Solving Business Queries using Cypher and Graph Database Modelling on the Crashes and Fatalities in Australia

Created By: Johan Carlo Ilagan, Student No. 23832843
Submitted on: 23 May 2025

Design and Implementation Process

Graph data modelling is a method of organizing data that emphasizes the relationships between entities [1]. It uses nodes to represent entities, which can also contain properties to help categorise or filter the data. Relationships, on the other hand, connect these nodes together and are assigned clear, descriptive labels. Graph data modelling allows efficient querying by enabling operations that naturally traverse the relationships between data items. One of the most common graph data models is the property graph model, which follows the principle that the logical structure should easily reflect the data and its relationships [2].

The dataset used in this report contains information about crashes that occurred over the past ten years. It provides information about the setting of the crash, the people involved, outcome, and many more. By applying graph data modelling, the aim is to answer business-related queries by using the relationships and node structures within the dataset, executed through Cypher queries. The first step in this process is to design the property graph based on the dataset.

The initial stage involves defining the domain [3]. These include all relevant entities that need to be represented in the model. In this crash dataset, entities can be classified into five categories – crashes, people, location, datetime, and vehicles involved. The next step is to identify the key questions or business queries that the graph should help answer. In addition to the queries provided in the project brief, two further queries were developed:

1. For each age group, how many crashes occurred in WA on roads with a speed limit of 100 or higher during weekdays? Sort the age groups based on the total crash count from highest to lowest.
2. For each road type in NSW, how many fatal crashes involving multiple fatalities occurred in the post-COVID period (2023-2024)? Additionally, among these crashes, output how many occurred during the day and how many during the night.

After analyzing the queries, the relevant properties required to answer them must be identified from the dataset. Each node can have related properties that can be essential when Cypher querying is done. Once nodes and their properties are defined, the final step is to connect them using relationships, which enables query analysis across the dataset.

The diagram above is the property graph developed through the design process described in the previous paragraphs. Each node is connected with the *Crash* node, while *Location* nodes can also connect to other *Location* nodes – primarily to answer query (f).

This design has both strengths and limitations. On the one hand, it is straightforward and easy to interpret. Each node is distinct, and clearly represents a specific real-world entity, which justifies its separation from others. The relationships are also well-defined, as their names clearly explain the business logic they represent, making the model intuitive to query and reason about. On the other hand, its simplicity of the model can also be a disadvantage as the queries become more complex. All contextual nodes – *Person, Vehicle, Datetime, Location* – connect only to the *Crash* node. While this centralisation supports basic querying, it limits flexibility. A more interconnected structure where supporting nodes relate to each other as well, could help analyse more advanced queries in the future. However, this current model is sufficient enough to analyse the 8 business queries that need to be tackled in this report.

Now that there is a property graph, the next step is to get the data by the extract, transform, and load (ETL) process. Once that is done and the data is in Neo4j, Cypher queries can be made that can be used to get an output that solves the required queries.

Extract, Transform, and Load (ETL) Process

The original CSV file used for this project is titled *Project2_Dataset_Corrected.csv*. Initial data cleaning and the ETL (Extract, Transform, Load) process were done in Jupyter Notebook using the pandas library. The dataset was loaded using the command `pd.read_csv('Project2_Dataset_Corrected.csv')`. To ensure the reliability of the dataset, duplicates were checked to see if there are some irrelevancies in the data using `duplicates = df[df.duplicated()]`. As no duplicates were detected, the original dataset was considered reliable and was used to proceed.

The data types of each property were reviewed to ensure they matched the expected usage of their values. For vehicle type properties such as `bus_involvement`, `heavy_rigid_truck_involvement`, `articulated_truck_involvement`, and holiday type properties such as `christmas_period`, `easter_period`, the data types for these properties' values were changed from object to boolean, as the values are either 'Yes/No' or True/False.

Following this, the column headers were renamed for greater clarity and to facilitate easier referencing during the ETL and graph import process. This renaming was implemented using the `df.rename()` function in pandas.

```
df = df.rename(columns={'Crash ID': 'crash_id',
                'State': 'state',
                'Month':'month',
                'Year':'year',
                'Dayweek':'day_of_week',
                'Time':'time',
                'Crash Type':'crash_type',
                'Number Fatalities':'number_fatalities',
                'Bus Involvement':'bus_involved',
                'Heavy Rigid Truck Involvement':'heavy_rigid_truck_involved',
                'Articulated Truck Involvement':'articulated_truck_involved',
                'Speed Limit':'speed_limit',
                'National Remoteness Areas':'remoteness_area',
                'SA4 Name 2021':'sa4_name',
                'National LGA Name 2024':'lga_name',
                'National Road Type':'road_type',
                'Day of week':'weekday',
                'Time of day':'time_of_day',
                'Road User':'road_user',
                'Gender':'gender',
                'Age':'age',
                'Age Group':'age_grp',
                'Christmas Period':'christmas_period',
                'Easter Period':'easter_period'
 })
```

The Extract, Transform, Load (ETL) process involves extracting the data from the edited dataset CSV file, and organizing it based on the property graph designed using the Arrows app. Based on this model, data is grouped according to the relevant node types and their properties, with a separate CSV file for each node.

```
# Create location node/df
location_df = df[["state","remoteness_area","sa4_name", "lga_name",
"road_type"]].drop_duplicates()
loc_index = range(1,1+len(location_df))
location_df["location_id"] = ["L" + str(i) for i in loc_index]
location_df = location_df[["location_id","state","remoteness_area","sa4_name",
"lga_name", "road_type"]]
```

In this example, to generate the *Location* node data, properties associated with location – *state, sa4_name, lga_name, remoteness_area* – are extracted from the dataset. A unique *location_id* is assigned based on the number of distinct locations. The result is stored in a data frame called location_df. This same approach is repeated for the other core nodes – *Crash*, *Datetime*, *Person*, and *Vehicle.*

Once the different node-specific data frames have been created, the next step is to construct data frames for each relationship as well. To do this, the previously generated node IDs must be merged back into the main dataset using the `df.merge()` function. This allows each row to reference the correct node IDs needed to establish relationships. The data frames for each relationship can then be made using a format of the following code:

```
# Create :INVOLVED relationship
rel_involved = df[['crash_id','person_id']]
rel_involved
```

Since the *:INVOLVED* relationship connects Crash and Person nodes, both *crash_id* and *person_id* are needed to create the relationship in Neo4j. Similar processes are followed for relationships *:OCCURRED_AT, :OCCURRED_ON,* and *:INVOLVED_VEHICLE* as well. After data frames are generated for each node and relationship, each one of these will be exported into a CSV file that can be used to load data in Neo4j for the Cypher queries using the code below:

```
csv_name = ['crash.csv','datetime.csv','person.csv','location.csv',
'vehicle_type.csv','rel_involved.csv','rel_occurredAt.csv','rel_involvedVehicle.csv'
,'rel_occurredOn.csv']
dfs = [crash_df, datetime_df, person_df, location_df, vehicle_type_df, rel_involved,
rel_occurredAt, rel_involvedVehicle, rel_occurredOn]
for i in range(len(csv_name)):
```

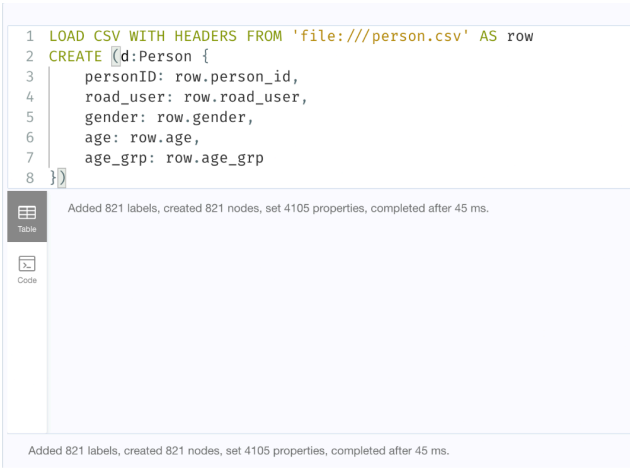
```
    dfs[i].to_csv(csv_name[i], index = False)
```

Graph Database Implementation

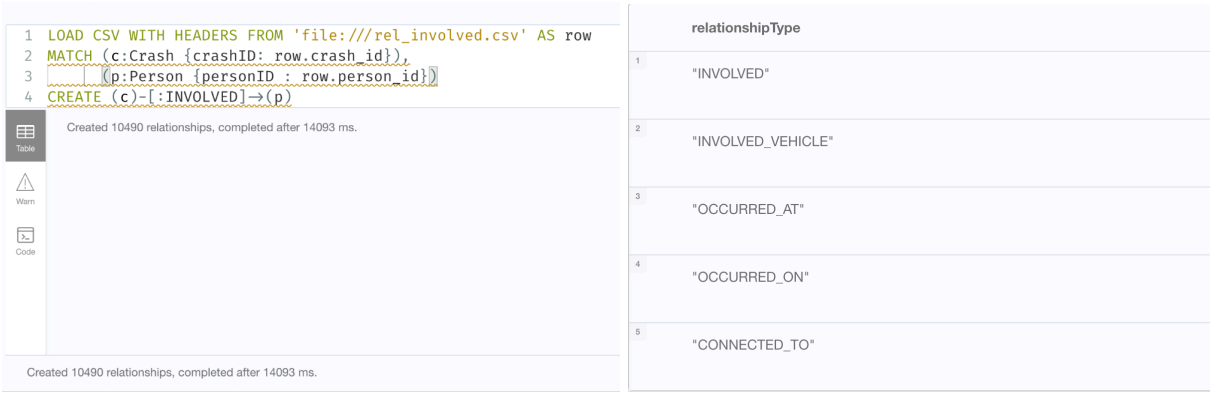
The graph database is implemented by loading the CSV files for each node and relationship, ensuring that all elements are properly connected within the graph structure. This process guarantees that each node is linked through its corresponding relationships. A brief format of the loading code used in Neo4j is inputted below:

```
// Loading a node CSV
LOAD CSV WITH HEADERS FROM 'file:///{file_node}.csv' AS row
CREATE (d:{node_name} {
    {property_name}: row.{property_name in the csv},
})

// Loading a relationship CSV – connecting Node1 and Node2
LOAD CSV WITH HEADERS FROM 'file:///{file_relationship}.csv' AS row
MATCH (c:{node1_name} {{property_name}: row.{property_name in the csv}}),
      (p:{node2_name} {{property_name}: row.{property_name in the csv}})
CREATE (c)-[:{RELATIONSHIP_NAME}]->(p)
```



The figures above demonstrate how a CSV file of a node can be loaded into Neo4j. The same structure of code is used for *Crash*, *Location*, *Vehicle*, and *Datetime*. Upon querying the graph, the expected properties from the CSV files are successfully returned, confirming the import process was carried out correctly.

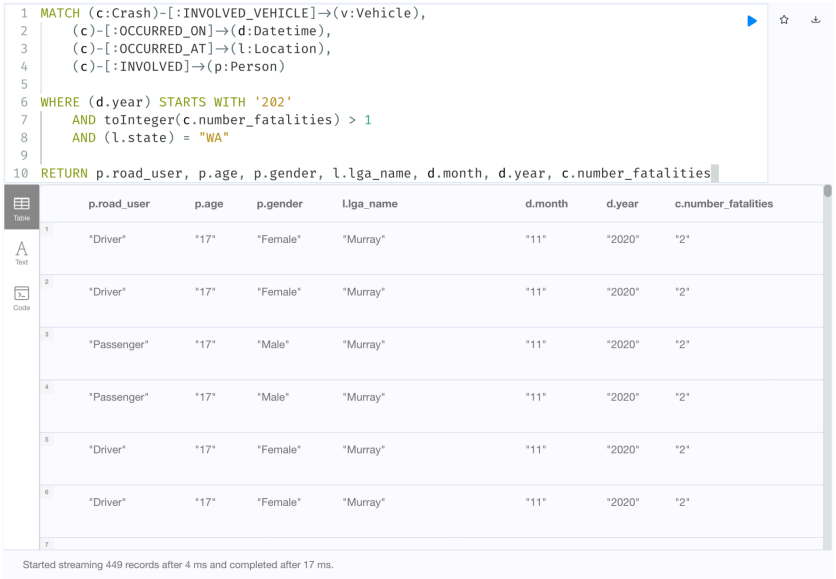


The figures above show an example of how relationship CSV files are imported into Neo4j. The four primary relationships – *:INVOLVED, :INVOLVED_VEHICLE, :OCCURRED_AT, :OCCURRED_ON* – are loaded using a similar Cypher format. The *:CONNECTED_TO* relationship was created in a different way which will be explained in the following section.

Answering Different Queries

With the complete set of nodes and relationships, along with their associated properties and values successfully loaded into the database from CSV files, the system is now ready to address the business queries using Cypher. Each query begins by matching the necessary nodes and relationships, centering around the Crash node, and connecting other nodes.

1. Find all crashes in WA from 2020-2024 where articulated trucks were involved and multiple fatalities (Number Fatalities>1) occurred. For each crash, provide the road user, age of each road user, gender of each road user, LGA Name, month and year of the crash, and the total number of fatalities.



The query identifies all crashes in WA from 2020 to 2024 that involved articulated trucks and resulted in multiple fatalities. The Cypher query applies these filters and retrieves the relevant information for each crash. The query successfully returns 449 records, each satisfying all specified conditions. While some rows may appear similar or identical even, this is expected. Multiple individuals may be involved in the same crash, resulting in duplicate information but different person-level attributes that are not returned.

2. Find the maximum and minimum age for female and male motorcycle riders who were involved in fatal crashes during the Christmas Period or Easter Period in inner regional Australia. Output the following information: gender, maximum age and minimum age.

```
MATCH (c:Crash)-[:INVOLVED_VEHICLE]→(v:Vehicle),
    (c)-[:OCCURRED_ON]→(d:Datetime),
    (c)-[:OCCURRED_AT]→(l:Location),
    (c)-[:INVOLVED]→(p:Person)

WHERE (d.christmas_period="True" OR d.easter_period="True")
    AND (l.remoteness_area='Inner Regional Australia')
    AND (p.road_user='Motorcycle rider')

RETURN p.gender,max(toInteger(p.age)), min(toInteger(p.age))
```

	p.gender	max(toInteger(p.age))	min(toInteger(p.age))
1	"Male"	73	14

The query identifies the maximum and minimum ages of male and female motorcycle riders who were involved in crashes during either the Christmas Period or Easter Period, specifically in Inner Regional Australia. It filters the data based on the required conditions and then groups the data by gender and returns the maximum and minimum age for each group.

The query returns results only for the male gender, indicating that there are no records matching all conditions where the gender is *'Female'*. This suggests that within the given conditions, no female motorcycle rides were involved in fatal crashes that occurred during these holiday periods in inner regional areas.

3. How many young **drivers** (**Age Group = '17_to_25'**) were involved in fatal crashes on weekends vs. weekdays in each state during **2024**? Output 4 columns: **State name**, **weekends**, **weekdays**, and the **average age for all young drivers (Age Group = '17_to_25') who were involved in fatal crashes in each State**.

```
MATCH (c:Crash)-[:INVOLVED_VEHICLE]->(v:Vehicle),
    (c)-[:OCCURRED_ON]->(d:Datetime),
    (c)-[:OCCURRED_AT]->(l:Location),
    (c)-[:INVOLVED]->(p:Person)
WHERE p.age_grp = '17_to_25'
```

```
    AND d.year = '2024'
    AND p.road_user = 'Driver'
WITH l.state AS state,
    d.weekday = 'Weekday' AS is_weekday,
    d.weekday = 'Weekend' AS is_weekend,
    toInteger(p.age) AS age
WITH state,
    SUM(CASE WHEN is_weekday THEN 1 ELSE 0 END) AS num_weekday,
    SUM(CASE WHEN is_weekend THEN 1 ELSE 0 END) AS num_weekend,
    ROUND(avg(age),2) AS avg_age
RETURN state, num_weekday, num_weekend, avg_age;
```

	state	num_weekday	num_weekend	avg_age
1	"NSW"	47	27	21.92
2	"QLD"	14	22	19.25
3	"SA"	30	1	24.29
4	"VIC"	41	7	20.73
5	"TAS"	9	0	23.56

The query determines how many young drivers in age groups 17 to 25 were involved in fatal crashes during weekdays and weekends across each state in 2024. The query first applies filters based on the required conditions. The query then uses the *WITH* clause to prepare values as for aggregation. To get the number of the weekdays and weekends, a *CASE WHEN* clause is applied inside a *SUM()* function. The use of CASE WHEN was guided by generative AI explanations to ensure correct application [4]. The query calculates the average age of the young drivers using the *avg()* function. The final output includes four columns – state name, number of weekday crashes, number of weekend crashes, and the average age of the drivers.

4. Identify all crashes in **WA** that occurred **Friday** (but categorised as a **weekend**) and resulted in **multiple deaths**, with victims being **both male and female**. For each crash, output the **SA4 name**, **national remoteness areas**, and **national road type**.

```
MATCH (c:Crash)-[:INVOLVED_VEHICLE]→(v:Vehicle),
    (c)-[:OCCURRED_ON]→(d:Datetime),
    (c)-[:OCCURRED_AT]→(l:Location),
    (c)-[:INVOLVED]→(p:Person)
WHERE l.state = 'WA'
    AND d.day_of_week = 'Friday'
    AND d.weekday ='Weekend'
    AND toInteger(c.number_fatalities) > 1
WITH c,d,l, COLLECT(DISTINCT p.gender) AS genders
WHERE 'Male' IN genders AND 'Female' IN genders
RETURN l.state, d.day_of_week,d.weekday, c.number_fatalities, genders
```

Table Text Code

	l.state	d.day_of_week	d.weekday	c.number_fatalities	genders
1	"WA"	"Friday"	"Weekend"	"2"	["Female", "Male"]
2	"WA"	"Friday"	"Weekend"	"2"	["Male", "Female"]

This query identifies all crashes in WA that occurred on a Friday (but were classified as a weekend) and involved multiple fatalities, with victims of both male and female genders. The query first filters the data using the required conditions. Next, the WITH clause is used to carry forward the required nodes and properties – specifically *Crash*, *Datetime*, and *Location* nodes. A second WHERE clause is then applied to ensure that both 'Male' and 'Female' appear in the list of involved genders. This confirms that the crash had victims of both genders. The final output includes the SA4 name, national remoteness area, and national road type for each qualifying crash. The query successfully returns two records, each representing a crash in WA on a Friday, involving multiple fatalities and included both male and female victims.

5. Find the top 5 SA4 regions where the highest number of fatal crashes occur during peak hours (**Time between 07:00-09:00 and 16:00-18:00**). For each SA4 region, output the **name** of the region and the separate number of crashes that occurred during morning peak hours and afternoon peak hours (Renamed Morning Peak and Afternoon Peak).

```
MATCH
(c:Crash)-[:INVOLVED_VEHICLE]->(v:Vehicle),
      (c)-[:OCCURRED_ON]->(d:Datetime),
      (c)-[:OCCURRED_AT]->(l:Location),
      (c)-[:INVOLVED]->(p:Person)
WITH l.sa4_name AS sa4, d.time AS time
WITH sa4,
   SUM(CASE WHEN time STARTS WITH "7"
      OR time STARTS WITH "8"
      OR time = "9:00"
      THEN 1 ELSE 0 END) AS `Morning Peak`,
   SUM(CASE WHEN time STARTS WITH "16"
      OR time STARTS WITH "17"
      OR time = "18:00"
      THEN 1 ELSE 0 END) AS `Afternoon Peak`
WITH sa4, `Morning Peak`, `Afternoon Peak`,
     (`Morning Peak` + `Afternoon Peak`) AS
total_crashes
RETURN sa4, `Morning Peak`, `Afternoon Peak`
ORDER BY total_crashes DESC
LIMIT 5;
```

sa4	Morning Peak	Afternoon Peak
"Darling Downs - Maranoa"	158	370
"Shepparton"	24	268
"Wide Bay"	141	122
"Barossa - Yorke - Mid North"	40	191
"Western Australia - Wheat Belt"	132	84

This query identifies the top five SA4 regions with the highest number of fatal crashes that occurred during peak hours, specifically between 7:00-9:00 and 16:00-18:00. To classify each crash by time, the query uses *CASE WHEN* to check whether the time value falls within the peak hours. Crashes are then conditionally counted as either 'Morning Peak' or 'Afternoon Peak'. The individual counts for each peak period are aggregated using SUM(), and their sum is labelled as *total_crashes*. This value is used to sort the results in descending order. The query returns the top five SA4 region names, along with the number of crashes during morning peak and afternoon peak hours.

6. Find paths with a length of 3 between any two LGAs. **Return the top 3 paths, including the starting LGA and ending LGA for each path**. Order results alphabetically by starting LGA and then ending LGA.

This query aims to identify paths of length 3 between any two LGAs and return the top three paths, sorted alphabetically by the LGAs. Initially, the property graph did not contain any direct paths of this length between LGAs. To enable such queries, a new relationship called *[:CONNECTED_TO]* was introduced, linking Location nodes that have meaningful connections based on crash data. Given the size of the dataset, data filtering was applied to improve query performance. Specifically, only crashes that occurred in 2024 and involved a bus or a truck were considered. The connections were established using the following code:

```
MATCH (c1:Crash)-[:INVOLVED_VEHICLE]->(v1:Vehicle),
    (c1)-[:OCCURRED_ON]->(d1:Datetime),
    (c1)-[:OCCURRED_AT]->(l1:Location),
    (c2:Crash)-[:INVOLVED_VEHICLE]->(v2:Vehicle),
    (c2)-[:OCCURRED_ON]->(d2:Datetime),
    (c2)-[:OCCURRED_AT]->(l2:Location)
WHERE c1.crash_type = c2.crash_type
    AND c1.crashID <> c2.crashID
    AND c1.speed_limit = c2.speed_limit
    AND (
      v1.bus_involved = 'True' OR
      v1.articulated_truck_involved = 'True' OR
      v1.heavy_rigid_truck_involved = 'True'
    )
    AND (
      v2.bus_involved = 'True' OR
      v2.articulated_truck_involved = 'True' OR
      v2.heavy_rigid_truck_involved = 'True'
    )
    AND d1.year = '2024'
    AND d2.year = '2024'
    AND id(l1) < id(l2)
WITH DISTINCT l1, l2
MERGE (l1)-[:CONNECTED_TO]->(l2);
```

```
MATCH path = (lga1:Location)-[:CONNECTED_TO*3]→(lga2:Location)
WHERE lga1.lga_name < lga2.lga_name
WITH DISTINCT path, lga1.lga_name AS startLGA, lga2.lga_name AS endLGA
ORDER BY startLGA, endLGA
LIMIT 3
RETURN path, startLGA, endLGA;
```

	path	startLGA	endLGA
1	{ "start": { "identity": 19259, "labels": ["Location"], "properties": { "lga_name": "Alpine", "locationID": "L153", "road_type": "Local Road", "sa4_name": "Hume", "state": "VIC", "remoteness_area": "Inner Regional Australia" }, "elementId": "4:dc07f6d7-e5bf-48f5-b8c6-5a63fc678651:19259" },	"Alpine"	"Buloke"

Once the :CONNECTED_TO relationships were created, paths of length 3 between LGAs became traversable. The final query successfully returns the top three paths, including the starting LGA, ending LGA, and the complete path ordered alphabetically by LGAs.

The next task involves creating and implementing new business queries using Cypher. The two queries created are as follows:

1. For each age group, how many crashes occurred in WA on roads with a speed limit of 100 or higher during weekdays? Sort the age groups based on the total crash count from highest to lowest.

```
MATCH (c:Crash)-[:OCCURRED_AT]→(l:Location),
    (c)-[:OCCURRED_ON]→(d:Datetime),
    (c)-[:INVOLVED]→(p:Person)
WHERE l.state = 'WA'
    AND (toInteger(c.speed_limit)⩾100)
    AND (d.weekday='Weekday')
RETURN p.age_grp, COUNT(DISTINCT c) AS crashCount
ORDER BY crashCount DESC;
```

	p.age_grp	crashCount
1	"40_to_64"	123
2	"26_to_39"	82
3	"17_to_25"	64
4	"65_to_74"	35
5	"75_or_older"	28
6	"0_to_16"	13

The query uses a combination of *WHERE, COUNT(DISTINCT …)*, and *ORDER BY* clauses to filter and aggregate the relevant data. The query is looking for crashes in WA that occurred on a weekday and at a road that has a speed limit of 100 or above. It then groups the data by age

group and counts the number of distinct crashes that satisfy these conditions. The output successfully returns a ranked list of age groups, showing which was most frequently involved in high-speed weekday crashes in WA.

2. For each road type in NSW, how many fatal crashes involving multiple fatalities occurred in the post-COVID period (2023-2024)? Additionally, among these crashes, output how many occurred during the day and how many during the night.

```
MATCH
(c:Crash)-[:OCCURRED_AT]->(l:Location),
    (c)-[:OCCURRED_ON]->(d:Datetime),
    (c)-[:INVOLVED]->(p:Person)
WHERE (d.year IN ['2023','2024'])
    AND (toInteger(c.number_fatalities)>1)
    AND (l.state = 'NSW')
WITH l.road_type AS roadType,
    d.time_of_day AS timeOfDay
WITH roadType,
    SUM(CASE WHEN timeOfDay = 'Day' THEN 1
ELSE 0 END) AS daytimeCrashes,
    SUM(CASE WHEN timeOfDay = 'Night' THEN 1
ELSE 0 END) AS nighttimeCrashes,
    COUNT(*) AS totalCrashes
RETURN roadType, totalCrashes,
daytimeCrashes, nighttimeCrashes
```

roadType	totalCrashes	daytimeCrashes	nighttimeCrashes
"National or State Highway"	1324	185	1139
"Local Road"	176	16	160
"Arterial Road"	123	51	72
"Sub-arterial Road"	67	8	59
"Collector Road"	16	16	0

The query aims to identify, for each road type in NSW, the number of fatal crashes involving multiple fatalities that occurred during the post-COVID period (2023-2024). Additionally, it distinguishes how many of these crashes occurred during the day, and how many during the night.

The query uses *CASE WHEN* to classify crashes based on the time of day property, separating them into daytime and nighttime crashes. These are aggregated using SUM() to count the number of crashes in each category. The output returns for each road type, the total number of fatal crashes that involve multiple fatalities, the number of crashes that occurred during the day and the number of crashes that occurred during the night.

Graph Data Science

Graph Data Science (GDS) involves using graph theory and algorithms to analyse connected data [4]. It emphasizes the importance of relationships just as much as it highlights the importance of entities themselves. By leveraging nodes, properties, and relationships, GDS helps uncover hidden patterns and structures that may not be obvious through traditional data analysis methods.

In the context of the crashes dataset that is used in this report, GDS can be used to help identify locations that play an important role in the network of the most fatal crashes. Even if these locations might not have the highest number of crashes, they might act as key links between other high-risk areas. Through this, upgrades can be made to ensure and improve road safety.

One useful algorithm for this type of analysis is Breadth-First Search (BFS). The BFS is a graph data science algorithm that essentially involves beginning with a specific node, then traversing to all adjacent nodes [5], and repeating it until either a conclusion has been made or all nodes are examined. It can be used to find the shortest path, detect cycles, and find hidden patterns and networks that can be helpful in data analysis. Applied to the crashes dataset, BFS can help explore each node one at a time, and can help us identify which locations are a few steps away from a crash-prone area. It provides a whole new insight that might not be visible with just one look at the dataset, and can help provide the bigger picture while giving the opportunity to remove hazards and increase warnings so that the public can experience less fatal crashes.

References

[1] “What Are Graph Models?,” Hypermode, 2022. [Online]. Available: https://hypermode.com/blog/graph-models

[2] J. Webber, “RDF vs. Property Graphs: Choosing the Right Approach for Implementing a Knowledge Graph,” *Neo4j Blog*, June 4, 2024. [Online]. Available: https://neo4j.com/blog/knowledge-graph/rdf-vs-property-graphs-knowledge-graphs/

[3] “Neo4j Documentation – Data Modeling Tutorial”, Neo4j, 2024. [Online]. Available: https://neo4j.com/docs/getting-started/data-modeling/tutorial-data-modeling/

[4] “ChatGPT,” OpenAI, 2025. [Online]. Available: https://chat.openai.com/

[5] “Breadth-First Search or BFS for a Graph,” *GeeksforGeeks*, April 21, 2025. [Online]. Available: https://www.geeksforgeeks.org/breadth-first-search-or-bfs-for-a-graph/#what-is-breadth-first-search