Project 2: CITS5504 – Data Warehousing

Flavian Jerotich

Student ID: 24001784

Graph Database Design and Cypher Query

---

1. Graph Database Design

Design structure

To support flexible and multidimensional analysis of fatal crash data, I designed a **property graph model** that reflects the real-world relationships embedded in the dataset. At the core of the design is the **Crash** node, which acts as the central fact-like entity. It is connected to various **dimension-like nodes** that provide spatial, temporal, and contextual information.

The schema was built using the **Arrows App**, which allowed me to visually map out entities, attributes, and relationships. My goal was to create a design that would support **complex Cypher queries** involving crash conditions, demographics, geography, and infrastructure while avoiding data duplication and maintaining readability.

Normalization and Scalability

While graph databases don't follow formal normalization rules like relational databases, I still applied **normalization principles** in my graph model to reduce redundancy and make the structure more efficient. Instead of storing repeated information such as the state name, remoteness category, or road type as simple properties inside each Crash node, I modeled these as **separate, connected nodes** (State, Remoteness, RoadType, etc.).

This approach improves the graph overall in several important ways [1]:

- **Reduces redundancy**: Common values like "WA", "Remote Australia", or "National or State Highway" only appear once as nodes, rather than being repeated across thousands of crash records.
- **Improves query performance**: Queries involving filtering or grouping by these attributes (e.g., crashes in a specific SA4 region or remoteness area) can traverse indexed relationships rather than scanning text properties across all crash nodes.

- **Maintains consistency**: Using shared nodes avoids issues like inconsistent naming (e.g., “W.A.” vs. “WA”) and helps preserve data quality.
- **Supports scalability**: If new crashes are added, they can link to the existing State, LGA, SA4, and Remoteness nodes without duplicating values making the graph easier to maintain and expand in the future.

Nodes and Their Properties

I designed the property graph schema using Arrows App to represent the relationships between fatal crash events and their associated entities. The schema places the **Crash** node at the center, connected to six other nodes: **RoadUser, State, LGA, SA4, Remoteness, and RoadType**.

Each node contains attributes based on the data dictionary provided:

- The **Crash node** includes crash-specific details such as ID, Crash ID, Month, Year, Dayweek, Time, Crash Type, Number Fatalities, Bus Involvement, Heavy Rigid Truck Involvement, Articulated Truck Involvement, Speed Limit, Christmas Period, Easter Period, Day of week, and Time of day.
- The **RoadUser node** represents individuals involved in crashes, including their Road User type, Gender, Age, and Age Group. The RoadUser node was modeled separately because:
    - Crashes can involve **multiple road users**, each with their own demographics.
    - Capturing this as a separate node allows me to store detailed attributes like Age, Gender, Age Group, and Road User Type per individual.
    - It also supports **many-to-one** or **many-to-many** relationships: one crash can involve several people, and the same person type (e.g. "Pedestrian") may appear across different crashes.
- The **State, LGA, SA4, Remoteness, and RoadType nodes** represent the spatial and road attributes associated with the crash.
    - ✓ State- State
    - ✓ LGA- National_LGA_Name
    - ✓ SA4 - SA4_Name
    - ✓ Remoteness - Remoteness
    - ✓ RoadType - Road_Type

I chose to model State, LGA, SA4, Remoteness, and RoadType as **distinct nodes** rather than storing them as string properties inside the Crash node for several reasons:

- **Avoiding redundancy**: Each of these attributes appears repeatedly across thousands of crashes. By creating a single shared node for each unique value (e.g., the same "WA" or "Arterial Road"), I reduced storage and improved consistency.
- **Supporting reusability and scalability**: As more crashes are added, they can simply link to existing location or road type nodes, instead of duplicating values. This makes the graph more maintainable.
- **Improved query flexibility**: With dimension-like nodes, I can easily traverse and group data — e.g., "find all crashes in remote areas on arterial roads" without scanning strings or filtering raw properties.
- **Semantic clarity**: Modeling State or RoadType as entities emphasizes their importance as real-world classifiers and enables future graph extensions, such as linking these nodes to external datasets or metadata.

Relationships

The Crash node serves as the **central fact-like node** because it contains the event-level data: what happened, when, and how many fatalities occurred. All relationships stem from this central event, making it the best anchor for the graph. To link the central Crash node to other entities, the following descriptive relationships were used:

INVOLVED, OCCURRED_IN_STATE, OCCURRED_IN, LOCATED_IN_SA4, IN_REMOTENESS_AREA, and HAPPENED_ON.

This schema design normalizes repeated information (like location names) to avoid redundancy and allows querying patterns across different dimensions.

Relationship Type	**From → To**	**Meaning**
INVOLVED	Crash → RoadUser	A person involved in a crash. Links a crash to each individual affected, enabling demographic and behavioral analysis.
OCCURRED_IN_STATE	Crash → State	The state where the crash occurred. Classifies each crash by state jurisdiction (e.g., WA, NSW), supporting state-level analysis and aggregation.
OCCURRED_IN	Crash → LGA	The local government area (LGA). Anchors the crash to a Local Government Area, enabling local policy or council-level insights.
LOCATED_IN_SA4	Crash → SA4	Statistical Area Level 4. Connects crashes to LGAs, enabling analysis by local regions.
IN_REMOTENESS_AREA	Crash → Remoteness	ABS remoteness classification. Reflects accessibility and geography (e.g., remote vs urban), essential for rural safety analysis.
HAPPENED_ON	Crash → RoadType	The road type where the crash occurred. Associates a crash with the type of road, important for infrastructure-related queries (e.g., highways vs local roads).

These relationships are directional from Crash to each contextual node. This makes querying easier and keeps the model semantically clear.

Why This Design Works

This graph design effectively supports **complex, multi-dimensional analysis** of crash data [1]. By separating key attributes into their own nodes and connecting them through meaningful relationships, the structure allows for intuitive and powerful queries. For example, I can easily answer questions like:

"Which crashes involved young drivers, occurred at night, took place in remote areas, and happened on national highways?"

Such queries would be more difficult or less efficient in a flat or denormalized structure.

Additionally, by modeling attributes like State, Remoteness, and RoadType as **dedicated nodes** rather than repeated string properties, the design [1]:

- **Reduces redundancy**, saving storage and improving consistency.
- **Enables reuse** of shared dimension nodes across thousands of crash records.
- **Improves query readability and performance**, especially for filtering, aggregation, and pattern matching.

This normalized property graph approach aligns with best practices in graph data modeling [1], [5], and has the added benefit of being semantically clear and extensible. Overall, this approach enhances **flexibility**, **scalability**, and **data integrity**, making the graph well-suited for both descriptive insights and future extensions such as linking to external datasets or applying Graph Data Science techniques.

Design Trade-offs

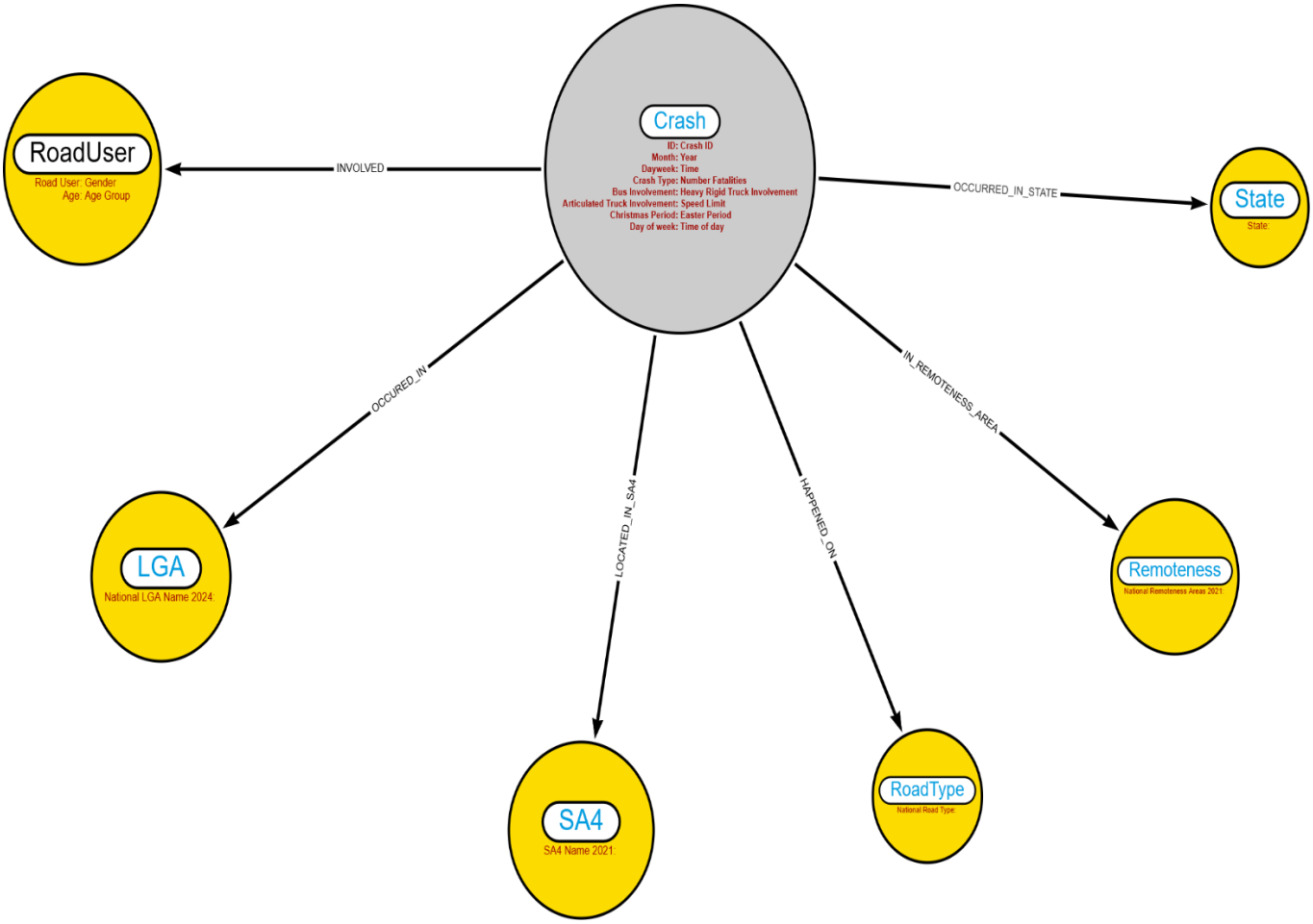
While the normalized graph structure offers many benefits in terms of scalability and query flexibility, it also introduces some trade-offs:

- **Increased query complexity**: Because key attributes like State, Road Type, and Remoteness are modeled as separate nodes, even simple queries often require multiple MATCH clauses and joins across nodes [1].
- **More relationships to manage**: With each crash connecting to several dimension nodes, the total number of relationships in the graph grows significantly. This can impact visual clarity when viewing parts of the graph in Neo4j Browser and may require query optimization as the dataset scales further.
- **ETL overhead**: Transforming the flat CSV file into multiple node and relationship files for import required additional preprocessing in Jupyter Notebook. Ensuring consistency in the mapping and avoiding duplication added extra steps to the ETL process.

Despite these trade-offs, this model was chosen because it best supports **analytical flexibility**, **modular data expansion**, and **high-level pattern discovery**, which are key goals in using a graph database [1], [5].

Schema Visualization

Below is the image created by using the Arrows app.



This model captures the essential entities and relationships in a way that supports both descriptive and analytical querying. To support reproducibility, I used clear labels for all node types and relationship directions. This schema was used as the foundation for creating the node and relationship CSVs in the ETL process and directly reflects the structure implemented in Neo4j

2. ETL Process

To prepare the data for use in Neo4j, I performed a custom Extract, Transform, Load (ETL) process using Jupyter Notebook. The goal was to convert the original flat CSV file **(Project2_Dataset_Corrected.csv),** which was obtained from the Australian Road Deaths Database maintained by the Bureau of Infrastructure, Transport and Regional Economics [3]. into multiple normalized files for nodes and relationships, based on the property graph schema I designed in the Arrows App.

Data Understanding and Transformation

The dataset provided in Project2_Dataset_Corrected.csv contained 10,490 records with 25 fields. After understanding the data dictionary, I used Python in Jupyter Notebook to clean and reshape the dataset into multiple **normalized CSV files**—one per node type and one for each relationship type.

Steps Taken

1. Extract – Loading the Original Dataset

- Loaded the dataset into a DataFrame using pandas.

```
import pandas as pd

# Update the file path if necessary
df = pd.read_csv('Project2_Dataset_Corrected.csv')

# Display the first 5 rows
df.head()
```

	ID	Crash ID	State	Month	Year	Dayweek	Time	Crash Type	Number Fatalities	Bus Involvement	...	Age	National Remoteness Areas	SA4 Name 2021	National LGA Name 2024	National Road Type	Christmas Period	Easter Period	Age Group	Day of week	Time of day
0	1	20241115	NSW	12	2024	Friday	4:00	Single	1	No	...	74	Inner Regional Australia	Riverina	Wagga Wagga	Arterial Road	Yes	No	65_to_74	Weekday	Night
1	2	20241125	NSW	12	2024	Friday	6:15	Single	1	No	...	19	Inner Regional Australia	Sydney - Baulkham Hills and Hawkesbury	Hawkesbury	Local Road	No	No	17_to_25	Weekday	Day
2	3	20246013	TAS	12	2024	Friday	9:43	Single	1	No	...	33	Inner Regional Australia	Launceston and North East	Northern Midlands	Local Road	Yes	No	26_to_39	Weekday	Day

- Verified column consistency, checked for missing or unknown values

After loading, I inspected the dataset shape (10,490 rows × 25 columns) and confirmed there were **no missing values**. I also used .nunique() to check which columns would make good unique nodes (e.g., State, SA4, LGA, Remoteness).

Checking data summary

```
# See column names
print(df.columns)

# See number of rows and columns
print(df.shape)

# Get quick stats (especially useful for numerical columns)
print(df.describe())

# Check for any missing/null values
print(df.isnull().sum())
```

```
Index(['ID', 'Crash ID', 'State', 'Month', 'Year', 'Dayweek', 'Time',
       'Crash Type', 'Number Fatalities', 'Bus Involvement',
       'Heavy Rigid Truck Involvement', 'Articulated Truck Involvement',
       'Speed Limit', 'Road User', 'Gender', 'Age',
       'National Remoteness Areas', 'SA4 Name 2021', 'National LGA Name 2024',
       'National Road Type', 'Christmas Period', 'Easter Period', 'Age Group',
       'Day of week', 'Time of day'],
      dtype='object')
(18498, 25)
                ID      Crash ID         Month          Year  \
count  18498.000000  1.849800e+04  18498.000000  18498.000000
mean    5245.500000  2.026606e+07      6.549388   2019.397521
std     3828.346496  3.543390e+06      3.469614      2.889095
min        1.000000  2.014400e+07      1.000000   2014.000000
25%     2623.250000  2.017383e+07      3.000000   2017.000000
50%     5245.500000  2.019587e+07      7.000000   2019.000000
75%     7867.750000  2.022284e+07     10.000000   2022.000000
max    18490.000000  2.018500e+08     12.000000   2024.000000

       Number Fatalities   Speed Limit           Age
count       18498.000000  18498.000000  18498.000000
mean            1.191992     82.358810     44.953956
std             0.582528     21.878222     22.124257
min             1.000000      5.000000      0.000000
25%             1.000000     60.000000     26.000000
50%             1.000000     80.000000     42.000000
75%             1.000000    100.000000     62.000000
max            10.000000    130.000000    101.000000
ID                             0
Crash ID                       0
State                          0
Month                          0
Year                           0
Dayweek                        0
Time                           0
Crash Type                     0
Number Fatalities              0
Bus Involvement                0
Heavy Rigid Truck Involvement  0
Articulated Truck Involvement  0
Speed Limit                    0
Road User                      0
Gender                         0
Age                            0
National Remoteness Areas      0
SA4 Name 2021                  0
National LGA Name 2024         0
National Road Type             0
Christmas Period               0
Easter Period                  0
Age Group                      0
Day of week                    0
Time of day                    0
dtype: int64
```

- Extracted unique records for each node type

```
# Check unique values in 'State' column
print(df['State'].unique())

# Check unique values in 'Crash Type'
print(df['Crash Type'].unique())

# Same for any column you're curious about
```

```
['NSW' 'TAS' 'QLD' 'SA' 'VIC' 'ACT' 'NT' 'WA']
['Single' 'Multiple']
```

2. Transform – Creating CSVs for Nodes and Relationships

I separated the dataset into different dataframes representing each **node type**:

- **Crash**: All core attributes (Crash_ID, Time, Date, vehicle involvement flags, etc.)
- **RoadUser**: Gender, Age, Age Group, Road User Type
- **State**
- **LGA**
- **SA4**
- **Remoteness**
- **RoadType**

Each dataframe was **deduplicated** using .drop_duplicates() and saved as its own CSV file.

CREATE NODE CSVS

```
#crash nodes
df_crash_nodes = df[['ID', 'Crash ID', 'Month', 'Year', 'Dayweek', 'Time', 'Crash Type',
                     'Number Fatalities', 'Bus Involvement', 'Heavy Rigid Truck Involvement',
                     'Articulated Truck Involvement', 'Speed Limit', 'Christmas Period',
                     'Easter Period', 'Day of week', 'Time of day']].drop_duplicates()

df_crash_nodes.to_csv('Crash_Nodes.csv', index=False)

#Creating road user nodes
df_roaduser_nodes = df[['Road User', 'Gender', 'Age', 'Age Group']].drop_duplicates().reset_index(drop=True)
df_roaduser_nodes['RoadUser_ID'] = df_roaduser_nodes.index + 1  # unique IDs starting at 1

df_roaduser_nodes.to_csv('RoadUser_Nodes.csv', index=False)

#state nodes
df_state_nodes = df[['State']].drop_duplicates().reset_index(drop=True)
df_state_nodes['State_ID'] = df_state_nodes.index + 1

df_state_nodes.to_csv('State_Nodes.csv', index=False)

#LGA nodes
df_lga_nodes = df[['National LGA Name 2024']].drop_duplicates().reset_index(drop=True)
df_lga_nodes['LGA_ID'] = df_lga_nodes.index + 1

df_lga_nodes.to_csv('LGA_Nodes.csv', index=False)

#SA4 nodes
df_sa4_nodes = df[['SA4 Name 2021']].drop_duplicates().reset_index(drop=True)
df_sa4_nodes['SA4_ID'] = df_sa4_nodes.index + 1

df_sa4_nodes.to_csv('SA4_Nodes.csv', index=False)

#Remoteness noeds
df_remoteness_nodes = df[['National Remoteness Areas']].drop_duplicates().reset_index(drop=True)
df_remoteness_nodes['Remoteness_ID'] = df_remoteness_nodes.index + 1

df_remoteness_nodes.to_csv('Remoteness_Nodes.csv', index=False)

#road type nodes
df_roadtype_nodes = df[['National Road Type']].drop_duplicates().reset_index(drop=True)
df_roadtype_nodes['RoadType_ID'] = df_roadtype_nodes.index + 1

df_roadtype_nodes.to_csv('RoadType_Nodes.csv', index=False)
```

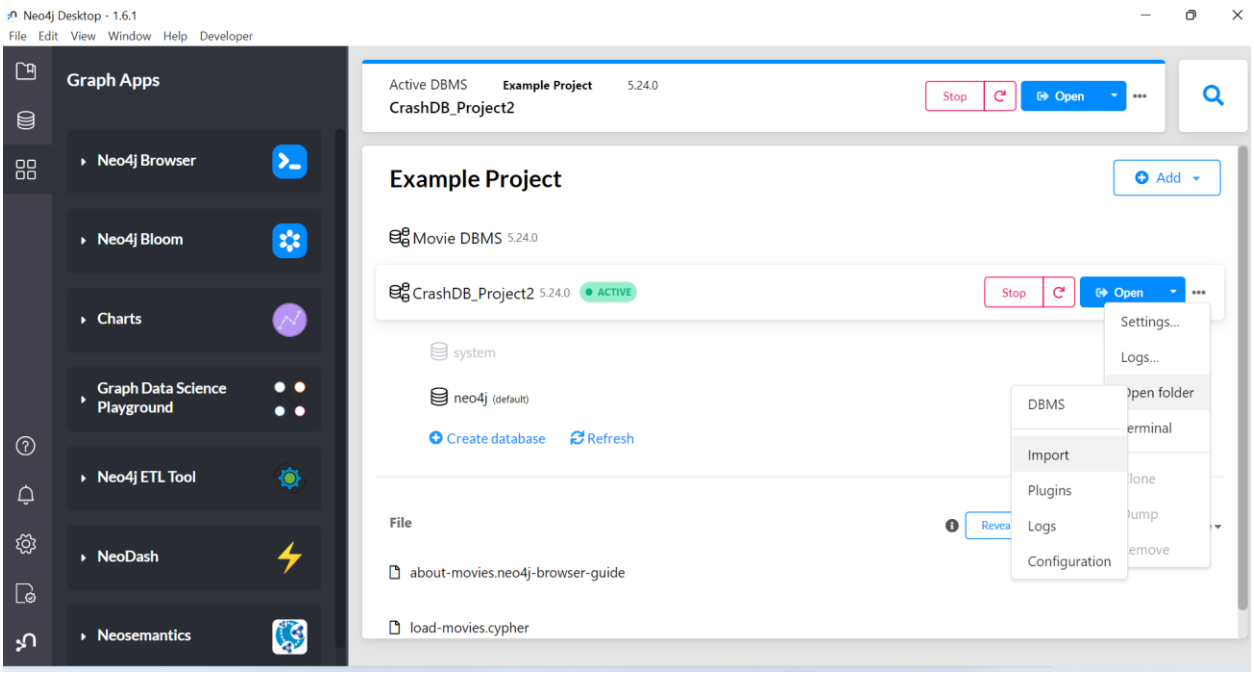
I also created **relationship files** that connected the Crash node to its dimensions using shared identifiers. For example, Crash_RoadUser.csv contained Crash_ID and RoadUser_ID, and similar structure was used for LGA, State, SA4, Remoteness, and RoadType.

```
# Create Crash → RoadUser Relationship
# ============================
df_crash_roaduser = df[['ID', 'Road User', 'Gender', 'Age', 'Age Group']]
df_crash_roaduser = df_crash_roaduser.merge(df_roaduser_nodes, on=['Road User', 'Gender', 'Age', 'Age Group'])

df_crash_roaduser_rel = df_crash_roaduser[['ID', 'RoadUser_ID']]
df_crash_roaduser_rel.to_csv('Crash_RoadUser_Relationship.csv', index=False)
print("Crash_RoadUser_Relationship.csv created.")

# ============================
# Create Crash → State Relationship
# ============================
df_crash_state = df[['ID', 'State']].merge(df_state_nodes, on='State')
df_crash_state_rel = df_crash_state[['ID', 'State_ID']]
df_crash_state_rel.to_csv('Crash_State_Relationship.csv', index=False)
print("Crash_State_Relationship.csv created.")

# ============================
# Create Crash → LGA Relationship
# ============================
df_crash_lga = df[['ID', 'National LGA Name 2024']].merge(df_lga_nodes, on='National LGA Name 2024')
df_crash_lga_rel = df_crash_lga[['ID', 'LGA_ID']]
df_crash_lga_rel.to_csv('Crash_LGA_Relationship.csv', index=False)
print("Crash_LGA_Relationship.csv created.")

# ============================
# Create Crash → SA4 Relationship
# ============================
df_crash_sa4 = df[['ID', 'SA4 Name 2021']].merge(df_sa4_nodes, on='SA4 Name 2021')
df_crash_sa4_rel = df_crash_sa4[['ID', 'SA4_ID']]
df_crash_sa4_rel.to_csv('Crash_SA4_Relationship.csv', index=False)
print("Crash_SA4_Relationship.csv created.")

# ============================
# Create Crash → Remoteness Relationship
# ============================
df_crash_remoteness = df[['ID', 'National Remoteness Areas']].merge(df_remoteness_nodes, on='National Remoteness Areas')
df_crash_remoteness_rel = df_crash_remoteness[['ID', 'Remoteness_ID']]
df_crash_remoteness_rel.to_csv('Crash_Remoteness_Relationship.csv', index=False)
print("Crash_Remoteness_Relationship.csv created.")

# ============================
# Create Crash → RoadType Relationship
# ============================
df_crash_roadtype = df[['ID', 'National Road Type']].merge(df_roadtype_nodes, on='National Road Type')
df_crash_roadtype_rel = df_crash_roadtype[['ID', 'RoadType_ID']]
df_crash_roadtype_rel.to_csv('Crash_RoadType_Relationship.csv', index=False)
print("Crash_RoadType_Relationship.csv created.")
```

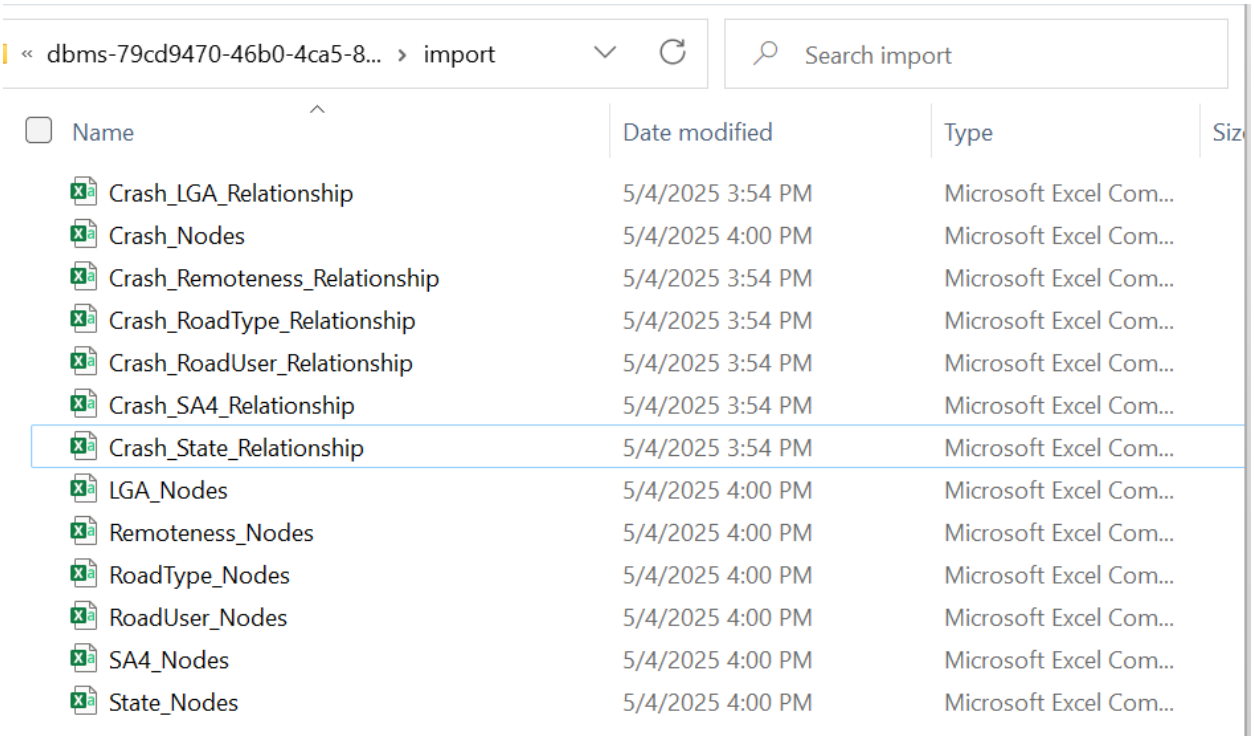
```
Crash_RoadUser_Relationship.csv created.
Crash_State_Relationship.csv created.
Crash_LGA_Relationship.csv created.
Crash_SA4_Relationship.csv created.
Crash_Remoteness_Relationship.csv created.
Crash_RoadType_Relationship.csv created.
```

3. Data Import into Neo4j

In Neo4j, I created my project called CrashDB_Project2.



I copy pasted the csv files created using Jupyter notebook into the folder named ‘import’ in my project files.

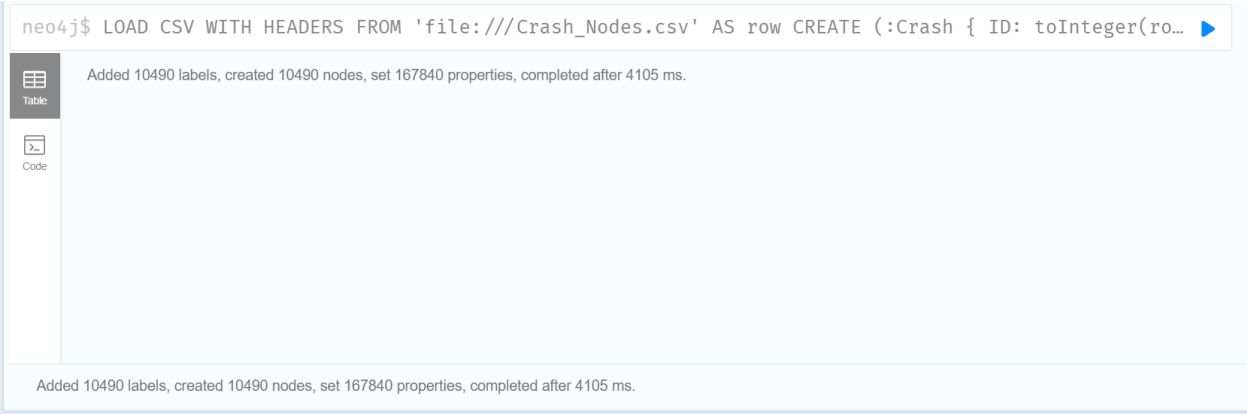


Constraints and Data Loading

Before importing data, I created **unique constraints** for each node using the modern REQUIRE syntax to ensure data integrity.

```
CREATE CONSTRAINT crash_id_unique IF NOT EXISTS
  FOR (c:Crash)
  REQUIRE c.ID IS UNIQUE;

CREATE CONSTRAINT roaduser_id_unique IF NOT EXISTS
  FOR (r:RoadUser)
  REQUIRE r.RoadUser_ID IS UNIQUE;

CREATE CONSTRAINT state_id_unique IF NOT EXISTS
  FOR (s:State)
  REQUIRE s.State_ID IS UNIQUE;

CREATE CONSTRAINT lga_id_unique IF NOT EXISTS
  FOR (l:LGA)
  REQUIRE l.LGA_ID IS UNIQUE;

CREATE CONSTRAINT sa4_id_unique IF NOT EXISTS
  FOR (s:SA4)
  REQUIRE s.SA4_ID IS UNIQUE;

CREATE CONSTRAINT remoteness_id_unique IF NOT EXISTS
  FOR (r:Remoteness)
  REQUIRE r.Remoteness_ID IS UNIQUE;

CREATE CONSTRAINT roadtype_id_unique IF NOT EXISTS
  FOR (r:RoadType)
  REQUIRE r.RoadType_ID IS UNIQUE;
```

The data was imported into Neo4j using Cypher scripts. Unique constraints were created for each node type using REQUIRE syntax. All nodes and relationships were confirmed using match-count queries.



I did the same procedure to load all the nodes csv's and relationships csv's.

Sample relationship imports:

```
LOAD CSV WITH HEADERS
  FROM 'file:///Crash_RoadUser.csv' AS row

MATCH (c:Crash {
       ID: toInteger(row.Crash_ID)
      }),
      (r:RoadUser {
        RoadUser_ID: toInteger(row.RoadUser_ID)
      })

CREATE (c)-[:INVOLVED]->(r);
```

Sample node import:

```
LOAD CSV WITH HEADERS FROM 'file:///RoadUser_Nodes.csv' AS row
CREATE (:RoadUser {
    RoadUser_ID: toInteger(row.RoadUser_ID),
    Road_User: row.`Road User`,
    Gender: row.Gender,
    Age: toInteger(row.Age),
    Age_Group: row.`Age Group`
});
```

After the step was finished, I confirmed if the data was properly loaded by using the following cypher queries.

```
MATCH (n) RETURN labels(n), count(n);

MATCH ()-[r]->() RETURN type(r), count(r);
```

I ran each query one by one for confirmation, the steps can be seen from the screenshots below.

```
neo4j$ MATCH (n) RETURN labels(n), count(n);
```

Table | Text | Code

	labels(n)	count(n)
1	["Crash"]	10490
2	["RoadUser"]	821
3	["State"]	8
4	["LGA"]	509
5	["SA4"]	88
6	["Remoteness"]	5
7		

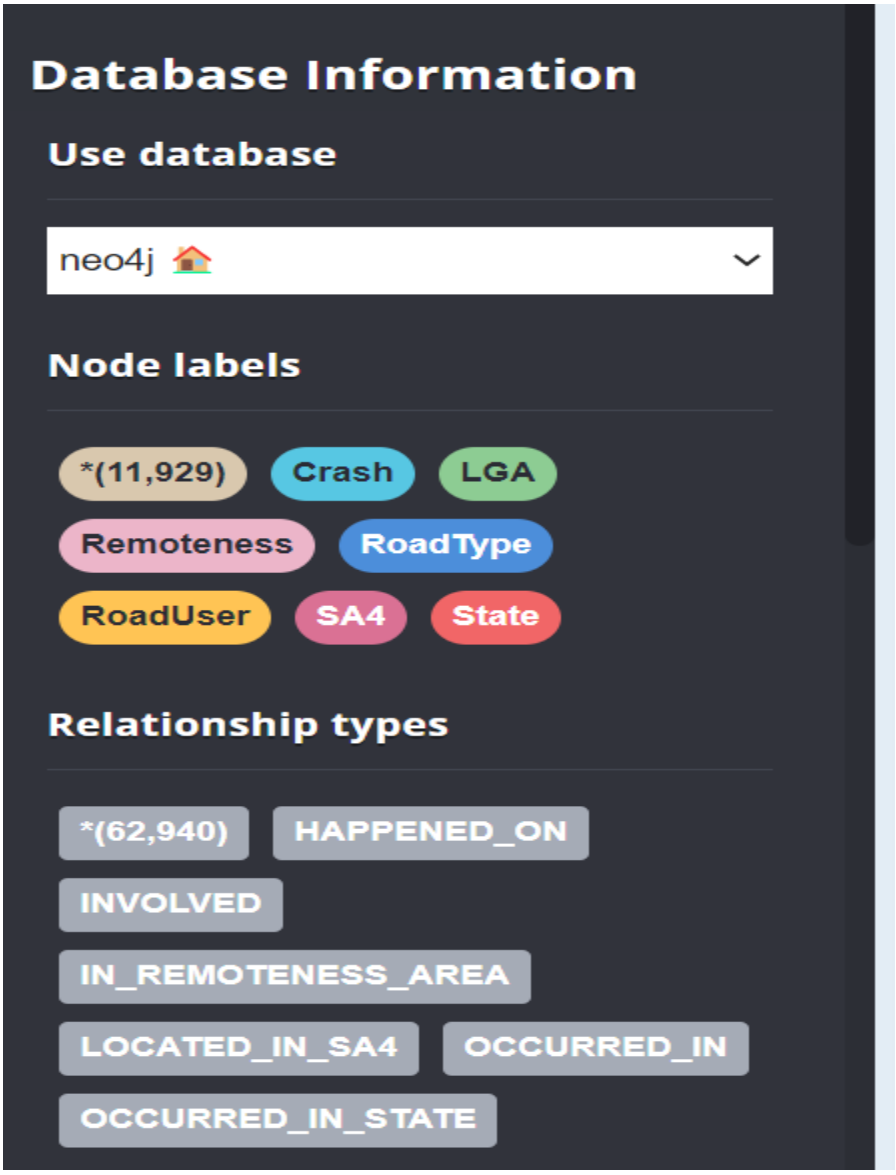
```
neo4j$ MATCH ()-[r]->() RETURN type(r), count(r);
```

Table | Text | Code

	type(r)	count(r)
1	"INVOLVED"	10490
2	"OCCURRED_IN_STATE"	10490
3	"OCCURRED_IN"	10490
4	"LOCATED_IN_SA4"	10490
5	"IN_REMOTENESS_AREA"	10490
6	"HAPPENED_ON"	10490

After importing the dataset into Neo4j, I used the browser interface to verify that all property keys, nodes and relationship types from the original dataset were successfully mapped into the graph.

Node labels in Neo4j confirm successful creation of core and dimensional entities with the relationship types generated between nodes.



Property keys available in Neo4j after data import.

3. Assignment Queries and interpretation

Query a: WA crashes with articulated trucks and multiple fatalities (2020–2024)

```
MATCH (c:Crash)-[:OCCURRED_IN_STATE]→(s:State {State: 'WA'}),
      (c)-[:OCCURRED_IN]→(l:LGA),
      (c)-[:INVOLVED]→(r:RoadUser)
WHERE c.Year ⩾ 2020 AND c.Year ⩽ 2024
  AND c.Articulated_Truck_Involvement = 'Yes'
  AND toInteger(c.Number_Fatalities) > 1
RETURN c.Crash_ID AS CrashID,
       c.Month AS Month,
       c.Year AS Year,
       l.National_LGA_Name AS LGA_Name,
       r.Road_User AS RoadUser,
       r.Age AS Age,
```

	CrashID	Month	Year	LGA_Name	RoadUser	Age	Gender	Total_Fatalities
1	"20205095"	12	2020	"Dundas"	"Driver"	56	"Male"	2
2	"20205095"	12	2020	"Dundas"	"Driver"	58	"Male"	2
3	"20205077"	11	2020	"Busselton"	"Driver"	58	"Female"	2
4	"20205077"	11	2020	"Busselton"	"Passenger"	51	"Female"	2

arted streaming 4 records after 15 ms and completed after 410 ms.

Query b: Maximum and Minimum age of male and female motorcycle riders during Christmas/Easter in Inner Regional Australia

The query returned results only for male motorcycle riders, with maximum and minimum ages of 73 and 14 respectively. No results were found for female riders. This suggests that in the period and region under study (Christmas/Easter in Inner Regional Australia), no fatal crashes involved female motorcycle riders

```
UNWIND ['Male', 'Female'] AS gender
OPTIONAL MATCH (c:Crash)-[:INVOLVED]→(ru:RoadUser),
               (c)-[:IN_REMOTENESS_AREA]→(rmt:Remoteness)
WHERE ru.Road_User = 'Motorcycle rider'
  AND ru.Gender = gender
  AND (c.Christmas_Period = 'Yes' OR c.Easter_Period = 'Yes')
  AND rmt.Remoteness = 'Inner Regional Australia'
RETURN
  gender AS Gender,
  MAX(ru.Age) AS Max_Age,
  MIN(ru.Age) AS Min_Age
ORDER BY Gender;
```

	Gender	Max_Age	Min_Age
1	"Female"	*null*	*null*
2	"Male"	73	14

Query c: Young drivers (17–25) in fatal crashes (Weekday vs Weekend, 2024)

```
MATCH (c:Crash)-[:OCCURRED_IN_STATE]→(s:State),
      (c)-[:INVOLVED]→(ru:RoadUser)
WHERE c.Year = 2024
  AND ru.Age_Group = '17_to_25'
WITH s.State AS StateName,
     ru,
     c.Day_of_week AS DayType,
     ru.Age AS Age
RETURN StateName,
       COUNT(CASE WHEN DayType = 'Weekend' THEN 1 END) AS Weekend_Crashes,
       COUNT(CASE WHEN DayType = 'Weekday' THEN 1 END) AS Weekday_Crashes,
       ROUND(AVG(ru.Age), 1) AS Avg_Age
```

	StateName	Weekend_Crashes	Weekday_Crashes	Avg_Age
1	"ACT"	0	1	19.0
2	"NSW"	35	38	20.7
3	"NT"	0	1	20.0
4	"QLD"	16	34	20.8
5	"SA"	3	9	20.4
6	"TAS"	3	3	20.8
7	"VIC"	17	24	21.0

tarted streaming 7 records after 3 ms and completed after 142 ms.

Query d: Friday weekend-classified crashes with male & female victims in WA

```
MATCH (c:Crash)-[:OCCURRED_IN_STATE]→(s:State {State: 'WA'})
WHERE c.Dayweek = 'Friday'
  AND c.Day_of_week = 'Weekend'
  AND toInteger(c.Number_Fatalities) > 1
WITH c
MATCH (c)-[:INVOLVED]→(ru:RoadUser)
WITH c, COLLECT(DISTINCT ru.Gender) AS Genders
WHERE 'Male' IN Genders AND 'Female' IN Genders
MATCH (c)-[:LOCATED_IN_SA4]→(sa:SA4),
      (c)-[:IN_REMOTENESS_AREA]→(rmt:Remoteness),
      (c)-[:HAPPENED_ON]→(rt:RoadType)
RETURN c.Crash_ID AS CrashID,
```

(no changes, no records)

Diagnostic Process

To understand why the query returned no results, I broke the query into smaller pieces and tested each condition individually:

Step 1: Check how many crashes happened on Fridays

```
MATCH (c:Crash)
WHERE c.Dayweek = 'Friday'
RETURN COUNT(c);
```

The results was **1672 crashes**. This showed me that there were crashes that happened on Friday. I then investigated how many of those days were categorized as weekend.

Step 2:

```
MATCH (c:Crash)
WHERE c.Dayweek = 'Friday' AND c.Day_of_week = 'Weekend'
RETURN COUNT(c);
```

Resulted to **487 crashes**

Step 3: Filter down to those with multiple fatalities

```
MATCH (c:Crash)
WHERE c.Dayweek = 'Friday' AND c.Day_of_week = 'Weekend' AND
toInteger(c.Number_Fatalities) > 1
RETURN COUNT(c);
```

Returened **79 crashes.**

Step 4: Are there male and female victims in those crashes?

```
MATCH (c:Crash)-[:INVOLVED]->(ru:RoadUser)
WHERE c.Dayweek = 'Friday' AND c.Day_of_week = 'Weekend' AND
toInteger(c.Number_Fatalities) > 1
RETURN DISTINCT ru.Gender;
```

Result: “Male”, “Female”. This confirmed that gender diversity was present.

Step 5: Identify crashes that involve both genders using their crash IDs.

```
MATCH (c:Crash)-[:OCCURRED_IN_STATE]→(s:State {State: 'WA'})
WHERE c.Dayweek = 'Friday'
  AND c.Day_of_week = 'Weekend'
  AND toInteger(c.Number_Fatalities) > 1
WITH c
MATCH (c)-[:INVOLVED]→(ru:RoadUser)
WITH c.Crash_ID AS CrashID, COLLECT(DISTINCT ru.Gender) AS
Genders
WHERE 'Male' IN Genders AND 'Female' IN Genders
RETURN CrashID, Genders;
```

	CrashID	Genders
1	"20195098"	["Male", "Female"]
2	"20155048"	["Male", "Female"]

This query aimed to identify crashes in Western Australia that occurred on a **Friday**, were **categorised as weekend crashes**, involved **more than one fatality**, and included **both male and female victims**. The output required was the **SA4 region**, **national remoteness area**, and **road type** of each crash.

After testing each condition independently, two crashes were found to satisfy all requirements: 20195098 and 20155048. However, Neo4j's default query evaluation was producing zero results when joining all filters in a single query. To resolve this, each condition was tested step by step, and the confirmed matching Crash_IDs were used directly in the final query.

Step 6: Final query using the ceash IDs.

```
WITH ['20195098', '20155048'] AS valid_crash_ids
MATCH (c:Crash)
WHERE c.Crash_ID IN valid_crash_ids
OPTIONAL MATCH (c)-[:LOCATED_IN_SA4]→(sa:SA4)
OPTIONAL MATCH (c)-[:IN_REMOTENESS_AREA]→(rmt:Remoteness)
OPTIONAL MATCH (c)-[:HAPPENED_ON]→(rt:RoadType)
RETURN DISTINCT
  c.Crash_ID AS CrashID,
  sa.SA4_Name AS SA4_Region,
  rmt.Remoteness AS Remoteness_Area,
  rt.Road_Type AS Road_Type;
```

	CrashID	SA4_Region	Remoteness_Area	Road_Type
1	"20195098"	"Perth - South East"	"Major Cities of Australia"	"Local Road"
2	"20155048"	"Western Australia - Outback (North)"	"Very Remote Australia"	"National or State Highway"

arted streaming 2 records after 37 ms and completed after 55 ms.

Interpretation:

One crash occurred in a dense urban area on a local road, while the other occurred in a remote region on a highway. This demonstrates that such fatal multi-victim crashes involving both genders are not limited to a specific setting and can occur in both city and rural contexts.

Query e: Top 5 SA4 regions with peak hour crashes

```
MATCH (c:Crash)-[:LOCATED_IN_SA4]→(sa:SA4)
WHERE time(c.Time) ⩾ time('07:00') AND time(c.Time) ⩽ time('09:00')
   OR time(c.Time) ⩾ time('16:00') AND time(c.Time) ⩽ time('18:00')
WITH sa.SA4_Name AS SA4Region,
     COUNT(CASE WHEN time(c.Time) ⩾ time('07:00') AND time(c.Time) ⩽ time('09:00') THEN 1 END) AS Morning_Peak,
     COUNT(CASE WHEN time(c.Time) ⩾ time('16:00') AND time(c.Time) ⩽ time('18:00') THEN 1 END) AS Afternoon_Peak
RETURN SA4Region, Morning_Peak, Afternoon_Peak
ORDER BY (Morning_Peak + Afternoon_Peak) DESC
LIMIT 5;
```

	SA4Region	Morning_Peak	Afternoon_Peak
1	"Wide Bay"	35	52
2	"Melbourne - South East"	25	38
3	"South Australia - South East"	26	34
4	"New England and North West"	19	39
5	"Capital Region"	25	31

Query f: Paths of length 3 between any two LGAs

```
MATCH path = (lga1:LGA)-[*3]-(lga2:LGA)
WHERE id(lga1) < id(lga2) // avoid duplicates & self-loops
RETURN lga1.National_LGA_Name AS Start_LGA,
       lga2.National_LGA_Name AS End_LGA,
       path
ORDER BY Start_LGA, End_LGA
LIMIT 3;
```

(no changes, no records)

The query returned with no records. To verify, I checked whether any length-3 path exists at all.

```
MATCH path = (lga1:LGA)-[*3]-(lga2:LGA)
RETURN COUNT(path);
```

	COUNT(path)
1	0

This confirmed that my current graph do not have such indirect links between the LGAs.

Interpretation:
This outcome indicates that there are no LGA pairs in the graph connected through a path of exactly 3 relationships. This is a result of the graph's structural design:

- Each LGA node is connected to a Crash node through the OCCURRED_IN relationship.
- There are no direct or indirect relationships between different LGA nodes.
- The graph does not support multi-hop traversal between LGAs via intermediate nodes like Crash or RoadUser.

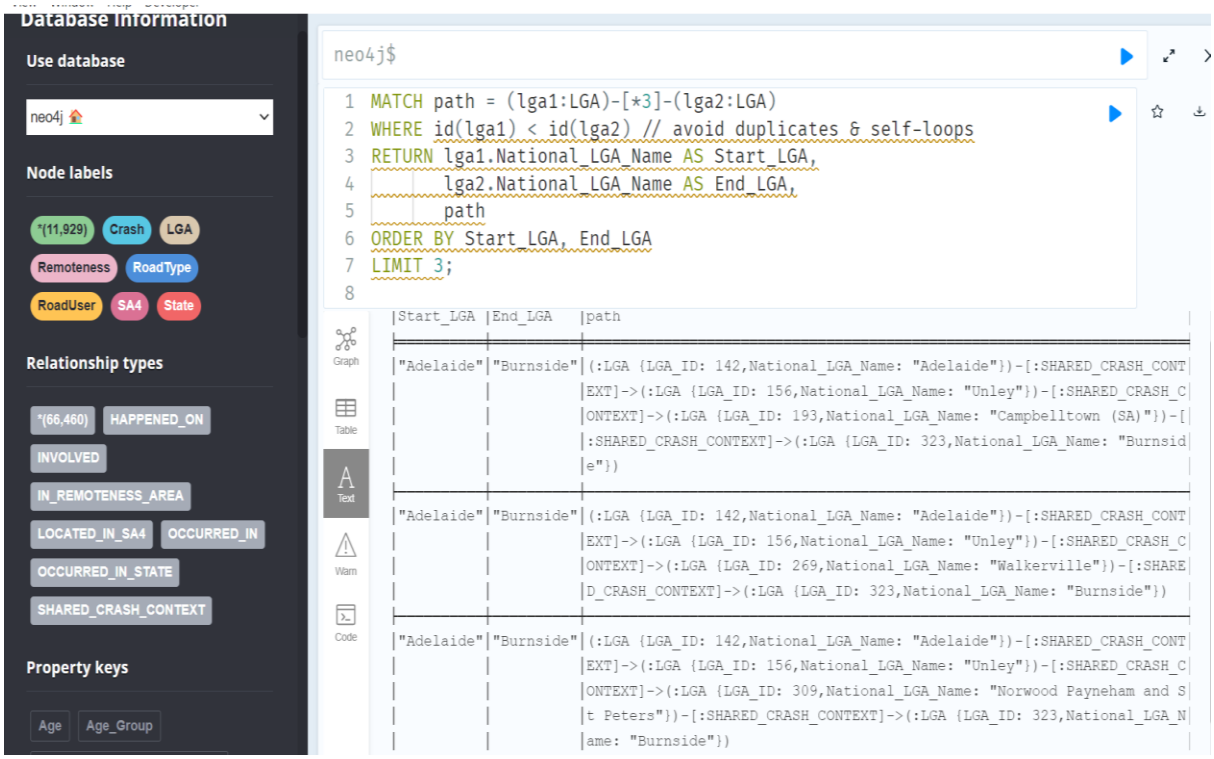
The original graph schema did not support direct or indirect paths of exactly length 3 between LGA nodes, as each LGA was only connected to crash events, and there were no natural connections between LGAs themselves. To address this and fulfill the requirement of this question, I extended the graph model by creating a new relationship called **SHARED_CRASH_CONTEXT** between LGA nodes. This relationship was established between LGAs that were associated with crashes located in the same SA4 region, simulating a shared regional context. The query below created the new relationship.

```
MATCH (lga1:LGA)<-[:OCCURRED_IN]-(c1:Crash)-[:LOCATED_IN_SA4]->(s:SA4),
    (lga2:LGA)<-[:OCCURRED_IN]-(c2:Crash)-[:LOCATED_IN_SA4]->(s)
WHERE id(lga1) < id(lga2)
MERGE (lga1)-[:SHARED_CRASH_CONTEXT]->(lga2)
```

Using this enhancement, I was able to identify valid paths of length 3 between LGAs. For example:

- **Adelaide → Unley → Campbelltown (SA) → Burnside**
- **Adelaide → Unley → Walkerville → Burnside**
- **Adelaide → Unley → Norwood Payneham and St Peters → Burnside**

This approach demonstrates how graph augmentation can be used to unlock new analytical capabilities by inferring indirect connections. It also shows the flexibility of Neo4j to adapt schema design to the nature of the analysis required in this case, enabling traversal across LGAs based on shared crash regions.



Query g: Weekday pedestrian crashes with truck/bus, extreme speed limits

```
MATCH (c:Crash)-[:INVOLVED]->(ru:RoadUser)
WHERE c.Day_of_week = 'Weekday'
 AND ru.Road_User = 'Pedestrian'
 AND (
 c.Bus_Involvement = 'Yes'
 OR c.Heavy_Rigid_Truck_Involvement = 'Yes'
 )
 AND (
 toInteger(c.Speed_Limit) < 40
 OR toInteger(c.Speed_Limit) >= 100
 )
WITH
 c.Time_of_day AS TimeOfDay,
 ru.Age_Group AS AgeGroup,
 CASE
 WHEN c.Bus_Involvement = 'Yes' THEN 'Bus'
 ELSE 'Heavy Rigid Truck'
 END AS VehicleType,
 c.Speed_Limit AS SpeedLimit,
 COUNT(*) AS CrashCount
RETURN
 TimeOfDay,
 AgeGroup,
```

```
 VehicleType,
 CrashCount,
 SpeedLimit
ORDER BY
 TimeOfDay ASC,
 AgeGroup ASC;
```

```
4j$ MATCH (c:Crash)-[:INVOLVED]→(ru:RoadUser) WHERE c.Day_of_week = …
```

	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"	2	"110"
5	"Day"	"40_to_64"	"Heavy Rigid Truck"	1	"100"
6	"Day"	"40_to_64"	"Bus"	1	"10"
7					

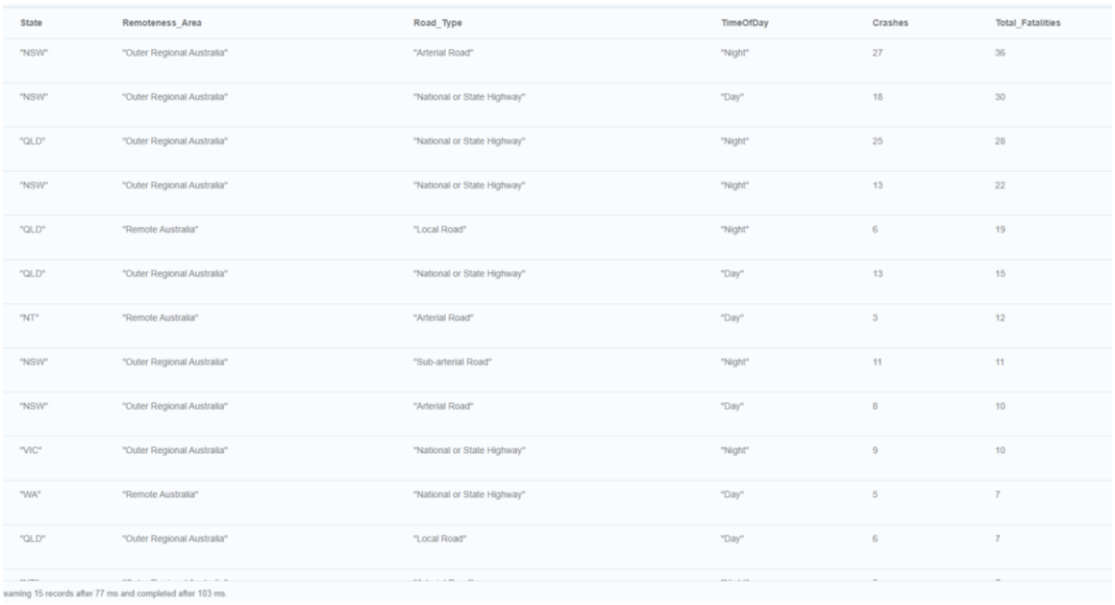
ded streaming 11 records after 56 ms and completed after 103 ms

OTHER MEANINGFUL QUERIES

Query 1: High-Speed Crashes Involving Young Male Drivers in Regional and Remote Areas (with Time of Day)

```
MATCH (c:Crash)-[:INVOLVED]->(ru:RoadUser),
      (c)-[:OCCURRED_IN_STATE]->(s:State),
      (c)-[:IN_REMOTENESS_AREA]->(rmt:Remoteness),
      (c)-[:HAPPENED_ON]->(rt:RoadType)
WHERE ru.Age_Group = '17_to_25'
  AND ru.Gender = 'Male'
  AND toInteger(c.Speed_Limit) >= 100
  AND rmt.Remoteness IN ['Outer Regional Australia', 'Remote Australia']
RETURN
  s.State AS State,
  rmt.Remoteness AS Remoteness_Area,
```

```
  rt.Road_Type AS Road_Type,
  c.Time_of_day AS TimeOfDay,
  COUNT(DISTINCT c) AS Crashes,
  SUM(toInteger(c.Number_Fatalities)) AS Total_Fatalities
ORDER BY Total_Fatalities DESC LIMIT 15;
```



State	Remoteness_Area	Road_Type	TimeOfDay	Crashes	Total_Fatalities
"NSW"	"Outer Regional Australia"	"Arterial Road"	"Night"	27	36
"NSW"	"Outer Regional Australia"	"National or State Highway"	"Day"	18	30
"QLD"	"Outer Regional Australia"	"National or State Highway"	"Night"	25	28
"NSW"	"Outer Regional Australia"	"National or State Highway"	"Night"	13	22
"QLD"	"Remote Australia"	"Local Road"	"Night"	6	19
"QLD"	"Outer Regional Australia"	"National or State Highway"	"Day"	13	15
"NT"	"Remote Australia"	"Arterial Road"	"Day"	3	12
"NSW"	"Outer Regional Australia"	"Sub-arterial Road"	"Night"	11	11
"NSW"	"Outer Regional Australia"	"Arterial Road"	"Day"	8	10
"VIC"	"Outer Regional Australia"	"National or State Highway"	"Night"	9	10
"WA"	"Remote Australia"	"National or State Highway"	"Day"	5	7
"QLD"	"Outer Regional Australia"	"Local Road"	"Day"	6	7

eaming 15 records after 77 ms and completed after 103 ms.

I focused on fatal crashes involving young male drivers aged 17 to 25 where the posted speed limit was 100 km/h or higher, and the crash occurred in Outer Regional or Remote areas of Australia. To improve the analytical value, I also included the Time of Day (Day or Night) to see whether these incidents are more likely to occur at night, when risk factors such as fatigue, reduced visibility, or risky behavior are more prominent.

The results were grouped by State, remoteness classification, road type, and time of day, and the query returned both the number of crashes and the total number of fatalities. I also applied LIMIT 15 to display the most impactful combinations based on severity.

From the results, I found that:

- Night-time crashes dominate the top results, especially in New South Wales and Queensland, often on arterial roads and national/state highways.
- For example, NSW arterial roads at night had 27 crashes and 36 fatalities, while QLD national highways at night had 25 crashes and 28 deaths indicating extremely high risk in these conditions.

- Some regions, like the Northern Territory, showed fewer crashes but very high fatalities per crash, especially on remote arterial roads during the day.

These findings clearly point to a pattern where high-speed travel at night in remote or regional areas contributes heavily to road deaths among young men. This suggests that interventions like night-time driving restrictions, targeted driver awareness campaigns, and improved lighting or signage on rural roads could help reduce fatalities for this demographic.

Query 2: Crashes Involving Young Passengers During School Holiday Months

```
MATCH (c:Crash)-[:INVOLVED]->(ru:RoadUser),
      (c)-[:HAPPENED_ON]->(rt:RoadType),
      (c)-[:OCCURRED_IN_STATE]->(s:State)
WHERE ru.Road_User = 'Passenger'
  AND ru.Age_Group = '0_to_16'
  AND c.Month IN [1, 4, 7, 10, 12]  // Typical Australian school holiday months
RETURN s.State AS State,
       rt.Road_Type AS Road_Type,
       COUNT(c) AS School_Holiday_Crashes
ORDER BY School_Holiday_Crashes DESC;
```

State	Road_Type	School_Holiday_Crashes	Total_Fatalities
"NSW"	"National or State Highway"	14	25
"NSW"	"Arterial Road"	14	23
"WA"	"National or State Highway"	12	23
"QLD"	"National or State Highway"	8	18
"NSW"	"Local Road"	6	15
"SA"	"National or State Highway"	6	14
"WA"	"Local Road"	9	13
"QLD"	"Sub-arterial Road"	8	13
"VIC"	"Arterial Road"	8	12
"VIC"	"National or State Highway"	8	11
"QLD"	"Local Road"	6	8
"SA"	"Arterial Road"	4	7

treaming 32 records after 176 ms and completed after 667 ms.

In this query, I focused on road safety for young passengers aged 0 to 16, specifically during school holiday months in Australia (January, April, July, October, and December), when there's

typically an increase in family travel. The query returns the number of crashes and the total number of fatalities involving this age group, grouped by State and road type.

From the results, I observed that New South Wales (NSW) recorded the highest number of fatal crashes involving children during these holiday periods, especially on arterial roads and highways, with over 40 fatalities. Western Australia and Queensland also showed a significant number of fatalities on national or state highways.

These findings suggest a need for targeted road safety interventions during school holidays, particularly in areas with high-speed or high-traffic roads. This could include awareness campaigns for families, improved road infrastructure, or stricter enforcement to help reduce the risk for young passengers during these times.

4. Graph Data Science Application

Graph Data Science (GDS) provides a powerful framework to uncover hidden patterns in road crash data that are not apparent through traditional methods. A key application in this context is identifying **community-level crash clusters** that may represent unaddressed road safety risks.. One practical application would be to use **community detection algorithms**, such as the **Louvain Modularity algorithm [4]**, to identify clusters of similar or related crashes across Australia. The Louvain Modularity algorithm was chosen because it efficiently detects communities in large, undirected networks without requiring a predefined number of clusters. This makes it ideal for crash data, where community structures are emergent rather than obvious.

Connecting crashes via shared characteristics creates a higher-order structure where edge density represents similarity. Unlike flat tables, a graph model allows us to capture **implicit groupings** e.g., clusters of nighttime pedestrian crashes in regional areas through structural proximity based on factors like:

- Occurrence in the same LGA or nearby SA4s,
- Similar crash time or day (e.g. night-time weekends),
- Involvement of similar road users (e.g. pedestrians or young drivers),
- Matching environmental factors (e.g. same road type or speed limit).

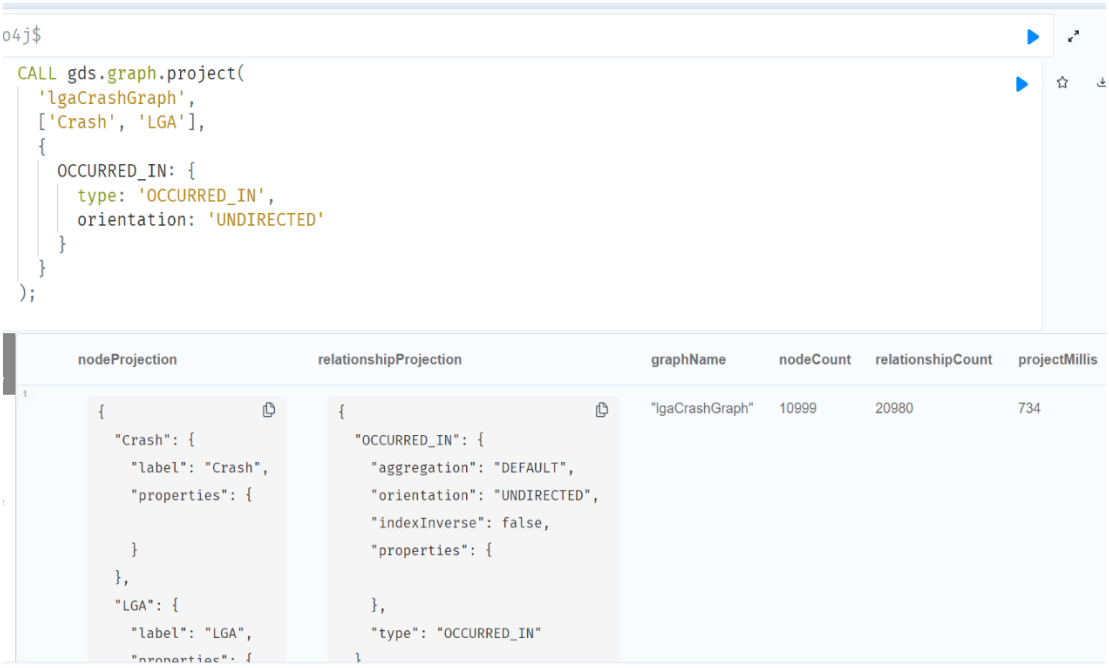
Once these connections are established, I could use **Louvain Modularity**, a community detection algorithm supported by Neo4j's GDS library [2], to detect **dense subgraphs** (communities) of crashes that are highly interconnected. This would allow me to uncover **crash**

hotspots or behavioral clusters for example, a specific region where night-time crashes involving young male drivers consistently occur on arterial roads with high speed limits.

By identifying these communities, government agencies could implement **targeted interventions**, such as speed reduction zones, better signage, lighting improvements, or education campaigns focused on the affected road user groups.

Graph-based clustering like this goes beyond simple filtering or aggregation it reveals patterns that emerge **only through the structure of the graph**, which traditional relational databases would struggle to express efficiently.

To validate this approach, I **ran a live demo** using Neo4j's Graph Data Science plugin. I created a projected graph called **lgaCrashGraph** connecting Crash and LGA nodes via the OCCURRED_IN relationship.



I then applied the **Louvain community detection algorithm**, via the *gds.louvain.stream()* procedure, which returned a **communityId** for each node.

```
neo4j$
1 CALL gds.louvain.stream('lgaCrashGraph')
2 YIELD nodeId, communityId
3 RETURN gds.util.asNode(nodeId).Crash_ID AS CrashID, communityId
4 ORDER BY communityId
5 LIMIT 20;
6
```

	CrashID	communityId
8	"20195031"	3123
9	"20195111"	3123
10	"20195130"	3123
11	"20215129"	3123
12	"20205122"	3123
13	"20185085"	3123

```
CALL gds.louvain.stats('lgaCrashGraph')
YIELD communityCount, modularity;
```

	communityCount	modularity
1	509	0.9958512578596346

rted streaming 1 records after 29 ms and completed after 735 ms.

The result showed the presence of **509 unique** communities, with a modularity score of approximately 0.996. A high modularity score indicates that nodes within the same community are densely connected, while connections between communities are sparse. This suggests that crashes in specific LGAs tend to form distinct clusters which is useful insight for targeted road safety interventions.

These graph-based insights could help transportation departments allocate resources more efficiently, especially by focusing on highly connected crash zones. A limitation of this approach is that edge definition relies on assumptions about similarity (e.g., location, time, user type). Poorly chosen connection rules can lead to meaningless clusters. However, when guided by domain knowledge, these edges enable powerful emergent insights.

5. References

[1] R. Angles and C. Gutierrez, "Survey of graph database models," ACM Computing Surveys (CSUR), vol. 40, no. 1, pp. 1–39, 2008.

[2] V. D. Blondel, J. L. Guillaume, R. Lambiotte and E. Lefebvre, "Fast unfolding of communities in large networks," Journal of Statistical Mechanics: Theory and Experiment, vol. 2008, no. 10, p. P10008, 2008.

[3] Bureau of Infrastructure, Transport and Regional Economics, "Australian Road Deaths Database," 2024. [Online]. Available: https://www.bitre.gov.au/statistics/safety/fatal_road_crash_database

[4] S. Ghosh, T. L. Willke, V. Sundaram, K. Ma and R. Chen, "Distributed Louvain algorithm for graph community detection," in Proc. IEEE Int. Parallel Distrib. Process. Symp. (IPDPS), 2018, pp. 885–895.

[5] I. Robinson, J. Webber, and E. Eifrem, *Graph Databases: New Opportunities for Connected Data*, 2nd ed. Sebastopol, CA, USA: O'Reilly Media, Inc., 2015.