Graph Time

A proof-of-concept data engineering project

Petter Lovehagen

2024-08-29

Table of contents

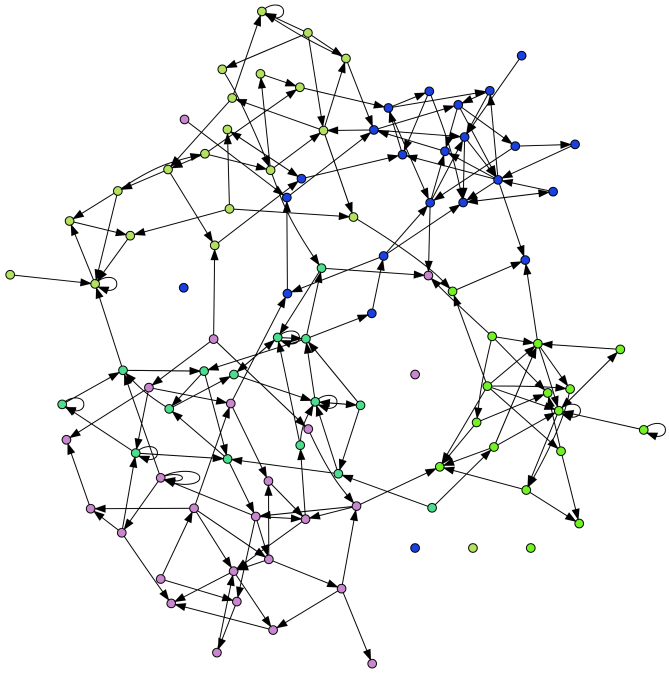
1 Exploring Graph Data Models for Timetabling Insights

A proof-of-concept data engineering project

Supervisor: Xiaodong Li

Programme: MSc Data Science

Word Count: [add when finished]



This is a randomly generated graph for visual purposes only.

See Appendix for graph generator code

2 Introduction

> ⚠ TODO
>
> - consider adding 'operational challenges - see 01-01a'

University timetabling is the process of scheduling curriculum elements (modules, programmes), locations, and 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, countless modules, 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 those timetables can lead to deeper insights and ultimately, improved quality for all stakeholders.

3 Background and Motivation

3.1 TODO

- review - reduce?

Many years ago I slipped into the world of timetabling and scheduling. 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 **ADD** producing a 'decent' timetable for the students. Following this, I spent some years as a timetabling data manager, where my view needed to be slightly broader, encompassing the whole dataset. These experiences exposed me not only to the intricacies of the data, systems and processes but also the intense pressures faced by timetabling teams. The constant scrutiny, stakeholder demands and impossibility of achieving universal satisfaction left an indelible mark, highlighting the need for robust tools and metrics to understand and assess timetable quality - a factor often overshadowed by the pursuit of mere feasibility.

3.2 Research Gap: Bridging Theory and Practice

Much of 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. The computationally-driven optimisation research is often referred to as the university course timetabing problem (UCTTP) and is categorised as NP-hard[1], meaning finding the absolute "best" timetable is exceptionally challenging (Babaei, Karimpour and Hadidi, 2015; Herres and Schmitz, 2021; Wikipedia contributors, 2024). Consequently, significant effort has been dedicated to developing algorithms like constraint programming (Holm et al., 2022) and local search techniques such as Tabu Search and simulated annealing (Oude Vrielink et al., 2019), aiming to create workable timetables within reasonable timeframes.

However, this emphasis on computational optimisation makes use of standardised datasets and predefined constraints. While crucial for advancing algorithmic development, these idealised

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

scenarios do not fully capture the dynamic complexity of real-world university timetabling. Universities grapple with constantly shifting demands: fluctuating student populations, evolving college 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).

This is where I believe graph data structures 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 excel at generating 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 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).

This potential for deeper analysis, coupled with the limitations of optimisation-centric approaches, motivates this exploration of graph data structures for enhancing timetable understanding and, ultimately, improving timetable quality for all stakeholders.

4 What is a “good” timetable?

One of the most challenging aspects of university timetabling is defining what constitutes a “good” timetable. Despite best efforts, it is virtually impossible to deliver universal satisfaction from a university timetable. The quality of a timetable is inherently subjective and varies among stakeholders depending on preferences and competing 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 looks like; individual stakeholders often have conflicting priorities:

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

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

4.1 Examples


Week 13 - w/c 13/11/2023 or 11/11/2024
8:00 8:30 9:00 9:30 10:00 10:30 11:00 11:30 12:00 12:30 13:00 13:30 14:00 14:30 15:00 15:30 16:00 16:30 17:00 17:30 18:00 18:30
Mon
SoE_PW1 L2 PreRec_ol/01 Pre-recorded Material Engineering Practice 2 UFMFQS-15-2 13
SoE_PW1 L2 W1 Mon am_oc/06 Workshop (On Campus) Engineering Practice 2 UFMFQS-15-2 2Q48 FR 13
SoE_PW1 Spotlight on PAL 1_oc/01 JTA Workshop (On Campus) Spotlight on PAL - Z-Block Atrium 1Z001 FR 13
SoE_PW1 L2 W1 Mon pm_oc/06 Self-Directed Study (On Campus) Engineering Practice 2 UFMFQS-15-2 2Q48 FR 13
Tue
SoE_PW1 L2 PreRec_ol/01 Pre-recorded Material Engineering Practice 2 UFMFQS-15-2 13
SoE_PW1 L2 W2 Tue am_oc/06 Workshop (On Campus) Engineering Practice 2 UFMFQS-15-2 4Q05 FR 13
SoE_PW1 Employers' Fair_oc/01 JTA Workshop (On Campus) Employers' Fair - Z-Block Atrium 1Z001 FR 13
SoE_PW1 L2 W2 Tue pm_oc/06 Self-Directed Study (On Campus) Engineering Practice 2 UFMFQS-15-2 4Q04 FR 13
Wed
SoE_PW1 L2 PreRec_ol/01 Pre-recorded Material Engineering Practice 2 UFMFQS-15-2 13
SoE_PW1 L2 W3 Wed am_oc/06 Workshop (On Campus) Engineering Practice 2 UFMFQS-15-2 3Q68 FR 13
SoE_PW1 Spotlight on PAL 2_oc/01 JTA Workshop (On Campus) Spotlight on PAL - Z-Block Atrium 1Z001 FR 13
Thu
SoE_PW1 L2 PreRec_ol/01 Pre-recorded Material Engineering Practice 2 UFMFQS-15-2 13
SoE_PW1 L2 W4 Thu am_oc/06 Workshop (On Campus) Engineering Practice 2 UFMFQS-15-2 6X110 FR 13
SoE_PW1 Careers DI_oc/01 Event (On Campus) Careers Drop In with Placements Focus 1Z001 FR 13
SoE_PW1 L2 W4 Thu pm_oc/06 Self-Directed Study (On Campus) Engineering Practice 2 UFMFQS-15-2 6X110 FR 13
Fri
SoE_PW1 L2 W5 Fri am Pres_oc/06 Presentation (On Campus) Engineering Practice 2 UFMFQS-15-2 4Z013 FR 13

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

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

Consider the above timetables:

- The first timetable is evenly spread over five days. Tuesday afternoons are heavily scheduled; Fridays have one two-hour activity over lunch.
- The second timetable has two days free of activities (Wednesday and Thursday). Monday has a single activity at 18:00-19:00.
- The third timetable has activities on five days. There are large gaps between activities on Tuesday and Wednesday.

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

5 Project Aims and Scope

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

Objectives include:

- Designing an extensible, system-agnostic graph data model for university timetables
- Developing a mapping pipeline to transition from relational to graph database representations of timetables
- Implementing and evaluating a set of proof-of-concept analytical metrics leveraging the graph data model
- Discussing the performance and capabilities of graph-based analytics against traditional relational database approaches
- Exploring how graph-based approaches might contribute to measuring and improving timetable 'quality' from various stakeholder perspectives

It is important to note that this project is positioned as a proof-of-concept and exploratory study. I will not be implementing a full-scale timetabling system or focusing on real-time timetable generation or optimisation. Instead, the scope is limited to demonstrating the potential of graph-based approaches in enhancing understanding and analysis of university timetables.

Let's graph!

6 Graph vs Relational Data Models

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

6.1 Relational Models

6.1.1 Tables, Joins and the Limits of Interconnectedness

Relational databases, using SQL 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 their 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 (see below in both SQL and graph queries). 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.

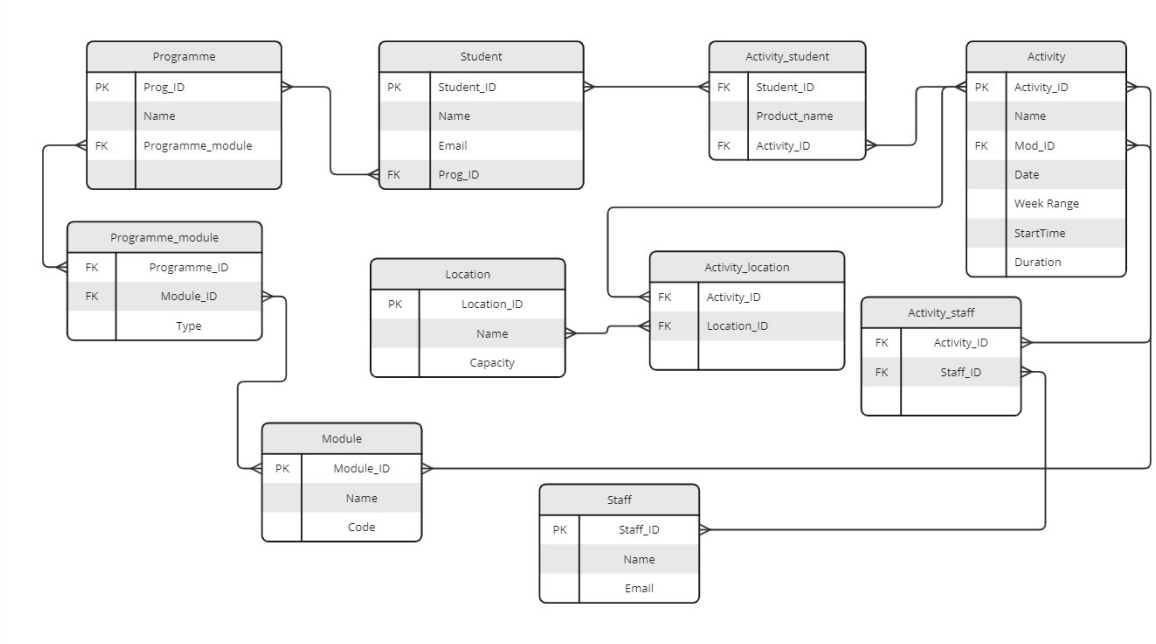


Figure 6.1: Example Simple Entity Relationship Diagram

Example Query in SQL

```
-- Assuming "BuildingName" is in V_BUILDING and linked to V_LOCATION
-- find students attending a specific lecturer's class in a particular building
-- requires 6 JOINS


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.[StudentS
INNER JOIN [RDB_MAIN2223].[rdowner].[V_ACTIVITY] a ON acts.[ActivityId] = a.[Id]
INNER JOIN [RDB_MAIN2223].[rdowner].[V_ACTIVITY_LOCATION] al ON a.[Id] = al.[ActivityId]
INNER JOIN [RDB_MAIN2223].[rdowner].[V_LOCATION] l ON al.[LocationId] = l.[Id]
INNER JOIN [RDB_MAIN2223].[rdowner].[V_BUILDING] b ON l.[BuildingId] = b.[Id]
INNER JOIN [RDB_MAIN2223].[rdowner].[V_ACTIVITY_STAFF] ast ON a.[Id] = ast.[ActivityId]
WHERE ast.[StaffId] = 'StaffID'
  AND b.[Name] = 'BuildingName';
```

6.2 Graph Models

6.2.1 Embracing interconnectedness

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

- **Nodes**: Represent entities (e.g., activity, room, staff, student). These are often "nouns."
- **Edges**: Represent relationships between nodes (e.g., "taught by," "enrolled in," "scheduled at"). These are often "verbs."

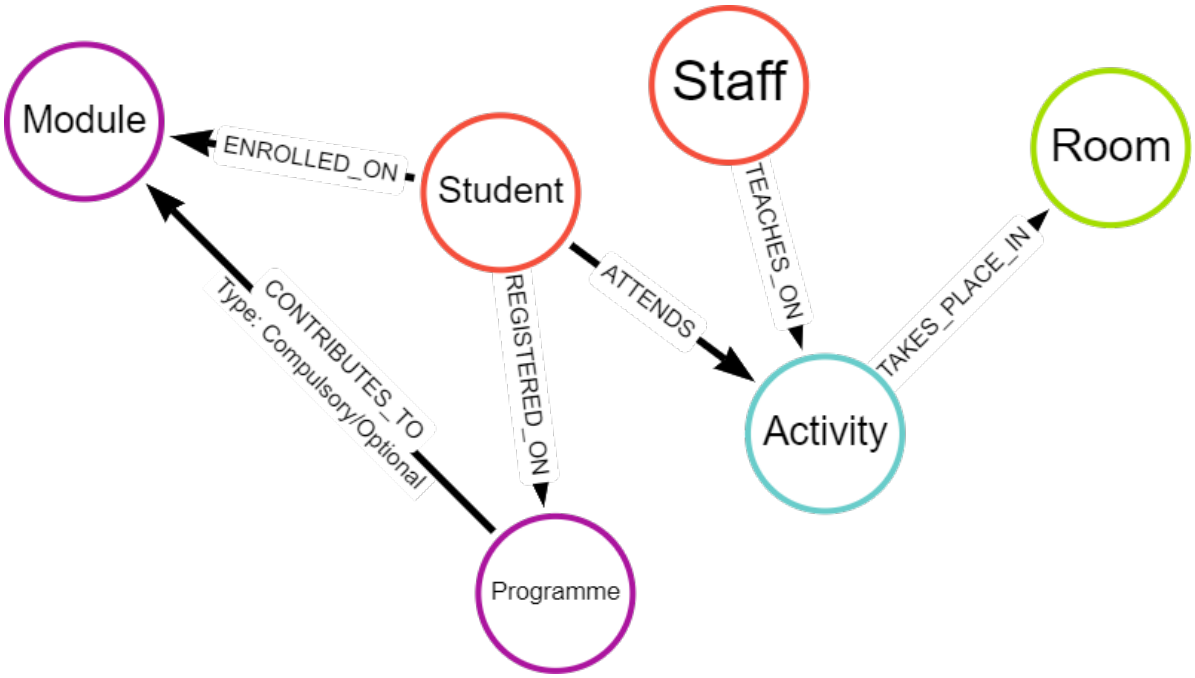


Figure 6.2: Simple Graph Data Model

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

- **Natural Representation**: Graph databases visually and conceptually mirror the relationships inherent in timetables, making them easier to understand and query.
- **Relationship-Centric Queries**: Graph databases are optimised for traversing and analysing relationships. Queries that would require multiple joins in a relational database often become significantly simpler and faster in a graph database - see below.

- **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).

Example Query in Cypher

```
// Assuming properties on nodes
// Much simpler query pattern

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

6.3 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.

7 Graph Data Model for Timebling

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

7.1 An Iterative Approach

The graph data model design was an iterative process: design -> build -> test -> review -> revise -> …and repeat. Using Neo4j Desktop, the first model was small scope in scope, incorporating *minimal* nodes and properties. Eventually, the expanded model was created in an cloud-based Neo4j Aura instance.

7.2 Core Nodes - Building Blocks

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

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

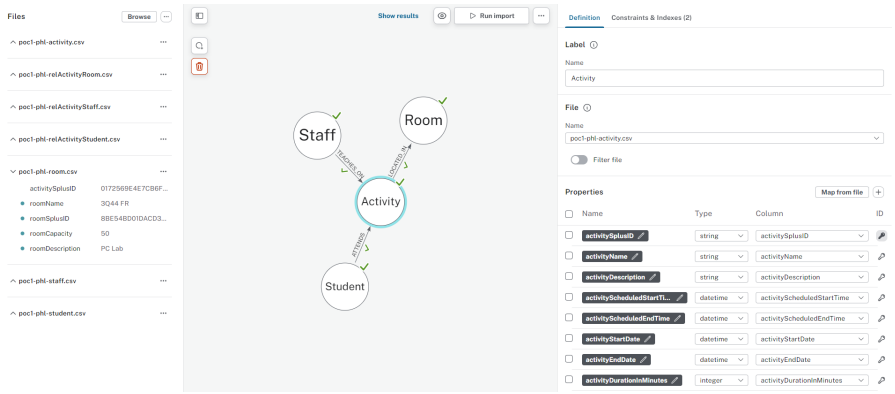


Figure 7.1: Neo4j Interface showing basic nodes and properties

7.3 Relationships - Connecting the Dots

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

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

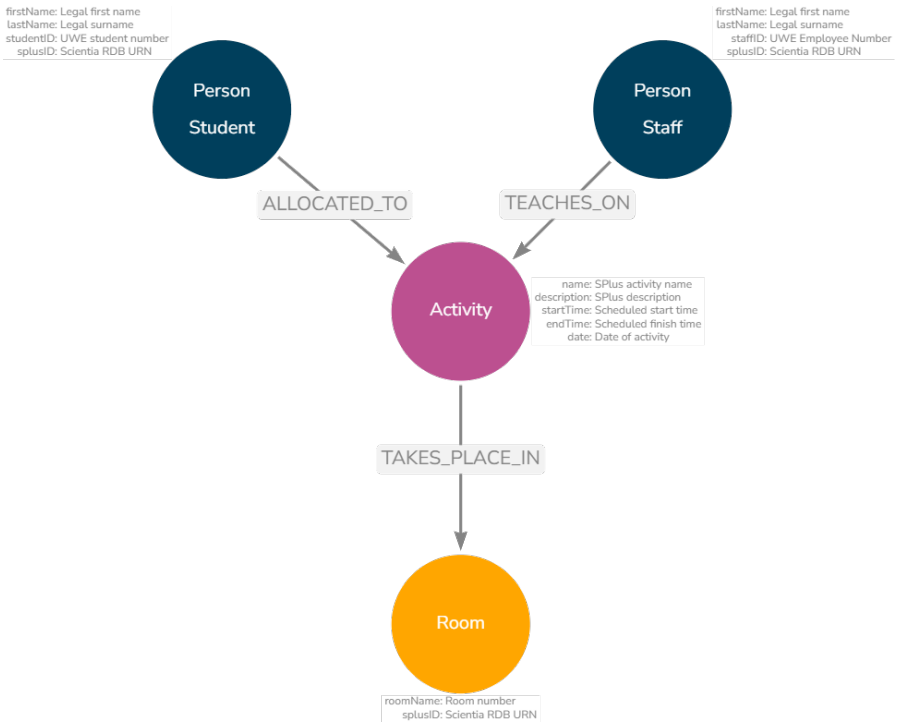


Figure 7.2: Core Nodes and Properties

8 Early Insights and Future Expansion

> ⚠ TODO
>
> - add cypher output results when reload data

8.1 Unveiling Basic Patterns

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

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

8.1.1 Example code

Busiest location overall

```
MATCH (r:Room)<-[:TAKES_PLACE_IN]-(a:Activity)
WITH r, sum(a.duration) AS totalDuration
RETURN r.name AS Room, totalDuration
ORDER BY totalDuration DESC
LIMIT 1
```

Busiest location for a specific time

```
MATCH (r:Room)<-[:TAKES_PLACE_IN]-(a:Activity)
WHERE a.startTime = "11:00"
WITH r, count(a) AS activityCount
RETURN r.name AS Room, activityCount
ORDER BY activityCount DESC
LIMIT 1
```

Students with below/above average hours

```
// Calculate program averages and standard deviations
MATCH (s:Student)-[:IS_ALLOCATED_TO]->(a:Activity)
WITH s.prog AS programme, AVG(a.duration) AS avgDuration, STDEV(a.duration) AS stdDev
GROUP BY programme

// Identify students outside the 10% margin
MATCH (s:Student)-[:IS_ALLOCATED_TO]->(a:Activity)
WITH s.studentID AS studentID, s.prog AS program, SUM(a.duration) AS totalDuration
MATCH (avgData)
WHERE avgData.programme = programme
WITH studentID, programme, totalDuration, avgData.avgDuration AS avgDuration, avgData.stdDev
WHERE totalDuration < avgDuration - (0.1 * avgDuration) OR totalDuration > avgDuration + (0.
RETURN studentID, programme, totalDuration, avgDuration, stdDev
ORDER BY programme, totalDuration DESC
```

8.2 Expanding the Model: Towards Richer Insights

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.

8.2.1 Potential Expansions:

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

- **Student Attributes**: Add properties like “international student” or “first-year student” to explore potential student clusters.

The below image shows an graph model expanded with some of the additional data contained within the timetable database. It is much richer and therefore more complex, but this allows for richer analysis.

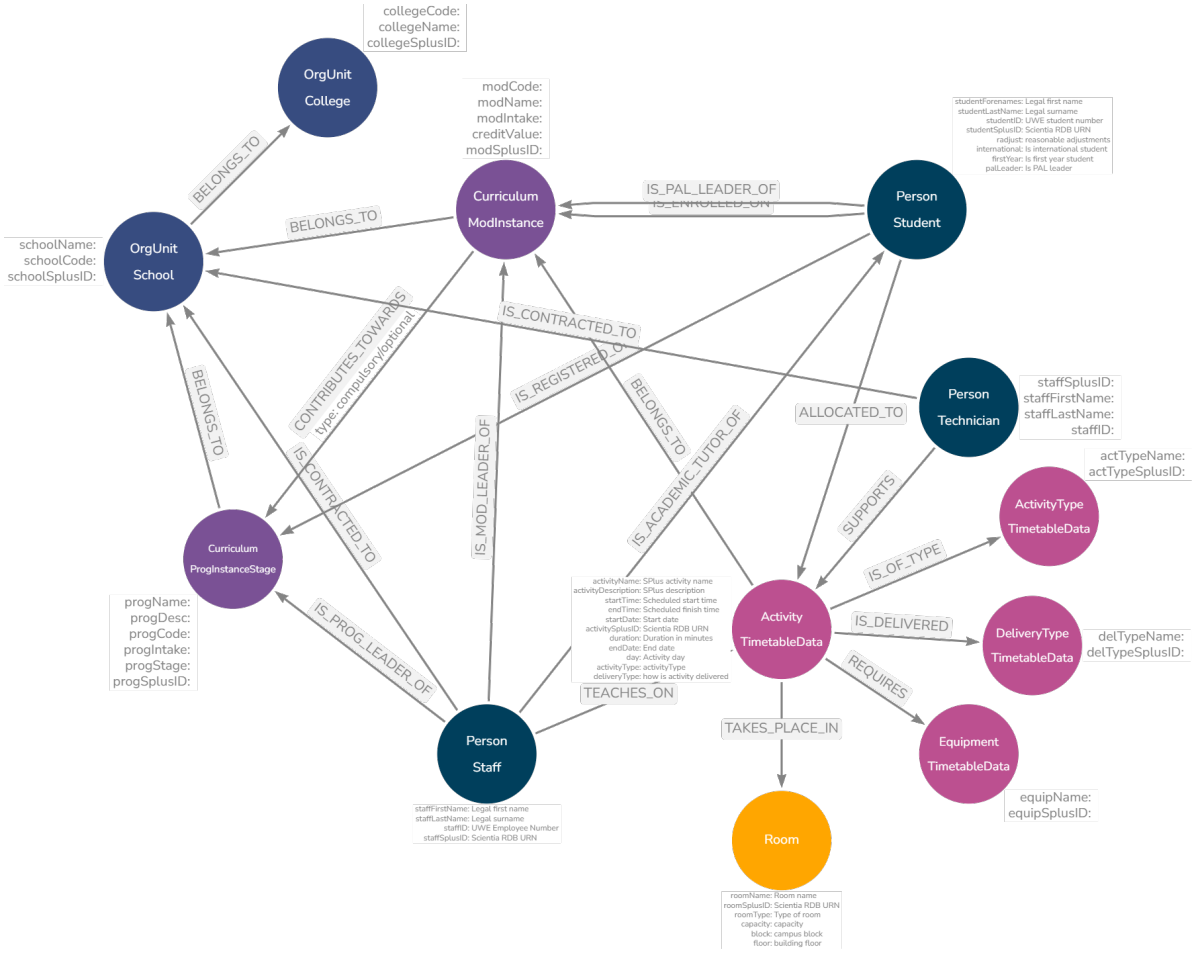


Figure 8.1: Example of Expanded Timetable Graph Model

9 Graphing Time

> ⚠ TODO
>
> - quote on normalisation (footnote)
> - add arrows for each model

The biggest challenge I encountered when modelling timetables into graph data structures involved temporal elements - that is, start and end times, dates, weeks, reccurences, durations, etc. Whether a data element is a node or a property of a node, or perhaps both is relatively trivial to model and test, and depends heavily on use cases. Time, on the other hand, is complicated, especially when you want to write aggregation queries on time elements.

While the conceptual flexibility of graphs is appealing, 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.

9.1 The Problem of Normalised Time

Traditional relational databases often store timetable information in a highly normalised 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.

9.2 Exploring Potential Solutions

To address these challenges, several time modelling approaches were explored, each with its own trade-offs. Let's explore using a fictional example - `Introduction to Graph Databases`:

Table 9.1: Example Source Data (Relational)

Name	Day	StartTime	EndTime	Weeks (11 weeks total)
ITGD	Wednesday	09:00	11:00	1-3, 5-7, 9-13
ITGD	Wednesday	10:00	13:00	4, 7-8, 15
ITGD	Monday	13:00	16:00	4, 7-8, 15

9.2.1 Option 0: Proof-of-concept activity

The basic model created nodes for each activity 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 activities because of the week ranges (number of occurrences).

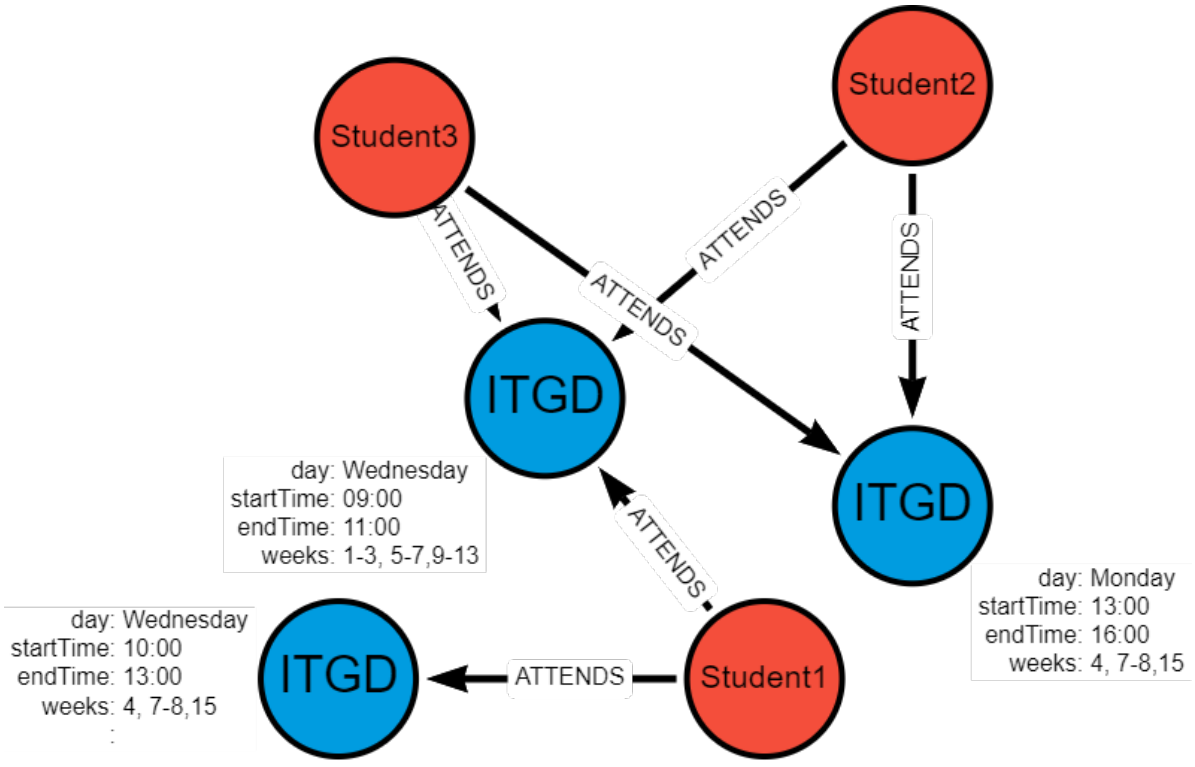


Figure 9.1: Basic example

If all students attend on Wednesday 09:00-11:00 (lecture) and only one of Monday or Thursday (seminar), some students will have a clash in week 7 between 10:00-11:00 . Unpicking this from the normalised activities is very challenging and not immediately obvious at a glance.

9.2.2 Option 1: Unique Activity Nodes

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

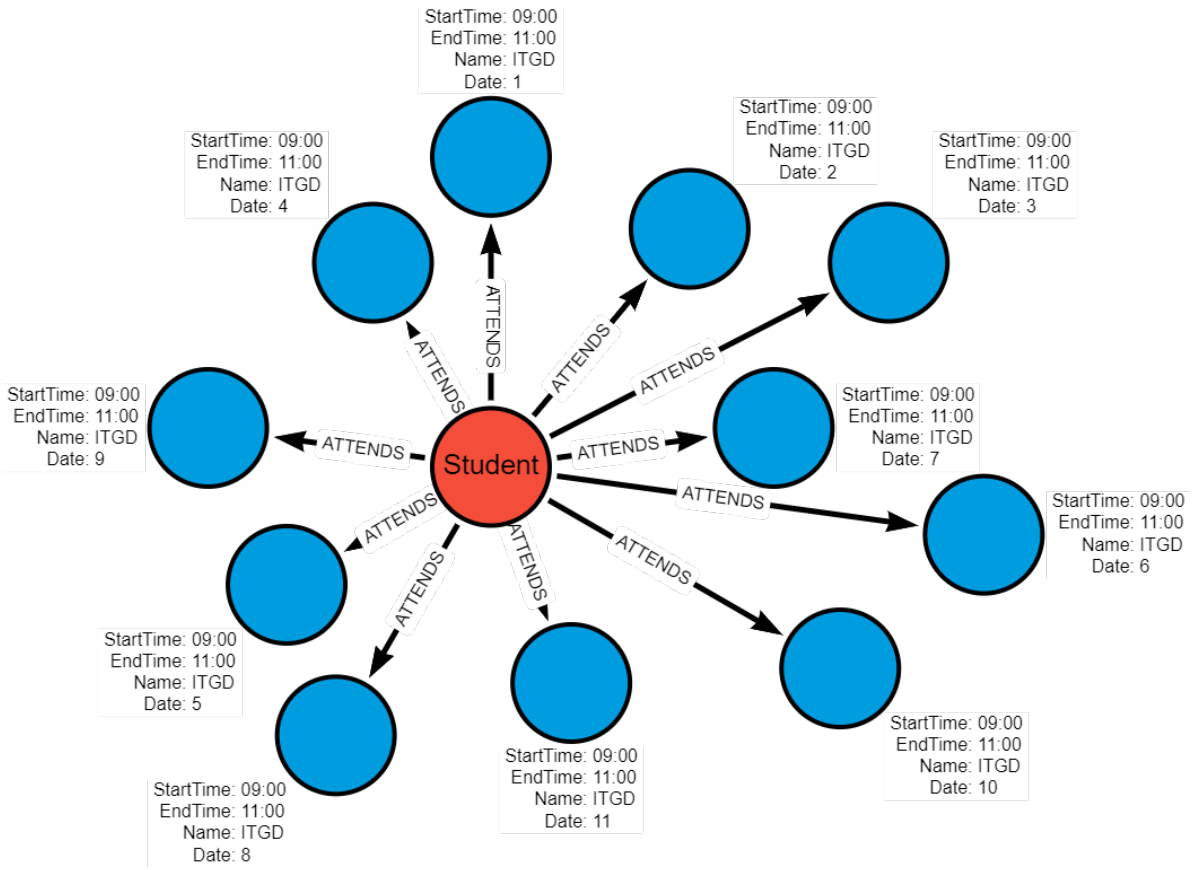


Figure 9.2: Unique Activity Nodes Graph Example

Graph Structure:

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

```
(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.
- **Complex Time-Based Queries**: Answering questions about time patterns or conflicts requires traversing numerous nodes and relationships.

9.2.3 Option 2: Date and Time Nodes

Option 2 creates a single activity node but also additional `date` and `time` nodes, as required.

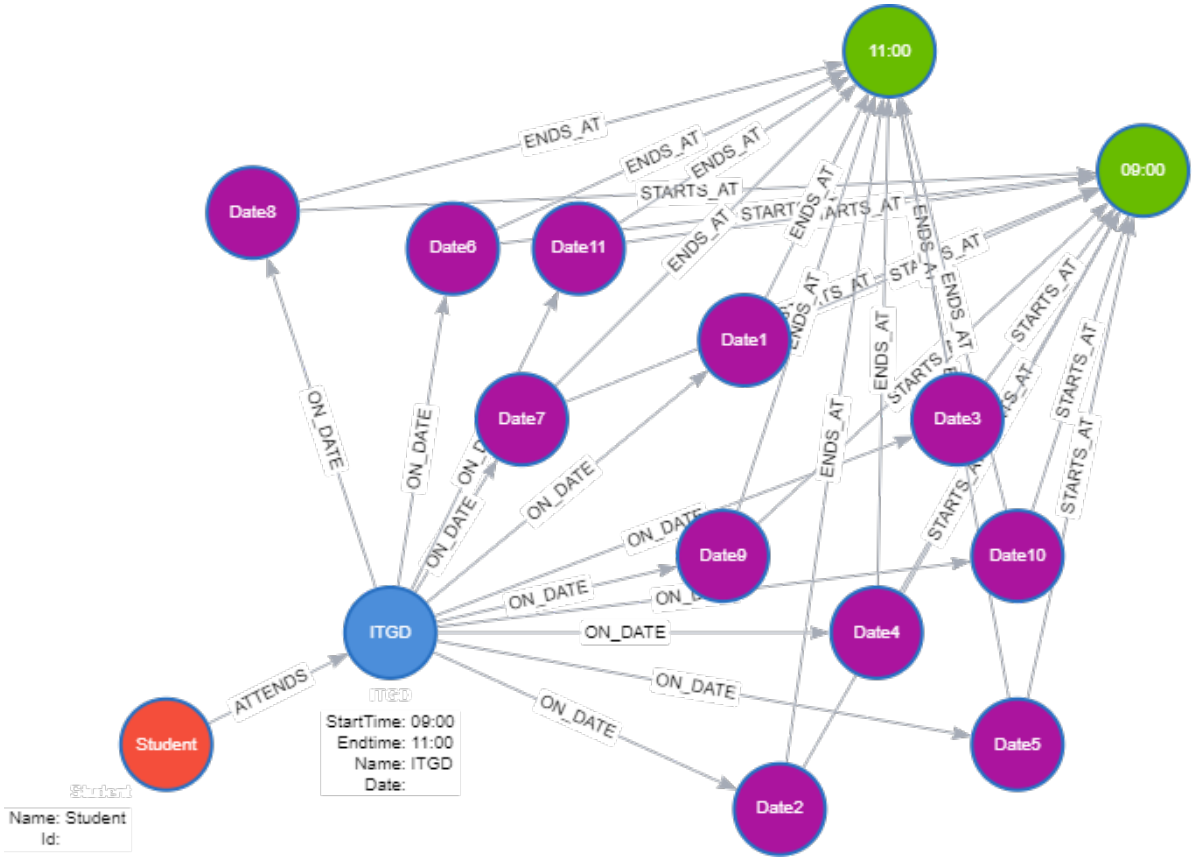


Figure 9.3: Time and Date Nodes

Graph Structure:

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

```
(Activity {Name: "ITGD"}) -[:SCHEDULED_ON]-> (Date {date: "2024-01-03"}) -[:STARTS_AT]-> (Tir
                                             (Date {date: "2024-01-03"}) -[:ENDS_AT]->   (Tir
(Activity {Name: "ITGD"}) -[:SCHEDULED_ON]-> (Date {date: "2024-01-10"}) -[:STARTS_AT]-> (Tir
                                             (Date {date: "2024-01-10"}) -[:ENDS_AT]->   (Tir
...
```

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.

9.2.4 Option 3: Date and Time Block Nodes

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

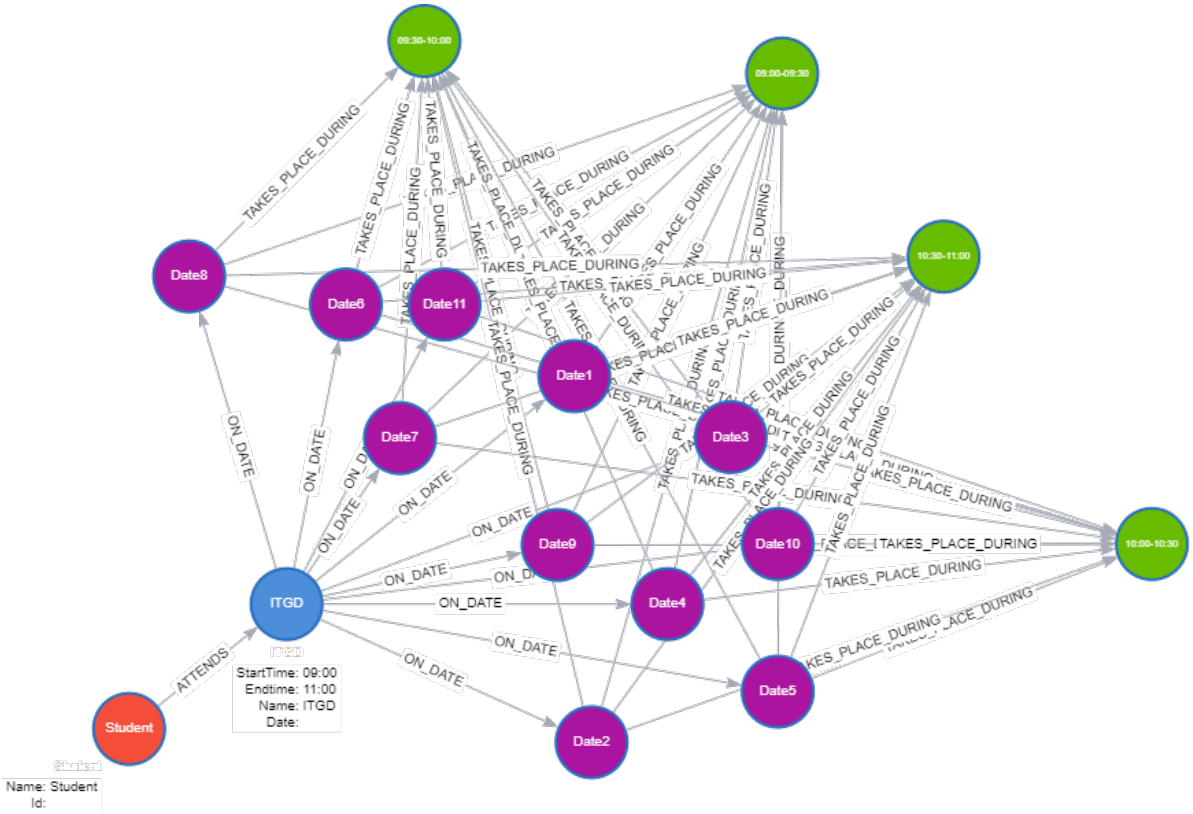


Figure 9.4: TimeBlock and Date Nodes

Graph Structure:

- 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)
  – `Date` -[:TAKES_PLACE_DURING]-> `timeBlock 09:00-09:30` (11 relationships)
  – `Date` -[:TAKES_PLACE_DURING]-> `timeBlock 09:30-10:00` (11 relationships)

```
(Activity {Name: "ITGD"}) -[:SCHEDULED_ON]-> (Date {date: "2024-01-03"}) -[:OCCUPIES]-> (Time
                                                                                -[:OCCUPIES]-> (Ti
...
```

Pros:

- **Granular Time Representation**: Enables analysis at specific time intervals

- **Easier Time Calculations**: Duration is encoded and allows for easy calculations.
- **Reduced Node Count Compared to Option 2**: Offers a balance between granularity and node proliferation.

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

9.2.5 Option: 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.

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:

```
(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})
```

9.2.6 Summary

Table 9.2: Option summary

Option	Pros	Cons
Unique Activities	Simple, direct	High node count, complex time pattern queries
Date & Time	Lower node count, good for time-based queries	More complex relationships
Date & TimeBlock	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 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.

10 Data Engineering Overview

> ⚠ TODO
>
> - add references - neo4j, graphviz

10.1 Overview of the Data Pipeline

The data engineering pipeline is designed to efficiently and securely transfer selected university timetabling data from a relational database (MS SQL) to a graph database (Neo4j), enabling advanced analytics and insights.

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

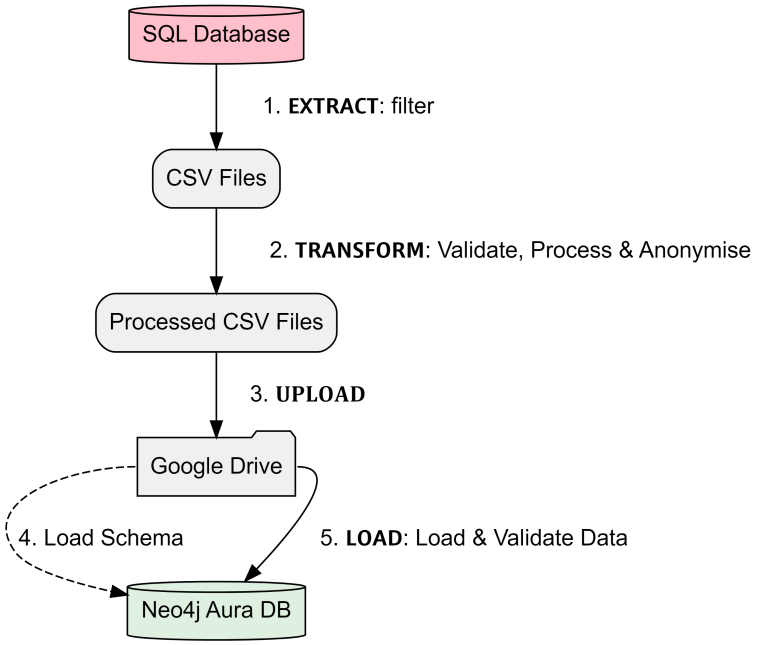
10.1.1 High-level Architecture

The data pipeline consists of these core stages:

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

10.1.2 Design Principles

This pipeline represents a comprehensive approach to data engineering, incorporating several best practices in data handling, processing, and database management.



Data Pipeline Overview

Figure 10.1

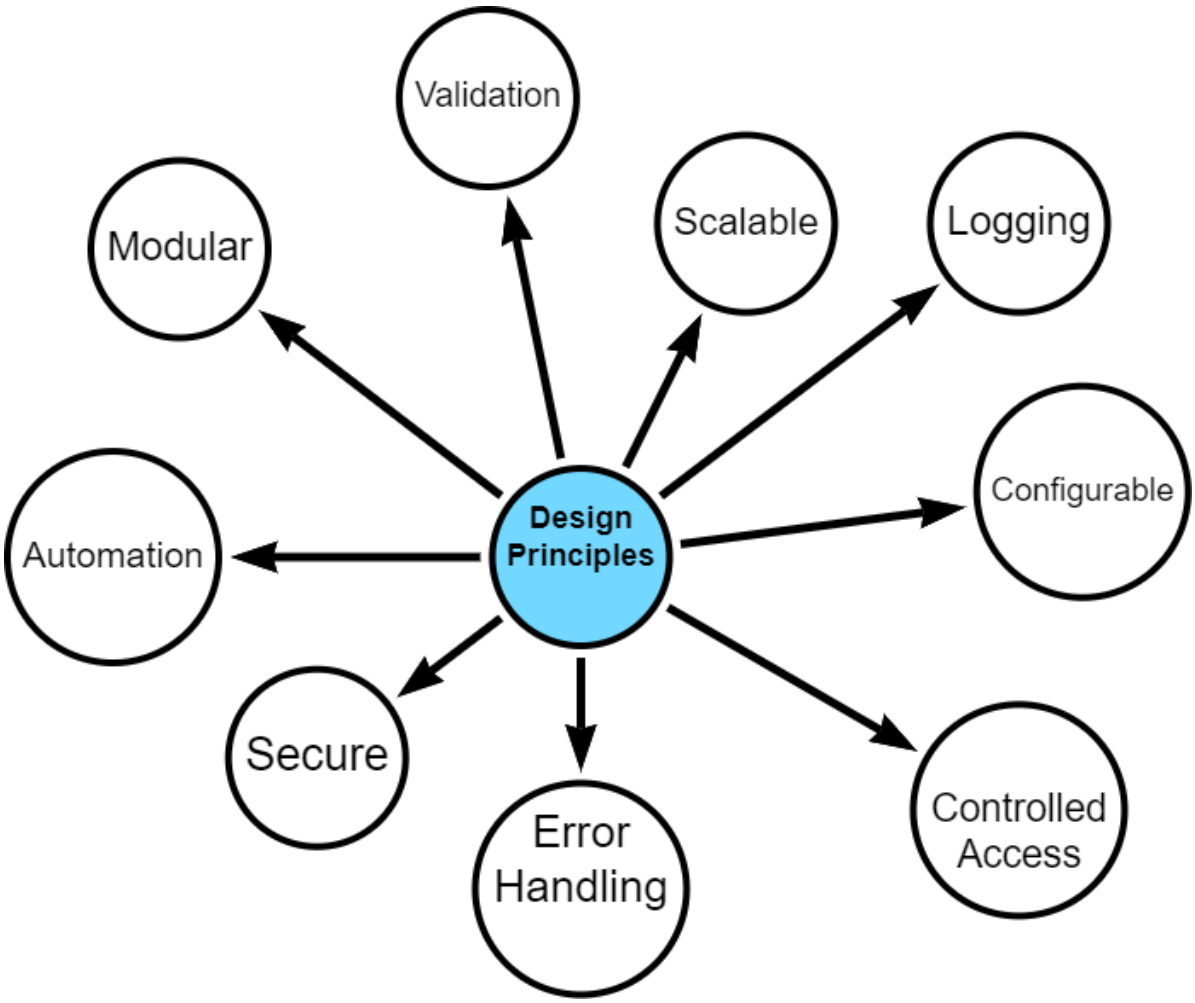


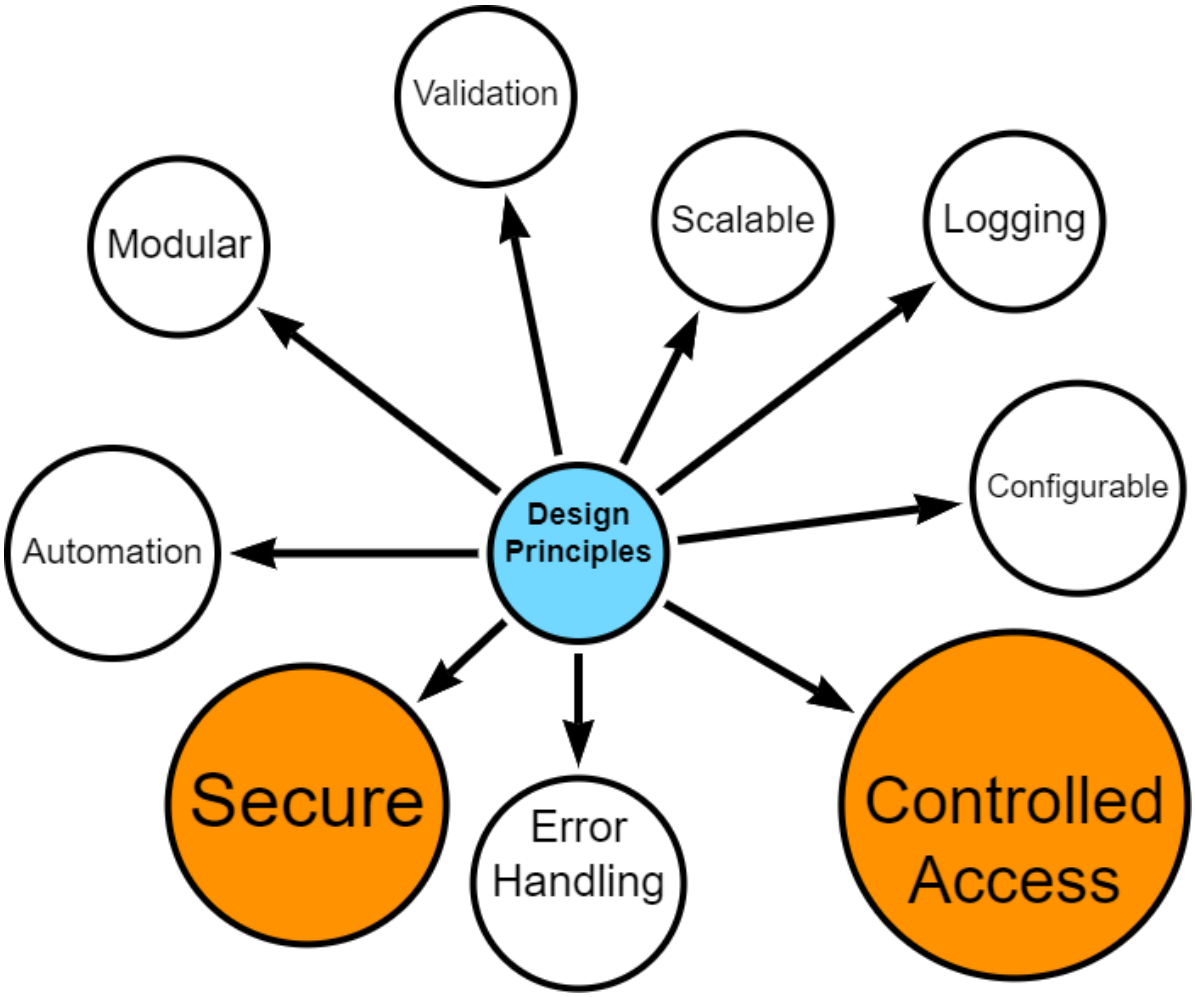
Figure 10.2: Design Principles

The data pipeline is built on several core design principles. I started with a strong sense of what I wanted to achieve - a modular, scalable, secure and configurable design - however, what *exactly* this meant was discovered during the development process.

Given that my project bound by time and word-limits and has additional resource and technology constraints, it was important to make the final artefact one which can be built upon after submission, including potential further development within operational contexts.

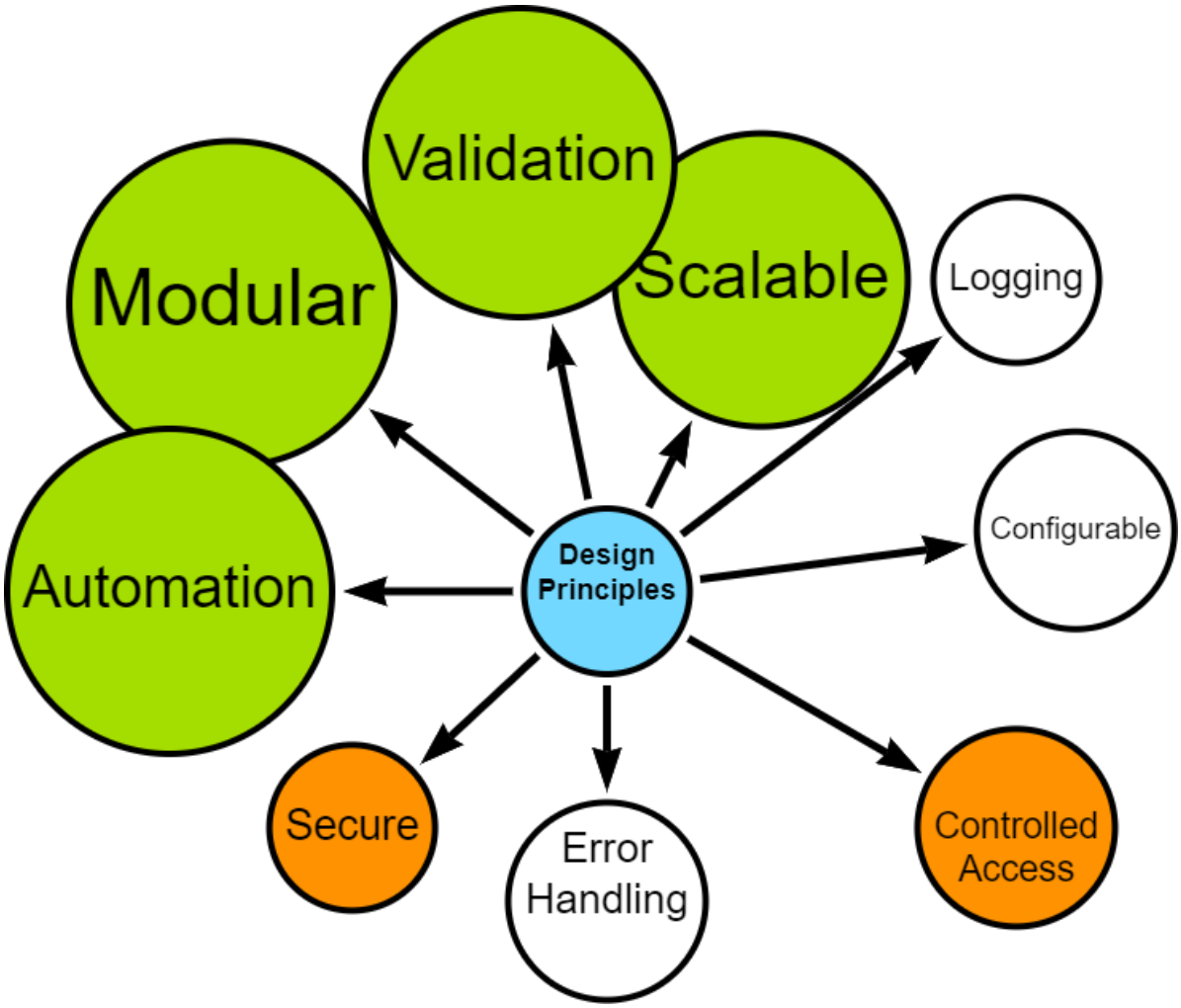
However, the project is also a *proof-of-concept* and as such, some design opportunities were eschewed in favour of simplicity and progress.

10.1.2.1 Security and Data Protection



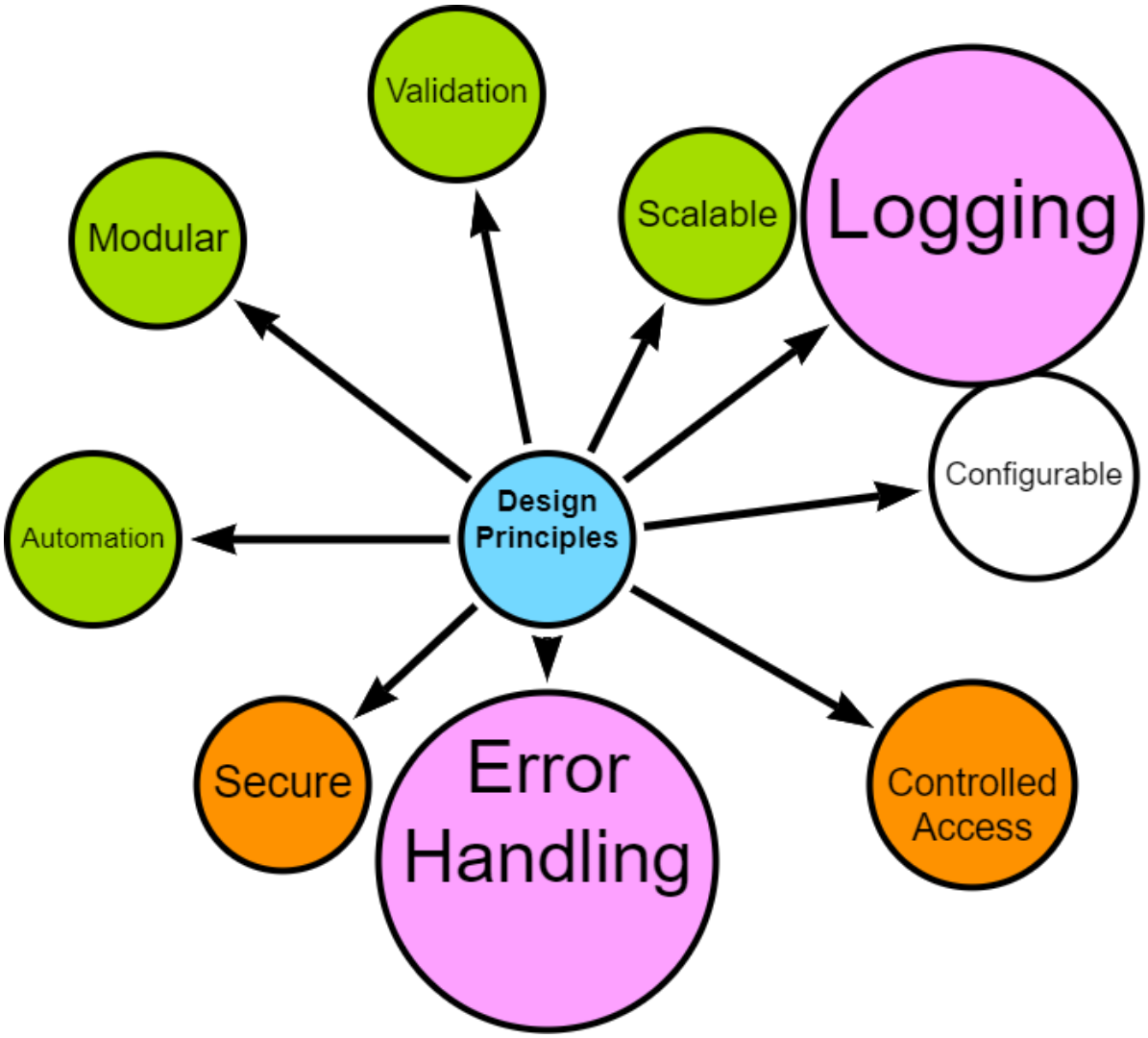
- Secure access controls
- Data anonymisation
- Controlled handling of personally identifiable information

10.1.2.2 Modularity, Scalability and Automation



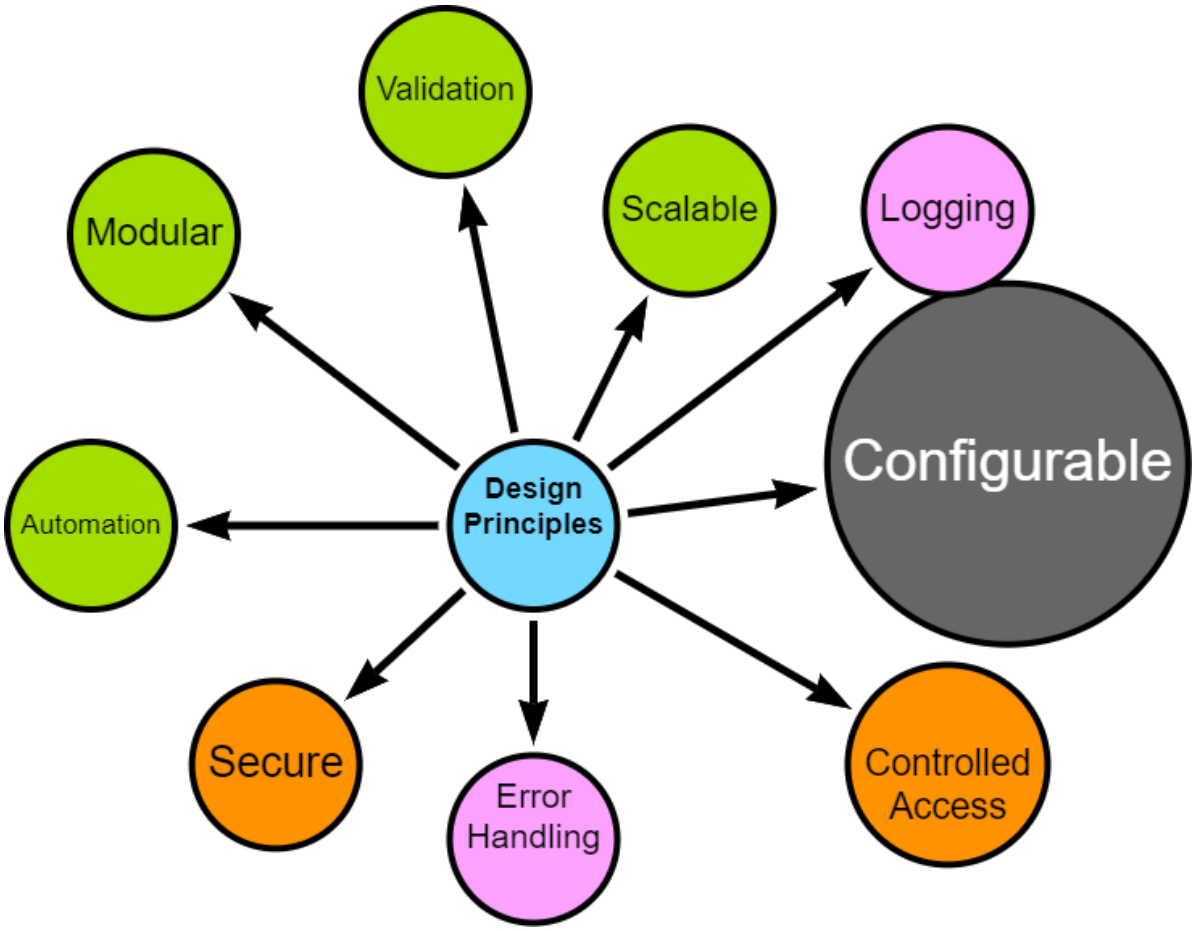
- Distinct, interoperable modules (extract, transform, load)
- Ability to handle increased data volume and complexity
- Automation, where possible
- Configurable data processing options (e.g., data chunking, row processing)
- Optimised, where possible

10.1.2.3 Error Handling and Logging



- Robust error handling
- Comprehensive logging for troubleshooting and auditing

10.1.2.4 User configurable



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

10.1.3 Implementation Approach

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

Crucial aspects of the implementation include:

- **Technology Stack**: Python for data processing, MS SQL for source data, Neo4j for the target graph database. See Appendix 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 componentsare behaving as expected.

10.1.4 Upcoming Sections

The following sections will delve into the specific implementation details of each stage in the pipeline, demonstrating how these principles are put into practice.

I will explore the iterative development process, configuration management, extraction techniques, transformation processes, loading strategies, and automation workflows.

Finally, I will reflect on lessons learned and potential future enhancements to the data engineering components.

11 Data Engineering Approach

> ⚠ TODO
>
> - Add agile reference

I followed an interative, agile-inspired approach when developing the data pipeline, despite being a team of one. This allowed for flexibility, continuous improvement and the opportunity to adapt to new insights during the process. The bulk of my effort was spent *prototyping*, *testing* and *reviewing* with each iteration resulting in a new challenge, issue, opportunity or occasionally, success.

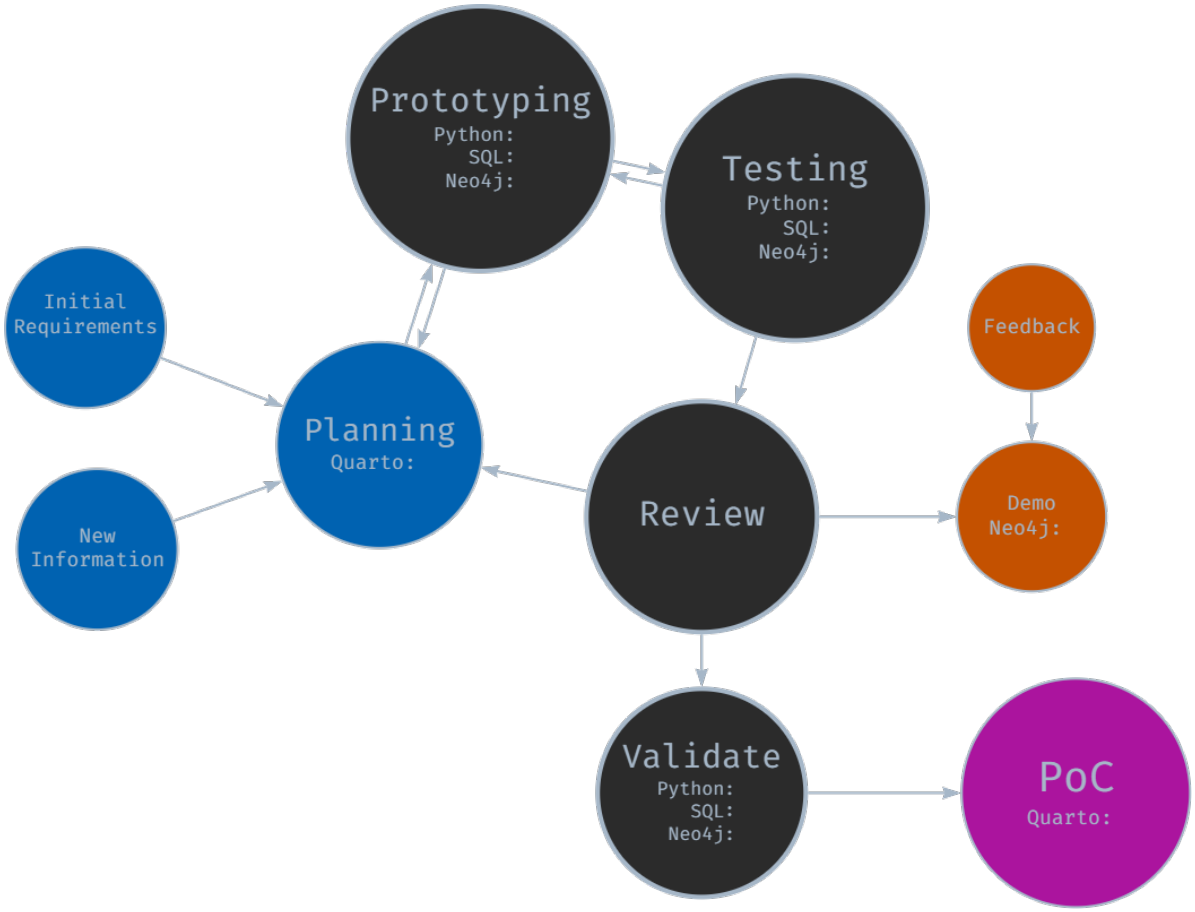


Figure 11.1: Iterative Development Approach

11.0.1 Initial Planning and Requirements Gathering

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

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

11.0.2 Prototyping

Following the 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.

11.0.3 Component-Based Development and Testing

Development proceeded with a focus on individual components:

- 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.

11.0.4 Integration -> Review -> Demo -> Feedback -> Repeat

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

- Components were combined to form larger functional units
- Integrated functionality was occasionally demonstrated to subject matter experts (operational timetablers, timetable manager, 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.

With each new piece of information or change, decisions were required and made - but not always the right ones.

11.0.5 Version Validation and Documentation

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

- End-to-end validation of the entire pipeline was performed.
- Results were documented in notebooks, including opportunities for improvement.
- Any issues (or opportunities) identified were logged for the next iteration.

11.0.6 Continuous Learning and Adaptation

Throughout the development process, learning and adaptation became central to the project's evolution. 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, resisting the temptation to make overly ambitious changes or indeed resisting the temptation to carry on when it would have been better to pause and shore up progress. With practice, I became better at recognising when refactoring was necessary. These experiences underscored the importance of incremental progress and consistent testing in maintaining project stability and direction.

This iterative journey was far from linear. There were many moments of frustration, periods of painstaking troubleshooting, and the constant urge to overdeliver, often exceeding the original proof-of-concept scope. Yet, with each stumble, the process itself became more refined, transforming into a powerful tool for identifying and resolving issues.

While the core MVP (minimum viable product) requirements remained relatively stable (I set them after all!), the iterative approach empowered 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.

This dedication to continuous refinement, while time-consuming, ultimately resulted in a robust, flexible solution that can adapt gracefully to unforeseen challenges and serve as the starting point for future opportunities.

However, this rigorous development process naturally led to a greater focus on data engineering rather than delving into the potential insights offered by Neo4j, simply due to the allocation of time and resources.

The iterative approach proved to be more than just a development methodology. It facilitated personal growth, enhanced technical skills, and improved project management capabilities.

12 Configuration and Logging

TODO

- rewrite detailed options into a summary format
- update if YAML config implemented.

☐ revisit approach doc in final script folder

The configuration and logging approach was to centralise configuration parameters into a python scripts in order to allow the user to manage different aspects of the ETL pipeline. The configuration options are as a result of both initial design and discovery during development.

When I set options during prototyping and testing, I considered whether these are worth parameterising in the ETL by weighing up cost versus value within a proof-of-concept scope. In general, my design is set up to run *automatically* or *dynamically* with well structured data, but I included options to override these settings.

12.0.1 Main Configuration options

An YAML file containing configuration options can be viewed in the Appendix.

General settings include being able to filter which data to extract by way of a list of award codes. This is the default folder name for this ETL run, although default folder names, filepaths and directories can be overwritten and customised. This includes source data files and Google Drive folders for processed data.

Credentials to SQL databases, Neo4j instances and Google drives are stored securely via environment variables or keyring secure storage.

There is an option to specify additional data sources, that is data which does not originate in the timetable database. This has been configured for optionally adding additional location data, e.g. latitude and longitude, square meterage, etc.

12.0.2 Logging

I set up a mechanism which can create a separate loggers with ease. This can be expanded or contracted, as required - for example, during development I had sepaparate loggers for each stage of the ETL to allow me to understand errors. However, if the ETL was developed into a stable release, it can be configured to have one log.

Options include customising the log level (DEBUG, INFO, WARNING, ERROR, CRITICAL).

12.0.2.1 Example Extract Log

12.0.2.2 Example Google Drive Log

As part of the logging functionality, I created a timing function which tracks and stores various execution and elapsed times with a view to optimising performance or identifying bottlenecks in future development.

12.0.2.3 Example Process Log

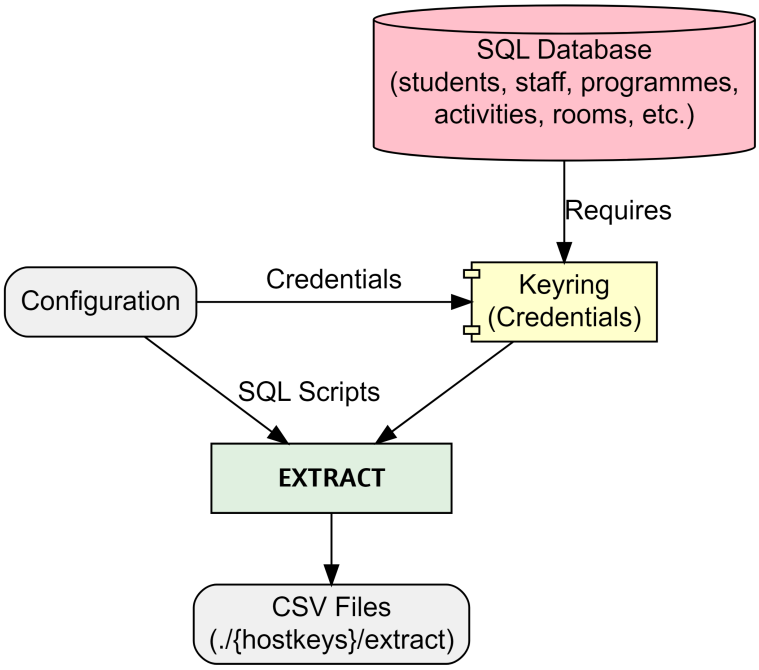
Logging also summarises loading results as can be seen in the snippet below.

12.0.2.4 Example Load Log

13 Extraction

TODO

-



Extract

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

13.0.1 SQL example

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

13.0.2 extract_main.py snippet

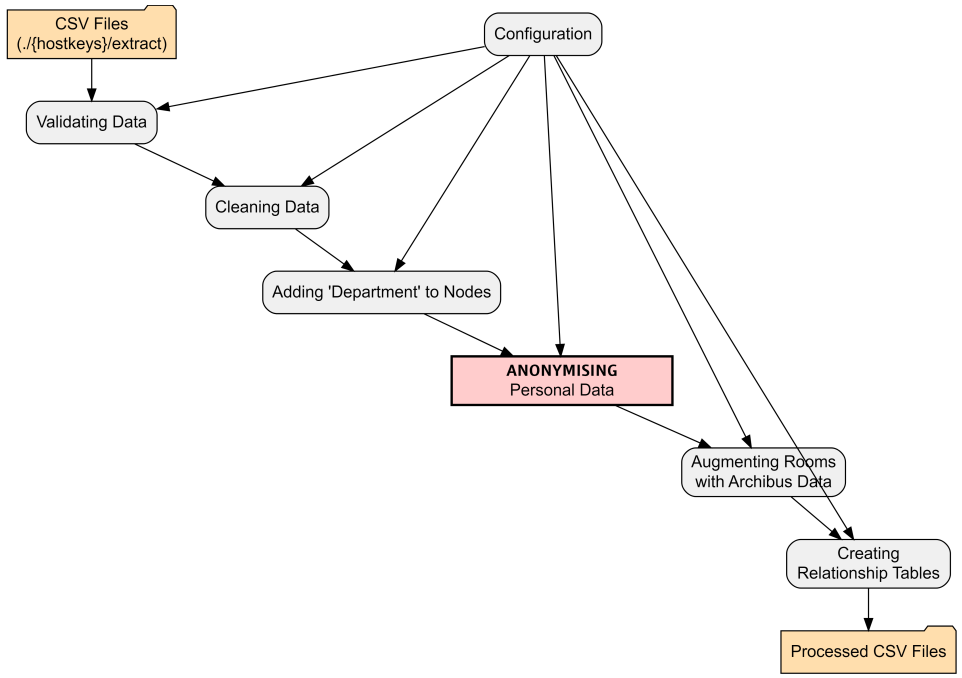
```
# extract_main.py
from logger_config import extract_logger
from extract_data import main as extract_main
from config import EXTRACT_DIR, HOSTKEYS, CHUNK_SIZE
from utils import execution_times

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()
```

14 Transformation

> ⚠ TODO
>
> -

The transformation section of the ETL pipeline picks up from where `extract` finished by using the extracted csv files as the source.



The configuration files allows the user to specify which columns should be used as the unique identifier when determining uniqueness, creating relationships between nodes and linking to additional datasets. It is also possible to specify datatypes - the load process will automatically load properties as `string` unless it is well formatted or the datatype is predetermined. The config file allows the user to specify how to handle certain datatypes like dates, times, boolean, etc.

14.1 All data

1. **Validation** - basic validation of the data is performed. Validation is extensible and can be expanded, as requirements are identified.
2. **Cleaned** - basic cleaning of all data is performed by stripping empty space and removing non-printable characters, etc. using regex. The cleaning functionality can be expanded.

With clean data, the transformation proper starts:

14.2 Nodes and relationships

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

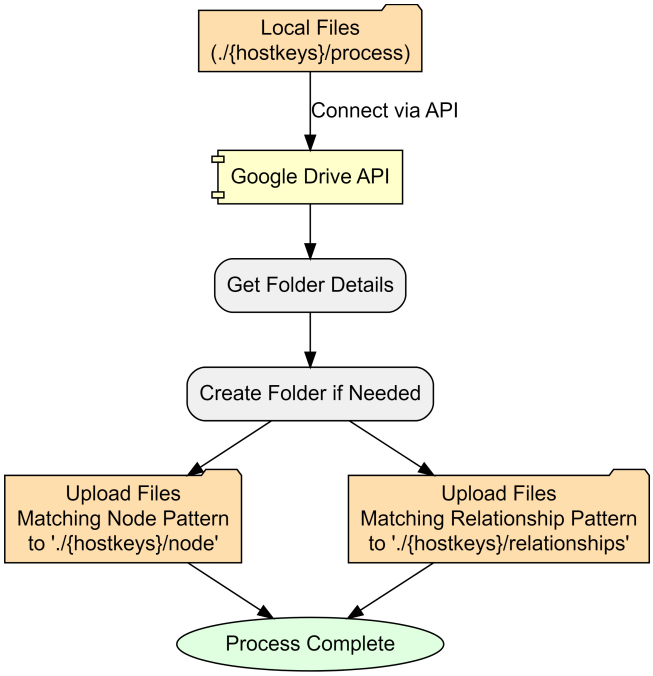
2. **Data Augmentation** - Room data is augmented with additional properties from the location master database, including latitude, longitude, square meterage, etc. Data augmentation is extensible.
3. **Anonymisation** - Personal data is anonymised. An anonymisation function was developed to remove and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

15 Google Load

> **TODO**
>
> - add log files?
> - add screenshot?

As I am using a free instance of Neo4j's graph database - called Neo4j Aura - I needed to overcome some limitations which are not relevant to desktop installations of Neo4j or paid-for cloud instances.

In order to load from csv files, Neo4j Aura requires that the csv files are stored in public cloud storage like Google Drive or Dropbox. Therefore, my project requires this intermediary step.



Configuration settings determine where processed node and relationship files are stored. I have made one folder in my drive public and all ETL files are stored within this root as follows:

- root Google Drive folder

- hostkeys (automatically created, unless override)
  - nodes
  - relationships

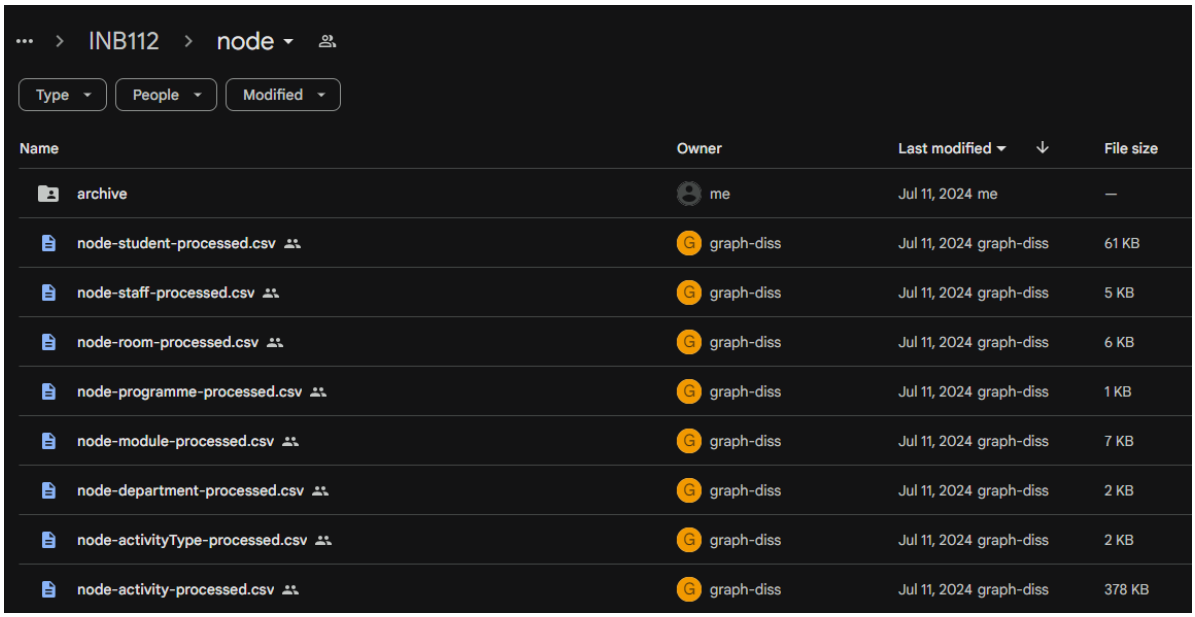
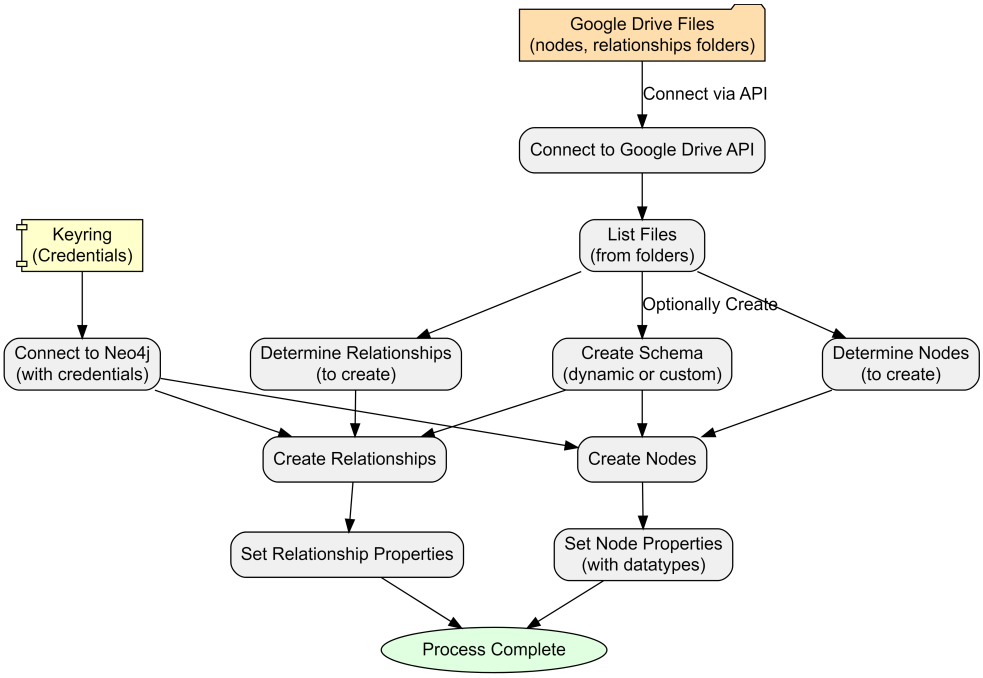


Figure 15.1: Screenshot of Google Drive

16 Neo4j Load

> **TODO**
>
> - add log files?
> - add screenshot?
> - add neo4j loaded data screenshot

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.

The process automatically processes nodes and relationships based on files in the specified folders by using a file-pattern matching approach. However, this can be overridden within configuration.

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!

17 Reflections

From the outset, I recognised that this data engineering project was ambitious in both scale and scope. However, the reality of its magnitude became increasingly apparent as development progressed. 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." 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 been incredibly rewarding, expanding my technical toolkit far beyond my initial expectations. However, it has also contributed to the project's expanding scope, as each new skill acquired opened up possibilities for further enhancements.

Unexpected challenges have been a constant companion throughout this process. From deleted servers and access issues to discrepancies between development environments (such as missing user 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.

17.1 Lessons Learned

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

18 Timetable Metrics

18.1 Timetable Quality Metrics and Insights (1500-2000 words)

18.1.1 4.1 Defining Timetable Quality

18.1.2 4.2 Implemented Metrics

- Constraint violations (max hours per day, days per week, lunch breaks, etc.)
- Distance-based metrics using room properties

18.1.3 4.3 Aggregation Methods

- Student-level, programme-level, and other relevant groupings

18.1.4 4.4 Cypher Queries for Metric Calculation

- Example queries with explanations

18.1.5 4.5 Visualization of Results

- Bloom visualisations or other relevant charts

19 Future Opportunities

19.1 Opportunities

Unveiling Hidden Patterns & Improving Student Experience: Problem: Timetable inefficiencies often remain hidden in relational data, impacting student experience. Graph Solution: Graph analysis can uncover patterns like students with excessive travel time between classes, those lacking adequate breaks, or those facing scheduling conflicts due to part-time work. This empowers universities to optimize timetables for improved student well-being and academic performance. Stakeholder-Centric Analysis & Enhanced Decision Making: Problem: Traditional timetabling often prioritizes one factor (e.g., room utilization) over others, neglecting holistic needs. Graph Solution: Graphs allow simultaneous modeling of student preferences (class times, travel distance), faculty constraints, and institutional priorities (resource allocation). This enables data-driven decisions that balance stakeholder needs and improve overall satisfaction. What-If Scenarios & Agile Timetable Management: Problem: Evaluating the impact of timetable changes in relational systems is cumbersome, hindering proactive planning. Graph Solution: Graph databases excel at simulating "what-if" scenarios. Adding hypothetical courses, adjusting room capacities, or modifying faculty availability becomes straightforward. This agility allows for rapid evaluation of multiple scenarios, enabling institutions to anticipate challenges and adapt timetables dynamically. Visual Exploration & Fostering Collaboration: Problem: Communicating complex timetable data to diverse stakeholders (students, faculty, administrators) is challenging. Graph Solution: Graph visualizations make complex relationships intuitive and accessible, fostering shared understanding. This transparency promotes collaboration, reduces misunderstandings, and facilitates informed decision-making.

19.2 Challenges

Data Migration & Integration: A Necessary Hurdle: Challenge: Migrating from existing relational systems to a graph database requires careful planning and data transformation. Mitigation: Employing robust ETL (Extract, Transform, Load) processes and leveraging graph database import tools can streamline the migration process. Prioritizing incremental migration, starting with core entities, can minimize disruption. Tooling and Expertise: Bridging the Skills Gap: Challenge: The graph database ecosystem, while maturing, might require specialized skills compared to traditional SQL. Mitigation: Investing in staff training, collaborating with experts, and leveraging online resources can address the skills gap. Open-source graph

databases like Neo4j offer ample learning material and community support. Performance at Scale: Ensuring Responsiveness with Large Datasets: Challenge: Graph databases, while generally performant for connected data, might face challenges with extremely large universities and complex queries. Mitigation: Employing performance tuning techniques like indexing, caching, and query optimization can enhance scalability. Exploring specialized graph database solutions designed for high-volume transactional systems might be necessary in extreme cases. "Soft" Constraint Modeling: Quantifying Subjective Preferences: Challenge: Graphs excel at explicit relationships but struggle with subjective preferences (e.g., student aversion to late classes). Mitigation: Combine graph analysis with techniques like sentiment analysis on student feedback or preference elicitation surveys. This hybrid approach allows incorporating both explicit relationships and quantified subjective factors.

2.4 Data Augmentation Opportunities

Data Augmentation Opportunities: You touch on this briefly; expanding this section could be very compelling. Example: Integrating room location data (latitude/longitude) with student address data could allow for powerful analyses of commute patterns and potential inequities. Ethical Considerations: As your project deals with student data, briefly mention the importance of data privacy, anonymization, and responsible use of insights.

2.5 Challenges and Considerations

Potential limitations of the graph approach Data migration considerations Performance considerations for large-scale timetabling systems

19.3 Future Opportunities and Potential Insights (500 words)

- Discussion of potential analyses (module combinations, student clustering, etc.)
- Integration of additional data sources

19.4 Exploring time

Future work could include evaluating the performance and scalability of different time modeling options, particularly:

- **Dynamic Node Creation (Option 2 or 3)**: This approach would create `Time` or `TimeBlock` nodes only when needed, potentially offering a good balance between flexibility and performance.
- **Direct Performance Comparisons**: Conducting benchmarks against specific use cases and datasets will provide valuable insights for choosing the optimal approach for large-scale deployments.

Modeling time effectively is crucial for unlocking the full potential of a graph database for university timetabling analysis. This section has outlined the challenges, explored potential solutions, and documented the chosen approach for this proof of concept.

Further exploration and optimisation of time modeling will be essential for developing robust, scalable, and insightful graph-based timetabling solutions. The next section will delve into the data engineering pipeline required to populate and maintain this model, bridging the gap between raw data and insightful analysis.

20 Conclusion

20.1 Conclusion (500 words)

- Summary of key achievements
- Reflection on the project's impact and potential for timetabling processes
- Future work and recommendations

21 Random Graph Generator

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

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

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

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

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

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

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

    for i in range(num_nodes):
```

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

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

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

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

    return dot
```

22 Technology Stack

22.1 Technology Stack

TO ADD:

- Technology Stack
  - Python used
  - Python packages used and why
  - Neo4j used
  - Google API
  - VS code
  - Quarto
  - SQL
- graphviz
- arrows
- mermaid
- git hub, git

23 Configuration YAML

23.1 Config

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

```yaml
# ETL Pipeline Configuration

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

file_paths:
  root_dir: '.'  # default to current working directory
  nodes_folder_url: # (Optional) override for dynamic lookup) eg "https://drive.google.com/d
  relationships_folder_url: # (Optional) override for dynamic lookup) eg."https://drive.goog
  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"
```

24 Anonymisation

> ⚠ TODO
>
> - wrtie a summary
> - show before and after
> - staff/student
> - columns
> - consistent changing - uses random seed to ensure

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


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

    # Determine if it's 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

    # Create a 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 DataFrame 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)

        # Remove 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 dataframe if an error occurs
```

25

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].

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].

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

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

Hutson, G. and Jackson, M. (2023) Graph Data Modeling in Python [online]. Packt Publishing. [Accessed 24 December 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].

Lee, V., Nguyen, P.K. and Thomas, A. (2023) Graph-Powered Analytics and Machine Learning with TigerGraph [online]. [Accessed 4 November 2023].

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].

Mandal, A.K. (2020) Development of an Interactive Tool based on Combining Graph Heuristic with Local Search for Examination Timetable Problem. International Journal of Advanced Computer Science and Applications [online]. 11 (3). Available from: https://www.proquest.com/docview/2655156280/abstract/33CF4A9244324D32PQ/1 [Accessed 18 July 2024].

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

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

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

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

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

Neo4j (2023) The Neo4j Cypher Manual v5.

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

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

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

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

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

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

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

Thomas, J.J., Khader, A.T. and Belaton, B. (2009) Visualization Techniques on the Examination Timetabling Pre-processing Data. In: Imaging and Visualization 2009 Sixth International Conference on Computer Graphics [online]Imaging and Visualization 2009 Sixth International Conference on Computer Graphics. pp. 454–458. Available from: https://ieeexplore.ieee.org/document/5298764 [Accessed 25 January 2024].

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

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