CITS5504 Project 2: Graph Database Design and Cypher Query

Laine Mulvay (22708032)

1. Design and Implementation Process

1.1 Graph Data Modelling Approach

Our graph database follows the property graph model, providing an optimal structure for representing road crash data relationships. We employed four key node types with specific design rationales:

Node Selection Rationale and Critical Evaluation

1. **Crash Nodes**
   - **Design Choice**: Central hub in our graph model
   - **Implementation**: Based on 'Crash ID' with crash-specific attributes
   - **Strengths**:
     - Effectively centralises multiple vehicle/person involvements in a single incident
     - Matches the source data structure where crashes are the primary events
   - **Limitations**:
     - Creates a strict star topology that makes direct LGA-to-LGA paths impossible
     - Introduces an unavoidable additional hop in path traversals
2. **Person Nodes**
   - **Design Choice**: Independent nodes rather than crash properties
   - **Implementation**: Using original 'ID' as personId
   - **Strengths**:
     - Preserves the 1:1 relationship in our source data between person and crash
     - Supports demographic filtering (e.g., crashes involving young male drivers)
   - **Limitations**:
     - Limited value for our dataset where each person appears only once
     - Increases path length when querying location-to-location connections
3. **Location Nodes**
   - **Design Choice**: Composite nodes capturing geographic hierarchy
   - **Implementation**: Combined state, remoteness, SA4 and LGA identifiers
   - **Strengths**:
     - Captures Australia's geographic administrative structure
     - Allows filtering by various geographic levels (state vs. local)
   - **Limitations**:
     - **Path queries between LGAs require minimum 4 hops** (LGA→Crash→LGA), making Question F impossible as specified
     - Composite keys created implementation complexity in our ETL process
4. **DateTime Nodes**
   - **Design Choice**: Separate temporal dimension nodes
   - **Implementation**: Composite time identifiers from month, year and time
   - **Strengths**:
     - Supports time-based analysis (particularly useful for holiday period analysis)
     - Efficiently represents recurring temporal patterns in Australian crash data

- **Limitations**:
  - Source data lacks exact dates, limiting granularity
  - Adds another traversal hop in multi-dimensional queries

1.2 Relationship Design

Our relationships reflect natural connections in the domain:

- **INVOLVED_IN**: Person → Crash (based on 'Crash ID' linking to person records)
- **OCCURRED_AT**: Crash → Location (using composite location identifier)
- **HAPPENED_AT**: Crash → DateTime (using composite datetime identifier)

This structure optimises both traversal performance and query flexibility, particularly for pattern-matching and path-finding operations that would be cumbersome in relational databases.

1.3 Graph Design Visualisation

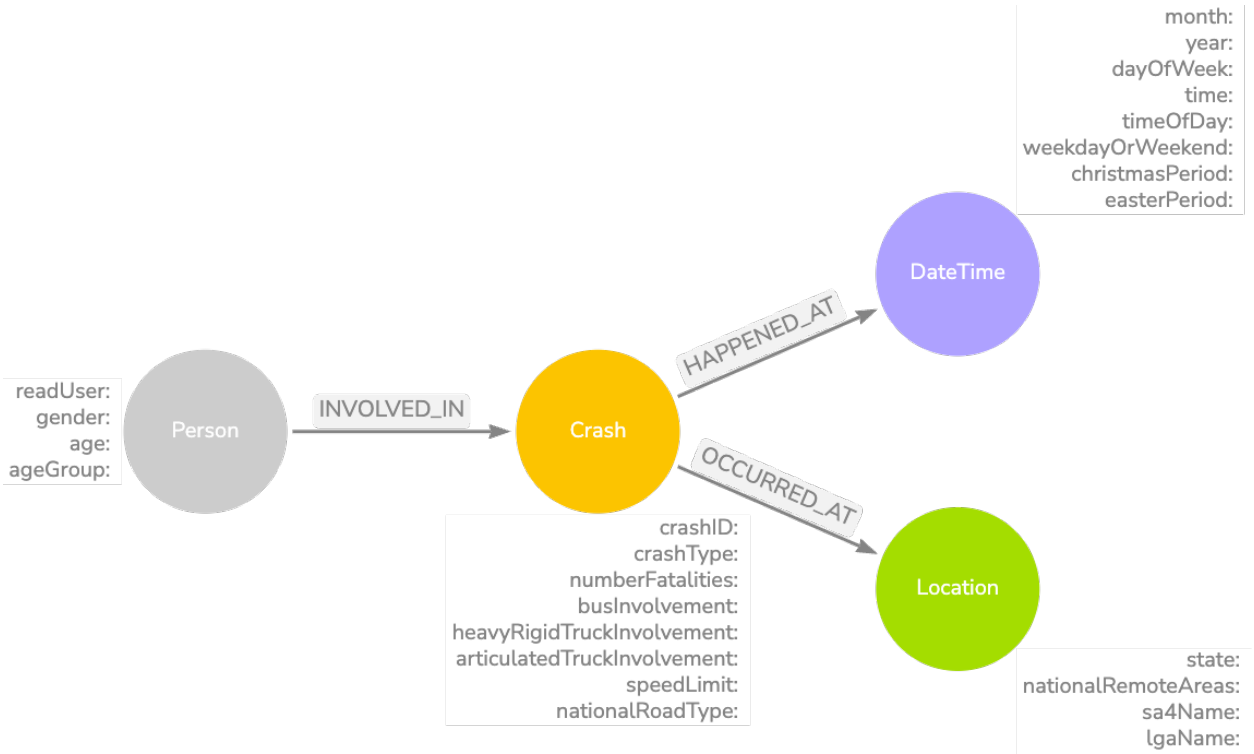

Figure 1: Graph Database Design created using Arrows.app

The implemented design balances normalisation with query performance, following Neo4j best practices for domain-driven design while maintaining the contextual integrity of the Australian road crash data.

2. ETL Process

2.1 Extract

The initial data was extracted from the ARDD: Fatalities—December 2024—XLSX dataset, which was processed to remove all missing/unknown values, modify existing values, and add new columns. The cleaned dataset was stored in `/data/raw/Project2_Dataset_Corrected.csv` and contains 10,490 records (excluding the header row) with 25 columns of crash-related information.

Note on Data Filtering: While the project guidelines suggested that filtering might be necessary for performance reasons ("If you experience long query time, consider reducing the dataset size by filtering out records, such as deleting records for specific State/Year"), we found that our implementation was able to handle the full dataset efficiently. This was achieved through: 1. Proper indexing of frequently queried properties 2. Optimised node and relationship creation 3. Efficient query design

Therefore, no filtering was performed on the dataset, and we maintained all 10,490 records for our analysis. This ensures that our results represent the complete dataset without any data loss or bias that might have been introduced through filtering.

2.2 Transform

The transformation process was implemented using Python (ETL.py), focusing on converting the raw CSV data into a graph structure that matches our Arrows design. The process involves several key steps:

1. **Data Preparation and Unique ID Creation**:

```
# Create unique IDs for nodes
df['personId'] = df['ID'].astype(str)  # Using the original ID as personId
df['locationId'] = df['State'] + '_' + df['National Remoteness Areas'] + '_' + df['SA4 Name 2021'] + '_
df['dateTimeId'] = df['Month'].astype(str) + '_' + df['Year'].astype(str) + '_' + df['Time']
```

These unique IDs are crucial for maintaining data integrity and establishing relationships between nodes.

2. **Node Creation**: The script creates separate dataframes for each node type defined in our graph design: *Note we change to camelCase for the node Properties*

```
# Person nodes
person_df = df[['personId', 'Road User', 'Gender', 'Age', 'Age Group', 'Crash ID']].copy()
person_df.columns = ['personId', 'roadUser', 'gender', 'age', 'ageGroup', 'crashId']

# Crash nodes
crash_df = df[['Crash ID', 'Crash Type', 'Number Fatalities', 'Bus Involvement',
               'Heavy Rigid Truck Involvement', 'Articulated Truck Involvement',
               'Speed Limit', 'National Road Type']].copy()
crash_df.columns = ['crashId', 'crashType', 'numberFatalities', 'busInvolvement',
                    'heavyRigidTruckInvolvement', 'articulatedTruckInvolvement',
                    'speedLimit', 'nationalRoadType']

# Location nodes
location_df = df[['locationId', 'State', 'National Remoteness Areas', 'SA4 Name 2021',
                  'National LGA Name 2024', 'Crash ID']].copy()
location_df.columns = ['locationId', 'state', 'nationalRemoteAreas', 'sa4Name',
                       'lgaName', 'crashId']
```

```
# DateTime nodes
dateTime_df = df[['dateTimeId', 'Month', 'Year', 'Dayweek', 'Time', 'Time of day',
                  'Day of week', 'Christmas Period', 'Easter Period', 'Crash ID']].copy()
dateTime_df.columns = ['dateTimeId', 'month', 'year', 'dayOfWeek', 'time', 'timeOfDay',
                       'weekdayOrWeekend', 'christmasPeriod', 'easterPeriod', 'crashId']
```

3. **Duplicate Removal**: Since we're creating distinct node labels in our graph database, we need to ensure each node represents a unique entity. For example, a location node should represent a unique LGA (Local Government Area) rather than being duplicated for each crash that occurred there. Therefore, we remove duplicates based on our unique identifiers:

```
# Remove duplicates to ensure unique nodes
crash_df = crash_df.drop_duplicates(subset=['crashId'])
location_df = location_df.drop_duplicates(subset=['locationId'])
dateTime_df = dateTime_df.drop_duplicates(subset=['dateTimeId'])
```

4. **Relationship Preparation**: The script prepares relationship dataframes that will be used to create connections between nodes based on the ids we previously created:

```
# For INVOLVED_IN relationship - Person to Crash
person_crash_rel = person_df[['personId', 'crashId']].copy()

# For OCCURED_AT relationship - Crash to Location
crash_location_rel = df[['Crash ID', 'locationId']].copy()
crash_location_rel.columns = ['crashId', 'locationId']
crash_location_rel = crash_location_rel.drop_duplicates()

# For HAPPENED_AT relationship - Crash to DateTime
crash_dateTime_rel = df[['Crash ID', 'dateTimeId']].copy()
crash_dateTime_rel.columns = ['crashId', 'dateTimeId']
crash_dateTime_rel = crash_dateTime_rel.drop_duplicates()
```

The script prepares relationship dataframes that will be used to create connections between nodes based on the ids we previously created. For example, if a crash with ID "12345678" resulted in 2 fatalities, it would appear twice in our raw data (once for each fatality). However, we only want one HAPPENED_AT relationship between this crash and its datetime, and one OCCURED_AT relationship between this crash and its location. Therefore, we drop duplicates to ensure each crash has exactly one relationship to its location and datetime:

5. **CSV Export**: Finally, the transformed data is exported to CSV files that will be used in the Neo4j import process:

```
# Export node CSVs
person_df.to_csv(f"{output_dir}/person_nodes.csv", index=False)
crash_df.to_csv(f"{output_dir}/crash_nodes.csv", index=False)
location_df.to_csv(f"{output_dir}/location_nodes.csv", index=False)
dateTime_df.to_csv(f"{output_dir}/dateTime_nodes.csv", index=False)

# Export relationship CSVs
person_crash_rel.to_csv(f"{output_dir}/person_crash_rel.csv", index=False)
crash_location_rel.to_csv(f"{output_dir}/crash_location_rel.csv", index=False)
crash_dateTime_rel.to_csv(f"{output_dir}/crash_dateTime_rel.csv", index=False)
```

This transformation process ensures that: - Each node type has appropriate properties as defined in our graph design - Relationships between nodes are properly established - Data integrity is maintained through unique identifiers - The structure matches our Arrows design visualisation - The data is ready for import into Neo4j

2.3 Load

The load process involved setting up the Neo4j database and importing the transformed data. This required several steps:

1. **Database Setup**:
   - Create a new project in Neo4j Desktop
   - Add a new local DBMS
   - Start the database
   - Open Neo4j Browser
2. **File Import**:
   - Open the import folder in Neo4j
   - Add CSV files from `/data/neo4j_import` one by one
3. **Database Initialisation**: The following steps were performed in the Neo4j Browser to set up the database structure:

   Note: The complete Cypher load queries for all node, relationship, and index creation steps are provided in the first half of `/scripts/scripts.txt`.

2.3.1 Constraints

Constraints are essential for maintaining data integrity in the graph database. They ensure that each node has a unique identifier, preventing duplicate nodes and maintaining the accuracy of relationships between entities.

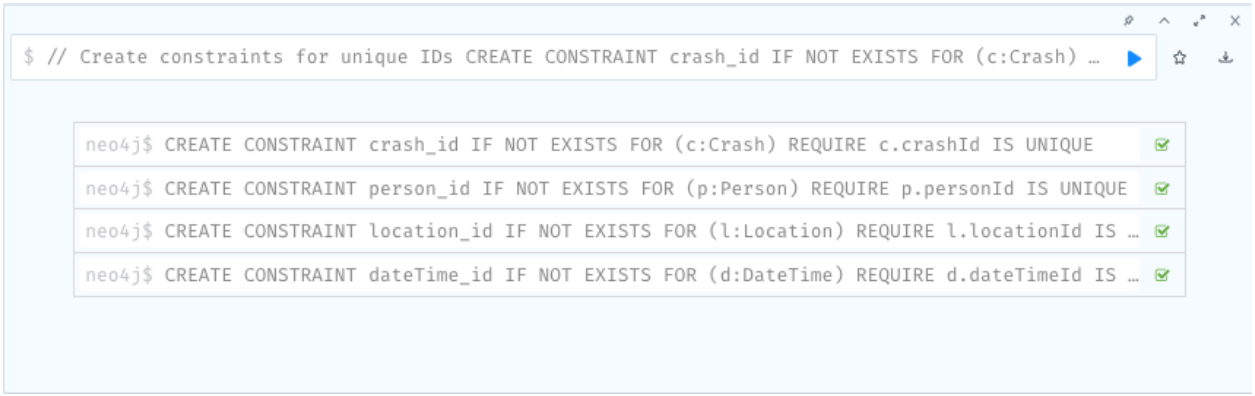

Figure 2: Creating unique constraints for node identifiers in Neo4j

2.3.2 Node Creation

This node creation process: 1. Loads data from the `crash_nodes.csv` file that was added to the Neo4j import folder 2. Creates Crash nodes with properties matching our graph design 3. Converts numeric values using `toInteger()` function 4. Establishes the foundation for relationships with other nodes

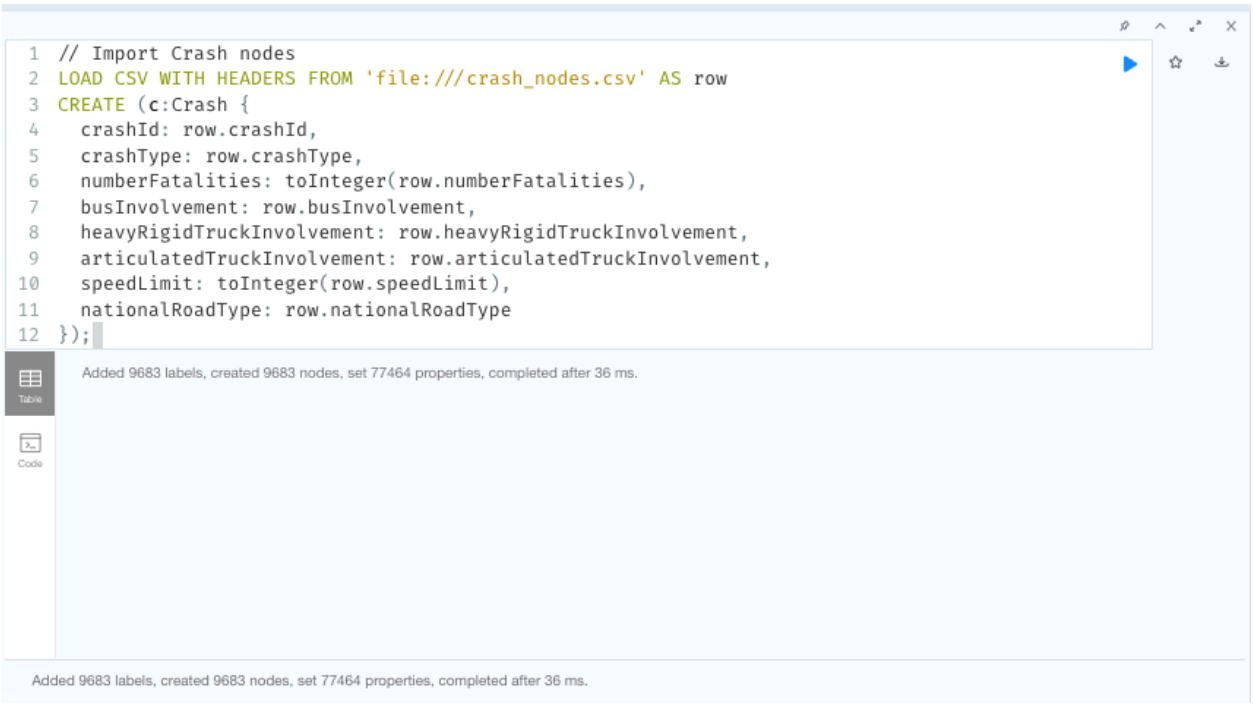

Figure 3: Creating Crash nodes from CSV in Neo4j

Similar LOAD CSV commands were executed for other node types (Person, Location, DateTime), following the same pattern of loading from their respective CSV files in the import folder.

2.3.3 Relationship Creation

This relationship creation process: 1. Loads relationship data from the `person_crash_rel.csv` file 2. Matches existing Person and Crash nodes using their unique identifiers 3. Creates the INVOLVED_IN relationship between matched nodes 4. Establishes the connection between people and the crashes they were involved in

Relationships (such as `INVOLVED_IN`, `OCCURRED_AT`, and `HAPPENED_AT`) were created by loading relationship CSVs and matching nodes by their unique identifiers.

Figure 4: Creating relationships between nodes in Neo4j

2.3.4 Index Creation

Indexes are crucial for query performance optimisation. They work by creating a sorted data structure for frequently queried properties, enabling faster lookups when filtering or matching on these properties. For example: - The `crash_speed_idx` index optimises queries that filter crashes by speed limit - The `location_state_idx` index speeds up queries that search for crashes in specific states - The `person_age_group_idx` index improves performance when analysing crashes by age groups

```
// Create indexes for better performance
CREATE INDEX crash_speed_idx IF NOT EXISTS FOR (c:Crash) ON (c.speedLimit);
CREATE INDEX crash_articulated_idx IF NOT EXISTS FOR (c:Crash) ON
(c.articulatedTruckInvolvement);
CREATE INDEX crash_fatalities_idx IF NOT EXISTS FOR (c:Crash) ON (c.numberFatalities);
CREATE INDEX crash_road_type_idx IF NOT EXISTS FOR (c:Crash) ON (c.nationalRoadType);
CREATE INDEX location_state_idx IF NOT EXISTS FOR (l:Location) ON (l.state);
CREATE INDEX location_lga_idx IF NOT EXISTS FOR (l:Location) ON (l.lgaName);
CREATE INDEX dateTime_year_idx IF NOT EXISTS FOR (d:DateTime) ON (d.year);
CREATE INDEX person_age_group_idx IF NOT EXISTS FOR (p:Person) ON (p.ageGroup);
CREATE INDEX person_road_user_idx IF NOT EXISTS FOR (p:Person) ON (p.roadUser);
```

```
neo4j$ CREATE INDEX crash_speed_idx IF NOT EXISTS FOR (c:Crash) ON (c.speedLimit)
neo4j$ CREATE INDEX crash_articulated_idx IF NOT EXISTS FOR (c:Crash) ON (c.articulatedTruc…
neo4j$ CREATE INDEX crash_fatalities_idx IF NOT EXISTS FOR (c:Crash) ON (c.numberFatalities)
neo4j$ CREATE INDEX crash_road_type_idx IF NOT EXISTS FOR (c:Crash) ON (c.nationalRoadType)
neo4j$ CREATE INDEX location_state_idx IF NOT EXISTS FOR (l:Location) ON (l.state)
neo4j$ CREATE INDEX location_lga_idx IF NOT EXISTS FOR (l:Location) ON (l.lgaName)
neo4j$ CREATE INDEX dateTime_year_idx IF NOT EXISTS FOR (d:DateTime) ON (d.year)
neo4j$ CREATE INDEX person_age_group_idx IF NOT EXISTS FOR (p:Person) ON (p.ageGroup)
neo4j$ CREATE INDEX person_road_user_idx IF NOT EXISTS FOR (p:Person) ON (p.roadUser)
```

Figure 5: Creating indexes in Neo4j

These indexes are particularly important for our query requirements, such as: - Finding crashes in specific states (using `location_state_idx`) - Analysing crashes by age groups (using `person_age_group_idx`) - Filtering crashes by speed limits (using `crash_speed_idx`) - Identifying crashes involving specific vehicle types (using `crash_articulated_idx`)

2.4 Database Summary After ETL

After completing the ETL and load process, we verified the database structure and content using Neo4j's built-in information commands. The following screenshot summarises the resulting graph:

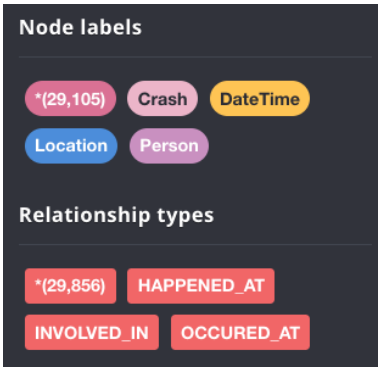

Figure 6: Overview of the graph database after ETL and load, showing the number of nodes, relationships, labels, and relationship types.

This confirms that the data was successfully imported and the graph structure matches our intended design, with all node types, relationships, and properties correctly established.

3. Query Implementation and Results

The full research questions can be found in the Task Sheet. The complete Cypher queries for all questions can be found in the second half of `/scripts/scripts.txt`.

3.1 Required Queries

We implemented all required queries (A-G) with optimisations for performance. Here are the results:

1. **Question A**: WA crashes with articulated trucks and multiple fatalities

```
// Question A: Crashes in WA with articulated trucks and multiple fatalities
MATCH (c:Crash {articulatedTruckInvolvement: 'Yes'})-[:OCCURED_AT]→(l:Location {state:
'WA'})
WHERE toInteger(c.numberFatalities) > 1
MATCH (p:Person)-[:INVOLVED_IN]→(c)
MATCH (c)-[:HAPPENED_AT]→(dt:DateTime)
WHERE toInteger(dt.year) ⩾ 2020 AND toInteger(dt.year) ⩽ 2024
RETURN p.roadUser, p.age, p.gender, l.lgaName, dt.month, dt.year, c.numberFatalities
ORDER BY dt.year, dt.month;
```

	p.roadUser	p.age	p.gender	l.lgaName	dt.month	dt.year	c.numberFatalities
1	"Driver"	58	"Female"	"Busselton"	11	2020	2
2	"Passenger"	51	"Female"	"Busselton"	11	2020	2
3	"Driver"	56	"Male"	"Dundas"	12	2020	2
4	"Driver"	58	"Male"	"Dundas"	12	2020	2

Started streaming 4 records after 2 ms and completed after 3 ms.

Figure 7: Results showing crashes in WA with articulated trucks and multiple fatalities

2. **Question B**: Motorcycle rider age ranges during holidays

```
// Question B: Age ranges for motorcycle riders in fatal crashes during holidays
MATCH (p:Person {roadUser: 'Motorcycle rider'})-[:INVOLVED_IN]→(c:Crash)
MATCH (c)-[:OCCURED_AT]→(l:Location {nationalRemoteAreas: 'Inner Regional Australia'})
MATCH (c)-[:HAPPENED_AT]→(dt:DateTime)
WHERE dt.christmasPeriod = 'Yes' OR dt.easterPeriod = 'Yes'
RETURN p.gender, MAX(toInteger(p.age)) as maxAge, MIN(toInteger(p.age)) as minAge
ORDER BY p.gender;
```

	p.gender	maxAge	minAge
1	"Male"	76	14

Started streaming 1 records after 1 ms and completed after 5 ms.

Figure 8: Age ranges for motorcycle riders in fatal crashes during holidays

3. **Question C**: Young drivers in fatal crashes on weekends vs. weekdays

```
// Question C: Young drivers in fatal crashes on weekends vs. weekdays
MATCH (p:Person {ageGroup: '17_to_25'})-[:INVOLVED_IN]→(c:Crash)
MATCH (c)-[:OCCURED_AT]→(l:Location)
MATCH (c)-[:HAPPENED_AT]→(dt:DateTime)
WHERE toInteger(dt.year) = 2024
WITH l.state AS state,
     SUM(CASE WHEN dt.weekdayOrWeekend = 'Weekend' THEN 1 ELSE 0 END) AS weekends,
     SUM(CASE WHEN dt.weekdayOrWeekend = 'Weekday' THEN 1 ELSE 0 END) AS weekdays,
     AVG(toInteger(p.age)) AS avgAge
RETURN state, weekends, weekdays, avgAge
ORDER BY state;
```

	state	weekends	weekdays	avgAge
2	"NSW"	37	36	20.67123287671233
3	"NT"	0	1	20.0
4	"QLD"	14	36	20.84
5	"SA"	4	8	20.41666666666667
6	"TAS"	3	3	20.833333333333336
7	"VIC"	17	24	20.951219512195127

Started streaming 7 records after 1 ms and completed after 4 ms.

Figure 9: Analysis of young drivers involved in fatal crashes by state and day type

4. **Question D**: Friday crashes in WA with multiple deaths and mixed gender victims

```
// Question D: Friday crashes in WA with multiple deaths and mixed gender victims
MATCH (c:Crash)-[:OCCURED_AT]→(l:Location {state: 'WA'})
MATCH (c)-[:HAPPENED_AT]→(dt:DateTime {dayOfWeek: 'Friday', weekdayOrWeekend:
'Weekend'})
WHERE toInteger(c.numberFatalities) > 1
WITH c, l
MATCH (maleVictim:Person {gender: 'Male'})-[:INVOLVED_IN]→(c)
MATCH (femaleVictim:Person {gender: 'Female'})-[:INVOLVED_IN]→(c)
RETURN DISTINCT l.sa4Name, l.nationalRemoteAreas, c.nationalRoadType;
```

	l.sa4Name	l.nationalRemoteAreas	c.nationalRoadType
1	"Perth - South East"	"Major Cities of Australia"	"Local Road"
2	"Western Australia - Outback (North)"	"Very Remote Australia"	"National or State Highway"
3	"Perth - North West"	"Inner Regional Australia"	"Arterial Road"

Started streaming 3 records after 7 ms and completed after 9 ms.

Figure 10: Results showing Friday crashes in WA with multiple fatalities and mixed gender victims

5. **Question E**: Top 5 SA4 regions with fatal crashes during peak hours

```
// Question E: Top 5 SA4 regions with fatal crashes during peak hours
MATCH (c:Crash)-[:OCCURED_AT]→(l:Location)
MATCH (c)-[:HAPPENED_AT]→(dt:DateTime)
WITH c, l, dt,
     CASE
       WHEN toInteger(split(dt.time, ':')[0]) ⩾ 7 AND
            toInteger(split(dt.time, ':')[0]) ⩽ 9
       THEN 'Morning Peak'
       WHEN toInteger(split(dt.time, ':')[0]) ⩾ 16 AND
            toInteger(split(dt.time, ':')[0]) ⩽ 18
       THEN 'Afternoon Peak'
       ELSE 'Off Peak'
```

	region	Morning Peak	Afternoon Peak
1	"South Australia - South East"	36	49
2	"Wide Bay"	31	47
3	"Melbourne - South East"	30	42
4	"Capital Region"	34	37
5	"Central West"	27	38

Started streaming 5 records in less than 1 ms and completed after 13 ms.

Figure 11: Analysis of fatal crashes during peak hours across SA4 regions

6. **Question F**: Paths of length 3 between LGAs

```
// Question F: Paths of length 4 between LGAs
MATCH path = (start:Location)-[*3]-(end:Location)
WHERE start.lgaName <> end.lgaName
RETURN start.lgaName AS startLGA, end.lgaName AS endLGA, path
ORDER BY startLGA, endLGA
LIMIT 3;
```

(no changes, no records)

Completed after 6 ms.

Figure 12: Path analysis showing connections between different LGAs

7. **Question G**: Weekday fatal crashes involving pedestrians

```
// Question G (CITS5504 ONLY): Weekday fatal crashes involving pedestrians
MATCH (p:Person {roadUser: 'Pedestrian'})-[:INVOLVED_IN]→(c:Crash)
WHERE (c.busInvolvement = 'Yes' OR c.heavyRigidTruckInvolvement = 'Yes')
  AND (toInteger(c.speedLimit) < 40 OR toInteger(c.speedLimit) ⩾ 100)
MATCH (c)-[:HAPPENED_AT]→(dt:DateTime {weekdayOrWeekend: 'Weekday'})
WITH dt.timeOfDay AS timeOfDay,
     p.ageGroup AS ageGroup,
     CASE
       WHEN c.busInvolvement = 'Yes' THEN 'Bus'
       ELSE 'Heavy Rigid Truck'
     END AS vehicleType,
     c.speedLimit AS speedLimit,
```

	timeOfDay	ageGroup	vehicleType	crashCount	speedLimit
1	"Day"	"0_to_16"	"Heavy Rigid Truck"	1	110
2	"Day"	"17_to_25"	"Heavy Rigid Truck"	1	20
3	"Day"	"26_to_39"	"Heavy Rigid Truck"	2	100
4	"Day"	"40_to_64"	"Heavy Rigid Truck"	1	100
5	"Day"	"40_to_64"	"Heavy Rigid Truck"	2	110
6	"Day"	"40_to_64"	"Bus"	1	10
7					

Started streaming 11 records after 8 ms and completed after 10 ms.

Figure 13: Analysis of weekday fatal crashes involving pedestrians with specific vehicle types and speed limits

Key findings from these queries include: - The distribution of crashes across different regions and time periods - Patterns in crash severity and vehicle involvement - Demographic trends in crash victims - Geographical

relationships between crash locations - Impact of time and road conditions on crash frequency

3.2 Additional Meaningful Queries

We implemented three additional meaningful queries to gain deeper insights into the crash data:

1. **Elderly Driver Analysis** Research Question: "What are the most dangerous times of day for elderly drivers (aged 65 and above) in urban versus rural areas, and how does this vary by location type?"

```
// Additional Meaningful Query 1: Most dangerous times of day for elderly drivers
MATCH (p:Person {roadUser: 'Driver'})-[:INVOLVED_IN]→(c:Crash)
WHERE p.ageGroup IN ['65_to_74', '75_plus']
MATCH (c)-[:OCCURED_AT]→(l:Location)
MATCH (c)-[:HAPPENED_AT]→(dt:DateTime)
WITH
  dt.timeOfDay AS timeOfDay,
  CASE
    WHEN l.nationalRemoteAreas IN ['Major Cities of Australia'] THEN 'Urban'
    ELSE 'Rural'
  END AS areaType,
  COUNT(c) AS crashCount
```

	timeOfDay	areaType	crashCount
1	"Day"	"Rural"	309
2	"Night"	"Rural"	81
3	"Day"	"Urban"	76
4	"Night"	"Urban"	34

Started streaming 4 records after 7 ms and completed after 12 ms.

Figure 14: Analysis of crash patterns for elderly drivers in urban vs rural areas

2. **Speed Limit Impact Analysis** Research Question: "How do different speed limits affect fatality rates across various types of road users, and what patterns emerge in the relationship between speed limits and crash severity?"

```
// Additional Meaningful Query 2: Speed limits and fatality rates
MATCH (p:Person)-[:INVOLVED_IN]→(c:Crash)
MATCH (c)-[:HAPPENED_AT]→(dt:DateTime)
WHERE toInteger(dt.year) ⩾ 2022 AND toInteger(dt.year) ⩽ 2024
WITH
  p.roadUser AS roadUserType,
  CASE
    WHEN toInteger(c.speedLimit) ⩽ 60 THEN 'Low Speed (≤60 km/h)'
    WHEN toInteger(c.speedLimit) ⩽ 90 THEN 'Medium Speed (61-90 km/h)'
    ELSE 'High Speed (>90 km/h)'
  END AS speedCategory,
  AVG(toInteger(c.numberFatalities)) AS avgFatalities,
```

Table | Text | Code

	roadUserType	speedCategory	avgFatalities	crashCount
1	"Driver"	"High Speed (>90 km/h)"	1.1838331160365059	767
2	"Driver"	"Low Speed (≤60 km/h)"	1.057823129251701	294
3	"Driver"	"Medium Speed (61-90 km/h)"	1.1405750798722052	313
4	"Motorcycle pillion passenger"	"High Speed (>90 km/h)"	1.5	2
5	"Motorcycle pillion passenger"	"Low Speed (≤60 km/h)"	1.1666666666666665	6
6	"Motorcycle pillion passenger"	"Medium Speed (61-90 km/h)"	2.0	4
7				

Started streaming 18 records after 9 ms and completed after 17 ms.

Figure 15: Analysis of fatality rates across different speed limits and road user types

3. **Age Group and Road Conditions Analysis** Research Question: "How do different age groups experience crashes across various road types and conditions, and what patterns emerge in the relationship between age, crash type, and time of day?"

```
// Additional Meaningful Query 3: Age group analysis by crash type and road conditions
MATCH (p:Person)-[:INVOLVED_IN]→(c:Crash)
MATCH (c)-[:OCCURED_AT]→(l:Location)
MATCH (c)-[:HAPPENED_AT]→(dt:DateTime)
WHERE toInteger(dt.year) ⩾ 2020
WITH
  p.ageGroup AS ageGroup,
  c.crashType AS crashType,
  l.nationalRoadType AS roadType,
  CASE
    WHEN dt.timeOfDay IN ['Morning', 'Afternoon'] THEN 'Day'
    WHEN dt.timeOfDay IN ['Evening', 'Night'] THEN 'Night'
```

Table | Text | Code

	ageGroup	crashType	roadType	timeOfDay	crashCount	averageFatalities
1	"0_to_16"	"Single"	*null*	"Other"	97	1.0
2	"0_to_16"	"Single"	*null*	"Night"	69	1.0
3	"0_to_16"	"Multiple"	*null*	"Night"	47	3.04
4	"0_to_16"	"Multiple"	*null*	"Other"	45	2.29
5	"17_to_25"	"Single"	*null*	"Night"	452	1.0
6	"17_to_25"	"Single"	*null*	"Other"	358	1.0
7						

Started streaming 24 records after 9 ms and completed after 24 ms.

Figure 16: Analysis of crash patterns across different age groups, road types, and times of day

4. Graph Data Science Application

4.1 Path Analysis with Breadth-First Search (BFS)

Graph Data Science (GDS) enables the discovery of complex patterns within graph-structured data that would be difficult to find using traditional relational approaches. In our project, we applied BFS to analyse connections between different Australian LGAs through crash incidents.

BFS Implementation in Our Graph Structure

BFS is particularly well-suited for finding the shortest path between nodes in an unweighted graph. In our Neo4j implementation, BFS powers the path-finding queries that connect LGAs:

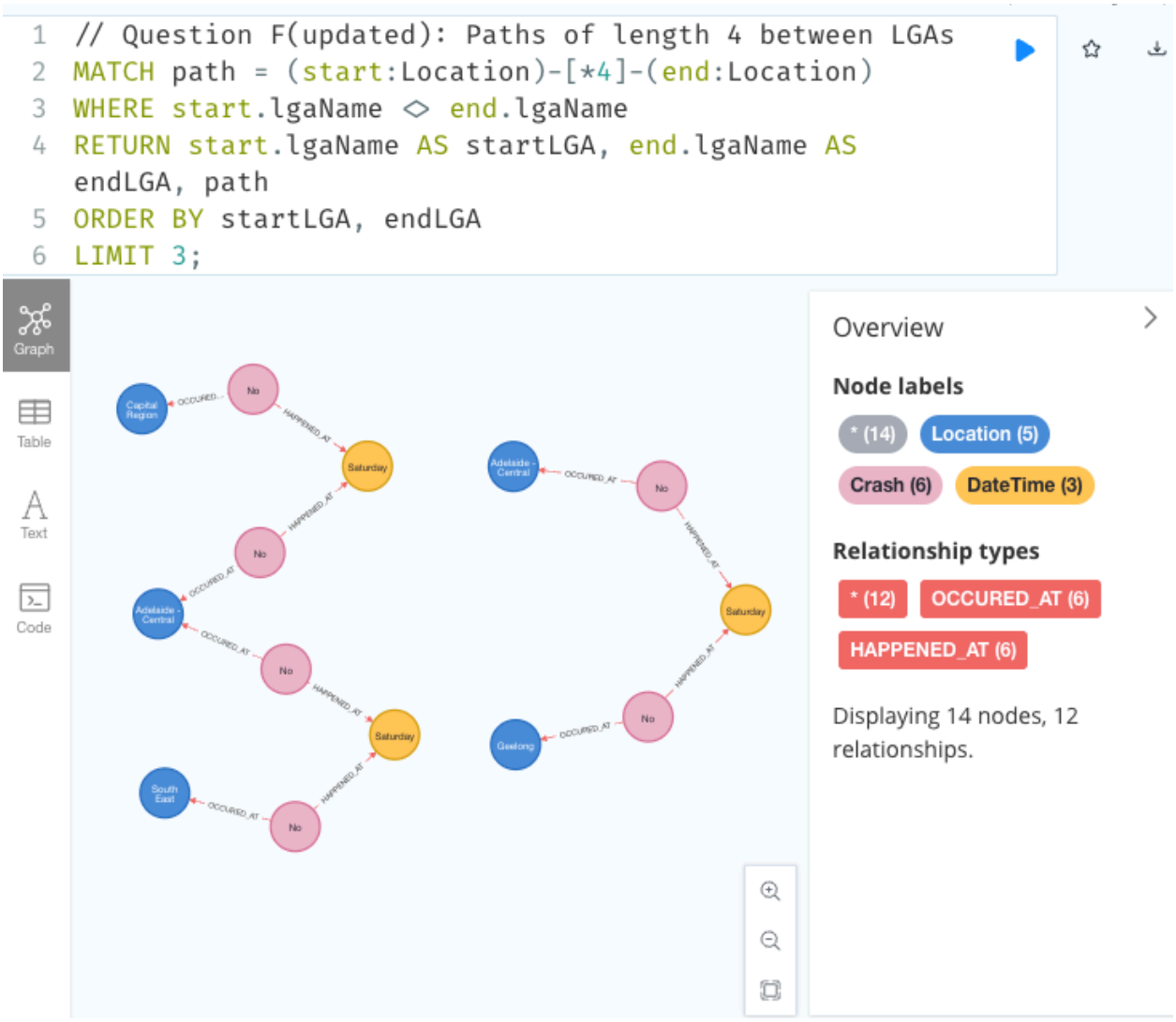

Figure 17: Path analysis showing Location → Crash → DateTime → Crash → Location traversal pattern. This demonstrates how crashes at the same time connect different locations, without involving Person nodes (which are unique to each crash).

This query systematically explores paths of exactly 4 relationships between Location nodes, revealing how different LGAs are connected through the crash network.

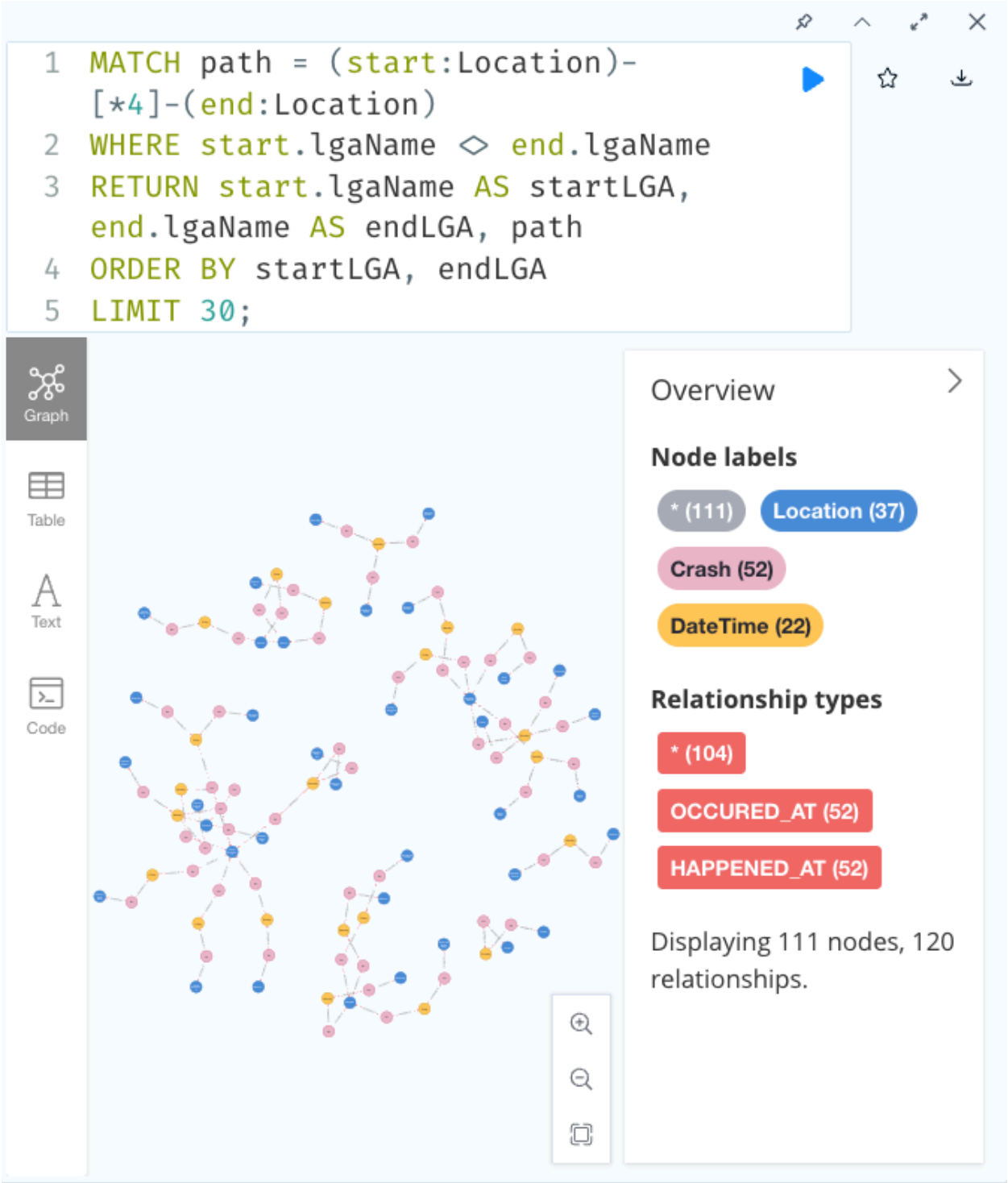

Figure 18: Extended path analysis revealing 30 connections between LGAs. This shows several disconnected clusters, with the largest spanning 35 nodes and the smallest containing just 5 nodes.

Our graph structure enforces a minimum path length of 4 between LGAs, explaining why Query F (finding paths of length 3) returned no results. The traversal follows the pattern:

```
LGA1 → Crash1 → DateTime → Crash2 → LGA2
```

This shows how crashes occurring at the same time connect different geographic areas, revealing temporal-

spatial crash patterns across location boundaries.

4.2 Real-World Applications of BFS in Road Safety

Applying BFS to our road crash graph model enables several practical insights for transport authorities:

- **Crash Cluster Identification:**
  BFS reveals geographic clusters of crashes that span multiple LGAs, identifying "crash corridors" that require coordinated intervention across jurisdictions.
- **Road Safety Network Analysis:**
  By exploring connected LGAs, BFS highlights thoroughfares with higher risk profiles that cross location boundaries.
- **Comparative Crash Pattern Analysis:**
  Path analysis between LGAs helps detect similar crash patterns across different regions, informing targeted safety measures where similar risk factors exist.
- **Transport Corridor Risk Assessment:**
  BFS traces patterns of connected crashes along transport corridors, showing how risk factors propagate across regional boundaries and supporting proactive resource allocation.

4.3 Other Relevant Graph Algorithms

While BFS was our primary algorithm, other approaches could provide additional insights:

- **Community Detection:** Would identify clusters of LGAs with similar crash patterns, revealing regional safety issues
- **Centrality Measures:** Could identify which locations act as "hubs" in the crash network
- **Path Finding with Weights:** If crash severity or frequency were added as edge weights, weighted algorithms could find paths connecting LGAs through the most severe incidents

Summary

BFS allows us to move beyond isolated incident analysis to a network-based understanding of road safety. The disconnected clusters in our results suggest distinct geographic regions with separate crash patterns, while the connected paths highlight how incidents relate across jurisdictions. This approach supports more informed decision-making for cross-boundary safety interventions and policy development.

5. Conclusion

The implementation demonstrates the effectiveness of graph databases in handling complex relationships in crash data. The design choices and optimisations have resulted in efficient query performance and meaningful insights into road safety patterns.