland  2022-07-19         5.5         5.5                2
```

44.0.1.5 Total hours per day

In contrast, calculating simple total hours per day is achieved by:

```
MATCH (s:student )-[:ATTENDS]->(a:activity)
WITH s, a.actStartDate AS Date, SUM(a.actDurationInMinutes) / 60.0 AS totalHours
```

```
RETURN s.stuFullName_anon AS Student, Date, totalHours
ORDER BY Date;
```

	Student	Date	totalHours
1	"Jose Ferrell"	2022-07-18	4.0
2	"James Moore"	2022-07-18	4.0
3	"Faith Bell"	2022-07-18	4.0
4	"Ashley Hughes"	2022-07-18	4.0
5	"Kayla Lee"	2022-07-18	4.0

Figure 44.2: Total hours per day

44.0.1.6 Longest consecutive block of activities per day

We can use the earlier cypher logic to identify the longest consecutive block of activities for a student, or student on a day, etc.

```
MATCH (s:student {stuFullName_anon: "Susan Lopez"})-[:ATTENDS]->(a:activity {actStartDate: da
WITH s, a
ORDER BY a.actStartTime
WITH s, COLLECT(a) AS activities
WITH s, activities,
     REDUCE(
       state = {currentBlock: {duration: 0, start: null, end: null}, longestBlock: {duration
       activity IN activities |
         CASE
           WHEN state.currentBlock.end IS NULL OR
                activity.actStartTime > state.currentBlock.end
           THEN {
             currentBlock: {
               duration: activity.actDurationInMinutes,
               start: activity.actStartTime,
               end: activity.actEndTime
             },
             longestBlock:
               CASE
                 WHEN activity.actDurationInMinutes > state.longestBlock.duration
                 THEN {
                   duration: activity.actDurationInMinutes,
```

```
                  start: activity.actStartTime,
                  end: activity.actEndTime
                }
                ELSE state.longestBlock
              END
          }
          ELSE {
            currentBlock: {
              duration: (activity.actEndTime.hour * 60 + activity.actEndTime.minute) -
                        (state.currentBlock.start.hour * 60 + state.currentBlock.start.minut
              start: state.currentBlock.start,
              end: activity.actEndTime
            },
            longestBlock:
              CASE
                WHEN ((activity.actEndTime.hour * 60 + activity.actEndTime.minute) -
                      (state.currentBlock.start.hour * 60 + state.currentBlock.start.minute)
                THEN {
                  duration: (activity.actEndTime.hour * 60 + activity.actEndTime.minute) -
                            (state.currentBlock.start.hour * 60 + state.currentBlock.start.m
                  start: state.currentBlock.start,
                  end: activity.actEndTime
                }
                ELSE state.longestBlock
              END
          }
        END
    ) AS finalState
RETURN
  s.stuFullName_anon AS stuName,
  activities[0].actStartDate AS date,
  finalState.longestBlock.duration AS longestConsecutiveBlockDuration,
  finalState.longestBlock.start AS blockStartTime,
  finalState.longestBlock.end AS blockEndTime
```

44.0.1.7 Example - longest consecutive block for 'Susan Lopez' on 2022-09-27

The below finds the longest consecutive block in a day for a student:

```
from connect_to_neo4j_db import connect_to_neo4j
from neo4j import GraphDatabase
```

```
import pandas as pd

# connect to Neo4j
driver = connect_to_neo4j()

# session
session = driver.session()

# run query (modified RETURN clause)
query = """
MATCH (s:student {stuFullName_anon: "Susan Lopez"})-[:ATTENDS]->(a:activity{actStartDate: dat
WITH s, a
ORDER BY a.actStartTime
WITH s, COLLECT(a) AS activities
WITH s, activities,
   REDUCE(
     state = {currentBlock: {duration: 0, start: null, end: null}, longestBlock: {duration: 0
     activity IN activities |
       CASE
         WHEN state.currentBlock.end IS NULL OR
              activity.actStartTime > state.currentBlock.end
         THEN {
           currentBlock: {
             duration: activity.actDurationInMinutes,
             start: activity.actStartTime,
             end: activity.actEndTime
           },
           longestBlock:
             CASE
               WHEN activity.actDurationInMinutes > state.longestBlock.duration
               THEN {
                 duration: activity.actDurationInMinutes,
                 start: activity.actStartTime,
                 end: activity.actEndTime
               }
               ELSE state.longestBlock
             END
         }
         ELSE {
           currentBlock: {
             duration: (activity.actEndTime.hour * 60 + activity.actEndTime.minute) -
                       (state.currentBlock.start.hour * 60 + state.currentBlock.start.minute)
```

```
            start: state.currentBlock.start,
            end: activity.actEndTime
          },
          longestBlock:
            CASE
              WHEN ((activity.actEndTime.hour * 60 + activity.actEndTime.minute) -
                    (state.currentBlock.start.hour * 60 + state.currentBlock.start.minute))
              THEN {
                duration: (activity.actEndTime.hour * 60 + activity.actEndTime.minute) -
                          (state.currentBlock.start.hour * 60 + state.currentBlock.start.mi
                start: state.currentBlock.start,
                end: activity.actEndTime
              }
              ELSE state.longestBlock
            END
        }
      END
  ) AS finalState
RETURN
s.stuFullName_anon AS student,
activities[0].actStartDate AS date,
finalState.longestBlock.duration AS longestConsecutiveBlockDuration,
finalState.longestBlock.start AS blockStartTime,
finalState.longestBlock.end AS blockEndTime
"""

print("Running query...\n")
result = session.run(query)

#  list to hold records
records = []
for record in result:
  records.append(record)

# df
df = pd.DataFrame(records, columns=["student", "date", "longestConsecutiveBlockDuration", "bl

# print
print(df.head(5))

# close session and driver
session.close()
```

```
driver.close()
```

```
Connecting to Neo4j database....
Connected to Neo4j database successfully! Driver: <neo4j._sync.driver.Neo4jDriver object at
Running query...

       student        date  longestConsecutiveBlockDuration  \
0  Susan Lopez  2022-09-27                               300

       blockStartTime        blockEndTime
0  13:00:00.000000000  18:00:00.000000000
```

44.0.2 Max hours in a day

This query calculates the total scheduled hours for each student on a day and returns the results ordered by date. This example filters for students who have more than 7 hours of activities in a day.

```
// sum of activity durations
// does not account for simultaneous activities (clashes) - so could be inflated, e.g. 12.5 h

MATCH (s:student)-[:ATTENDS]->(a:activity)
WITH s, a.actStartDate AS Date, SUM(a.actDurationInMinutes) / 60.0 AS totalHours
WHERE totalHours > 7 // Set  maximum here
RETURN s.stuFullName_anon AS Student, Date, totalHours
ORDER BY Date;
```

	Student	Date	totalHours
1	"Benjamin Macdonald"	2023-05-19	8.0
2	"David Mueller"	2023-05-19	8.0
3	"Dustin Oconnor"	2023-05-19	8.0
4	"Brian Castro"	2023-05-19	8.0
5	"Anthony Harrison"	2023-05-19	8.0

Figure 44.3: Max hours in a day

44.0.3 Travel time between activities

This query calculates the travel time between consecutive activities for a student on a specific date. It uses the lat/long coordinates of the buildings to calculate the distance and time taken to travel between them and a default walking speed of 1.4 m/s to calculate time.

This is a simple example and does not account for factors like traffic, walking speed, or other modes of transport.

```
// Calculate travel time between consecutive activities for a student on a specific date
MATCH (s:student {stuFullName_anon: "David Johnson"})-[:ATTENDS]->(a1:activity)-[:OCCUPIES]->
      (s)-[:ATTENDS]->(a2:activity)-[:OCCUPIES]->(r2:room)
WHERE a1.actEndTime = a2.actStartTime AND a1.actStartDate = a2.actStartDate AND a1 <> a2 AND
      a1.actStartDate IN [date("2023-01-11"), date("2022-09-27"), date("2023-03-14")]
RETURN DISTINCT
    s.stuFullName_anon,
    a1.actName AS act1, a1.actStartDate AS date, a1.actStartTime+"-"+a1.actEndTime AS act1Ti
    point.distance(r1.location, r2.location) AS distance,
    round(point.distance(r1.location, r2.location) / 1.4) AS walkingTimeSeconds // Calculate
```

	s.stuFullName_	act1	date	act1Times	act2Times	act2	distance	walkingTimeSec
1	"David Johnson"	"UFCEM1-60-MSB L_oc/01"	2022-09-27	"10:00:00-11:00:00"	"11:00:00-12:00:00"	"UFCEM1-60-MSB GT_oc/01"	0.0	0.0
2	"David Johnson"	"UFCEP1-30-MCP_oc/01"	2023-01-11	"14:00:00-17:00:00"	"17:00:00-19:00:00"	"UFMFHR-15-MTB1 CP_oc/01"	26.878321926	19.0
3	"David Johnson"	"UFCEM1-60-MW_oc/01"	2023-03-14	"09:00:00-12:00:00"	"12:00:00-14:00:00"	"UFCEN1-15-MW_oc Wk 34/01"	143.770144571	103.0

Figure 44.4: Travel time between activities

44.0.4 Lunch breaks

It might be expected that a student (or staff) has a lunch break. The cypher below calculates free and booked tie within a window, in this case 12:00 and 14:00. It can be used to find students who do not have a lunch break or count the number of days that a student does not have a lunch break.

```
MATCH (s:student)-[:ATTENDS]->(a:activity)
WITH s, a
ORDER BY a.actStartDate, a.actStartTime
WITH s, COLLECT(a) AS activities
```

```
UNWIND activities AS activity
WITH s.stuFullName_anon AS Student, activity.actStartDate AS Date, time(activity.actStartTime
WITH Student, Date, BreakWindow_12_14, COLLECT([StartTime, EndTime]) AS Activities
UNWIND Activities AS activity
WITH Student, Date, BreakWindow_12_14, activity[0] AS StartTime, activity[1] AS EndTime
WITH Student, Date, BreakWindow_12_14,
     CASE
       WHEN StartTime >= time('14:00') OR EndTime <= time('12:00') THEN 0
       WHEN StartTime < time('12:00') AND EndTime > time('14:00') THEN BreakWindow_12_14
       WHEN StartTime >= time('12:00') AND StartTime < time('14:00') THEN duration.between(St
       WHEN EndTime > time('12:00') AND EndTime <= time('14:00') THEN duration.between(time(
     END AS BookedDurationMinutes
RETURN Student, Date, BreakWindow_12_14, BreakWindow_12_14 - SUM(BookedDurationMinutes) AS Fr
ORDER BY Date
```

	Student	Date	BreakWindow_12_14	FreeTimeMinutes	BookedTimeMinutes
1	"Jose Ferrell"	2022-07-18	120	30	90
2	"James Moore"	2022-07-18	120	30	90
3	"Faith Bell"	2022-07-18	120	30	90
4	"Ashley Hughes"	2022-07-18	120	30	90
5	"Kayla Lee"	2022-07-18	120	30	90
6	"Michael Johnson"	2022-07-18	120	-60	180

Figure 44.5: Identifying lunch breaks

Interestingly, `Michael Johnson` has a negative lunch break! A quick look showed that there are actually two `Michael Johnsons` attending this class and they both have 30 minutes free time in the 2-hour lunch break window.

Because the query was written using student name, it is incorrectly aggregating the two students into one person as follows:

$$\text{2 hour lunch window} - (\text{1.5 hours/class} \times \text{2 students}) = -\text{1 hour}$$

To remedy this, the query can be updated to use student ID or a different unique identifier. I would also like to update the anonymisation function in the ETL so that it does not duplicate names in the output.

45 U: Rooms and Spaces

The timetabling database contains some information about rooms and spaces on campus, but the master data systems contain much more. I have added a few rooms and properties from the master source, as an experiment.

45.1 Room Geo-location

45.1.1 Import locations from file

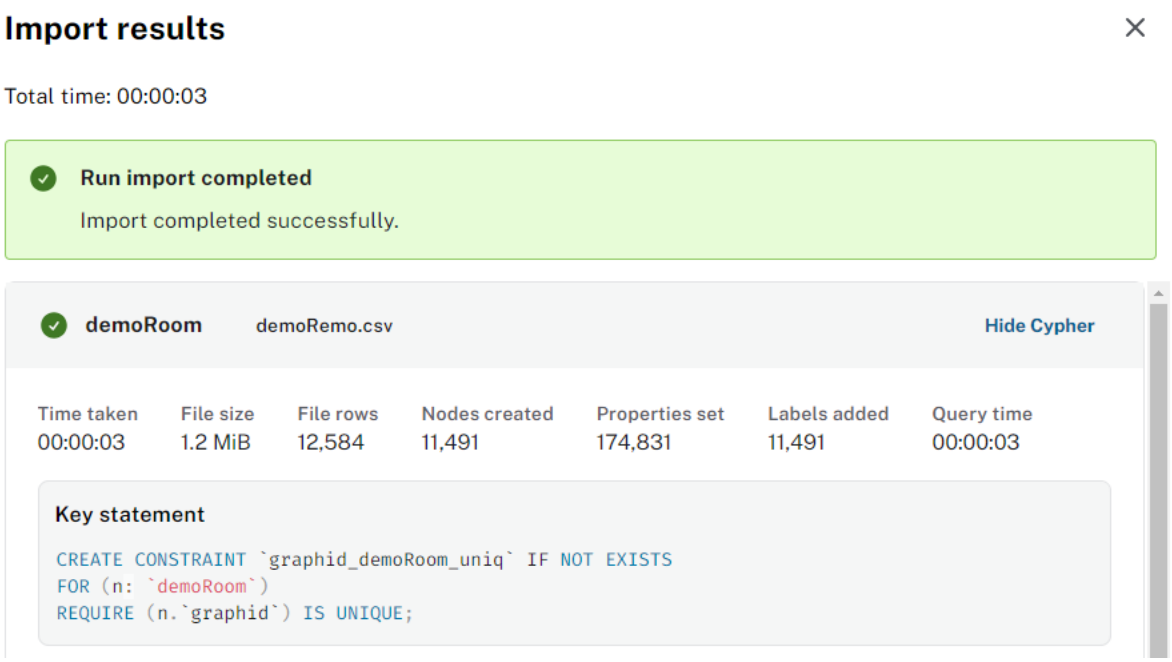


Figure 45.1: Screenshot of demoRoom import

45.1.2 load cypher

```
UNWIND $nodeRecords AS nodeRecord
WITH *
WHERE NOT nodeRecord.`graphid` IN $idsToSkip AND NOT nodeRecord.`graphid` IS NULL
MERGE (n: `demoRoom` { `graphid`: nodeRecord.`graphid` })
SET n.`#rm.bl_id` = nodeRecord.`#rm.bl_id`
SET n.`fl_id` = toInteger(trim(nodeRecord.`fl_id`))
SET n.`rm_id` = nodeRecord.`rm_id`
SET n.`rm_type` = nodeRecord.`rm_type`
SET n.`dp_id` = nodeRecord.`dp_id`
SET n.`bu_id` = nodeRecord.`bu_id`
SET n.`rm_std` = nodeRecord.`rm_std`
SET n.`rm_use` = nodeRecord.`rm_use`
SET n.`site_id` = nodeRecord.`site_id`
SET n.`dv_id` = nodeRecord.`dv_id`
SET n.`asb_risk` = nodeRecord.`asb_risk`
SET n.`rm_cat` = nodeRecord.`rm_cat`
SET n.`lon` = toFloat(trim(nodeRecord.`lon`))
SET n.`lat` = toFloat(trim(nodeRecord.`lat`))
SET n.`roomHostKey` = nodeRecord.`roomHostKey`;
```

45.1.3 Thoughts

Some rooms have longitude and latitude, which have been used to calculate distance. The screenshot below shows locations on Frenchay campus - you can clearly see rooms aligned into

the shape our the buildings.

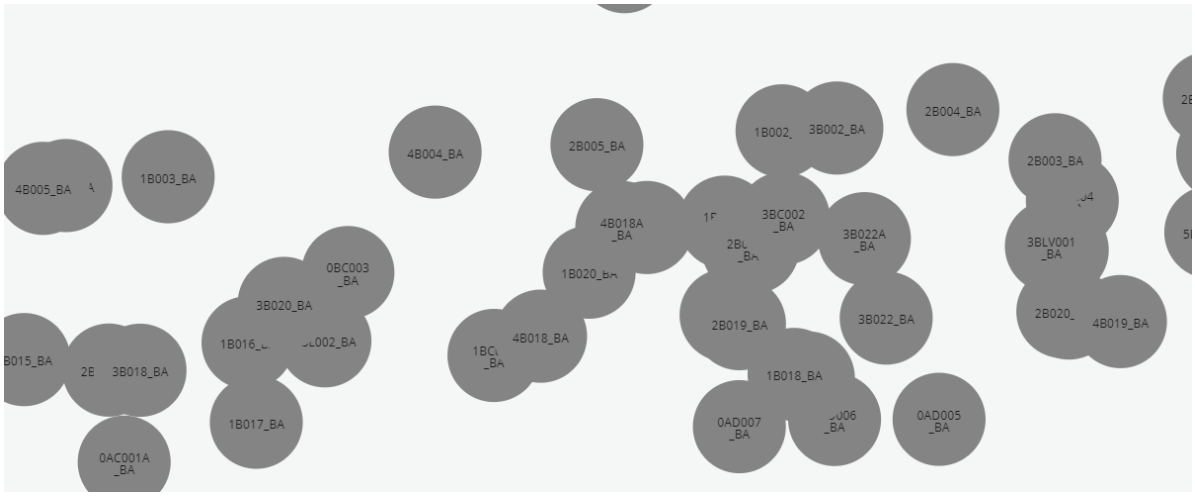


Figure 45.2: Close of up B Block rooms on Frenchay Campus

An alternative view includes locations in City Campus.

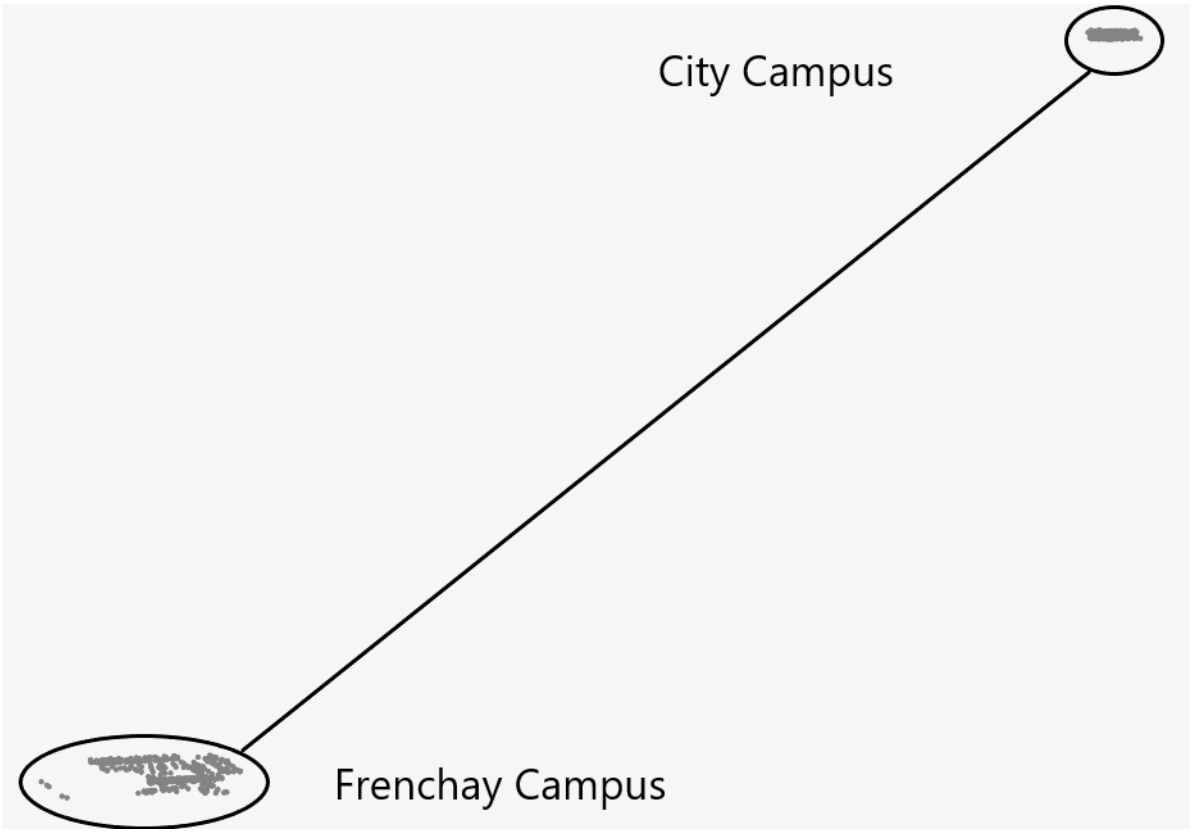


Figure 45.3: Frenchay and City Campus rooms

The above images show the potential of using coordinates although there is a lot more to consider including actual locations, accuracy, missing data, and coordinate transformations.

45.2 Room use

Being able to measure room usage - utilisation, frequency, occupancy - is a key metric for the university.

```
// room occupancy
MATCH (room:room )<-[:OCCUPIES]-(activity:activity)
WITH room, activity,
     activity.actDurationInMinutes / 60.0 AS occupancyHours
RETURN room.roomName AS room,
       SUM(occupancyHours) AS totalOccupancyHours
```

	room	totalOccupancyHours
1	"2Q12 FR"	650.0
2	"2Q42 FR"	209.0
3	"3E28 FR"	376.5
4	"3Q16 FR"	249.5
5	"3Q44 FR"	296.0

Figure 45.4: Example table of Room Occupancy Hours

```
// frequency
MATCH (room:room )<-[:OCCUPIES]-(activity:activity)
RETURN room.roomName AS room,
       COUNT(activity) AS totalActivities
```

room	totalActivities
"2Q12 FR"	267
"2Q42 FR"	114
"3E28 FR"	191
"3Q16 FR"	122
"3Q44 FR"	138
"3Q82 FR"	13

Figure 45.5: Example of Room frequency

```
// simple utilisation
MATCH (room:room {roomName: "2Q12 FR"})<-[:OCCUPIES]-(activity:activity)
WITH room, activity,
     CASE
         WHEN activity.actStartTime.hour = activity.actEndTime.hour
         THEN 1
         ELSE activity.actEndTime.hour - activity.actStartTime.hour + 1
     END AS occupiedHours
```

```
UNWIND range(activity.actStartTime.hour, activity.actEndTime.hour) AS hour
RETURN room.roomName AS room,
       hour AS hourBlockStart,
       hour + 1 AS hourBlockEnd,
       SUM(CASE WHEN hour IN range(activity.actStartTime.hour, activity.actEndTime.hour) THEN
       occupiedHours AS totalOccupiedHours
ORDER BY hourBlockStart
```

	room	hourBlockStart	hourBlockEnd	utilizationCount	totalOccupiedHours
1	"2Q12 FR"	9	10	26	5
2	"2Q12 FR"	9	10	60	3
3	"2Q12 FR"	10	11	26	5
4	"2Q12 FR"	10	11	60	3
5	"2Q12 FR"	11	12	26	5
6	"2Q12 FR"	11	12	84	3
7	"2Q12 FR"	12	13	35	3
8	"2Q12 FR"	12	13	26	5
9	"2Q12 FR"	13	14	71	3

Figure 45.6: Example of Room Utilisation

The above queries need to be developed further as they are very simplistic. For example, the utilisation query does not take into account the day of the week, nor does it consider the number of people in the room. The occupancy query does not consider the capacity of the room.

46 V: Perspectives and Scenes for Neo4j Explore Functionality

This page contains *example* scenarios for demonstrating Neo4j "Perspectives" and "Scenes" which are used to explore data in a graph database from the perspective of a particular user. Perspectives focus the view by selecting specific nodes and relationships, while scenes allow for visualising the data interactively.

46.1 Scenario 1: Programme Leader Perspective

Goal: Enable a programme leader to gain insights into their programme's structure, student engagement, and potential areas for improvement.

46.1.1 Perspective:

Nodes:

- Programme (central node)
- Module
- Student
- Activity
- Room
- Staff

Relationships:

- Programme <-[:BELONGS_TO]- Module
- Module <-[:BELONGS_TO]- Activity
- Student -[:REGISTERED_ON]-> Programme
- Student -[:ATTENDS]-> Activity
- Student-[:ENROLLED_ON]-> Module
- Activity -[:OCCUPIES]-> Room
- Staff -[:TEACHES]-> Activity

46.1.2 Scenes:

Programme Overview: Visualise the entire programme structure, showcasing modules, their connections, and associated activities. Highlight popular and less popular modules based on student enrolment.

Student Engagement: Focus on a specific module and visualise student attendance patterns. Identify students with low attendance and potential at-risk students.

Resource Allocation: Visualise room utilisation across the programme's activities. Identify potential scheduling conflicts or underutilised spaces.

Staff Workload: Analyse the distribution of teaching workload among staff members within the programme.

46.1.3 Example Cypher Queries:

```
//Programme Structure:

MATCH (p:programme {name: 'Computer Science'})<-[:BELONGS_TO]-(m:module)
RETURN p, m


//Student Attendance:

MATCH (m:module {name: 'Data Structures'})-[:BELONGS_TO]->(p:programme)<-[:REGISTERED_ON]-(s
RETURN m, s, a


//Room Utilisation:

MATCH (p:programme)<-[:BELONGS_TO]-(m:module)<-[:BELONGS_TO]-(a:activity)-[:OCCUPIES]->(r:roo
RETURN p, m, a, r

Staff Workload:

MATCH (p:programme)<-[:BELONGS_TO]-(m:module)<-[:BELONGS_TO]-(a:activity)<-[:TEACHES]-(s:sta
RETURN p, m, a, s
```

46.2 Scenario 2: Module Leader Perspective

Goal: Provide a module leader with insights into their module's scheduling, student performance, and room allocation.

46.2.1 Perspective:

Nodes:

- Module (central node)
- Activity
- Student
- Room
- Assessment

Relationships:

- module <-[:BELONGS_TO]- activity
- student -[:ATTENDS]-> activity
- activity -[:OCCUPIES]-> room
- student -[:HAS_RESULT_FOR]-> assessment
- module -[:HAS_ASSESSMENT]-> assessment

46.2.2 Scenes:

- **Module Schedule**: Visualise the module's activities, their timeslots, and associated rooms. Highlight potential scheduling conflicts or overlaps.
- **Student Performance**: Focus on a specific assessment and visualise student results. Identify students who may need additional support.
- **Room Suitability**: Analyse the rooms used for the module's activities and their capacities. Identify potential issues with room size or suitability.

46.3 Scenario 3: Administrator Perspective (Room Usage)

Goal: Ability to analyse room usage, identify underutilised spaces, and optimise room allocation.

46.3.1 Perspective:

Nodes:

- Room (central node)
- Activity
- Module
- Programme

Relationships:

- activity -[:OCCUPIES]-> room
- module <-[:BELONGS_TO]- activity
- programme <-[:BELONGS_TO]- module

46.3.2 Scenes:

- **Room Utilization Overview**: Visualise all rooms and their occupancy levels across different time periods. Highlight rooms with low utilisation or frequent conflicts.
- **Room Activity Breakdown**: Focus on a specific room and visualise the activities scheduled there. Identify peak usage times and potential scheduling issues.
- **Programme Room Usage**: Analyse room usage by different programmes. Identify programmes with high room demands or potential conflicts.

47 Blue Sky Opportunities

Here I imagine how the timetabling data *could* be combined with other data sources to unlock insights. This does not mean that any of this *should* be done. It also does not mean that it can't be done very effectively using non-graph solutions. It is very much *blue-skies* thinking in that it is not constrained by the realities of data, systems, technology, or policy.

If any of these ideas are taken forward, the usual data governance, ethics and privacy considerations need to be addressed, in addition to the technical and practical challenges like data quality, integration, and scalability.

47.1 Popular Module Combinations and Student Choice:

- **Data**: Analyse student enrolment data within programmes and identify popular module combinations. Combine with student feedback and performance (outcomes) and engagement data.

- **Insights**: Understand student preferences and identify frequently chosen module combinations. This can inform program design, curriculum development, and elective module offerings.
- **Action**: Adapt programme structures to align with student choices, and offer targeted module recommendations based on popular combinations and individual student interests.

47.2 Modules and Library Resources:

- **Data**: Integrate module data with library resources such as reading lists, e-resources, and physical collections.
- **Insights**: Identify the most frequently accessed resources for each module and assess their impact on student learning outcomes. Make recommendations for additional resources or support services.
- **Action**: Enhance module delivery by aligning resources with learning objectives, providing targeted support, and improving access to relevant materials.

47.3 Student Travel and Engagement:

- **Data**: Include student term-time addresses and combine with attendance and engagement data. Combine with public transport data, e.g. bus and train schedules.
- **Insights**: Understand the impact of travel distances on student engagement and academic performance. Identify potential transport barriers for specific student groups or areas. This is particularly relevant to Bristol where there is a large student population and accommodation challenges.
- **Action**: Optimise timetable scheduling to minimise travel distances, particularly for students with long commutes. Explore targeted transportation solutions or support services.

47.4 Equitable Access and Outcomes Analysis:

- **Data**: Combine timetable data with various pre and post student datasets, like SES, POLAR, Free Lunch data, and graduate outcomes.
- **Insights**: Identify if timetable structures disproportionately impact specific student groups. Does this impact their academic performance or future prospects?
- **Action**: Use insights to proactively address disparities in access and outcomes by adjusting timetables, providing targeted support, or informing policy changes.

47.5 Self-Serve Timetable Changes and Student Behaviour:

- **Data**: Analyse student self-serve timetable changes[1] and compare original and modified schedules.
- **Insights**: Gain insights into student preferences and identify common reasons for timetable modifications. This can inform timetable design and optimise scheduling processes - this could be particularly useful to inform the Timetable Quality Index.
- **Action**: Address common issues identified through timetable adjustments, such as frequent clashes or gaps in scheduling. Optimise the self-serve process based on student behaviour.

47.6 Facilities Optimisation and Space Utilisation:

- **Data**: Integrate estates/facilities datasets like master location data, wifi hits, etc. with timetable and occupancy data.

[1] Students are able to make certain changes to their timetables, at activity and module level. These changes are recorded and can be analysed.

- **Insights**: Identify underutilised spaces and peak usage times. Optimise space allocation and improve campus resource management. Understand flow of students and staff around campus.
- **Action**: Adjust timetables to balance space utilisation, explore flexible learning environments, and inform future campus development plans.

47.7 Student Clustering and Community Building:

- **Data**: Incorporate student demographic (with appropriate data privacy safeguards)
- **Insights**: Identify clusters of students with shared characteristics or interests. This can inform targeted support services, social events, and community-building initiatives.
- **Action**: Foster a sense of belonging by connecting students with similar backgrounds or interests. Tailor support services to meet the needs of specific student groups.

47.8 Module and Timeslot Recommendations:

- **Data**: Combine timetable, enrolment, student performance, and feedback data.
- **Insights**: Develop a recommender system to suggest module combinations and timeslots based on student interests, past performance, and peer recommendations.
- **Action**: Personalise the student experience and improve module selection.

47.8.1 Unpopular Activity Analysis:

- **Data**: Analyse attendance data and identify unpopular activities, correlating them with time, location, staff, and other properties.
- **Insights**: Understand the factors contributing to low attendance and identify potential areas for improvement.
- **Action**: Adjust timetable scheduling, explore alternative teaching approaches, or provide additional support to address identified issues.

47.8.2 Identifying Anomalies and Opportunities:

- **Data**: Apply graph algorithms and machine learning techniques to identify patterns, anomalies, and trends in combined datasets.
- **Insights**: Discover hidden opportunities for improvement, detect potential issues proactively, and gain deeper insights into complex relationships between various factors.
- **Action**: Address identified issues, optimise processes, and leverage opportunities for innovation and continuous improvement.

48 Supervision

Project supervision was undertaken by Dr. Xiaodong Li who was very supportive and made himself available to me in-person and by email, etc.

In general, we met for 30-60 minutes every fortnight where I presented progress, issues/blockers, planned next steps, etc. and we discussed the project in general. Dr. Li was very engaged and helpful and I am indebted to his supervision. Following each meeting, I wrote up notes.

I have shared a few examples in the following pages. Please note that links within these pages will not necessarily work as they point to working files and repositories.

49 Fortnightly Update - 10 June 2024

49.1 Summary

This week has been about getting back on track after an extended period away from the dissertation. It has mainly revolved around refocusing, rescoping, reacquainting and rekindling motivation.

49.2 Accomplishments

- **Results:**
  - ☒ recreated single student graph timetable - see poc-1-basic
  - ☒ created SQL to csv pipeline
  - ☒ created cypher queries
- **Project Management**
  - ☒ new github repo (currently private) https://github.com/zoonalink/graph-project
    - ☒ draft readme
    - ☒ folder/file structure
  - ☒ weekly update template
- **Data Collection:**
  - ☒ SQL to CSV files for single student
  - ☒ raw SQL data tables
- **Analysis / Wrangling:**
  - ☒ Transformation in SQL query
- **Model Development:**
- **Investigation**
  - ☒ pipelines to automate data flow - prefect
  - ☒ how to represent time in graph

49.3 Issues/Blockers

- **Technical:**
  - Original servers were deleted and access to new servers was gone
  - Some original work no longer works
- **Methodological:**
  - I need to ensure that scope does not creep.
  - I need to stay focused and not start solving problems which are out of scope or are interesting.
- **Data-Related:**
  - develop a robust anonymisation process
  - representing time - see representing time

49.4 Next Steps

49.4.1 Weekly Goals

1. Bigger cohort of data (MSc DataScience?)
2. Explore time in Graph
3. Data pipeline documentation and steps (anonymisation)

- **Project Management** Project tasks - planning, admin.
  - ☐ add supervisor to github repo
- **Data:**
  - ☐ Anonymisation or generation
  - ☐ MSc Data Science cohort isolated into separate csv files
  - ☐ splitting data into single rows
  - ☐ Pipeline steps plotted
- **Analysis / Wrangling:**
  - ☐ More advanced cypher queries
- **Modeling:**
  - ☐ Comparing different representations of time
- **Validation:**
- **Deadlines:**

49.5 Post-Meeting Notes

We met on 13 June 2024 for approximately 45 minutes. I showed Xiaodong what I have been up to referencing my working files, poc files, and some code. I also showed what I currently have in my free instance of neo4j aura - which was MSc DS students activities and rooms. Staff and Students to be loaded but I want to ensure my anonymisation function is tested first.

We discussed what my aims are with this project, clarifying that it is not building a timetable schedule based on graph structure; it is also not a timetable optimiser. Instead it is a data engineering project that ultimately explores whether representing timetables in graph format can bring opportunities for reporting and insight which is currently difficult to produce using the current system.

We agreed to meet in two weeks' time. I will schedule a meeting for us.

49.6 Additional Notes

Summary from intial meeting

I met with Xiaodong Li my supervisor on Wednesday 01 May 2024 where we introduced ourselves and our backgrounds. We discussed my proposal which Xiaodong had read in advance of our meeting.

I showed Xiaodong my work so far, most of it completed in January which included proof-of-concept data engineering steps to extract, transform and load data from a relational database to a graph structure. I showed the steps I had taken, the challenges encountered and the possible opportunities of graph data structures which I am hoping to explore in more detail.

We discussed project scope and outcomes and recognised that I need to ensure that I keep within scope.

The next few weeks will require attention on other matters (work, taught modules, etc.) but I will look to pick up project work soon.

If possible, Xiaodong was going to investigate Neo4j and graph to get a bit of context.

50 Fortnightly Update - 2024-06-24

50.1 Summary

Significant progress in the backend and more proof-of-concept in front end.

50.2 Accomplishments

- **Project Management**
  - ☒ post supervisor meeting notes
- **Data Collection:**
  - ☒ Anonymisation function for staff/student personal data developed and tested.
  - ☒ MSc Data Science cohort isolated into separate csv files
  - ☒ Data extraction pipeline designed, developed, tested. This allows for extracting data filtered by programme on demand.
    - modularised, scalable, configurable, efficient pipeline
  - ☒ Transformation pipeline (preprocessing, anonymising)
    - More than half way completed - same principles
    - need to add staff, student, activity nodes and test outcomes
    - need to add relationships and test.
  - ☒ Loading to Neo4j pipeline developed - suitable for version 1
- **Analysis / Wrangling:**
  - ☒ More advanced cypher queries - constraint violation queries developed using cypher for version 1. some are very complex. more testing needed.
  - ☒ list of insights which can be derived
- **Model Development:**
  - ☐ Comparing different representations of time
- **Results:**
- Pipeline development
- Cypher queries for version 1
- conversation with business owners to validate work

50.3 Next Steps

50.3.1 Weekly Goal: What is goal of next fortnight?

- write up and benchmark v1 notes
- finish developing and testing pipeline:
  - extract
  - transform
  - load
- document pipeline - written and visual (mermaid?)
- develop cypher queries for version 2
- design (theory) timetable quality index
- consider scaling dataset
- start writing project notes

50.4 Issues/Blockers

- **Technical:**
  - Digital certificates on machines preventing load
  - Neo4j Aura (free) limitations - loading issues
- **Methodological:** Concerns about approach or analysis?
  - Spending a lot of time getting ETL 'right'
  - Not sure about balance of project and what I will deliver at the end
- **Data-Related:** Issues with data quality, access, or quantity?
  - Working on pipelines which will mean I can scale accordingly (within constraints of free instance)

50.5 Post-Meeting Notes

My supervisor and I spoke about where I am at in the project at the moment and next steps. We discussed what I am attempting to do and why, including graph data structures, proof-of-concept, etc. and that it is becoming a data engineering project. I stated that my timeline looks to complete a robust data ETL pipeline by the middle of July, with a view to shifting towards more work within Neo4j from the end of July to the middle of August.

We agreed to meet in two weeks.

51 Fortnightly Update - 2024-07-08

51.1 Summary

- This week has been primarily based on getting the ETL pipeline working.
- Some thought and planning about the overall project

51.2 Accomplishments

- **Project Management**
  - Planned next 6 weeks - high level
- **Data Collection:**
  - N/A
- **Analysis / Wrangling:**
  - N/A
- **Model Development:**
  - The ETL is (hopefully) production ready.
- **Results:**
  - Functioning ETL which:
    * Extracts -> Filters -> Cleans -> Transforms -> Anonymises -> Uploads to Google Drive -> Loads to Neo4j
    * Configurable, scalable, modular
    * Logging, Error-handling

51.3 Next Steps

51.3.1 Weekly Goal: What is goal of week?

- **Project Management**
  - Plan next six weeks - with tasks
  - see project-structure
- **Data:**
  - Clear out database and repopulate
  - Run ETL for some programme - e.g. UG maths, PG Data Sci, PG AI, PG cyber
  - Consider Unit Tests
- **Analysis / Wrangling:**
  - Develop and explore Cypher quality and violation queries
  - see queries
- **Modeling:**
  - Consider different model of time as nodes (but probably will not develop)
- **Writing**
  - After ETL confirmation - fully documentation of code
    - Also write up into QMD doc
    - Visualiation - Data Flow Diagram?
  - Model section write up
    - SQL data model
    - Graph data model
    - Visualisations
    - see code-critique
  - Draft Intro
    - Proof of concept
    - Research question
    - Domain

51.4 Issues/Blockers

- **Technical:**
  - Main limitations are free Neo4j Aura software and other restrictions
- **Methodological:**
  - Need to think about what I want to deliver and how it looks

51.5 Post-Meeting Notes

- **Key Decisions:** What were the main takeaways from meeting?
  - I need to focus on making progress and think about what the final produce will be.
- **Action Items:** What tasks do you need to complete?
  - see above - write up sections, even if draft

51.6 Additional Notes

- **Literature Review:** Relevant papers read.
- **Experiments:** If applicable, describe any experiments conducted.
- **Code:** Embedding example snippets of code.

52 References

Abdipoor, S., Yaakob, R., Goh, S.L. and Abdullah, S. (2023) Meta-heuristic approaches for the University Course Timetabling Problem. Intelligent Systems with Applications [online]. 19, p. 200253. Available from: https://www.sciencedirect.com/science/article/pii/S2667305323000789 [Accessed 25 July 2024].

Anon. (no date) CPB Projects [online]. Available from: https://www.cpbprojects.co.uk/solutions/timetabling-and-teaching-space [Accessed 25 July 2024a].

Babaei, H., Karimpour, J. and Hadidi, A. (2015)'A survey of approaches for university course timetabling problem' Computers & Industrial EngineeringApplications of Computational Intelligence and Fuzzy Logic to Manufacturing and Service Systems [online]. 86, pp. 43–59. Available from: https://www.sciencedirect.com/science/article/pii/S0360835214003714 [Accessed 28 July 2024].

Beck, K., et al. (2001) The Agile Manifesto. Agile Alliance. http://agilemanifesto.org/

Bellio, R., Ceschia, S., Di Gaspero, L., Schaerf, A. and Urli, T. (2016) Feature-based tuning of simulated annealing applied to the curriculum-based course timetabling problem. Computers & Operations Research [online]. 65, pp. 83–92. Available from: https://linkinghub.elsevier.com/retrieve/pii/S0305054815001690 [Accessed 26 January 2024].

Bonutti, A., De Cesco, F., Di Gaspero, L. and Schaerf, A. (2012) Benchmarking curriculum-based course timetabling: formulations, data formats, instances, validation, visualization, and results. Annals of Operations Research [online]. 194 (1), pp. 59–70. Available from: https://doi.org/10.1007/s10479-010-0707-0 [Accessed 3 February 2024].

Bruggen, R. van (2014)'Learning Neo4j: run blazingly fast queries on complex graph datasets with the power of the Neo4j graph database'Community Experience Distilled. 1st edition. Birmingham, England: Packt Publishing.

Burke, E., Mccollum, B., Meisels, A., Petrovic, S. and Qu, R. (2007) A graph-based hyper-heuristic for educational timetabling problems. European Journal of Operational Research [online]. 176, pp. 177–192.

Ceschia, S., Di Gaspero, L. and Schaerf, A. (2023) Educational timetabling: Problems, benchmarks, and state-of-the-art results. European Journal of Operational Research [online]. 308 (1), pp. 1–18. Available from: https://linkinghub.elsevier.com/retrieve/pii/S0377221722005641 [Accessed 25 January 2024].

Chen, M., Sze, S., Goh, S.L., Sabar, N. and Kendall, G. (2021) A Survey of University Course Timetabling Problem: Perspectives, Trends and Opportunities. IEEE Access [online]. PP, pp. 1–1.

Chicken, S., Fogg Rogers, L., Hobbs, L., Hunt-Fraisse, T. and Lewis, D. (2023) Amplifying the voices of neurodivergent students in relation to higher education assessment at UWE Bristol. [online]. Available from: https://uwe-repository.worktribe.com/output/10879555 [Accessed 25 July 2024].

Dammak, A., Elloumi, A. and Kamoun, H. (2007) An enterprise system component based on graph colouring for exam timetabling: A case study in a Tunisian university. Transforming Government: People, Process and Policy [online]. 1 (3), pp. 255–270. Available from: https://www.emerald.com/insight/content/doi/10.1108/17506160710778095/full/html [Accessed 19 February 2024].

de Werra, D. (1997) The combinatorics of timetabling. European Journal of Operational Research [online]. 96 (3), pp. 504–513.

Don State Technical University, Rostov-on-Don, Russian Federation and Al-Gabri, W.M. (2017) Literature review for the topic of automation of scheduling classes and exams in higher education institutions. Vestnik of Don State Technical University [online]. 17 (1), pp. 132–143. Available from: https://vestnik.donstu.ru/jour/article/view/255 [Accessed 25 January 2024].

Dowland, D. (2018) Rubik's cube or Battenburg? The university timetable Wonkhe. 11 January 2018 [online]. Available from: https://wonkhe.com/blogs/rubiks-cube-or-battenburg-the-university-timetable/ [Accessed 25 July 2024].

Foung, D. and Chen, J. (2019) Discovering disciplinary differences: blending data sources to explore the student online behaviors in a University English course. Information Discovery and Delivery [online]. 47 (2), pp. 106–114. Available from: https://www.emerald.com/insight/content/doi/10.1108/IDD-10-2018-0053/full/html [Accessed 19 February 2024].

helenclu (2024) Database normalization description - Microsoft 365 Apps. 6 June 2024 [online]. Available from: https://learn.microsoft.com/en-us/office/troubleshoot/access/database-normalization-description [Accessed 11 August 2024].

Herres, B. and Schmitz, H. (2021) Decomposition of university course timetabling: A systematic study of subproblems and their complexities. Annals of Operations Research [online]. 302 (2), pp. 405–423. Available from: https://ezproxy.uwe.ac.uk/login?url=https://search.ebscohost.com/login.aspx
live [Accessed 28 July 2024].

Holm, D.S., Mikkelsen, R.Ø., Sørensen, M. and Stidsen, T.J.R. (2022) A graph-based MIP formulation of the International Timetabling Competition 2019. Journal of Scheduling [online]. 25 (4), pp. 405–428. Available from: https://doi.org/10.1007/s10951-022-00724-y [Accessed 2 November 2023].

Johnson, D. (1993) A Database Approach to Course Timetabling. The Journal of the Operational Research Society [online]. 44 (5), pp. 425–433. Available from: https://www.jstor.org.ezproxy.uwe.ac.uk/stable/2583909 [Accessed 28 July 2024].

Khan, W., Kumar, T., Zhang, C., Raj, K., Roy, A.M. and Luo, B. (2023) SQL and NoSQL Database Software Architecture Performance Analysis and Assessments—A Systematic Literature Review. Big Data and Cognitive Computing [online]. 7 (2), p. 97. Available from: https://www.mdpi.com/2504-2289/7/2/97 [Accessed 28 July 2024].

Lemos, A., Melo, F.S., Monteiro, P.T. and Lynce, I. (2019) Room usage optimization in timetabling: A case study at Universidade de Lisboa. Operations Research Perspectives [online]. 6, p. 100092. Available from: https://linkinghub.elsevier.com/retrieve/pii/S2214716018301696 [Accessed 26 January 2024].

Lindahl, M., Mason, A.J., Stidsen, T. and Sørensen, M. (2018) A strategic view of University timetabling. European Journal of Operational Research [online]. 266 (1), pp. 35–45. Available from: https://www.sciencedirect.com/science/article/pii/S0377221717308433 [Accessed 28 July 2024].

MirHassani, S.A. and Habibi, F. (2013) Solution approaches to the course timetabling problem. Artificial Intelligence Review [online]. 39 (2), pp. 133–149. Available from: http://link.springer.com/10.1007/s10462-011-9262-6 [Accessed 26 January 2024].

Mühlenthaler, M. and Wanka, R. (2016) Fairness in academic course timetabling. Annals of Operations Research [online]. 239 (1), pp. 171–188. Available from: https://doi.org/10.1007/s10479-014-1553-2 [Accessed 3 February 2024].

Müller, T. and Murray, K. (2010) Comprehensive approach to student sectioning. Annals of Operations Research [online]. 181 (1), pp. 249–269. Available from: http://link.springer.com/10.1007/s10479-010-0735-9 [Accessed 26 January 2024].

Nan 1, Z., Bai, X. 1 1 C. of I. and Economics, T.Y. (2019) The study on data migration from relational database to graph database. [online]. Available from: https://www.proquest.com/docview/2568058349?pq-origsite=primo [Accessed 4 November 2023].

Negro, A. (2021) Graph-Powered Machine Learning [online]. O'Reilly Media, Inc. [Accessed 4 November 2023].

Neo4j (2023) The Neo4j Cypher Manual v5.

Nguyen, V.D. and Nguyen, T. (2021) An SHO-based approach to timetable scheduling: a case study. Journal of Information and Telecommunication [online]. 5 (4), pp. 421–439. Available from: https://doi.org/10.1080/24751839.2021.1935644 [Accessed 2 November 2023].

Norman, R. and Williams, E. (no date) PSP Board Pack 220804 v1.3.pptx [online]. Available from: https://uweacuk-my.sharepoint.com/:p:/g/personal/richard2_norman_uwe_ac_uk/EWKNTqInQuRDo
758c-97f4-07dd-ff61808257cf [Accessed 19 February 2024].

Oude Vrielink, R.A., Jansen, E.A., Hans, E.W. and Van Hillegersberg, J. (2019) Practices in timetabling in higher education institutions: a systematic review. Annals of Operations Research [online]. 275 (1), pp. 145–160. Available from: http://link.springer.com/10.1007/s10479-017-2688-8 [Accessed 2 November 2023].

Ries, E. (2024) What Is an MVP? Eric Ries Explains Lean Startup Co. 28 February 2024 [online]. Available from: https://leanstartup.co/resources/articles/what-is-an-mvp/ [Accessed 11 August 2024].

Rudová, H., Müller, T. and Murray, K. (2011) Complex university course timetabling. Journal of Scheduling [online]. 14 (2), pp. 187–207. Available from: http://link.springer.com/10.1007/s10951-010-0171-3 [Accessed 26 January 2024].

Sanchez, C.A. (2015) An analytics based architecture and methodology for collaborative timetabling in higher education - ProQuest [online]. Available from: https://www.proquest.com/docview/177955
origsite=primo&parentSessionId=GLad2hOIbVrF%2F0eiOKHGi%2BO%2BFOyV9GXuQTQCxSfgWNw%3D&
[Accessed 25 January 2024].

Scifo, E. (2023) Graph Data Science with Neo4j [online]. [Accessed 4 November 2023].

Sokolova, Marina V., Francisco J. Gómez, and Larisa N. Borisoglebskaya. 'Migration from an SQL to a Hybrid SQL/NoSQL Data Model'. Journal of Management Analytics 7, no. 1 (March 2020): 1–11. https://doi.org/10.1080/23270012.2019.1700401.

Webber, J., Eifrem, E. and Robinson, I. (2013) Graph Databases [online]. [Accessed 4 November 2023].

Wikipedia contributors (2024) Cypher (query language) — Wikipedia, the free encyclopedia [online]. Available from: https://en.wikipedia.org/w/index.php?title=Cypher_(query_language)&oldid=12197

Wikipedia contributors (2024) NP-hardness — Wikipedia, the free encyclopedia [online]. Available from: https://en.wikipedia.org/w/index.php?title=NP-hardness&oldid=1236371945.

Wikipedia contributors (2024) SQL Wikipedia, The Free Encyclopedia [online]. Available from: https://en.wikipedia.org/w/index.php?title=SQL&oldid=1238737606 [Accessed 11 August 2024].

53 Acknowledgements

First, I would like to express my gratitude to my supervisor, Dr. Xiaodong Li, for his guidance, support and feedback throughout this project. He was particular helpful in keeping me focused and staying on topic, allowing me to incrementally achieve my goals.

I extend my sincere thanks to the faculty and staff of the MSc Data Science programme for providing the environment and resources necessary for my project:

- Dr Paul Matthews - Course lead for MSc Data Science
- Prakash Chatterjee - Data Management Fundamentals
  - Reintroduced the concept of graph databases and noSQL.
- Dr David Wyatt - Programming for Data Science
  - Provided foundations of Python programming, Git version control, markdown.
- Dr Hisham Ihshaish - Machine Learning and Predictive Analytics
  - Introduced me to Machine Learning, marrying programming and statistics.
- Dr Deirdre Toher - Advanced Statistics
  - Pulled me through challenging statistics, introduced me to Quarto (in an R context) and was generally very supportive over the years. Now working for Central Statistics Office in Ireland.
- Dr Jason Anquandah - Interdisciplinary Group Project
  - Supported me through the challenges of a group project.
- Dr Mahmoud Elbattah - Interdisciplinary Group Project
  - Supported me through the challenges of a group project.

I am indebted to my colleagues in SDS for their support, camaraderie, intellectual discussions and willingness to share knowledge, expertise and time. I am especially grateful to my mentor, colleague, manager and friend, Esther Williams, who has supported me throughout this journey and others, always magnanimously sharing her wisdom and unwavering encouragement.

Finally, I would like to thank my family and friends for their unwavering love and support throughout my graduate studies. Their encouragement and understanding have been invaluable in helping me navigate the challenges and celebrate the successes along the way.

In the spirit of graphs...

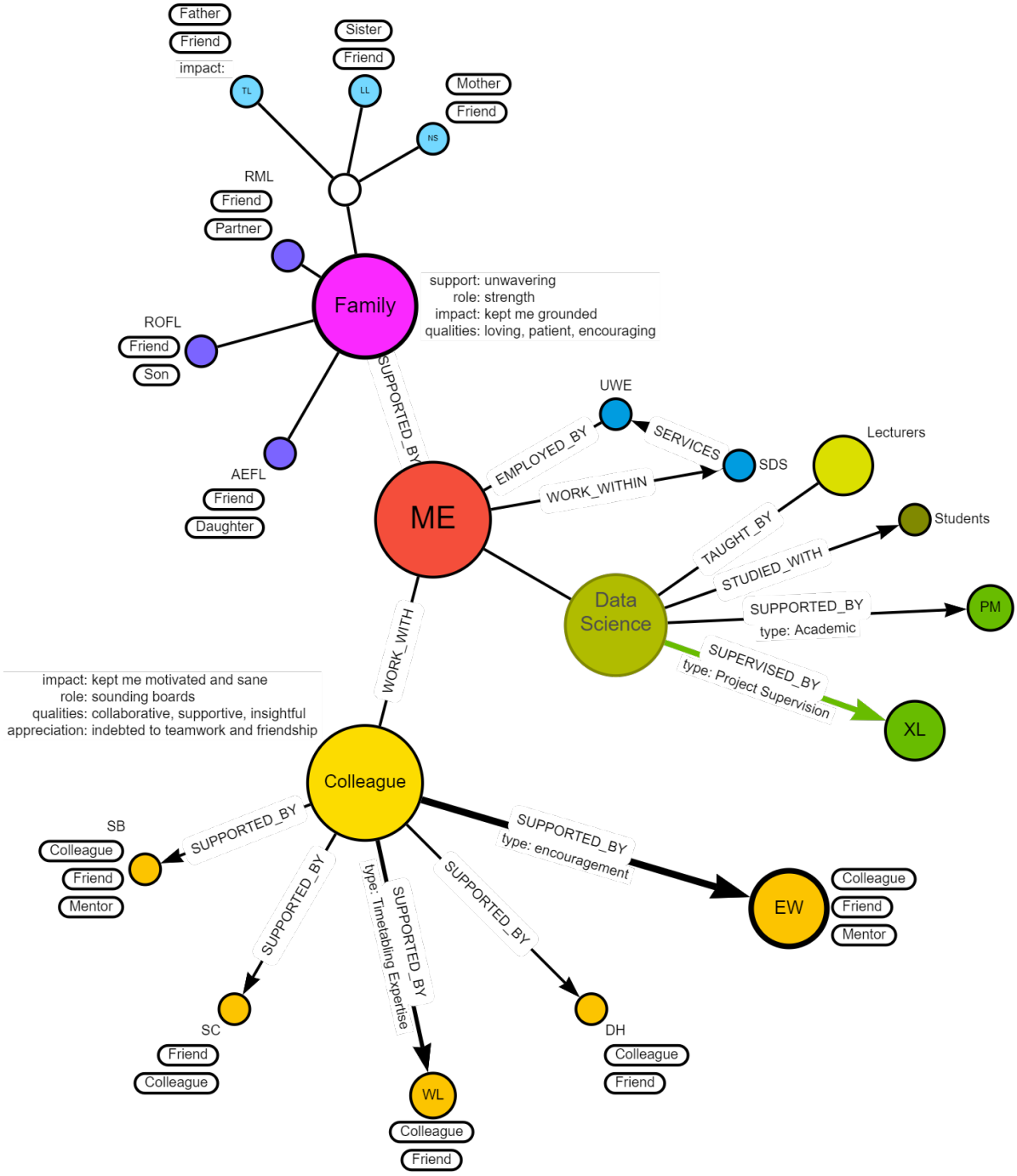


Figure 53.1: Acknowledgement graph