Graphing University Timetables

A proof-of-concept data engineering project

Petter Lovehagen

2024-08-29

Contents

[Graph Time 1](#_Toc175146068)

[Exploring Graph Data Models for Timetabling Insights 6](#_Toc175146069)

[Introduction 7](#_Toc175146070)

[Background and Motivation 8](#_Toc175146071)

[Personal 8](#_Toc175146072)

[Research Gap: Bridging Theory and Practice 8](#_Toc175146073)

[The hypothesis…Enter the Graph 9](#_Toc175146074)

[What is a “good” timetable? 10](#_Toc175146075)

[Consider these timetables: 10](#_Toc175146076)

[Project Aims and Scope 13](#_Toc175146077)

[Graph vs Relational Data Models 15](#_Toc175146078)

[Relational Models 15](#_Toc175146079)

[Graph Models 16](#_Toc175146080)

[Comparing queries 17](#_Toc175146081)

[Key Differences and Implications 18](#_Toc175146082)

[Graph Data Model for Timetabling 19](#_Toc175146083)

[An Iterative Approach 19](#_Toc175146084)

[Core Nodes - Building Blocks 19](#_Toc175146085)

[Relationships - Connecting the Dots 20](#_Toc175146086)

[MVP model 20](#_Toc175146087)

[Early Insights 22](#_Toc175146088)

[Unveiling Basic Patterns 22](#_Toc175146089)

[Model Expansion 25](#_Toc175146090)

[Graphing Time 27](#_Toc175146091)

[The Problem of Normalised Time 27](#_Toc175146092)

[Exploring Potential Solutions 27](#_Toc175146093)

[Data Engineering Overview 35](#_Toc175146094)

[High-level Architecture 35](#_Toc175146095)

[Design Principles 36](#_Toc175146096)

[Implementation Approach 40](#_Toc175146097)

[Upcoming Sections 41](#_Toc175146098)

[Data Engineering Approach 42](#_Toc175146099)

[Configuration and Logging 45](#_Toc175146100)

[Extraction 46](#_Toc175146101)

[Transformation 49](#_Toc175146102)

[All data 49](#_Toc175146103)

[Nodes and relationships 50](#_Toc175146104)

[Google Load 51](#_Toc175146105)

[Neo4j Load 53](#_Toc175146106)

[Reflections 55](#_Toc175146107)

[Lessons Learned 55](#_Toc175146108)

[Timetable Metrics 57](#_Toc175146109)

[Metric Aggregations 60](#_Toc175146110)

[Implementing TQI 61](#_Toc175146111)

[TQI Summary 63](#_Toc175146112)

[Final Thoughts 65](#_Toc175146113)

[Reflections on Journey 65](#_Toc175146114)

[Looking Ahead: The Future of Graph at Universities 65](#_Toc175146115)

[Appendix: Table of Contents 67](#_Toc175146116)

[A: Random Graph Generator 68](#_Toc175146117)

[B: Technology Stack 70](#_Toc175146118)

[Configuration YAML 72](#_Toc175146119)

[Anonymisation 76](#_Toc175146120)

[ETL Summary 79](#_Toc175146121)

[ETL Code Gists 82](#_Toc175146122)

[Config and Misc 83](#_Toc175146123)

[Extract-SQL 84](#_Toc175146124)

[Extract 85](#_Toc175146125)

[Google Drive Load 86](#_Toc175146126)

[Process 87](#_Toc175146127)

[Neo4j Load 88](#_Toc175146128)

[Cypher Queries 89](#_Toc175146129)

[Constraints 89](#_Toc175146130)

[Indexes 89](#_Toc175146131)

[Creating Nodes and Relationships 93](#_Toc175146132)

[Creating Nodes 93](#_Toc175146133)

[Creating Relationships 93](#_Toc175146134)

[Relationships created after ETL 94](#_Toc175146135)

[Deleting Nodes and Relationships 97](#_Toc175146136)

[Deleting Nodes 97](#_Toc175146137)

[Deleting Relationships 97](#_Toc175146138)

[General Queries 99](#_Toc175146139)

[Count Queries 105](#_Toc175146140)

[Hard Constraints 110](#_Toc175146141)

[Student Clashes - Deeper Dive 117](#_Toc175146142)

[Scenario 117](#_Toc175146143)

[Model 1 - Activity Occurrence 117](#_Toc175146144)

[Model 2 - Date and Time Nodes 120](#_Toc175146145)

[Model 3 - Date Nodes 123](#_Toc175146146)

[Conclusion 129](#_Toc175146147)

[Delete Data 129](#_Toc175146148)

[Soft Constraints 131](#_Toc175146149)

[Rooms and Spaces 146](#_Toc175146150)

[Room Geo-location 146](#_Toc175146151)

[Room use 148](#_Toc175146152)

[Perspectives and Scenes for Neo4j Explore Functionality 151](#_Toc175146153)

[Scenario 1: Programme Leader Perspective 151](#_Toc175146154)

[Scenario 2: Module Leader Perspective 152](#_Toc175146155)

[Scenario 3: Administrator Perspective (Room Usage) 153](#_Toc175146156)

[Supervision 155](#_Toc175146157)

[Weekly Update - 10 June 2024 156](#_Toc175146158)

[Summary 156](#_Toc175146159)

[Accomplishments 156](#_Toc175146160)

[Issues/Blockers 156](#_Toc175146161)

[Next Steps 157](#_Toc175146162)

[Post-Meeting Notes 158](#_Toc175146163)

[Additional Notes 158](#_Toc175146164)

[Fortnightly Update - 2024-06-24 159](#_Toc175146165)

[Summary 159](#_Toc175146166)

[Accomplishments 159](#_Toc175146167)

[Next Steps 160](#_Toc175146168)

[Issues/Blockers 160](#_Toc175146169)

[Post-Meeting Notes 160](#_Toc175146170)

[Weekly Update - 2024-07-08 161](#_Toc175146171)

[Summary 161](#_Toc175146172)

[Accomplishments 161](#_Toc175146173)

[Next Steps 161](#_Toc175146174)

[Issues/Blockers 162](#_Toc175146175)

[Post-Meeting Notes 162](#_Toc175146176)

[Additional Notes 163](#_Toc175146177)

[Acknowledgement 168](#_Toc175146178)

# Exploring Graph Data Models for Timetabling Insights

A proof-of-concept data engineering project

**Supervisor**: [Xiaodong Li](https://people.uwe.ac.uk/Person/XiaodongLi)

**Programme**: [MSc Data Science](https://courses.uwe.ac.uk/INB112/data-science)

**Word Count**: [add when finished - currently ~6100 words (excluding code, tables, references)]

[insert Loom video]

# 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.

# Background and Motivation

## 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.

## 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]](#footnote-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]](#footnote-2) do not fully capture the dynamic complexity of real-world university timetabling.

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).

## The 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.

# 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.

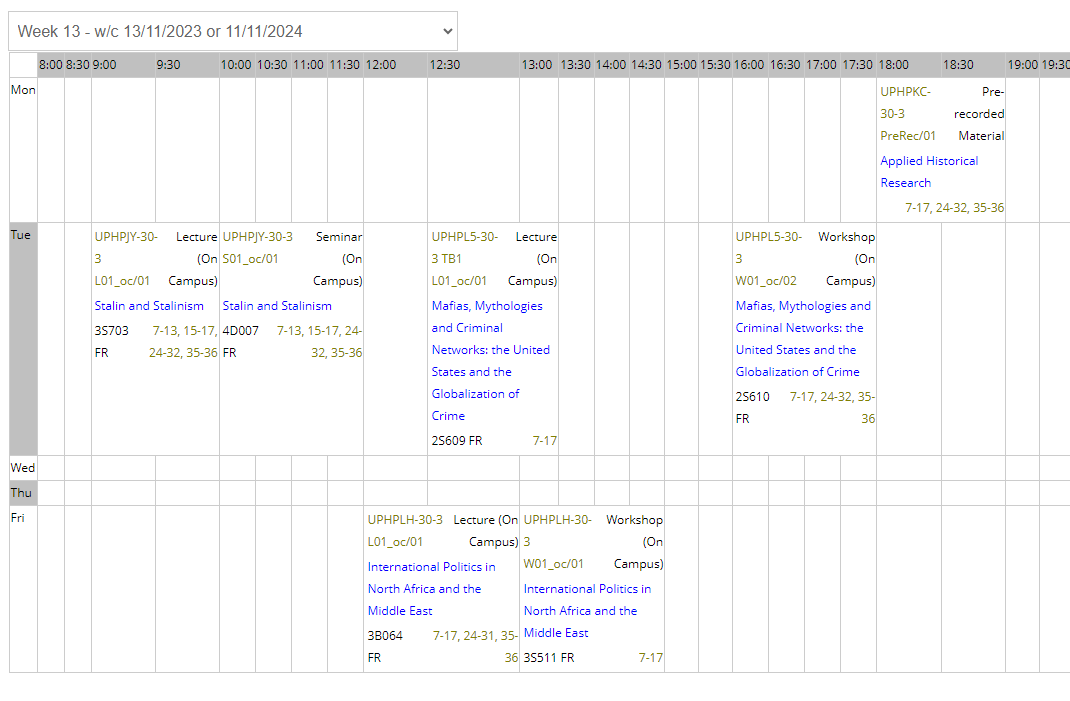
## Consider these timetables:

* 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)







# 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.

### 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.

In order 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.

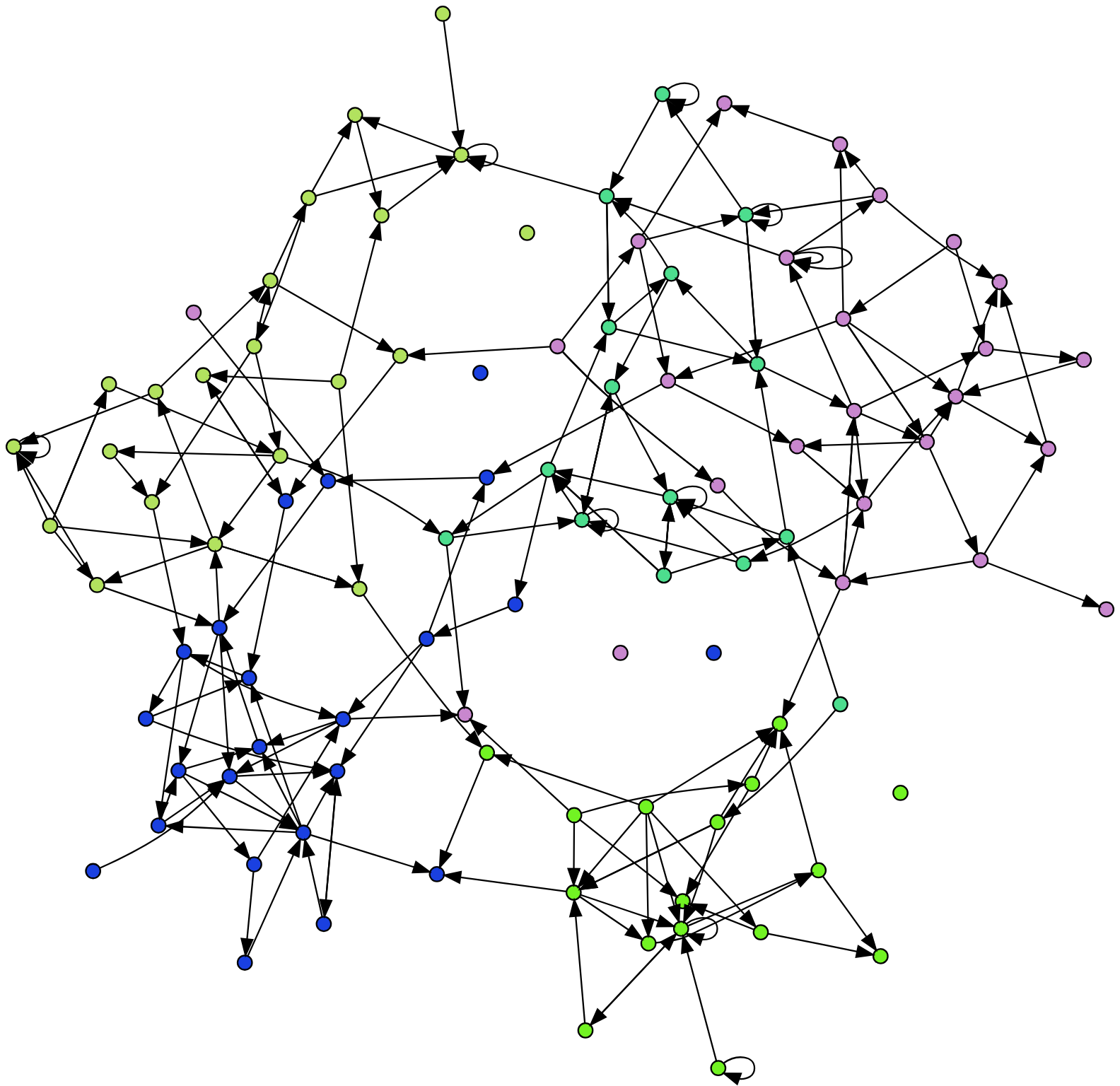
I will **not**:

* 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 visual purposes only.

[See Appendix for graph generator code](appendix-random-graph.qmd)

# Graph vs Relational Data Models

As outlined in the [project aims](01-04-project.qmd), 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.

## Relational Models

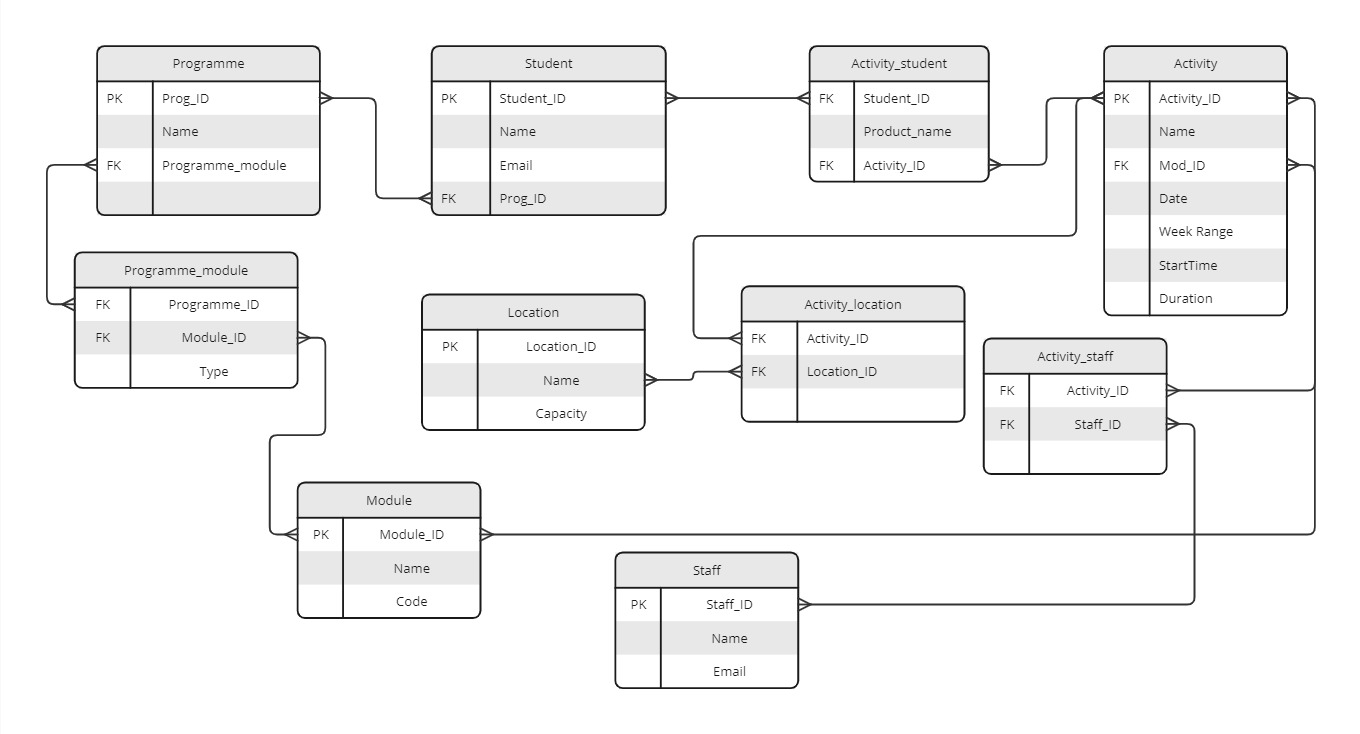
### Tables, Joins and the Limits of Interconnectedness

Relational databases, using SQL[[3]](#footnote-3) 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.



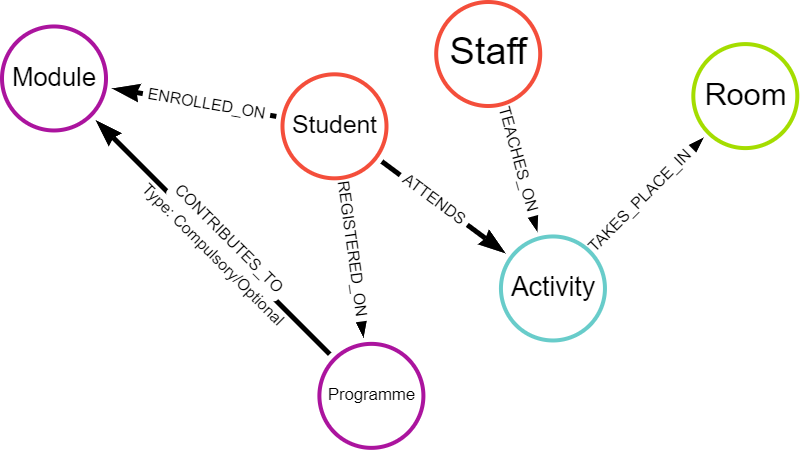
Example Simple Entity Relationship Diagram

## Graph Models

### 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.



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).

## 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.

### 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.[StudentSetId]  
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';

### 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

## 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) |
| 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.

# 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.

## 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[[4]](#footnote-4) approach. Eventually, my expanded model was created in a cloud-instance of Neo4j Aura.

## Core Nodes - Building Blocks

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

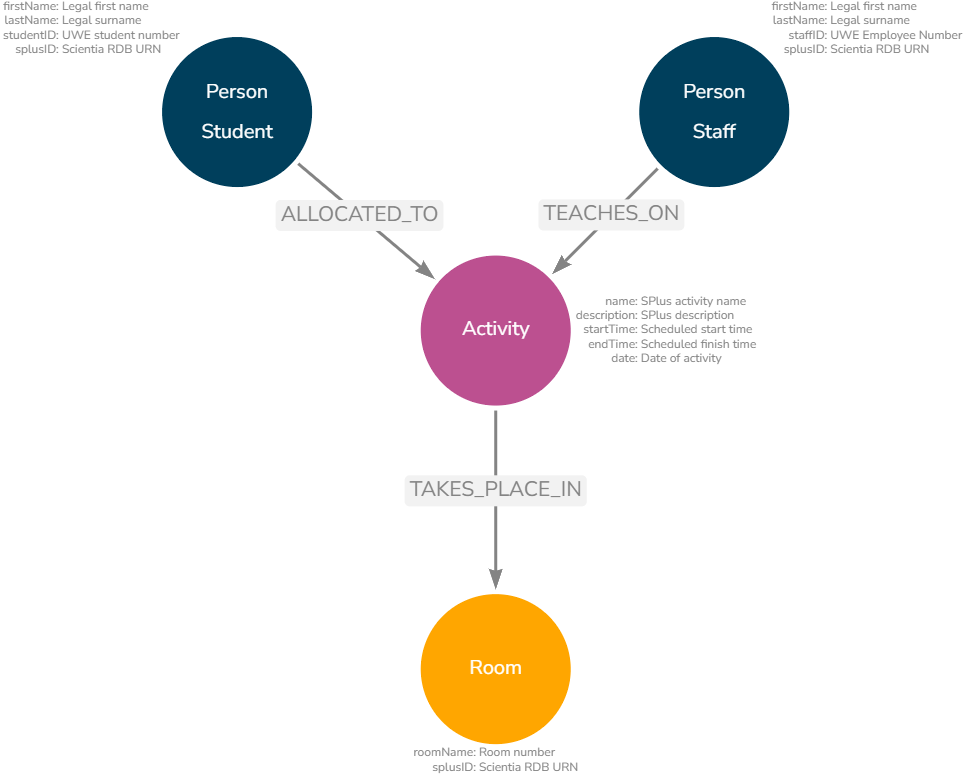
| Node | Property | Description | Data Type |
| --- | --- | --- | --- |
| Student | firstName | Legal first name | string |
|  | lastName | Legal last name | string |
|  | studentID | University identifier | integer |
|  | splusID | Timetable URN | string |
| Lecturer | firstName | First name | string |
|  | 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 |
|  | description | Activity description | string |
|  | startTime | Scheduled start time | datetime |
|  | endTime | Scheduled end time | datetime |
|  | date | Date of activity | date |

## Relationships - Connecting the Dots

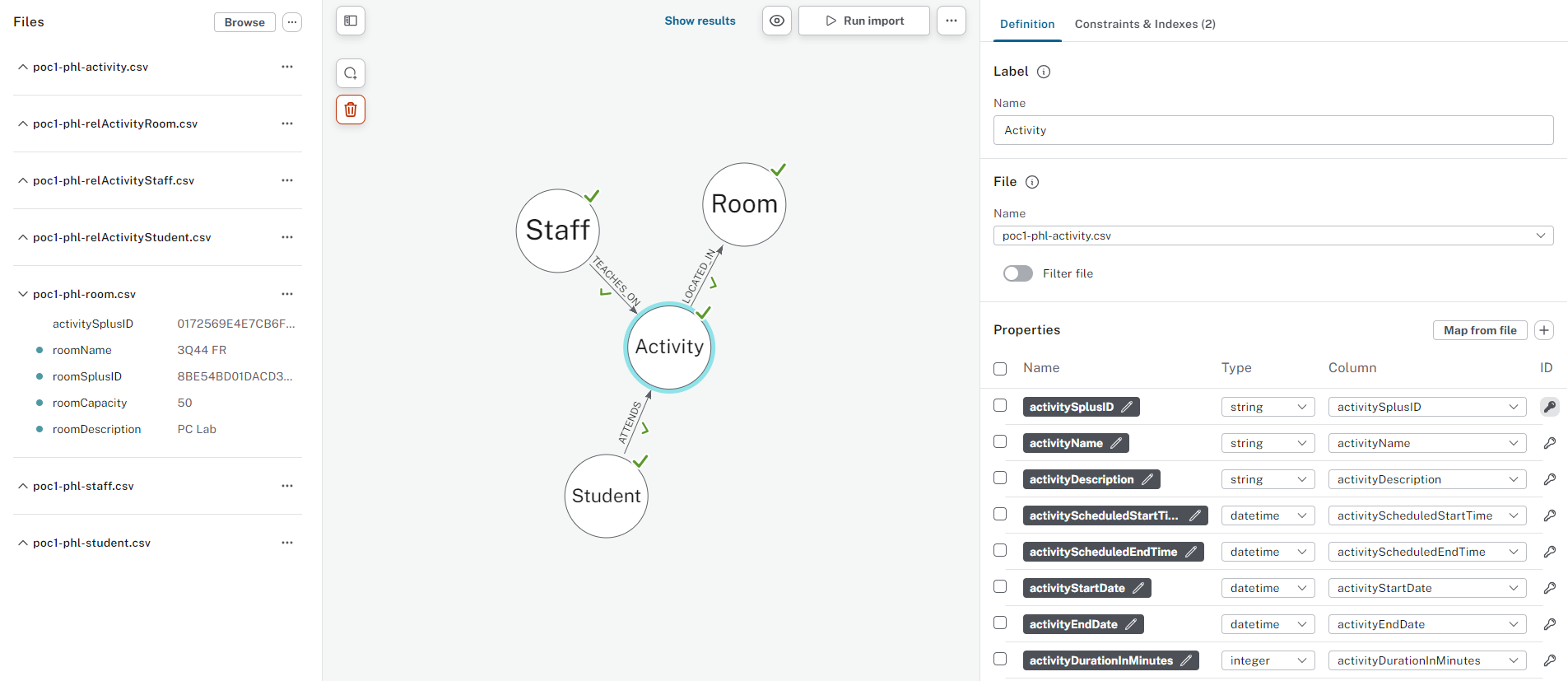
The core nodes are interconnected through relationships that reflect the dynamics of a timetable:

* (Student)-**[IS\_ALLOCATED\_TO]**->(Activity)
* (Staff)-**[TEACHES\_ON]**->(Activity)
* (Activity)-**[TAKES\_PLACE\_IN]**->(Room)

## MVP model



Core Nodes and Properties



Neo4j Interface showing basic nodes and properties

# Early Insights

## 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.

### Example code

**Busiest locations overall**

{cypher}{.scroll-cypher} 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, totalDurationInHours 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**

{cypher}{.scroll-cypher} 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. [[5]](#footnote-5)

{cypher}{.scroll-cypher} MATCH (s:student)-[:ATTENDS]->(a:activity) WITH s.stuProgName AS progName, s.stuID\_anon AS studentID, SUM(a.actDurationInMinutes) AS studentTotalDuration WITH progName, AVG(studentTotalDuration) / 60 AS progAverageHoursPerStudent, collect({studentID: studentID, studentTotalHours: studentTotalDuration / 60}) AS studentsData UNWIND studentsData AS studentData RETURN progName, progAverageHoursPerStudent, studentData.studentID AS studentID, studentData.studentTotalHours AS studentTotalHours, CASE WHEN studentData.studentTotalHours < (progAverageHoursPerStudent \* 0.9) THEN 'Below Average' WHEN studentData.studentTotalHours > (progAverageHoursPerStudent \* 1.1) THEN 'Above Average' ELSE 'Average' END AS compare

Using Percentage

| progName | progAverageHoursPerStudent | studentID | studentTotalHours | 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.

Using Standard Deviation

| progName | progAverageHoursPerStudent | progStDevHoursPerStudent | studentID | studentTotalHours | compare |
| --- | --- | --- | --- | --- | --- |
| “Maths NS” | 274.7692307692308 | 118.81231781219205 | “stu-23442558” | 361 | “Average” |
| “Maths NS” | 274.7692307692308 | 118.81231781219205 | “stu-91911371” | 126 | “Below Average” |
| “Maths NS” | 274.7692307692308 | 118.81231781219205 | “stu-42997469” | 251 | “Below Average” |

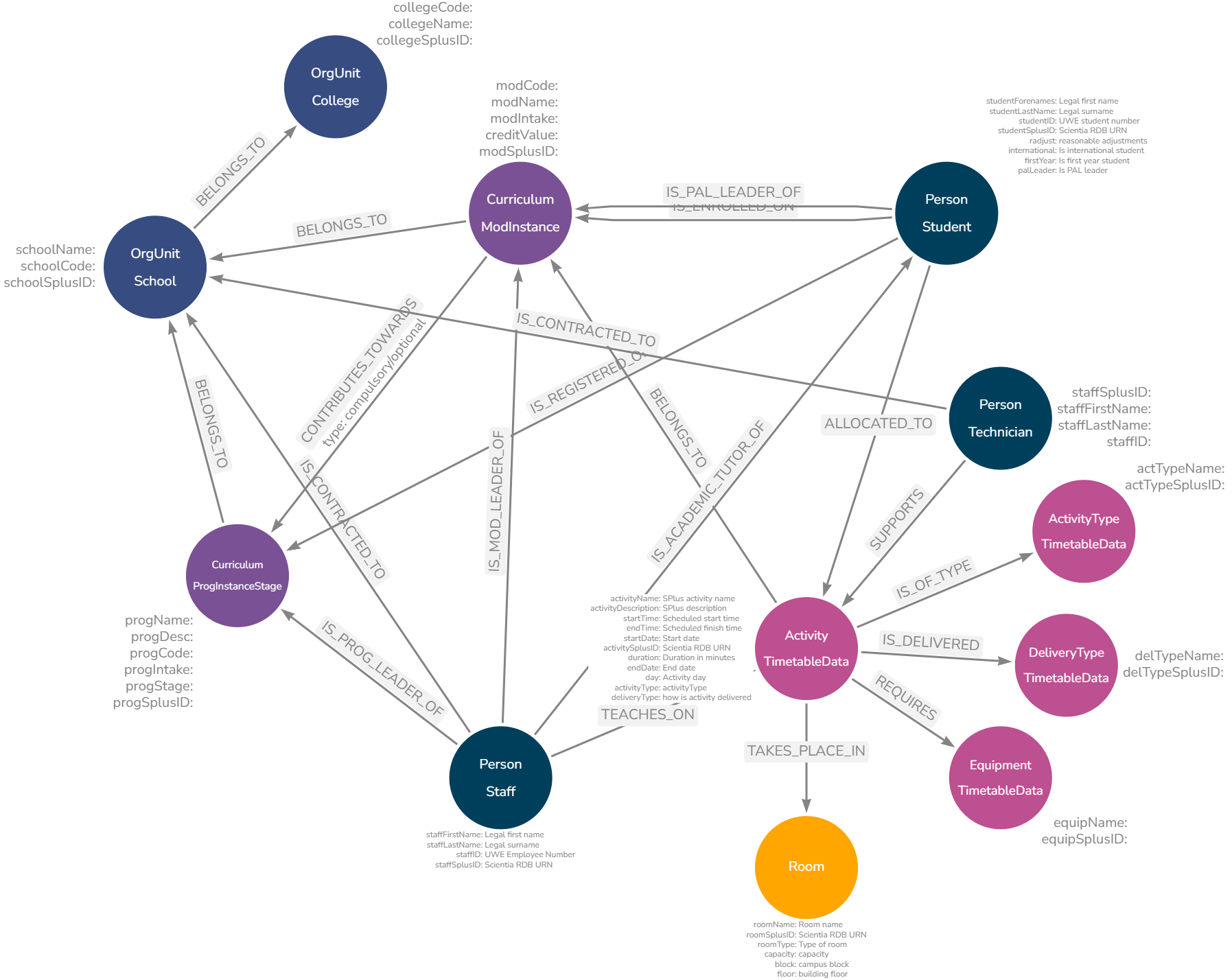
# Model Expansion

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:

### 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.



Example of Expanded Timetable Graph Model

# 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.

## The Problem of Normalised Time

Traditional relational databases often store timetable information in a highly normalised[[6]](#footnote-6) 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.

## Exploring Potential Solutions

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

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

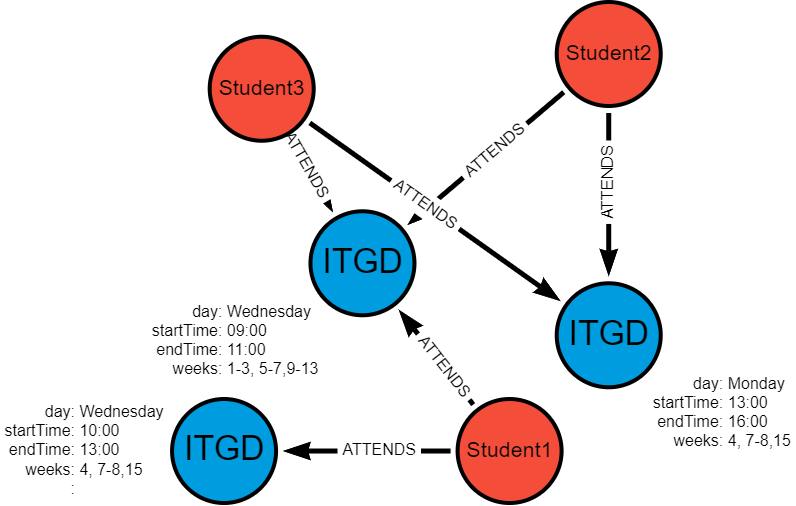
Example Source Data (Relational)

| Name | ActivityType | 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 |

### 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.

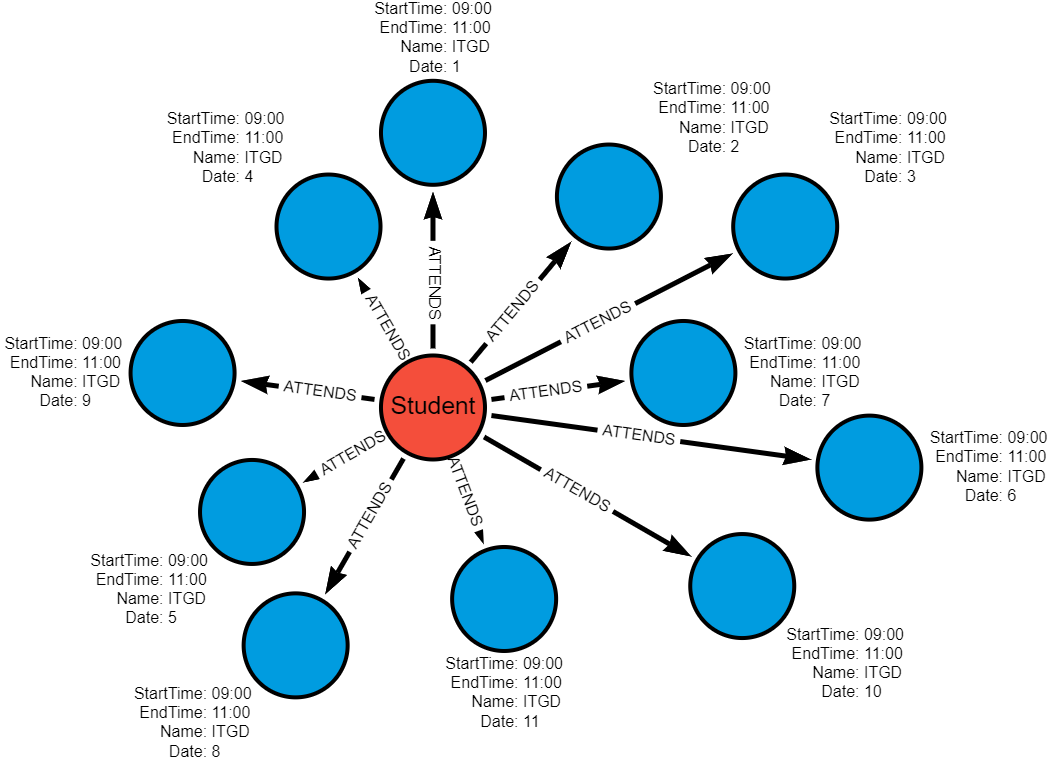


Basic example of Graphing Normalised Activities

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.

### 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.



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**:

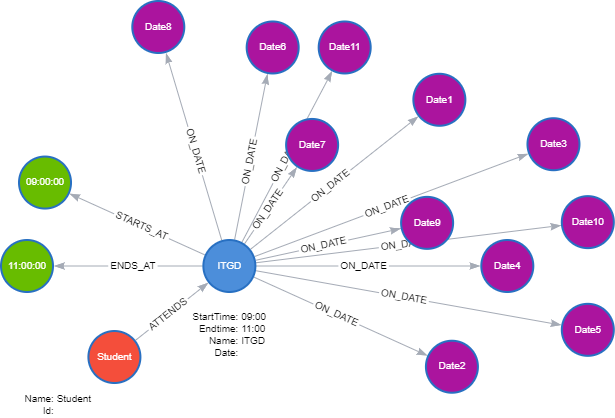
* *Node Proliferation*: Leads to a high volume of nodes, potentially impacting performance with large datasets.

**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.

### 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.



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**:

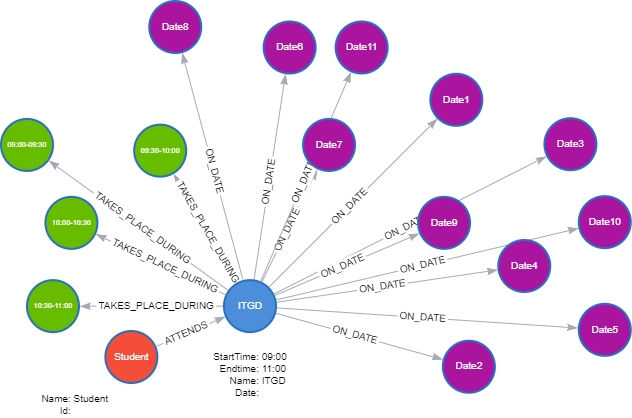
* *Increased Flexibility*: Facilitates queries across time ranges and aggregations across time slots.
* *Reduced Redundancy*: Avoids replicating time information for activities occurring on the same date and time.
* *Lower Node Count*: Potentially fewer nodes overall compared to Option 1 as date and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

### 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.



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.

### 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:

### Summary

Option summary

| # | 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.[[7]](#footnote-7)

# 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](https://www.microsoft.com/en-us/sql-server/sql-server-2019)) to a graph database ([Neo4j](https://neo4j.com/)).

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

## High-level Architecture

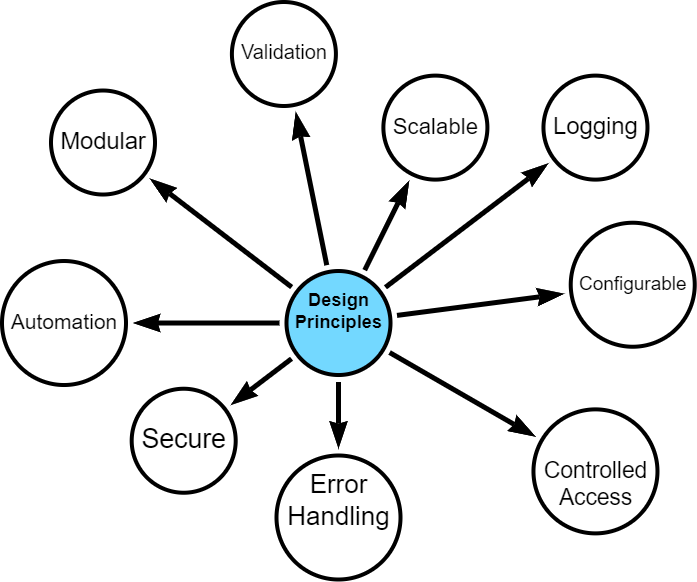
The data pipeline consists of these core stages:

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](https://neo4j.com/cloud/platform/aura-graph-database/) free instance).
4. **Loading**: Clean data is processed and loaded into Neo4j.

|  |
| --- |
| Data Pipeline Overview  Figure 11.1 |

## 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:

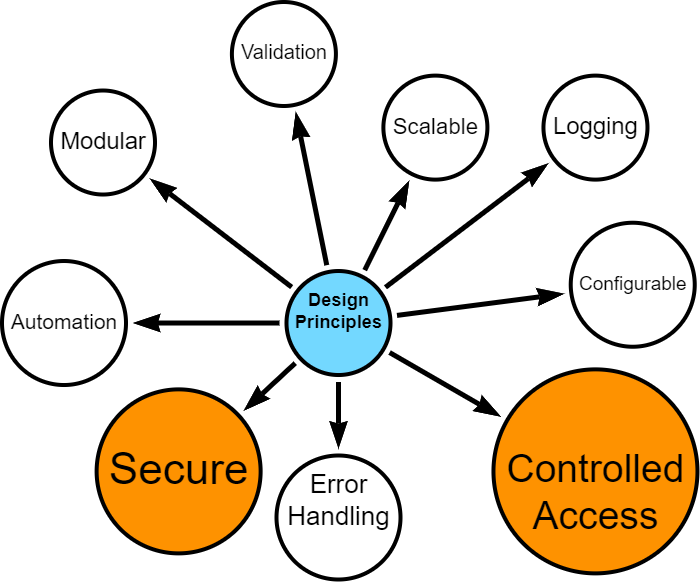


Design Principles

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.

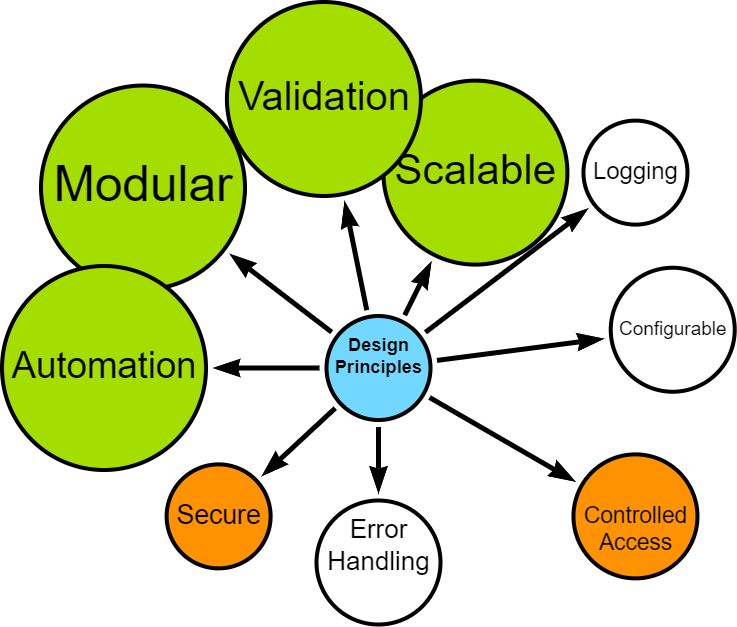
Given project constraints - deadline, word-limits, resources, data, technology - it is fair to say that compromises were made. That said, it was important that the final artefact is one that can be developed further for specific business use-cases.

### Security and Data Protection



* Secure access controls
* [Data anonymisation](appendix-anonymise.qmd)
* Controlled handling of personally identifiable information

### Modularity, Scalability and Automation



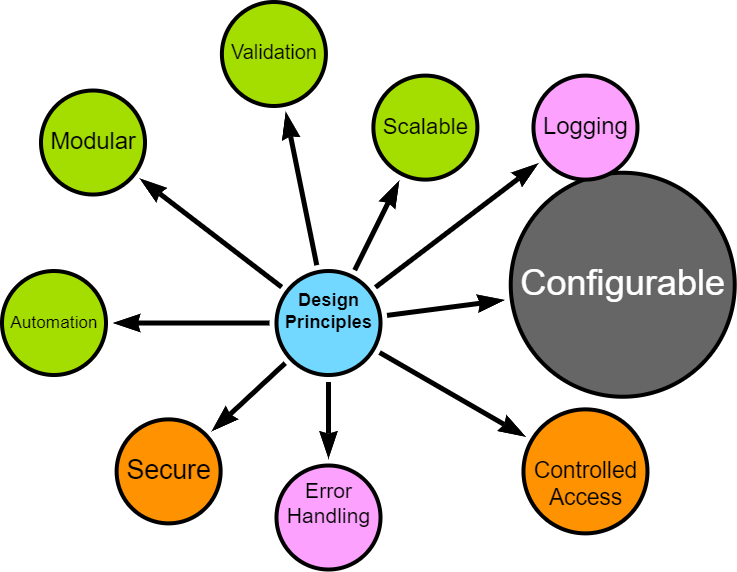
* Distinct, interoperable modules (extract, process, upload, load)
* Ability to handle increased data volume and complexity
* Automation, where possible
* Configurable data processing options (e.g., data chunking, row processing)
* Optimised, where possible

### Error Handling and Logging



* Robust error handling
* Comprehensive logging for troubleshooting and auditing

### User configurable



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

## 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](appendix-tech-stack.qmd) 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.

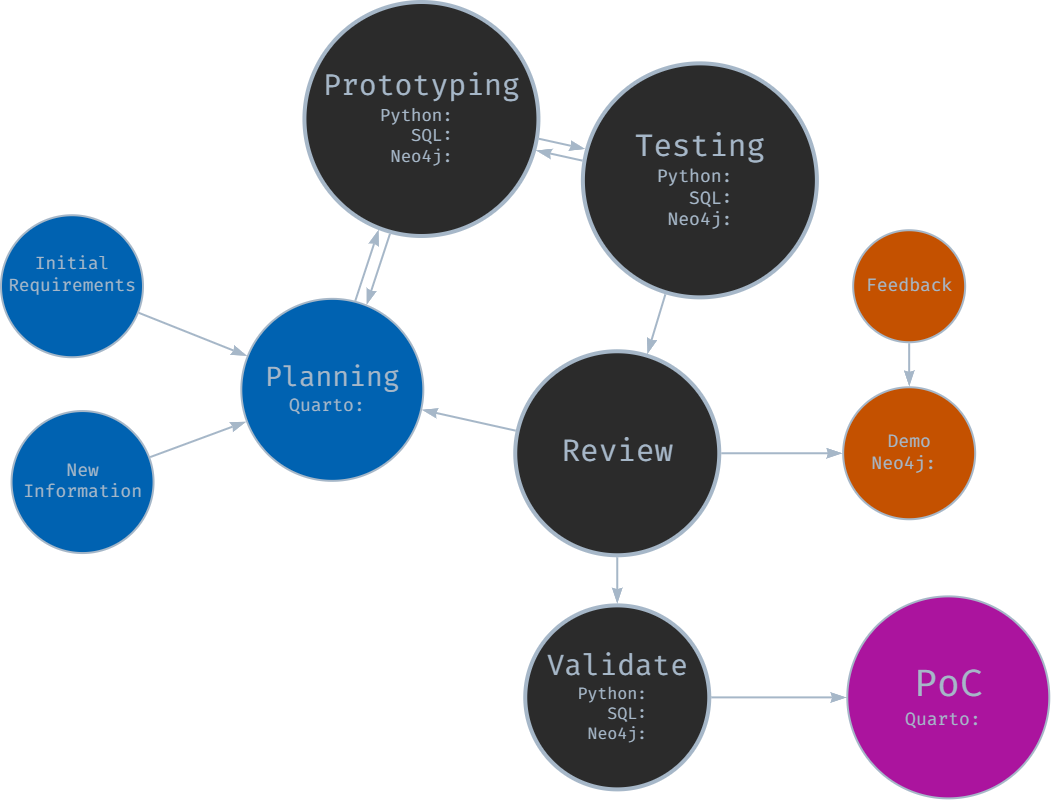
## Upcoming Sections

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.

# Data Engineering Approach

I followed an interative, 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.



Iterative Development Approach

### 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.

### 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.

### 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!

### 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!

### 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.

### 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.

# 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.

### 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](appendix-config.qmd)) 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.

### 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.

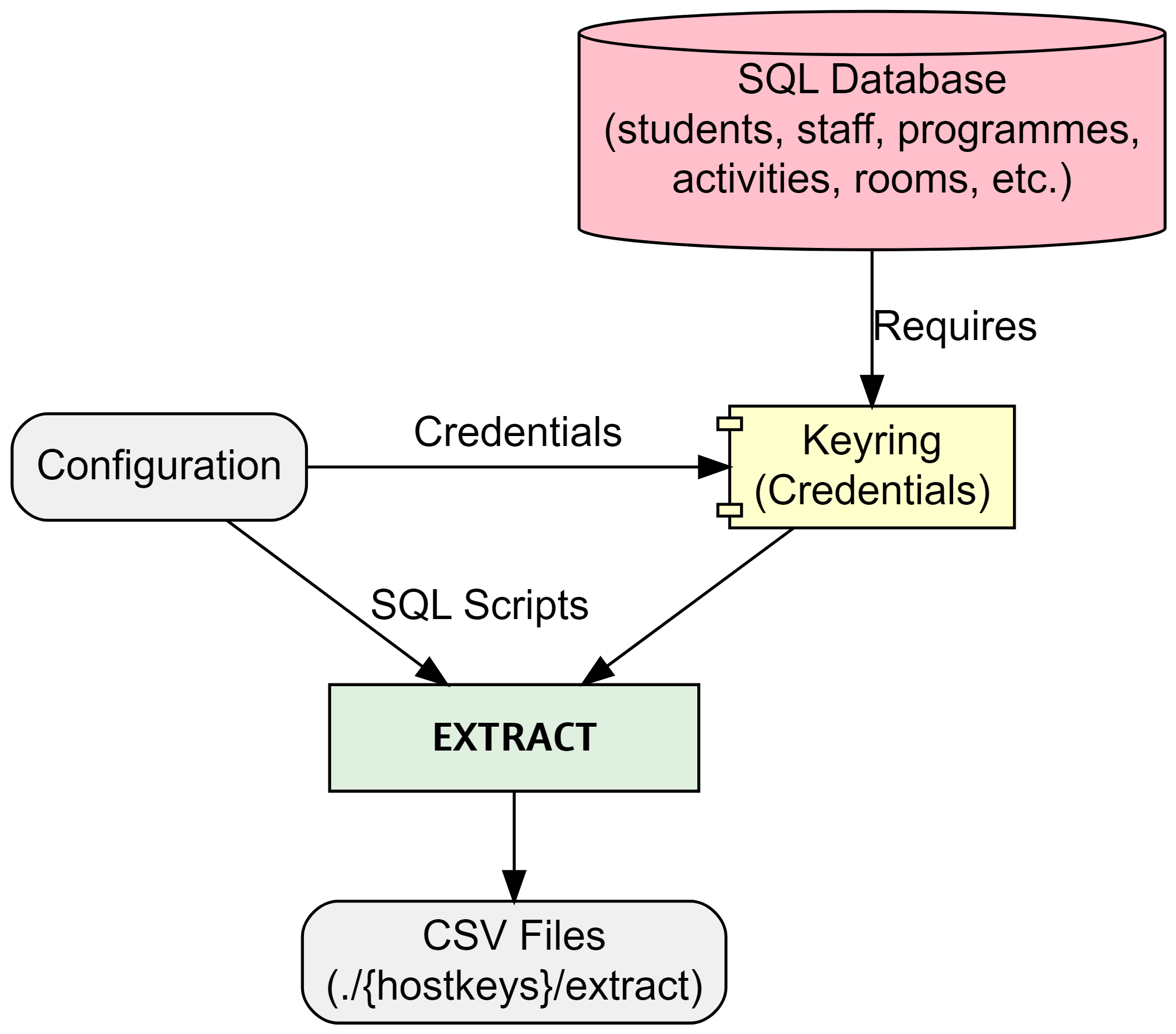
#### Example Extract Log

#### Example Google Drive Log

#### Example Process Log

#### Example Load Log

# Extraction



Extract

EXTRACT starts by securely connecting to the specified SQL database using encrypted credentials stored with [keyring](https://pypi.org/project/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 signficant 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.

### SQL example

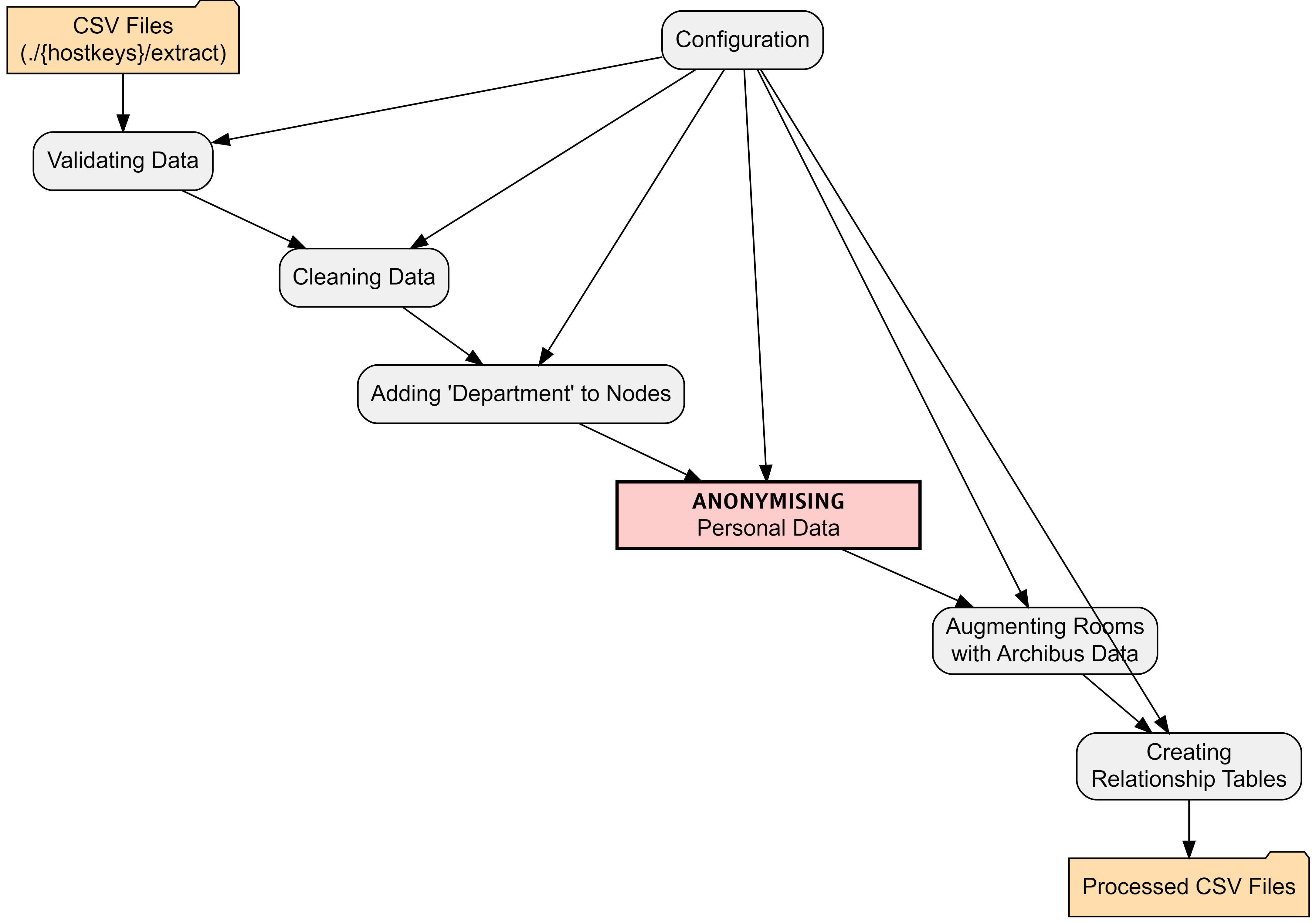
SELECT DISTINCT a.[Id] AS actSplusID,  
 CONCAT(a.[Id], '-', adt.[Week], '-', adt.[Day]) AS actGraphID,  
 a.[Name] AS actName,  
 a.[Description] AS actDescription,  
 a.[DepartmentId] AS actDeptSPlusID,  
 adt.[StartDateTime] AS actStartDateTime,  
 adt.[EndDateTime] AS actEndDateTime,  
 adt.[Week] 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];

### 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()

# 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.

## 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:

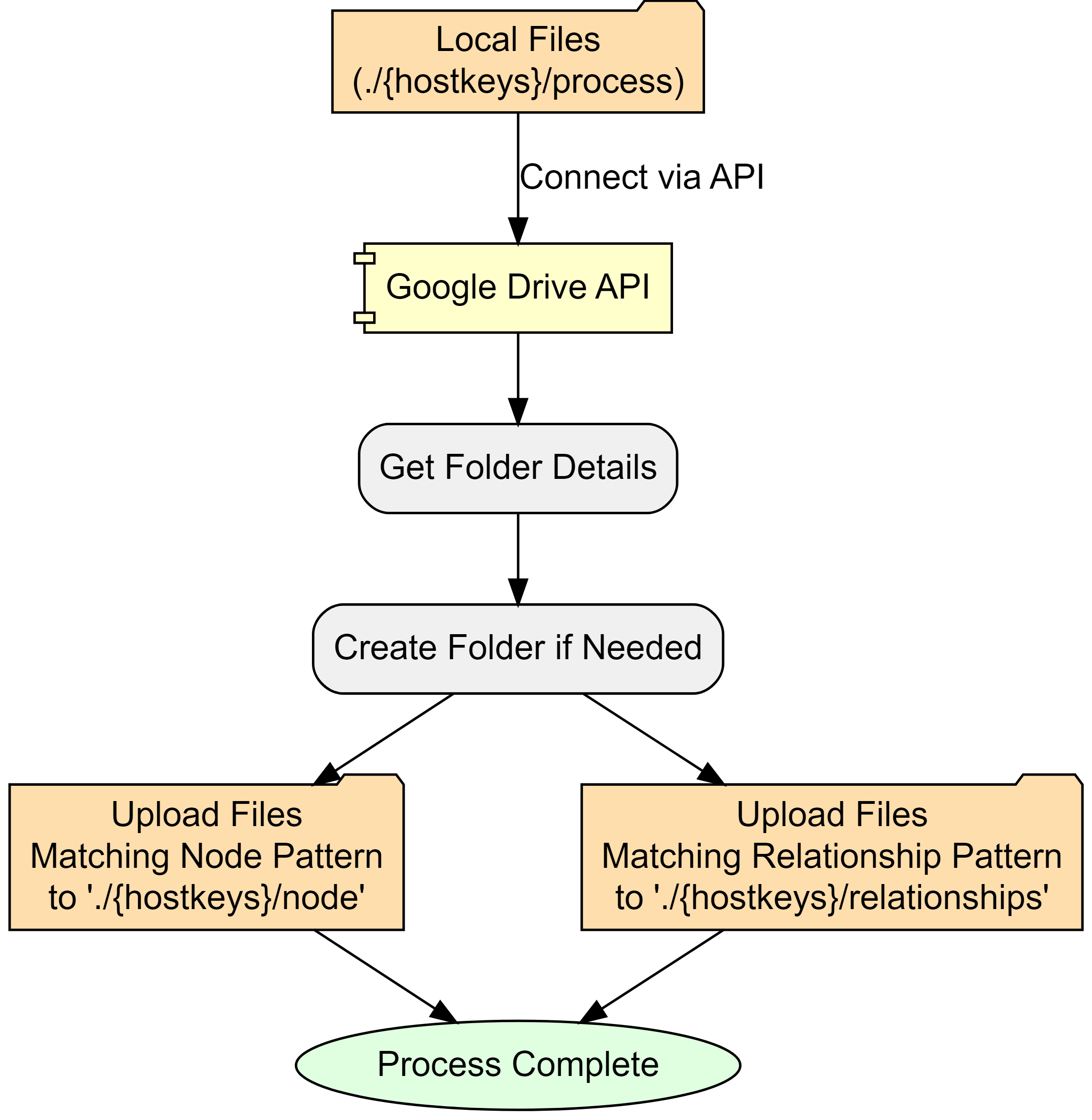
## Nodes and relationships

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](appendix-anonymise.qmd)
4. **Relationships** - Based on requirements in the configuration, relationships are extracted including optional relationship properties.

# Google Load

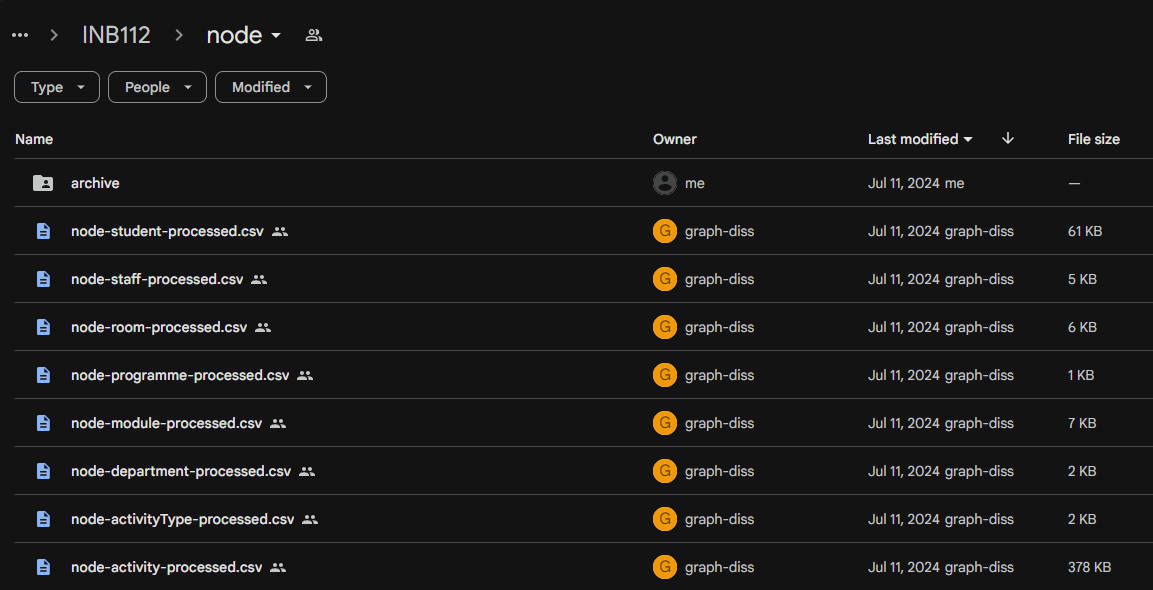
The free cloud instance of Neo4j (Aura) requires that csv files are stored in [public cloud storage](https://neo4j.com/docs/aura/aurads/importing-data/load-csv/) like Google Drive or Dropbox.

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:

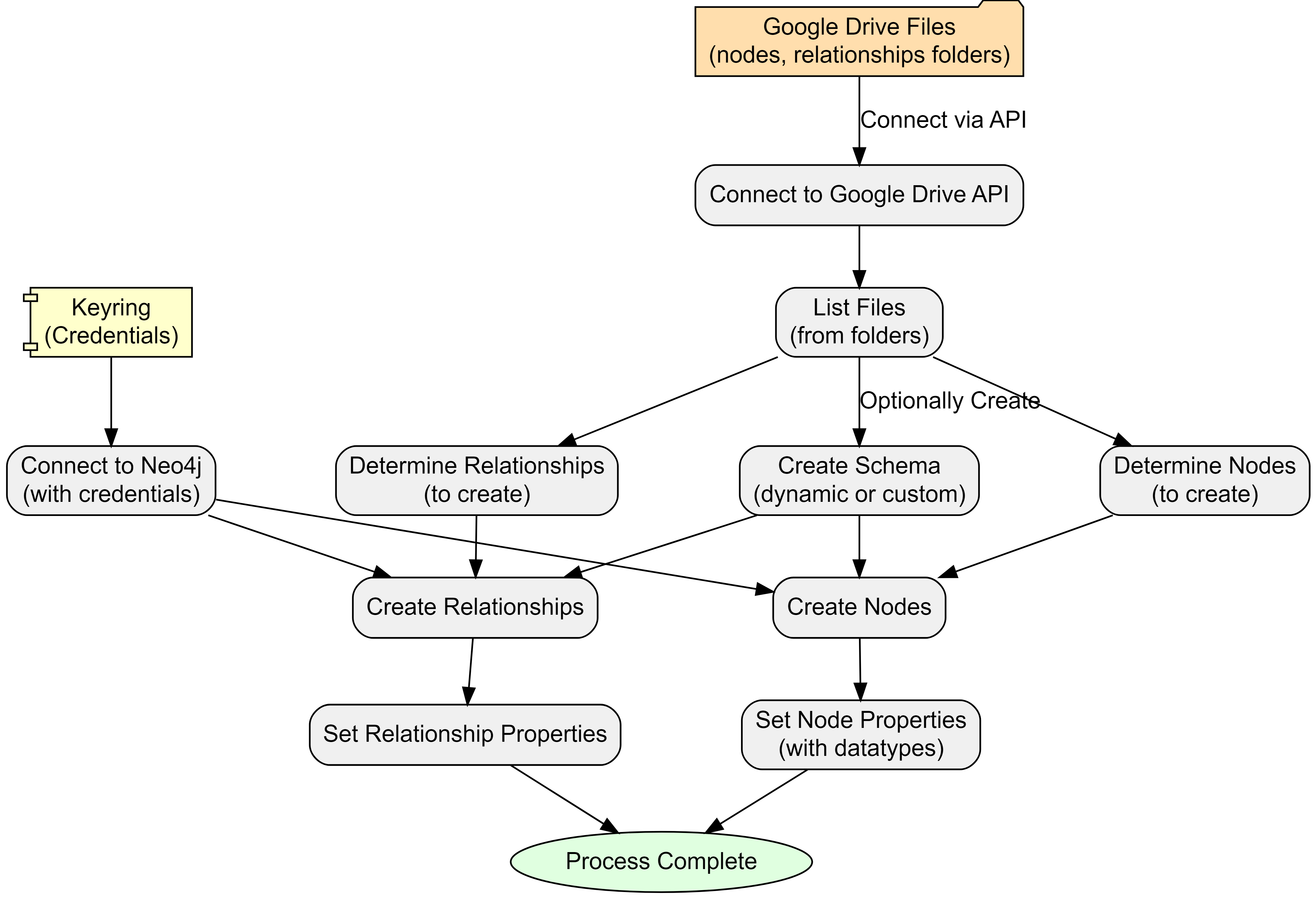
- root: Google Drive folder  
 - hostkeys (automatically created, unless override)   
 - nodes  
 - relationships



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

# 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.

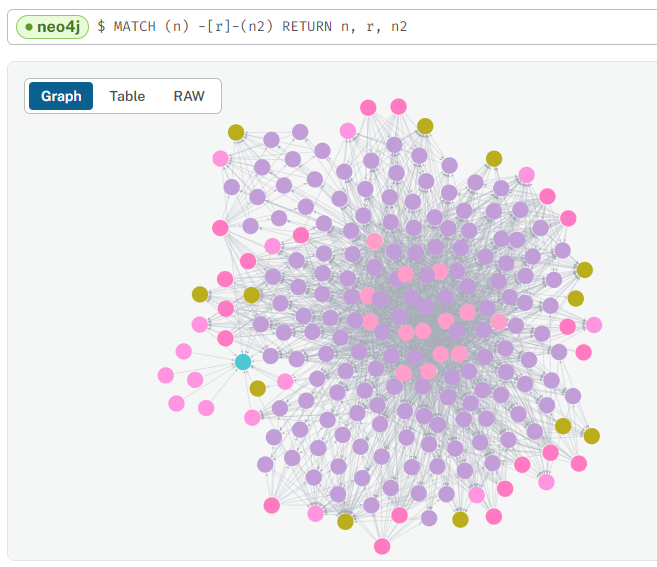
Nodes and relationships are dynamically processed by using a file-pattern matching approach. However, this can be overridden within configuration, if needed.

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!



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

# 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.” [[8]](#footnote-8) 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.

## 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.
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.

# 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.

### 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*.

### 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:

#### 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.

#### 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.

#### 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.

#### Data-driven decision making

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

#### 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.

### Implemented Metrics

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

#### 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

#### 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](appendix-cypher7.qmd) for more details.

#### 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.

#### 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.

# 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.

#### Student-level

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

#### Programme-level

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

#### Other groupings

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

# Implementing TQI

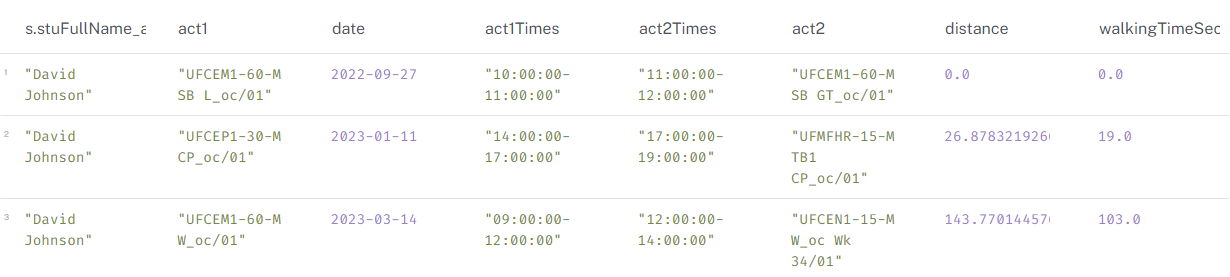
Prototype queries have been identified to identify constraint violations. Several of these queries are quite complex but their final form will dependent on the use-case as well as the graph data model.

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]->(r1:room),  
 (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 act1Times, a2.actStartTime+"-"+a2.actEndTime AS act2Times, a2.actName AS act2,  
 point.distance(r1.location, r2.location) AS distance,  
 round(point.distance(r1.location, r2.location) / 1.4) AS walkingTimeSeconds // Calculate walking time in seconds



Travel time between activities

See [Cypher Queries - Hard Constraints](appendix-cypher5.qmd) and [Cypher Queries - Soft Constraints](appendix-cypher6.qmd) for more examples.

### 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

# TQI Summary

### 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 [Appendix-perspectives.qmd] 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.

### 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.

### 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.

# 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**](02-02-graph-timetable.qmd) 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**](03-01-overview.qmd) 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**](04-timetable-metrics.qmd) 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](acknowledgements.qmd) I have received from many people along the way.

## 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.

## 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 Edducation 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[[9]](#footnote-9) - to illustrate, I have included some blue-skies thinking in the [Appendix](appendix-blue-sky.qmd).

# Appendix: Table of Contents

This project contains several appendices that provide additional information, resources and code.

The appendices are listed below:

A. [Random Graph Generator](appendix-random-graph.qmd) B. [Technology Stack](appendix-tech-stack.qmd) C. [Configuration](appendix-config.qmd) D. [Anonymisation](appendix-anonymise.qmd) E. [ETL Summary](appendix-code-intro.qmd) F. [ETL Code](appendix-code.qmd) G. [Config and Misc](appendix-code1-config.qmd) H. [Extract-SQL](appendix-code2-sql.qmd) I. [Extract](appendix-code3-extract.qmd) J. [Google Drive Load](appendix-code4-gdrive.qmd) K. [Transform](appendix-code5-transform.qmd) L. [Neo4j Load](appendix-code6-load.qmd) M. [Cypher Queries](appendix-cypher.qmd) N. [Creating Nodes and Relationships](appendix-cypher1.qmd) O. [Deleting Nodes and Relationships](appendix-cypher2.qmd) P. [General Queries](appendix-cypher3.qmd) Q. [Count Queries](appendix-cypher4.qmd) R. [Hard (timetabling) Constraints](appendix-cypher5.qmd) S. [Student Clashes](appendix-cypher5a.qmd) T. [Soft Constraints](appendix-cypher6.qmd) U. [Rooms and Spaces](appendix-cypher7.qmd) V. [Perspectives](appendix-perspectives.qmd) W. [Supervision](appendix-supervision.qmd) X. [Notes Example 1](appendix-supervision1.qmd) Y. [Notes Example 2](appendix-supervision2.qmd) Z. [Notes Example 3](appendix-supervision3.qmd)

# A: Random Graph Generator

The function below generates a random graph (dot file) using [Graphviz](https://graphviz.org/).

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 colours will be used.  
 """  
  
 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="1", fillcolor=color, style="filled")  
   
  
 # 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\_clusters - 1) # 80% chance of staying in cluster  
 src\_node = random.choice([node for i, node in enumerate(nodes) if cluster\_assignments[i] == src\_cluster])  
 dst\_node = random.choice([node for i, node in enumerate(nodes) if cluster\_assignments[i] == dst\_cluster])  
 edges.append((src\_node, dst\_node))  
  
 # edges to the graph  
 for edge in edges:  
 dot.edge(\*edge)  
  
 return dot

# 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:

### Programming

1. [Python](https://www.python.org/) - Main programming language.
2. [SQL](https://www.microsoft.com/en-us/sql-server/) - SELECT queries to extract source data from relational database.
3. [Cypher](https://neo4j.com/developer/cypher/) - Querying language for Neo4j Graph Databases.
4. [Batch](https://docs.microsoft.com/en-us/windows-server/administration/windows-commands/windows-commands) - Windows command language to handle yaml files and multi-format rendering.
5. [VSCode](https://code.visualstudio.com/) - Main IDE (Integrated Development Environment).

### Documentation

* [Quarto](https://quarto.org/) - Open-source technical publishing system.
* [Jupyter](https://jupyter.org/) - Open-source application for interactive notebooks.
* [Zotero](https://www.zotero.org/) - Open-source reference management system.

### Visualisation

* [Graphviz](https://graphviz.org/) - Open-source graph visualisation application.
* [Mermaid](https://mermaid-js.github.io/mermaid/) - Open-source JavaScript diagramming tool.
* [Arrows](https://neo4j.com/labs/arrows/) - Neo4j Labs diagramming tool.

### Versioning

* [Github](https://github.com/) - Web-based platform for version control and collaboration using Git.

### Python Libraries

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

#### Directory/File Handling

* [os](https://docs.python.org/3/library/os.html) - Interacting with the operating system for tasks like creating, deleting, and navigating directories and files.
* [glob](https://docs.python.org/3/library/glob.html) - Finding files and directories based on pattern matching.
* [io](https://docs.python.org/3/library/io.html) - Working with input/output streams for reading and writing data.

#### Data Handling

* [pandas](https://pandas.pydata.org/) - Handling tabular data for analysis and manipulation.
* [json](https://docs.python.org/3/library/json.html) - Encoding and decoding JSON (JavaScript Object Notation) data.

#### Typing and Logging

* [typing](https://docs.python.org/3/library/typing.html) - Adding type hints to code for better code readability, maintainability, and static type checking.
* [logging](https://docs.python.org/3/library/logging.html) - Configuring and managing logging for application.
* [time](https://docs.python.org/3/library/time.html) - Working with time-related functions, potentially used for logging timestamps.

#### Database Connectivity

* [keyring](https://keyring.readthedocs.io/en/latest/) - Securely storing and retrieving passwords and other sensitive information.
* [pyodbc](https://github.com/mkleehammer/pyodbc/wiki) - Connecting to and interacting with SQL databases using the Open Database Connectivity (ODBC) standard.
* [neo4j](https://neo4j.com/docs/api/python-driver/current/) - Interacting with Neo4j graph databases.

#### Google API Integration

* [googleapiclient](https://googleapis.dev/python/google-api-core/latest/index.html) - Interacting with various Google APIs.
* [google.oauth2](https://google-auth.readthedocs.io/en/latest/) - Handling OAuth 2.0 authentication for accessing Google services securely.

#### Anonymisation

* [random](https://docs.python.org/3/library/random.html) - Generating random numbers and making random choices.
* [hashlib](https://docs.python.org/3/library/hashlib.html) - Implementing various secure hash and message digest algorithms.
* [Faker](https://faker.readthedocs.io/en/master/) - Generating fake data for testing and development purposes.

# Configuration YAML

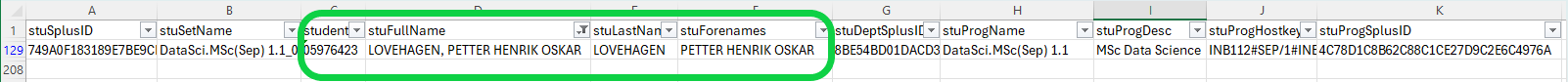
The below is an example of configuration options configured in more human readable YAML format.

# 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/drive/folders/1Rc3vQCF6CwxV3yNjfUTWXv61BgYD1j\_3"  
 relationships\_folder\_url: # (Optional) override for dynamic lookup) eg."https://drive.google.com/drive/folders/1w\_ea6ETzRcdYz71crLxL9khjLrEfcbuH"  
 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: ''  
 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"

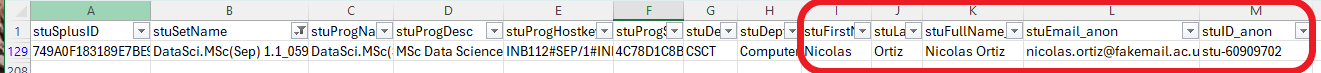
# Anonymisation

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.



Pre-anonymisation Extract



Post-anonymisation Extract

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

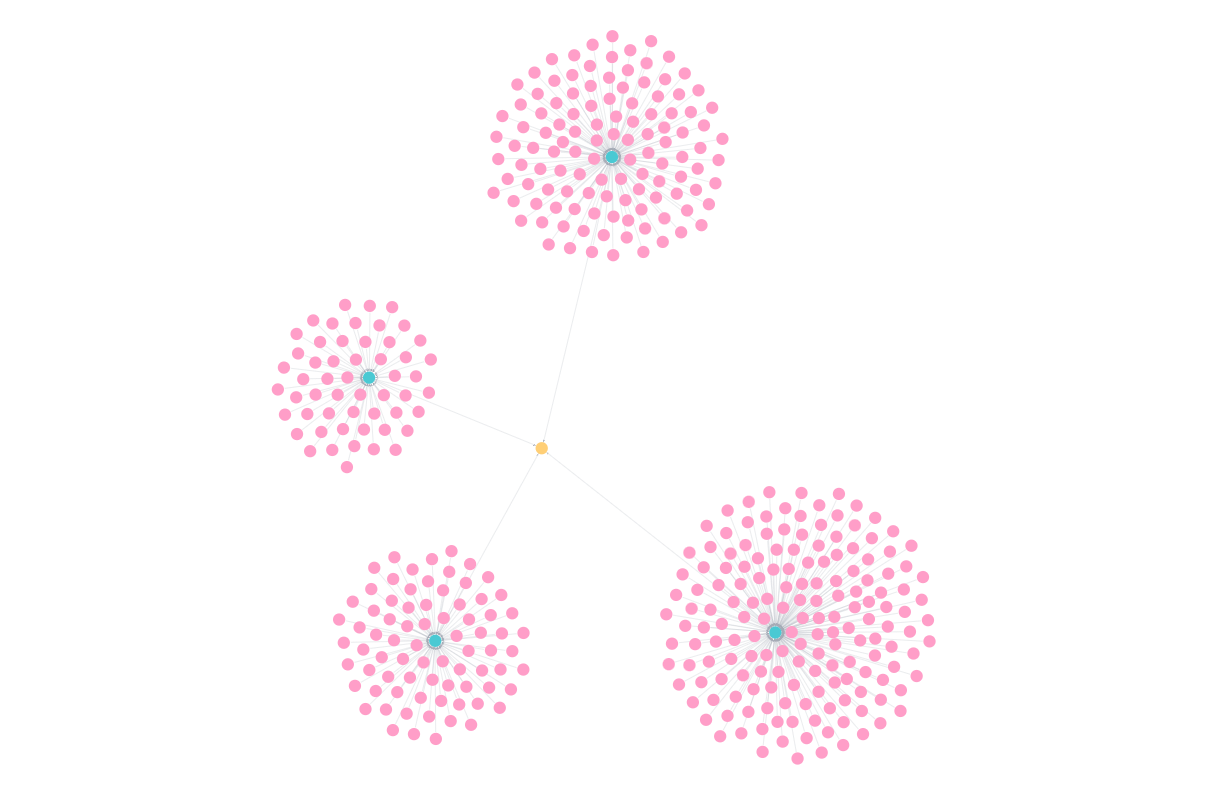
# ETL Summary

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.

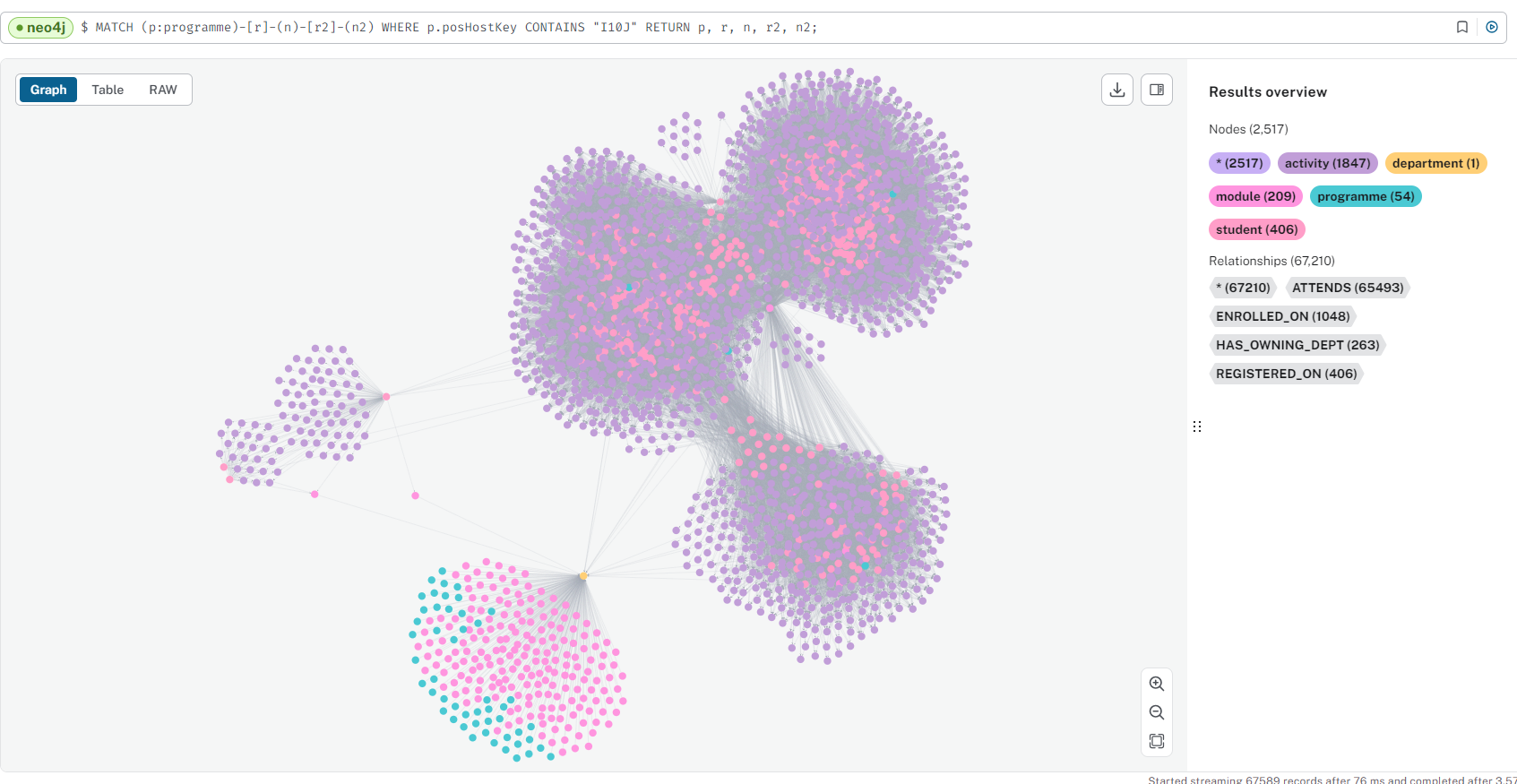
The table below summarises the time taken for each programme.

| 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.



Computer Science (I10J) - Department-Programme-Students



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 |

# 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**](https://gist.github.com/zoonalink/818cae9f20db1f1417725a1f297ff061)

[**Graph Timetable - Quarto - SQL Queries**](https://gist.github.com/zoonalink/4ba9fa5bf75ac21c1537d8176ee5c700)

[**Graph Timetable - Quarto - Extract**](https://gist.github.com/zoonalink/d3bafb3100a162c50694be699bfc8e3f)

[**Graph Timetable - Quarto - Transform**](https://gist.github.com/zoonalink/125673550192ff7515628b7d00879114)

[**Graph Timetable - Quarto - Upload to Google Drive**](https://gist.github.com/zoonalink/450bff1a92d2efcde58ab4b9edc3e7ae)

[**Graph Timetable - Quarto - Load**](https://gist.github.com/zoonalink/ce5e1f3e578a5eeed05144f3676af409)

[**Graph Timetable - Quarto - Config and Misc**](https://gist.github.com/zoonalink/818cae9f20db1f1417725a1f297ff061)

[**Graph Timetable - Quarto - SQL Queries**](https://gist.github.com/zoonalink/4ba9fa5bf75ac21c1537d8176ee5c700)

[**Graph Timetable - Quarto - Extract**](https://gist.github.com/zoonalink/d3bafb3100a162c50694be699bfc8e3f)

[**Graph Timetable - Quarto - Transform**](https://gist.github.com/zoonalink/125673550192ff7515628b7d00879114)

[**Graph Timetable - Quarto - Upload to Google Drive**](https://gist.github.com/zoonalink/450bff1a92d2efcde58ab4b9edc3e7ae)

[**Graph Timetable - Quarto - Load**](https://gist.github.com/zoonalink/ce5e1f3e578a5eeed05144f3676af409)

# Config and Misc

[**Graph Timetable - Quarto - Config and Misc**](https://gist.github.com/zoonalink/818cae9f20db1f1417725a1f297ff061)

1. config.py
2. logger\_config.py
3. neo4j\_schema.py
4. utils.py
5. connect\_to\_neo4j.py
6. connect\_to\_rdb.py

# Extract-SQL

[**Graph Timetable - Quarto - SQL Queries**](https://gist.github.com/zoonalink/4ba9fa5bf75ac21c1537d8176ee5c700)

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

# Extract

[**Graph Timetable - Quarto - Extract**](https://gist.github.com/zoonalink/d3bafb3100a162c50694be699bfc8e3f)

1. temp\_table\_loader
2. extract\_sql\_file
3. extract\_data
4. extract\_main

# Google Drive Load

[**Graph Timetable - Quarto - Upload to Google Drive**](https://gist.github.com/zoonalink/450bff1a92d2efcde58ab4b9edc3e7ae)

1. gdrive\_upload
2. google\_drive\_utils

# Process

[**Graph Timetable - Quarto - Process**](https://gist.github.com/zoonalink/125673550192ff7515628b7d00879114)

1. process\_utils
2. process\_node
3. process\_main

# Neo4j Load

[**Graph Timetable - Quarto - Load**](https://gist.github.com/zoonalink/ce5e1f3e578a5eeed05144f3676af409)

1. load\_schema
2. load\_utils
3. load\_nodes
4. load\_relationships
5. convert\_properties
6. set\_display\_properties
7. load\_main

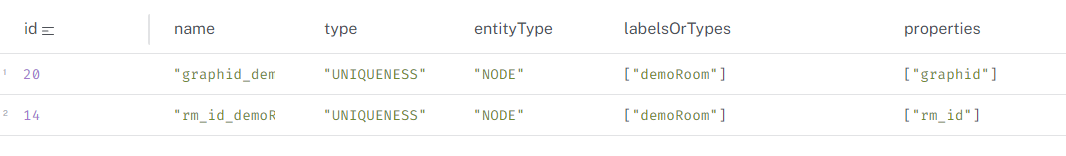
# Cypher Queries

These pages collate Cypher[[10]](#footnote-10) 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.

## 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.



Constraints

## 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.

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 0x00000137D0AB1190>  
Running query...  
  
<Record id=7 name='activity\_clash\_index' state='ONLINE' populationPercent=100.0 type='RANGE' entityType='NODE' labelsOrTypes=['activity'] properties=['actStartDate', 'actStartTime', 'actEndTime'] indexProvider='range-1.0' owningConstraint=None lastRead=None readCount=0>  
<Record id=19 name='graphid\_demoRoom\_uniq' state='ONLINE' populationPercent=100.0 type='RANGE' entityType='NODE' labelsOrTypes=['demoRoom'] properties=['graphid'] indexProvider='range-1.0' owningConstraint='graphid\_demoRoom\_uniq' lastRead=neo4j.time.DateTime(2024, 8, 20, 18, 16, 44, 231000000, tzinfo=<UTC>) readCount=3>  
<Record id=4 name='index\_12b79beb' state='ONLINE' populationPercent=100.0 type='RANGE' entityType='NODE' labelsOrTypes=['activity'] properties=['actEndTime'] indexProvider='range-1.0' owningConstraint=None lastRead=None readCount=0>  
<Record id=6 name='index\_207d313' state='ONLINE' populationPercent=100.0 type='RANGE' entityType='NODE' labelsOrTypes=['activity'] properties=['id'] indexProvider='range-1.0' owningConstraint=None lastRead=None readCount=0>  
<Record id=8 name='index\_241bd22f' state='ONLINE' populationPercent=100.0 type='RANGE' entityType='NODE' labelsOrTypes=['room'] properties=['roomName'] indexProvider='range-1.0' owningConstraint=None lastRead=neo4j.time.DateTime(2024, 8, 21, 12, 10, 21, 129000000, tzinfo=<UTC>) readCount=152>  
<Record id=9 name='index\_2d375d70' state='ONLINE' populationPercent=100.0 type='RANGE' entityType='NODE' labelsOrTypes=['room'] properties=['roomLatitude'] indexProvider='range-1.0' owningConstraint=None lastRead=neo4j.time.DateTime(2024, 8, 15, 18, 40, 6, 645000000, tzinfo=<UTC>) readCount=107>  
<Record id=0 name='index\_343aff4e' state='ONLINE' populationPercent=100.0 type='LOOKUP' entityType='NODE' labelsOrTypes=None properties=None indexProvider='token-lookup-1.0' owningConstraint=None lastRead=neo4j.time.DateTime(2024, 8, 21, 12, 8, 48, 160000000, tzinfo=<UTC>) readCount=117812>  
<Record id=10 name='index\_43c5c824' state='ONLINE' populationPercent=100.0 type='RANGE' entityType='NODE' labelsOrTypes=['room'] properties=['roomLongitude'] indexProvider='range-1.0' owningConstraint=None lastRead=neo4j.time.DateTime(2024, 8, 15, 18, 40, 6, 646000000, tzinfo=<UTC>) readCount=106>  
<Record id=2 name='index\_6acb9a84' state='ONLINE' populationPercent=100.0 type='RANGE' entityType='NODE' labelsOrTypes=['activity'] properties=['actStartDate'] indexProvider='range-1.0' owningConstraint=None lastRead=neo4j.time.DateTime(2024, 8, 21, 9, 9, 7, 681000000, tzinfo=<UTC>) readCount=116>  
<Record id=5 name='index\_7ade165f' state='ONLINE' populationPercent=100.0 type='RANGE' entityType='NODE' labelsOrTypes=['activity'] properties=['actModSplusID'] indexProvider='range-1.0' owningConstraint=None lastRead=neo4j.time.DateTime(2024, 8, 15, 18, 40, 6, 638000000, tzinfo=<UTC>) readCount=148>  
<Record id=1 name='index\_f7700477' state='ONLINE' populationPercent=100.0 type='LOOKUP' entityType='RELATIONSHIP' labelsOrTypes=None properties=None indexProvider='token-lookup-1.0' owningConstraint=None lastRead=neo4j.time.DateTime(2024, 8, 15, 13, 53, 33, 557000000, tzinfo=<UTC>) readCount=134>  
<Record id=3 name='index\_f86013e' state='ONLINE' populationPercent=100.0 type='RANGE' entityType='NODE' labelsOrTypes=['activity'] properties=['actStartTime'] indexProvider='range-1.0' owningConstraint=None lastRead=neo4j.time.DateTime(2024, 8, 21, 9, 8, 41, 307000000, tzinfo=<UTC>) readCount=18>  
<Record id=18 name='lat\_demoRoom' state='ONLINE' populationPercent=100.0 type='RANGE' entityType='NODE' labelsOrTypes=['demoRoom'] properties=['lat'] indexProvider='range-1.0' owningConstraint=None lastRead=None readCount=0>  
<Record id=17 name='lon\_demoRoom' state='ONLINE' populationPercent=100.0 type='RANGE' entityType='NODE' labelsOrTypes=['demoRoom'] properties=['lon'] indexProvider='range-1.0' owningConstraint=None lastRead=None readCount=0>  
<Record id=16 name='rm\_cat\_demoRoom' state='ONLINE' populationPercent=100.0 type='RANGE' entityType='NODE' labelsOrTypes=['demoRoom'] properties=['rm\_cat'] indexProvider='range-1.0' owningConstraint=None lastRead=neo4j.time.DateTime(2024, 8, 20, 18, 16, 44, 250000000, tzinfo=<UTC>) readCount=3>  
<Record id=15 name='rm\_type\_demoRoom' state='ONLINE' populationPercent=100.0 type='RANGE' entityType='NODE' labelsOrTypes=['demoRoom'] properties=['rm\_type'] indexProvider='range-1.0' owningConstraint=None lastRead=neo4j.time.DateTime(2024, 8, 20, 18, 16, 44, 258000000, tzinfo=<UTC>) readCount=3>  
<Record id=11 name='room\_fulltext\_index' state='ONLINE' populationPercent=100.0 type='FULLTEXT' entityType='NODE' labelsOrTypes=['room'] properties=['roomName', 'roomDescription', 'roomType', 'roomCapacity'] indexProvider='fulltext-1.0' owningConstraint=None lastRead=neo4j.time.DateTime(2024, 8, 20, 18, 16, 43, 786000000, tzinfo=<UTC>) readCount=99>  
<Record id=12 name='room\_geo\_index' state='ONLINE' populationPercent=100.0 type='FULLTEXT' entityType='NODE' labelsOrTypes=['room'] properties=['roomLatitude', 'roomLongitude'] indexProvider='fulltext-1.0' owningConstraint=None lastRead=neo4j.time.DateTime(2024, 8, 20, 18, 16, 43, 803000000, tzinfo=<UTC>) readCount=12>

### 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.actEndTime);  
  
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)

# 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.

## 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

### Example: Creating a Student Node

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

## 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

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

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

## 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. |

### (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

### (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

### (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

### (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

### (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

# Deleting Nodes and Relationships

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

## 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

### Example: Deleting a Student Node

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

## 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

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

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

# General Queries

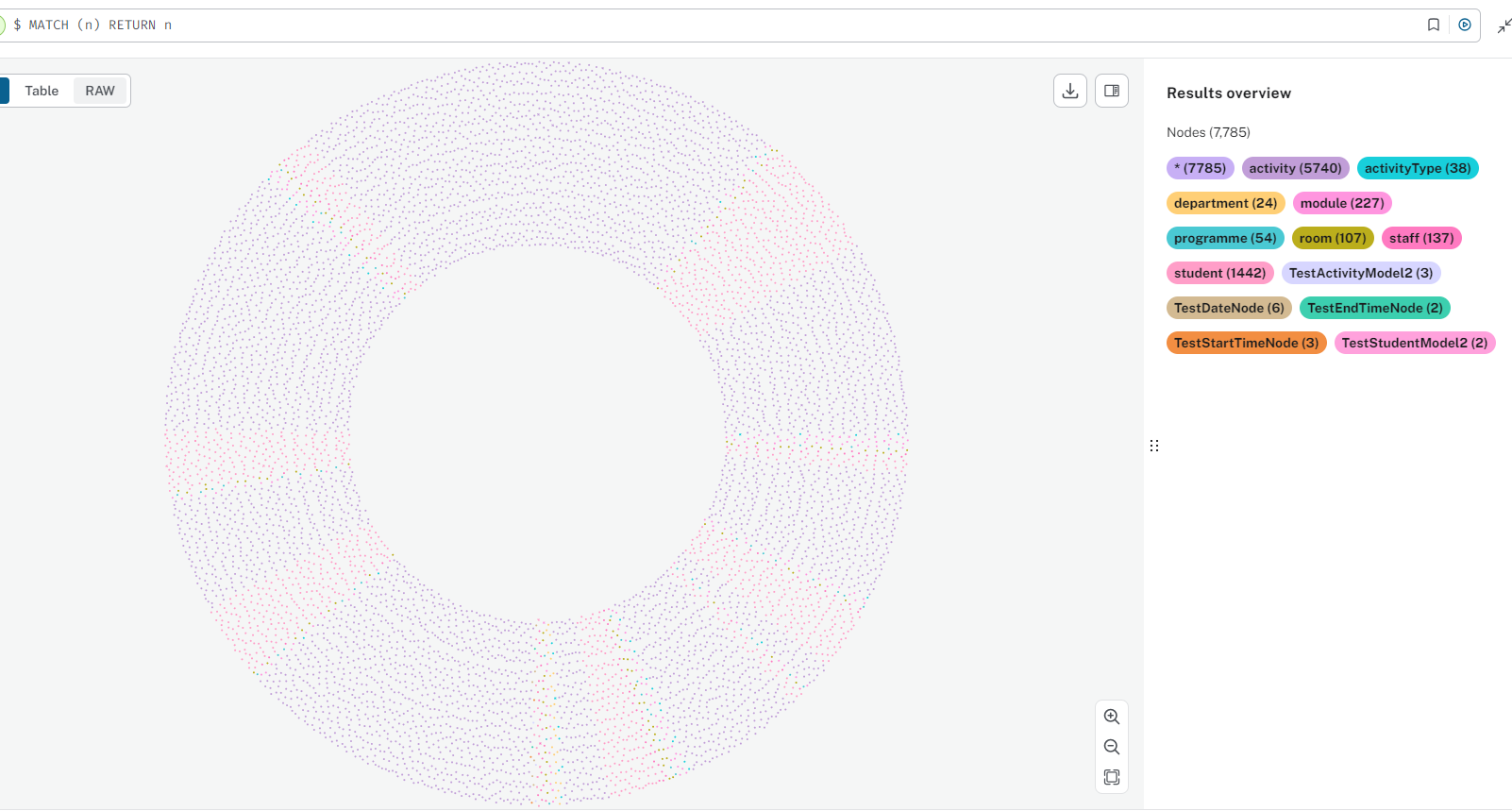
This page contains a *selection* general queries which can be used to explore the graph database. The queries are designed to provide insights into the data and relationships between nodes.

### 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

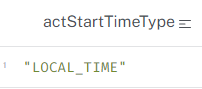


All Nodes

### 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

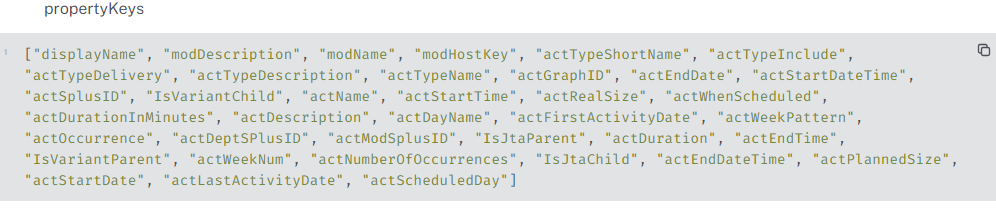


Datatype of Property

### 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



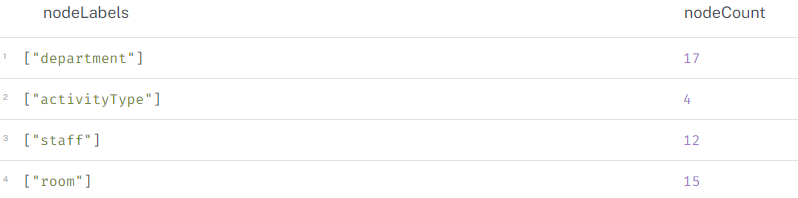
Unique Property Values

### 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

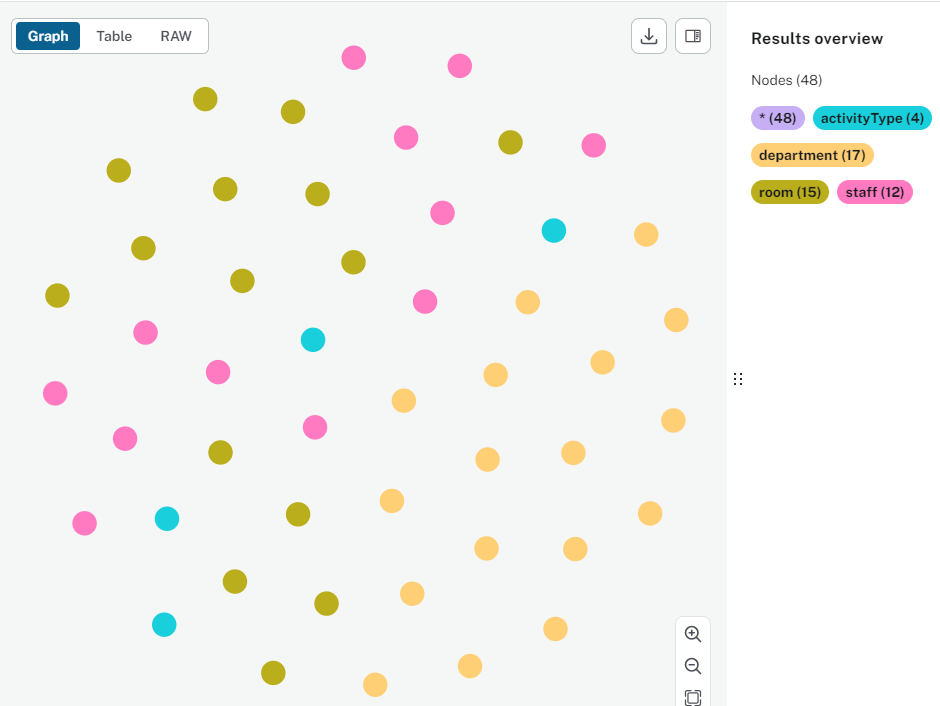


Node labels without Relationships

### 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

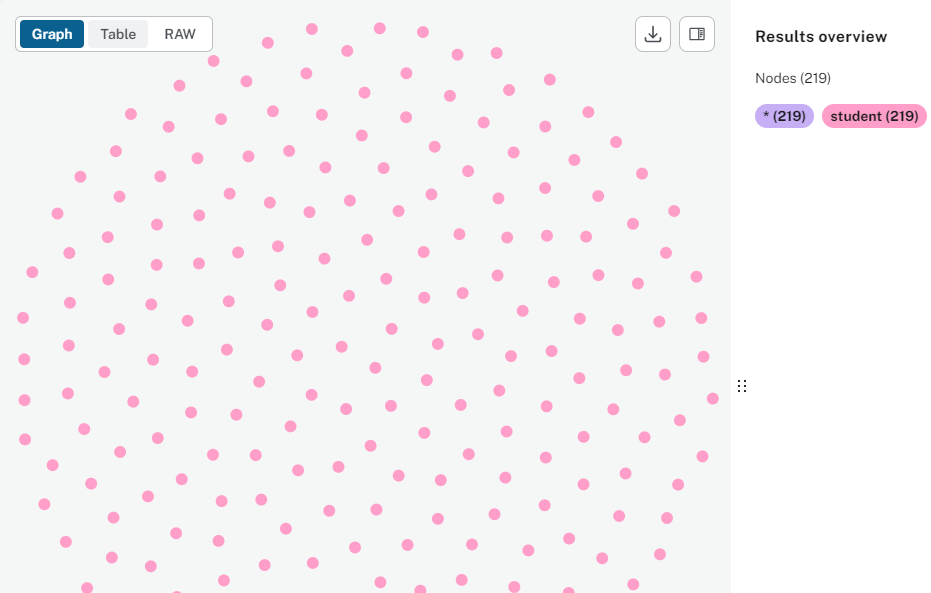


Nodes Without Relationships

### 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

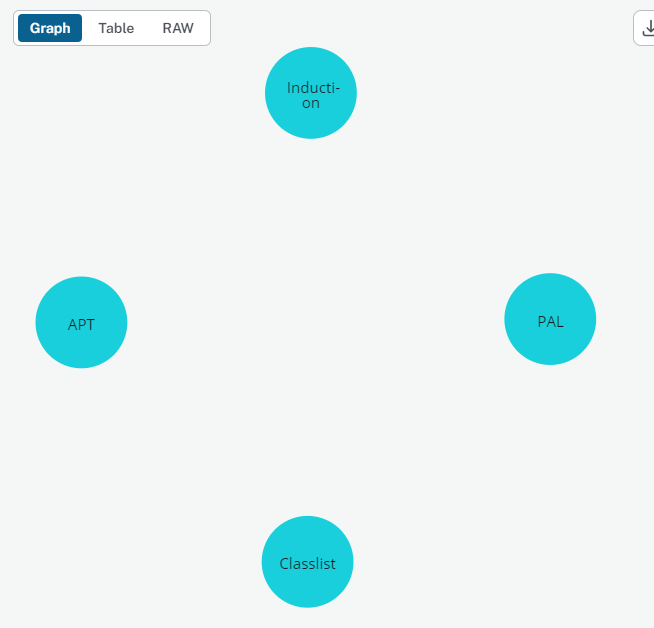


Students Without Activities

### 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 typewithout 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;

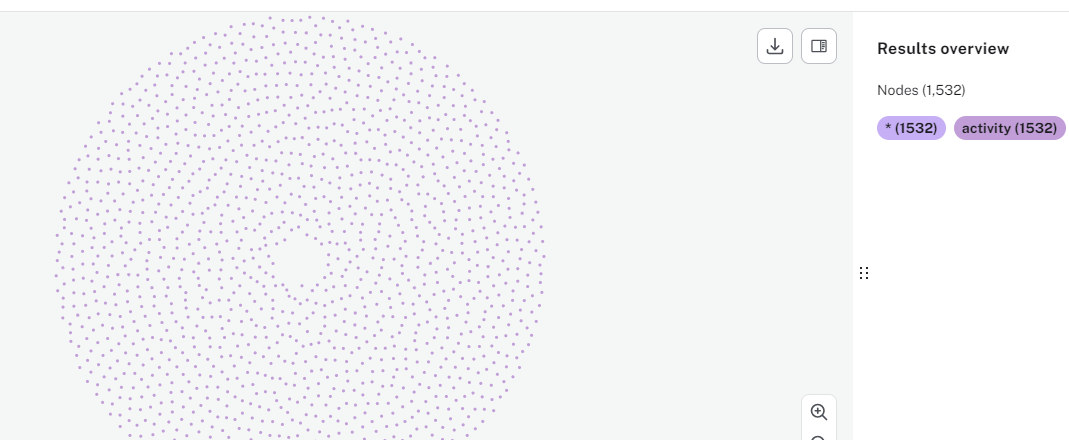


ActivityType Without Room

### 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;



Activities Without Rooms

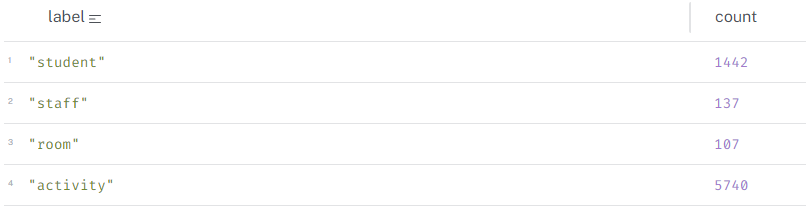
# Count Queries

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.

### Count all nodes - by label

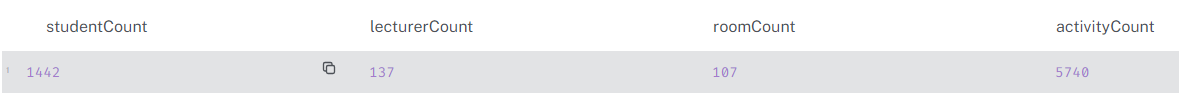
Below are two queries returning the same results - counts of nodes by node label.

// 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



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

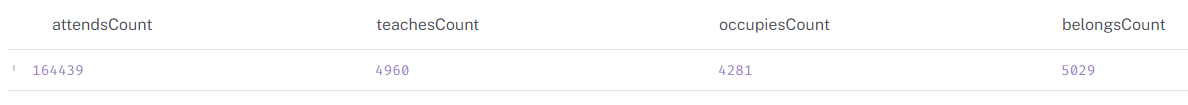


Count All Nodes

### 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



Count All Relationships

### Activity counts

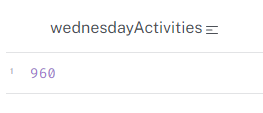
In this graph, an activity is an instance of an activity, that is, a unique combination of name, date, start, end, location, staff. It means *a lot* of activities!

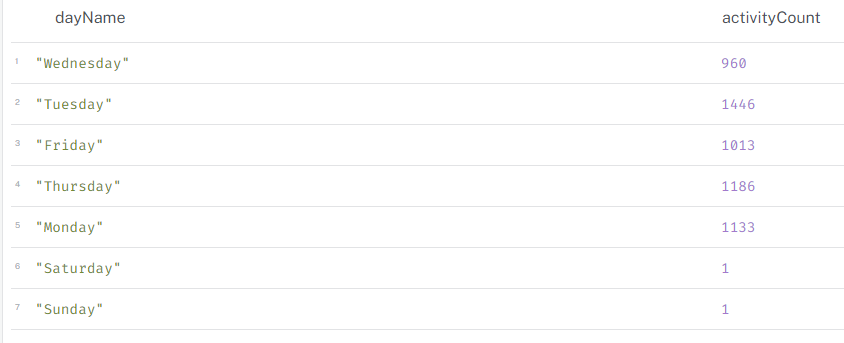
// Count of activities  
  
MATCH (a:activity)  
RETURN count(a) AS totalActivities;

// 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







### 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 0x000002389B7361D0>  
Running query...  
  
<Record activitiesStartingAt5pm=172>

### Staff activity count

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

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 0x000002389EB7A050>  
Running query...  
  
 staffName activityCount  
0 Debbie Nichols 145  
1 Marc Hernandez 127  
2 Eileen Allen 126  
3 Steven Perez 124  
4 Justin Alvarez 112

# 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

### Unscheduled activities

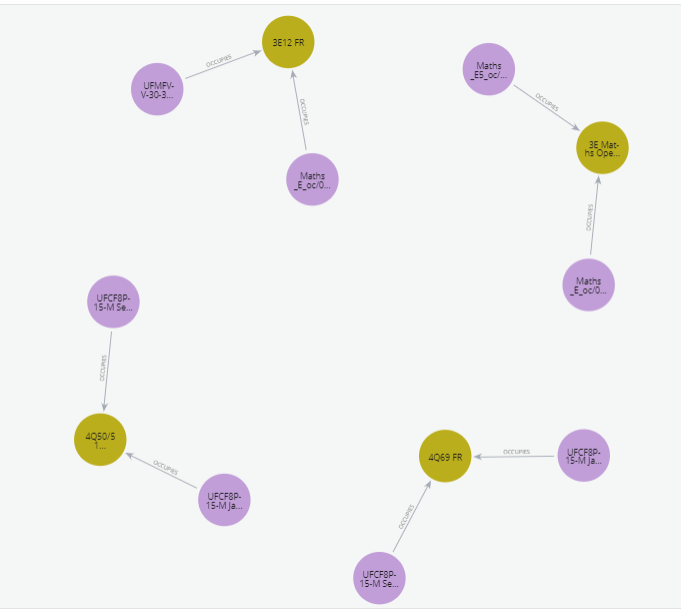
Unscheduled activities can be identified as follows. This query can be tweaked to also search for matches where the property equals ’’ - that is, a blank.

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

### 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



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 0x000002805B8012D0>  
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

### 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 AS extraNeeded  
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 0x000002805DA61CD0>  
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

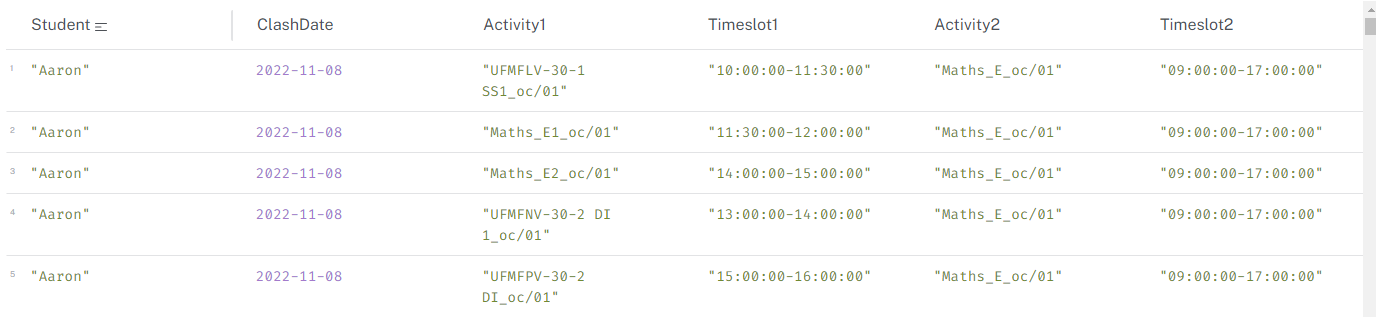
### Student clashes

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](appendix-cypher5a.qmd)

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 Clashes

# 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.

## Scenario

Each model below will use the same basic scenario:

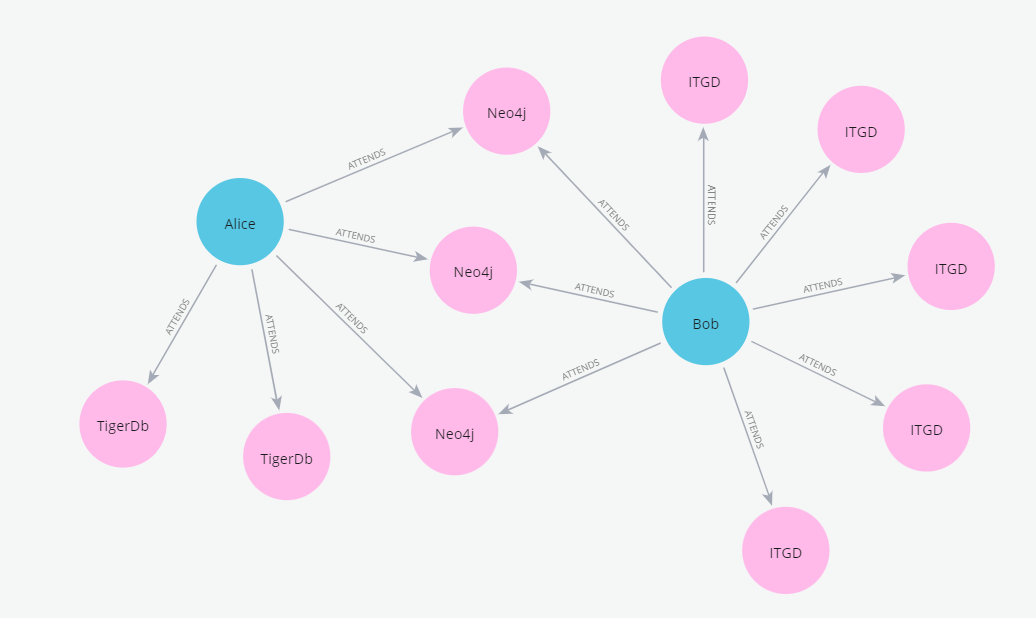
* Two students - Alice and Bob
* Three activities:
  + ITGD - Introduction to Graph Databases
  + Neo4j - Neo4j for Beginners
  + TigerDB - TigerGraph for Data Scientists
  + Each activity has a start and end time
  + Each activity is scheduled for several weeks
* There are deliberate clashes between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

## 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.

### Create data

// Create unique activity nodes (TestActivityModel1)  
CREATE (:TestActivityModel1 { actName: "ITGD", date: date("2024-08-06"), startTime: localtime("09:00:00"), endTime: localtime("11:00:00") })  
CREATE (:TestActivityModel1 { actName: "ITGD", date: date("2024-08-13"), startTime: localtime("09:00:00"), endTime: localtime("11:00:00") })  
CREATE (:TestActivityModel1 { actName: "ITGD", date: date("2024-08-20"), startTime: localtime("09:00:00"), endTime: localtime("11:00:00") })  
CREATE (:TestActivityModel1 { actName: "ITGD", date: date("2024-08-27"), startTime: localtime("09:00:00"), endTime: localtime("11:00:00") })  
CREATE (:TestActivityModel1 { actName: "ITGD", date: date("2024-09-03"), startTime: localtime("09:00:00"), endTime: localtime("11:00:00") })  
CREATE (:TestActivityModel1 { actName: "Neo4j", date: date("2024-07-30"), startTime: localtime("10:00:00"), endTime: localtime("11:00:00") })  
CREATE (:TestActivityModel1 { actName: "Neo4j", date: date("2024-08-13"), startTime: localtime("10:00:00"), endTime: localtime("11:00:00") })  
CREATE (:TestActivityModel1 { actName: "Neo4j", date: date("2024-08-27"), startTime: localtime("10:00:00"), endTime: localtime("11:00:00") })  
CREATE (:TestActivityModel1 { actName: "TigerDb", date: date("2024-08-06"), startTime: localtime("11:00:00"), endTime: localtime("12:00:00") })  
CREATE (:TestActivityModel1 { actName: "TigerDb", date: date("2024-08-13"), startTime: localtime("11:00:00"), endTime: localtime("12:00:00") });  
  
// 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) ;

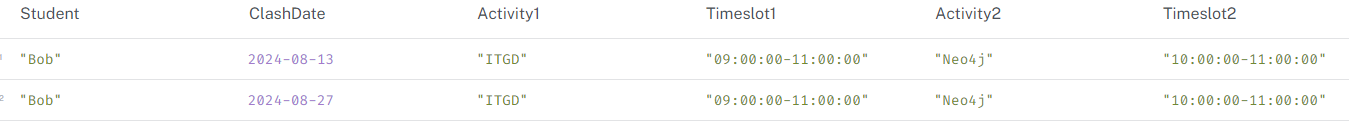


Model 1

To identify the clashes, this query can be run:

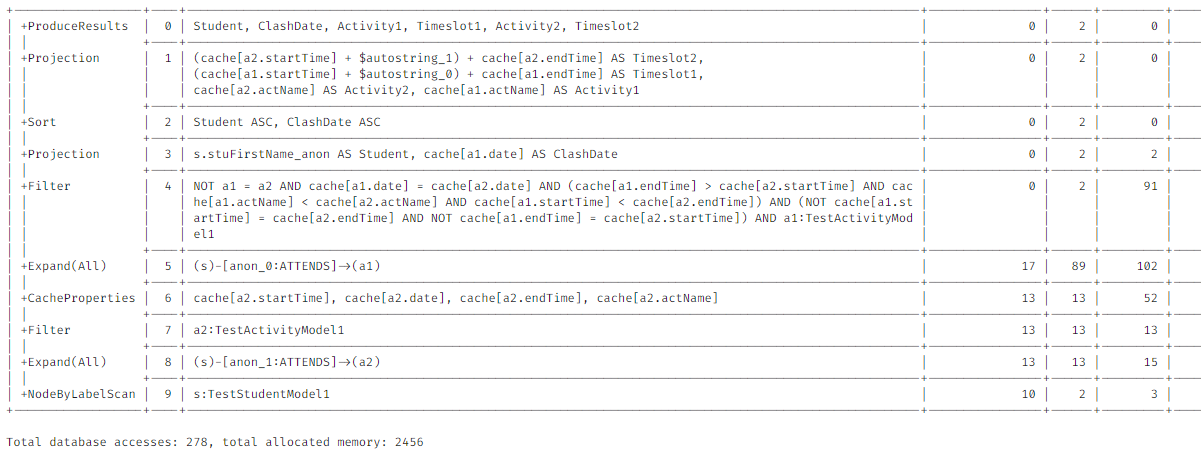
MATCH (s:TestStudentModel1)-[:ATTENDS]->(a1:TestActivityModel1)  
WITH s, a1  
MATCH (s)-[:ATTENDS]->(a2:TestActivityModel1)   
WHERE a1 <> a2   
 AND a1.date = a2.date   
 AND (a1.startTime < a2.endTime AND a1.endTime > a2.startTime) // overlap condition  
 AND NOT (a1.startTime = a2.endTime OR a1.endTime = a2.startTime) // xxclude "touching" cases  
 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:



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.



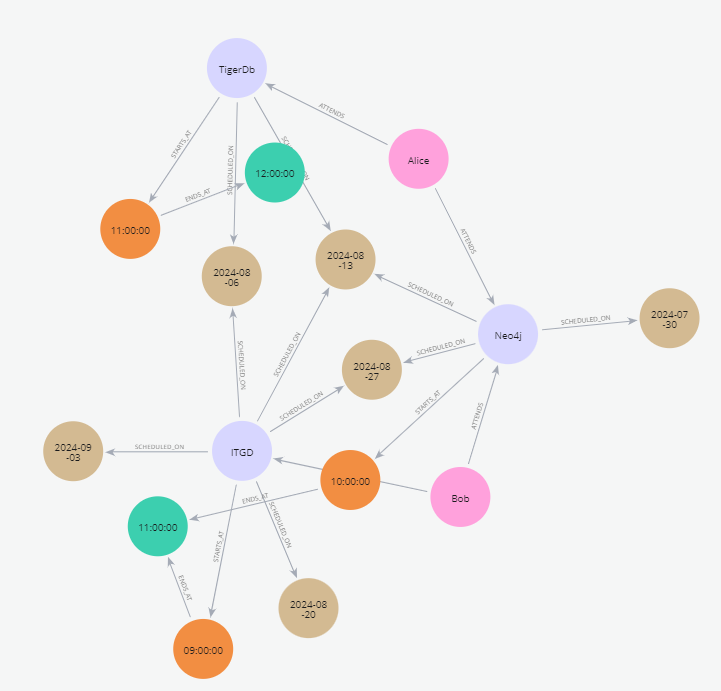
Model 1 profile

## 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.

### 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("2024-08-20"), date("2024-08-27"), date("2024-09-03")]  
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("2024-08-27")]  
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) ;

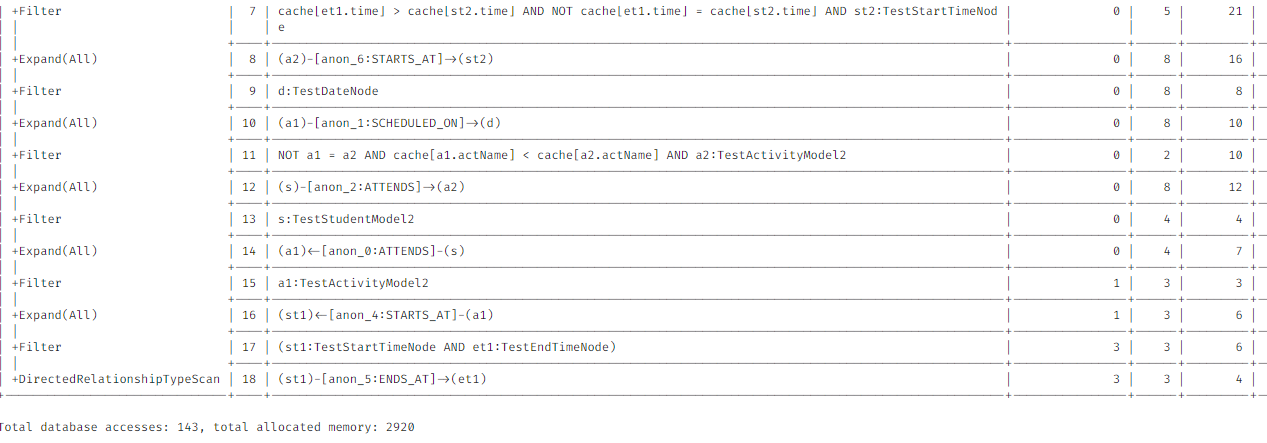


Model 2

To identify student clashes, this query can be run.

MATCH (s:TestStudentModel2)-[:ATTENDS]->(a1:TestActivityModel2)-[:SCHEDULED\_ON]->(d:TestDateNode)  
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):

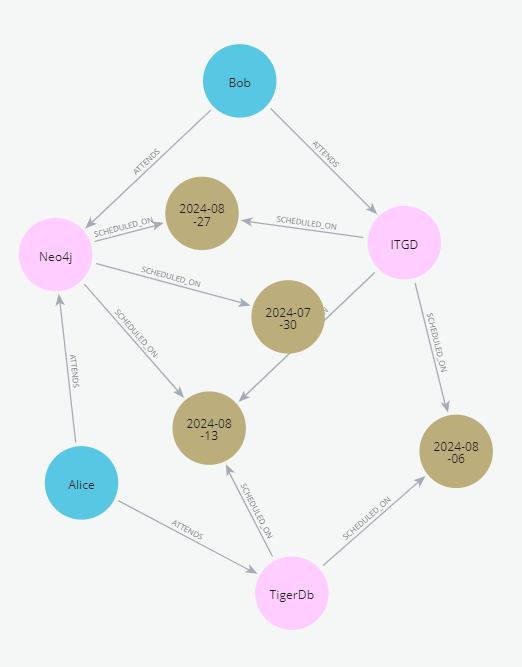


Model 2 profile

## Model 3 - Date Nodes

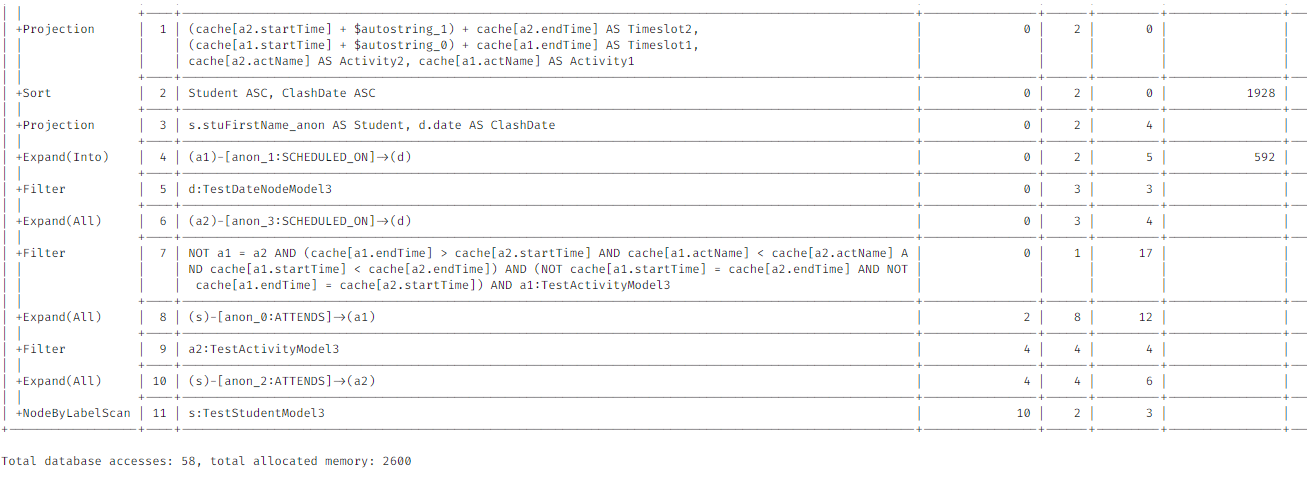
Model 3 uses a single node for each activity as well as date nodes - start and end times are properties of the activity.

// 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: localtime("11:00:00") })  
CREATE (:TestActivityModel3 { actName: "Neo4j", startTime: localtime("10:00:00"), endTime: localtime("11:00:00") })  
CREATE (:TestActivityModel3 { actName: "TigerDb", startTime: localtime("11:00:00"), endTime: localtime("12:00:00") });  
  
// 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("2024-08-20"), date("2024-08-27"), date("2024-09-03")]  
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("2024-08-27")]  
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) ;



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):

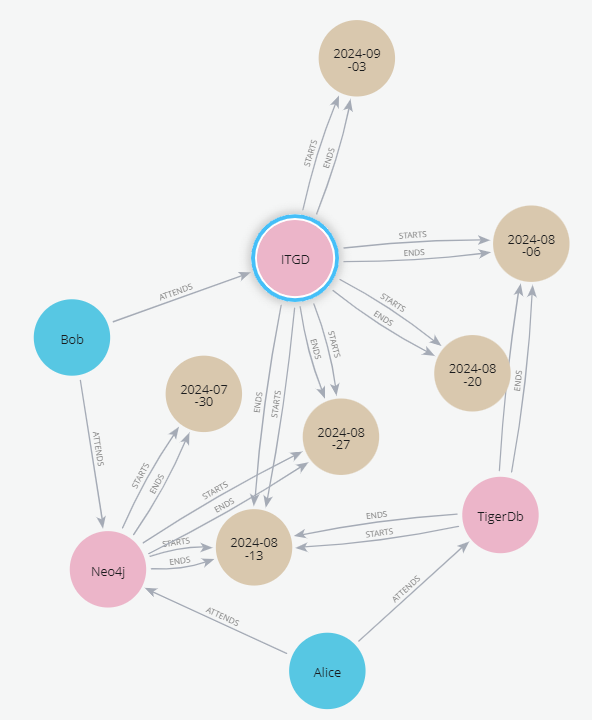


Model 3 profile

### Model 4 -

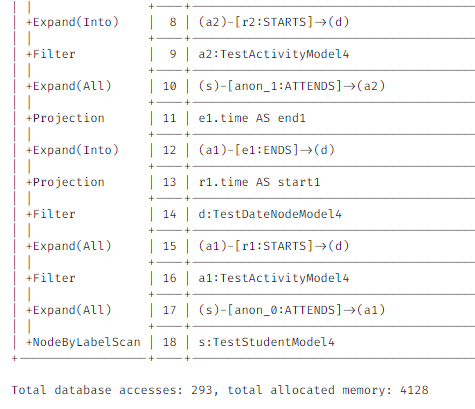
Model 4 uses a single node for each activity and date - start and end times are now properties of the *relationship* between the activity and the date.

// 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("2024-08-20"), date("2024-08-27"), date("2024-09-03")]  
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("2024-08-27")]  
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) ;



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):



Model 4 profile

## 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.

## Delete Data

The cypher below deletes all test data.

### Model 1

// Delete all TestActivityModel1 nodes  
MATCH (a:TestActivityModel1)  
DETACH DELETE a;   
  
// Delete all TestStudentModel1 nodes  
MATCH (s:TestStudentModel1)  
DETACH DELETE s;

### Model 2

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

### Model 3

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

### Model 4

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

# 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.

### Minimal idle time

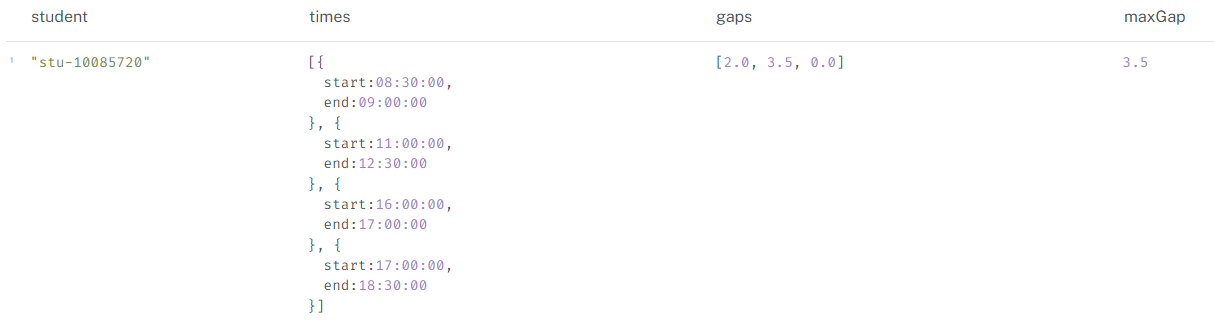
Identifying time gaps between scheduled activities is very complex and requires several steps, clauses and comprehensions within a single query:

* *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.

#### 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 gap > maxGap THEN gap ELSE maxGap END) AS maxGap



Example student with gaps between activities

#### 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: a}) AS times  
// 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}, gapRecord IN gaps |   
 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.firstActivity.actEndTime,  
 activity2: maxGapRecord.secondActivity.actName,  
 activity2\_time: maxGapRecord.secondActivity.actStartTime + "-" + maxGapRecord.secondActivity.actEndTime,  
 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 0x000001B61F23E750>  
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

#### 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) / 60.0 AS blockHours,  
 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", "blockActivities"])  
  
# 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 0x000001B622E13350>  
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

#### 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.

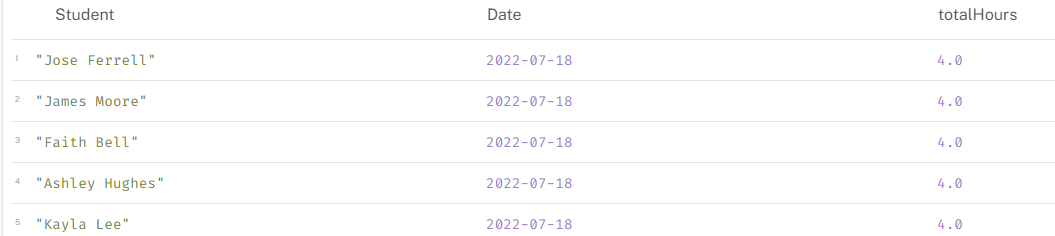
from connect\_to\_neo4j\_db import connect\_to\_neo4j  
from neo4j import GraphDatabase  
import pandas as pd  
  
# connect to Neo4j  
driver = connect\_to\_neo4j()  
  
# session  
session = driver.session()  
  
# run query (modified RETURN clause)  
query = """  
MATCH (s:student)-[:ATTENDS]->(a:activity)<-[:TEACHES]-(:staff) // Filter for teaching activities  
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) / 60.0 AS blockHours,  
 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", "blockActivities"])  
  
# 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 0x000001B622A10190>  
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

#### 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;



Total hours per day

#### 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: date("2022-09-27")})  
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, start: null, end: null}},  
 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)) > state.longestBlock.duration  
 THEN {  
 duration: (activity.actEndTime.hour \* 60 + activity.actEndTime.minute) -   
 (state.currentBlock.start.hour \* 60 + state.currentBlock.start.minute),  
 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

#### Example - longest consecutive block for ‘Susan Lopez’ on 2022-09-27

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

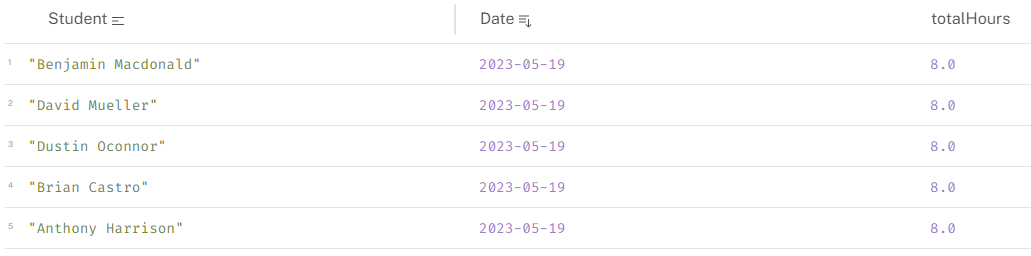
from connect\_to\_neo4j\_db import connect\_to\_neo4j  
from neo4j import GraphDatabase  
import pandas as pd  
  
# connect to Neo4j  
driver = connect\_to\_neo4j()  
  
# session  
session = driver.session()  
  
# run query (modified RETURN clause)  
query = """  
MATCH (s:student {stuFullName\_anon: "Susan Lopez"})-[:ATTENDS]->(a:activity{actStartDate: date("2022-09-27")})  
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, start: null, end: null}},  
 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)) > state.longestBlock.duration  
 THEN {  
 duration: (activity.actEndTime.hour \* 60 + activity.actEndTime.minute) -   
 (state.currentBlock.start.hour \* 60 + state.currentBlock.start.minute),  
 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", "blockStartTime", "blockEndTime"])  
  
# 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 0x000001B622A2D1D0>  
Running query...  
  
 student date longestConsecutiveBlockDuration \  
0 Susan Lopez 2022-09-27 300   
  
 blockStartTime blockEndTime   
0 13:00:00.000000000 18:00:00.000000000

### 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 hour students  
  
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;



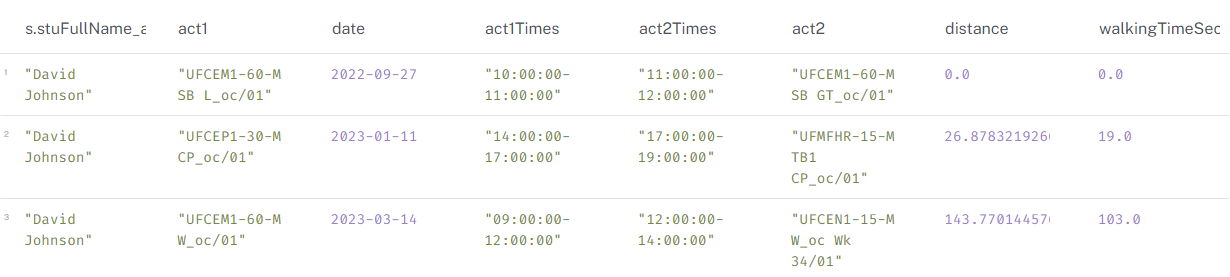
Max hours in a day

### 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]->(r1:room),  
 (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 act1Times, a2.actStartTime+"-"+a2.actEndTime AS act2Times, a2.actName AS act2,  
 point.distance(r1.location, r2.location) AS distance,  
 round(point.distance(r1.location, r2.location) / 1.4) AS walkingTimeSeconds // Calculate walking time in seconds

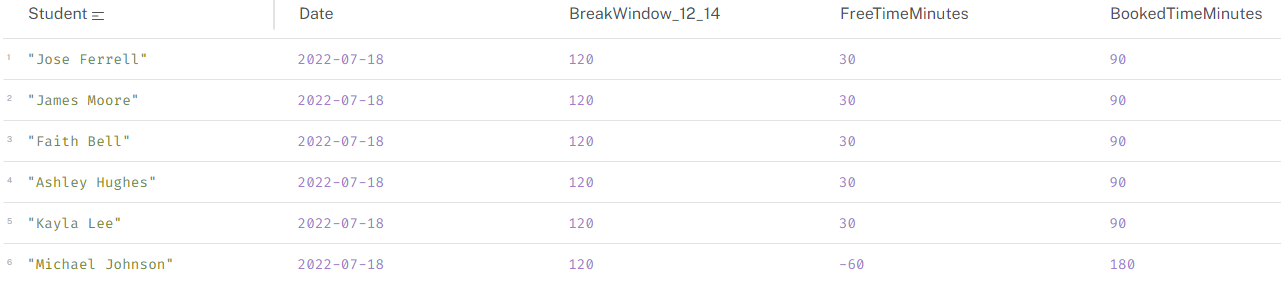


Travel time between activities

### 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) AS StartTime, time(activity.actEndTime) AS EndTime, duration.between(time('12:00'), time('14:00')).minutes AS BreakWindow\_12\_14  
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(StartTime, time('14:00')).minutes  
 WHEN EndTime > time('12:00') AND EndTime <= time('14:00') THEN duration.between(time('12:00'), EndTime).minutes  
 END AS BookedDurationMinutes  
RETURN Student, Date, BreakWindow\_12\_14, BreakWindow\_12\_14 - SUM(BookedDurationMinutes) AS FreeTimeMinutes, SUM(BookedDurationMinutes) AS BookedTimeMinutes  
ORDER BY Date



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:

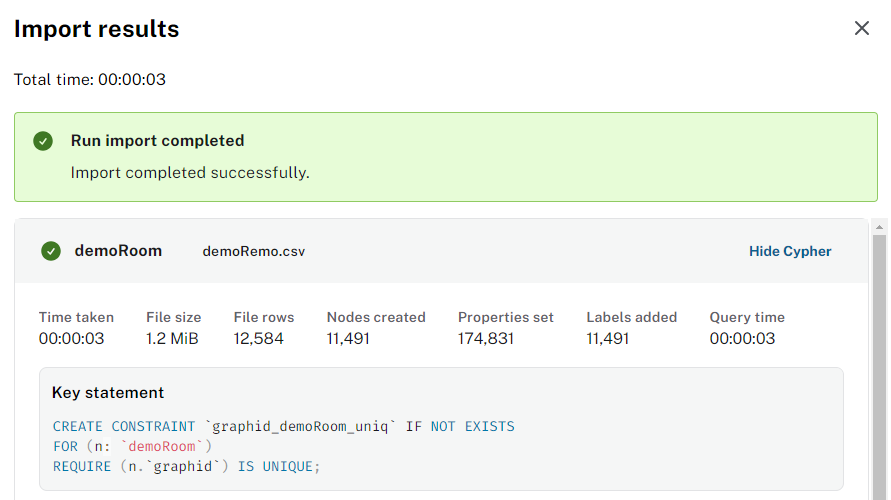
To remedy this, the query can be updated to use student ID or a different unique identifier. I would also like to update the anonymisation function in the ETL so that it does not duplicate names in the output.

# Rooms and Spaces

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

## Room Geo-location

### Import locations from file



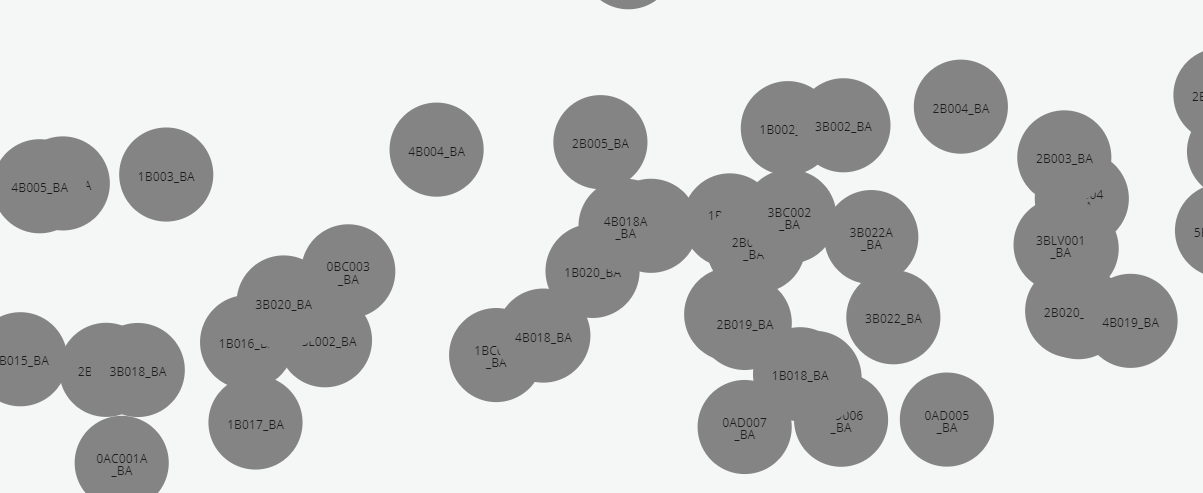
imported locations from file

### 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`;

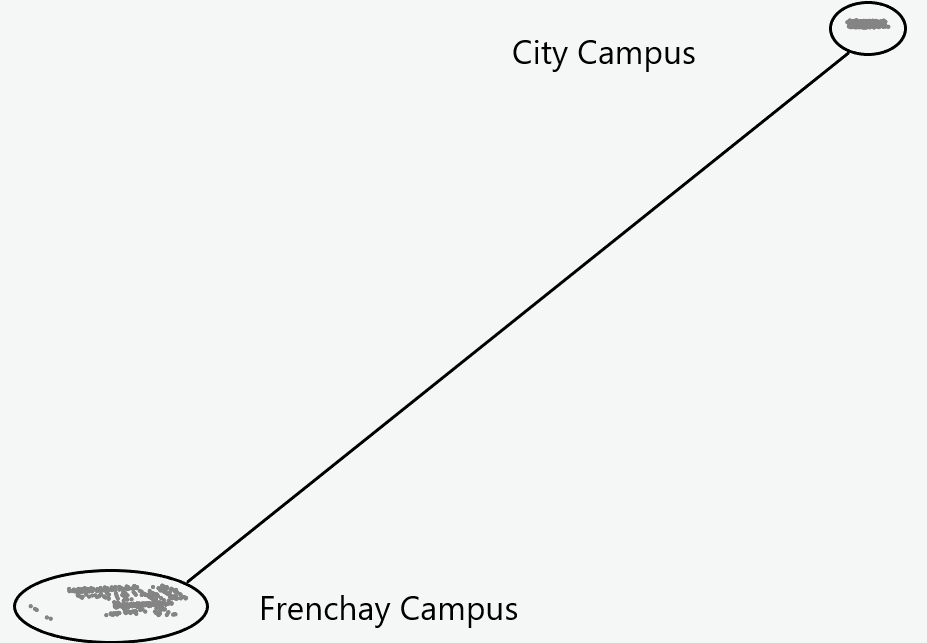
### Thoughts

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. 



Close of up B Block rooms on Frenchay Campus

An alternative view includes locations in City Campus.



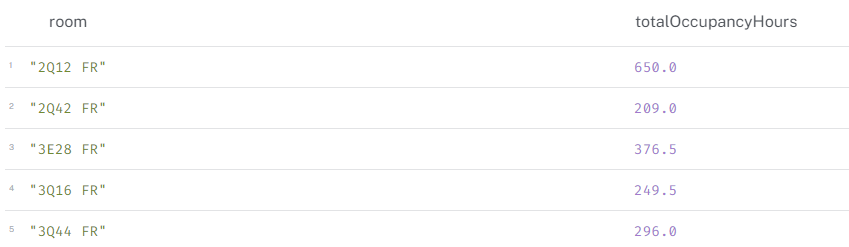
Frenchay and City Campus rooms

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.

## Room use

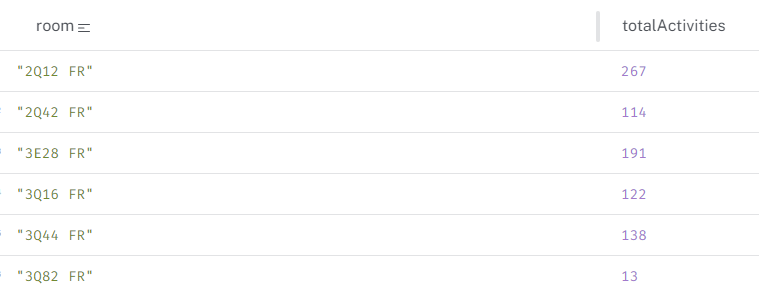
Being able to measure room usage - utilisation, frequency, occupancy - is a key metric for the university.

// 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



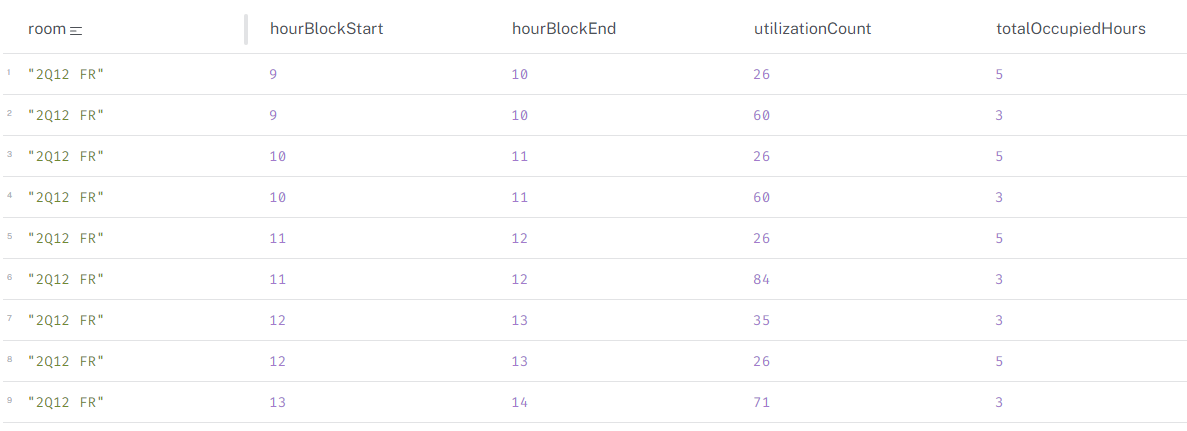
Room occupancy

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



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 1 ELSE 0 END) AS utilizationCount,  
 occupiedHours AS totalOccupiedHours  
ORDER BY hourBlockStart



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.

# 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. Perspectivies focus the view by selecting specific nodes and relationships, while scenes allow for visualising the data interactively.

## Scenario 1: Programme Leader Perspective

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

### 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

### 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.

### 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:student)-[:ATTENDS]->(a:Activity)  
RETURN m, s, a  
  
  
//Room Utilisation:  
  
MATCH (p:programme)<-[:BELONGS\_TO]-(m:module)<-[:BELONGS\_TO]-(a:activity)-[:OCCUPIES]->(r:room)  
RETURN p, m, a, r  
  
Staff Workload:  
  
MATCH (p:programme)<-[:BELONGS\_TO]-(m:module)<-[:BELONGS\_TO]-(a:activity)<-[:TEACHES]-(s:staff)  
RETURN p, m, a, s

## Scenario 2: Module Leader Perspective

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

### 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

### 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.

## Scenario 3: Administrator Perspective (Room Usage)

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

### Perspective:

**Nodes**:

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

**Relationships**:

* activity -[:OCCUPIES]-> room
* module <-[:BELONGS\_TO]- activity
* programme <-[:BELONGS\_TO]- module

### 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.

# 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.

# Weekly Update - 10 June 2024

## 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.

## Accomplishments

* **Results:**
  + ☒ recreated single student graph timetable - see [poc-1-basic](.\model/poc1-basic-phl/poc-1-basic.qmd)
  + ☒ 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

## 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](.\model/poc2-time/representing-time.qmd)

## Next Steps

### 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:**

## Post-Meeting Notes

We met on 13 June 2024 for approximatly 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 an 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.

## 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.

# Fortnightly Update - 2024-06-24

## Summary

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

## 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

## Next Steps

### 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

## 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)

## Post-Meeting Notes

My superviser 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 lookes 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.

# Weekly Update - 2024-07-08

## Summary

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

## 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

## Next Steps

### Weekly Goal: What is goal of week?

* **Project Management**
  + Plan next six weeks - with tasks
  + see [project-structure](../docs/notes/project-structure.ipynb)
* **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](../docs/notes/queries.ipynb)
* **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](../docs/notes/code-critique.ipynb)
  + Draft Intro
    - Proof of concept
    - Research question
    - Domain

## 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

## 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

## Additional Notes

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

# 

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?direct=true&db=bsu&AN=150973572&site=ehost-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/EWKNTqInQuRDoSaPmHCOp20B72Dp\_2vYpJ\_\_GbCzaY5tiA?email=Petter.Lovehagen%40uwe.ac.uk&e=4%3AwEUjpu&fromShare=true&at=31&CID=256ab09c-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/1779550151?pq-origsite=primo&parentSessionId=GLad2hOIbVrF%2F0eiOKHGi%2BO%2BFOyV9GXuQTQCxSfgWNw%3D&sourcetype=Dissertations%20&%20Theses [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=1219736900.

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].

# Acknowledgement

First, I would like to express my gratitude to my supervisor, [Dr. Xiaodong Li](https://people.uwe.ac.uk/Person/XiaodongLi), for their guidance, support and feedback throughout this project. They were particular helpful in keeping me focused and staying on topic, allowing me to incrementally achieve my goals.

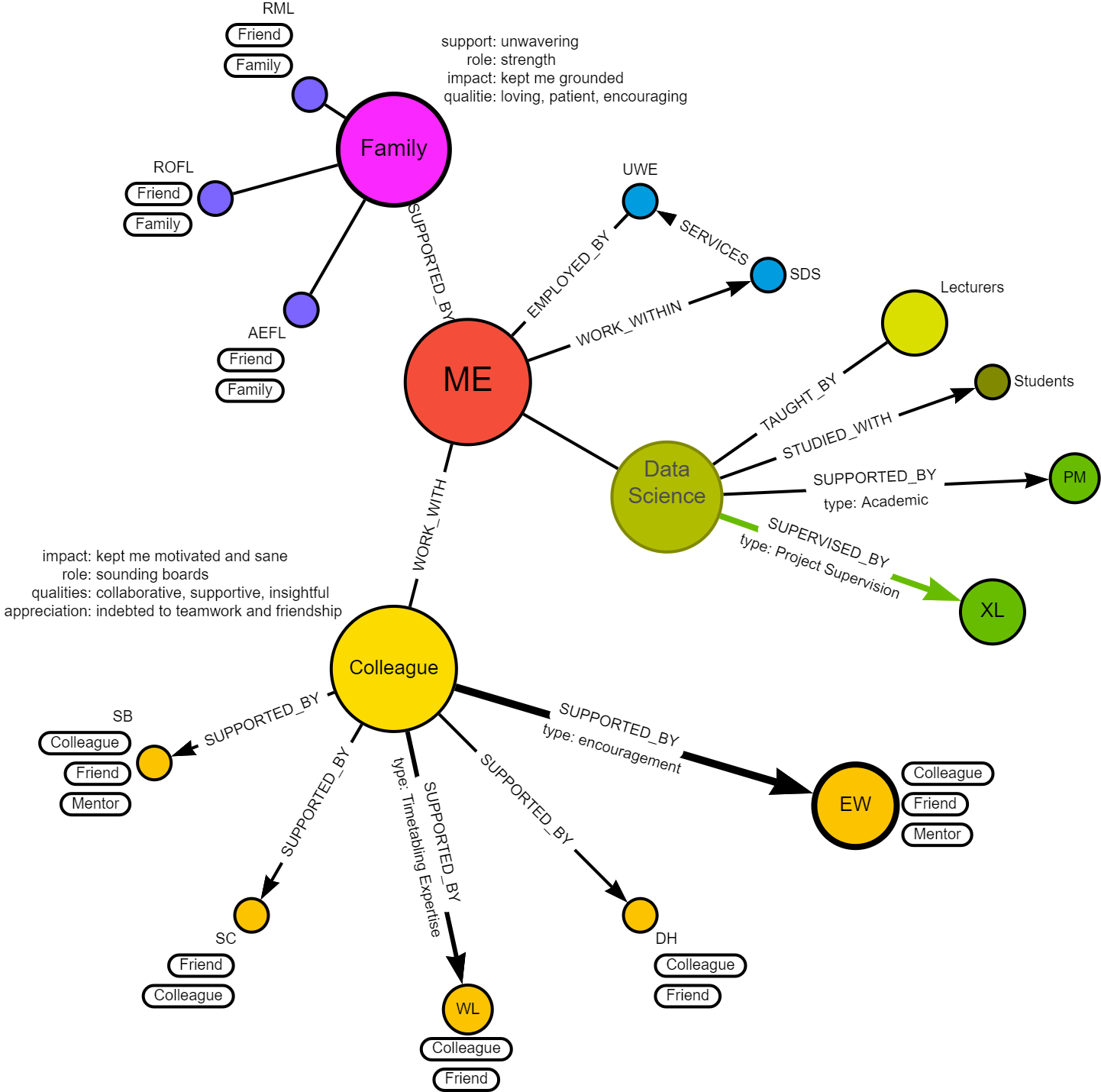
I extend my sincere thanks to the faculty and staff of the MSc Data Science programme for providing the environment and resources necessary for my project:

* [Dr Paul Matthews](https://people.uwe.ac.uk/Person/Paul2Matthews) - Course lead for MSc Data Science
* [Prakash Chatterjee](https://people.uwe.ac.uk/Person/PrakashChatterjee) - Data Management Fundamentals
  + Reintroduced the concept of graph databases and noSQL.
* [Dr David Wyatt](https://people.uwe.ac.uk/Person/David2Wyatt) - Programming for Data Science
  + Provided foundations of Python programming, Git version control, markdown.
* [Dr Hisham Ihshaish](https://people.uwe.ac.uk/Person/HishamIhshaish) - Machine Learning and Predictive Analytics
  + Introduced me to Machine Learning, marrying programming and statistics.
* [Dr Deirdre Toher](https://scholar.google.co.uk/citations?user=2P4klJcAAAAJ&hl=en) - 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](https://www.cso.ie/en/index.html) in Ireland.
* [Dr Jason Anquandah](https://people.uwe.ac.uk/Person/JasonAnquandah) - Interdisciplinary Group Project
  + Supported me through the challenges of a group project.
* [Dr Mahmoud Elbattah](https://people.uwe.ac.uk/Person/MahmoudElbattah) - 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…



Acknowledgement graph

1. “**Cypher** is a declarative graph query language that allows for expressive and efficient data querying in a property graph” (Wikipedia contributors, 2024) [↑](#footnote-ref-1)
2. Some of the above variations are explored in [Appendix-Student Clashes](appendix-cypher5a.qmd) [↑](#footnote-ref-2)
3. Structured Query Language (Wikipedia contributors, 2024) [↑](#footnote-ref-3)
4. 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) [↑](#footnote-ref-4)
5. “**Cypher** is a declarative graph query language that allows for expressive and efficient data querying in a property graph” (Wikipedia contributors, 2024) [↑](#footnote-ref-5)
6. “**Cypher** is a declarative graph query language that allows for expressive and efficient data querying in a property graph” (Wikipedia contributors, 2024) [↑](#footnote-ref-6)
7. Some of the above variations are explored in [Appendix-Student Clashes](appendix-cypher5a.qmd) [↑](#footnote-ref-7)
8. “**Cypher** is a declarative graph query language that allows for expressive and efficient data querying in a property graph” (Wikipedia contributors, 2024) [↑](#footnote-ref-8)
9. “**Cypher** is a declarative graph query language that allows for expressive and efficient data querying in a property graph” (Wikipedia contributors, 2024) [↑](#footnote-ref-9)
10. “**Cypher** is a declarative graph query language that allows for expressive and efficient data querying in a property graph” (Wikipedia contributors, 2024) [↑](#footnote-ref-10)