CITS5504 Project 1

Laine Mulvay (22708032) and Mohaed Ismail Khan (24894389)

Table of Contents

---

1. Introduction

1.1 Project Overview

In this project, we designed and implemented a data warehouse to facilitate business intelligence and analytics. This report outlines and discusses the project process, including key insights gained throughout the project. Along with the report, all relevant materials have been included in the `Project1_Fatalities` folder.

Within this folder, you will find the following:

- Scripts: All Python and SQL scripts used throughout the project, including the ETL process (`etl_process.py`) and database queries (`queries.sql`). These scripts are clearly commented to ensure they are well-structured and easy to understand.
- Power BI/Tableau Files: The Power BI/Tableau workbooks used for visualisations are included in the archive folder. Specifically, the file `query_visualisations.pdf` provides the visualisations, and the Tableau workbook query_workbook.twb offers the structured data representation.
- CSV Files: All CSV files used for building and populating the database are in the data/raw folder. These include essential datasets such as `bitre_fatalities_dec2024.xlsx` and Population estimates by LGA that contributed to the data warehouse's creation.
- README.md: Outlining the poject repository and how to set up the project on your own machine. This includes the set up of Docker, the local PostgreSQL database, pgAdmin and BI tooling

Note: Internal links for this report only work when it is opened at `report/Project1_Report.ipynb`

The full directory structure is as follows:

In [ ]:
```
Project1_Fatalities
.
├── Dockerfile
├── README.md
├── association_rules/
├── data
│   ├── geojson
│   │   ├── LGA_2021_AUST_GDA94.geojson
│   │   ├── SA4_2021_AUST_GDA2020.geojson
│   │   └── STE_2021_AUST_GDA2020.geojson
│   ├── processed/
│   └── raw
│       ├── LGA (count of dwellings).csv
│       ├── ardd_dictionary_sep2023.pdf
```

```
│       ├── bitre_fatal_crashes_dec2024.xlsx
│       └── bitre_fatalities_dec2024.xlsx
├── docker-compose.yml
├── images/
├── report
│   └── Project1_Report.ipynb
├── requirements.txt
├── scripts
│   ├── etl_process.py
│   ├── queries.sql
│   └── setup.sh
├── visualisations
│   ├── query_visualisations.pdf
│   └── query_workbook.twb
└── working_notebooks
    ├── ETL_Explained.ipynb
    └── Kimballs_steps.ipynb
```

1.2 Business Objectives

The primary objective of this project is to support decision-making by enabling the ability to ask insightful questions and perform effective data analysis. This will be achieved through the design of a snowflake schema data warehouse, which will be explored in detail, including the rationale behind its choice.

As part of the project, we will design six key business questions that leverage different combinations of data across various parts of the schema. These questions will draw from multiple dimensions and measures within the data warehouse, ensuring it can effectively support diverse analytical needs. By structuring the data in a snowflake design, we aim to provide a flexible and efficient framework for answering these queries, while also allowing for future analytical growth.

1.3 Tools and Technologies Used

This project utilises Docker to containerise the environment, providing a consistent setup across development and production. It includes a PostgreSQL database for data storage, with pgAdmin for database management and visualisation. The ETL pipeline runs within a Docker container, leveraging Python virtual environments to manage dependencies and ensure portability. For data visualisation, the project integrates with Power BI and Tableau, which connect to the PostgreSQL database for real-time insights. The use of Docker and virtual environments simplifies setup and ensures that the project can be easily reproduced and scaled across different systems.

2. Data Warehouse Design

2.1 Schema Overview

Note: `/working_notebooks/Kimballs_steps.ipynb` was the workbook used to design the following Schema (by implementing Kimballs 4 Steps)

2.1.1 Fact Table Design

The data warehouse design includes three fact tables, each serving different analytical purposes:

Fact_Crashes

Column Name	Description
`crash_id` (PK)	Unique identifier for the crash event.
`date_id` (FK)	Reference to `Dim_Date` table.
`lga_id` (FK)	Reference to `Dim_LGA` table, which itself joind `Dim_State`.
`state_id` (FK)	Reference to `Dim_State` table.

Fact_Fatalities

Column Name	Description
`fatality_id` (PK)	Unique identifier for each fatality.
`person_id` (FK)	Reference to `Dim_Person` table.
`crash_id` (FK)	Reference to `Fact_Crashes` table.

Fact_Number

Column Name	Description
`number_date_id` (PK)	Unique identifier for the daily summary record.
`date_id` (FK)	Reference to `Dim_Date` table.
`total_fatalities`	Total number of fatalities on the given date (and optionally by state). This is a `MEASURE`
`total_crashes`	Total number of crashes on the given date (and optionally by state). This is a `MEASURE`

The fact tables were designed with clear **grain** definitions:

- Fact_Crashes: One row per crash event
- Fact_Fatalities: One row per fatality associated with a crash
- Fact_Number: One row per date for aggregated metrics. This **pre-aggrigated** fact table **stores measures** for **opimised querying**

2.1.2 Dimension Tables and Conecpt Hierachies

The data warehouse incorporates several dimension tables with defined hierarchies:

Dimension Tables

Dim_Date *Time dimension - contains all date attributes*

Column Name	Description
`date_id` (PK)	Unique identifier for each date
`year`	Year of the crash/fatality
`month`	Month of the crash/fatality
`day`	Day of the month (e.g. 1-31)
`day_of_week`	Day of the week (e.g., Monday, Tuesday)
`is_weekend`	Boolean indicating if the crash occurred on a weekend

Dim_State *Geographic dimension - highest level geography*

Column Name	Description
`state_id` (PK)	Unique identifier for each state
`state_name`	Name of the state

Dim_LGA *Geographic dimension - local government area level*

Column Name	Description
`lga_id` (PK)	Unique identifier for each LGA
`lga_name`	Local Government Area name
`state_id` (FK)	Reference to `Dim_State` table
`national_remoteness_area`	Area classification based on remoteness
`dwelling_count`	Number of dwellings in the LGA. This is a `MEASURE`

Dim_Time *Time dimension - contains time of day attributes*

Column Name	Description
`crash_id` (PK)	Unique identifier for the time
`crash_time`	Exact time of the crash (in timestamp format)
`time_of_day`	Time of day (e.g., Morning, Afternoon, Evening)

Dim_Vehicle *Vehicle dimension - contains vehicle involvement attributes*

Column Name	Description
`crash_id` (PK)	Unique identifier for vehicle data related to a crash
`bus_involvement`	Boolean indicating if a bus was involved
`heavy_rigid_truck_involvement`	Boolean indicating if a heavy rigid truck was involved

Column Name	Description
`articulated_truck_involvement`	Boolean indicating if an articulated truck was involved

Dim_Person *Person dimension - contains demographic information about people involved*

Column Name	Description
`person_id` (PK)	Surrogate key made from a combination of `CrashID`, `Age`, `Gender`, `RoadUser`
`crash_id` (FK)	ID of the crash in which the person was involved
`gender`	Gender of the individual
`age`	Age of the individual
`age_group`	Age group of the individual (e.g., 18-25, 26-40)
`road_user`	Type of road user (e.g., Pedestrian, Driver, Passenger)

Dim_Event *Event dimension - contextual information about the crash*

Column Name	Description
`crash_id` (PK)	Unique identifier for the event
`christmas_period`	Boolean indicating if the crash occurred during the Christmas period
`easter_period`	Boolean indicating if the crash occurred during the Easter period

Dim_Road *Road dimension - attributes of the road where the crash occurred*

Column Name	Description
`crash_id` (PK)	Unique identifier for road data
`speed_limit`	Speed limit of the road where the crash occurred
`national_road_type`	Type of road (e.g., highway, local road)

Conceptual Hierarchies

Geographic Hierarchy

```
State → LGA
```

- This hierarchy allows analysis to drill down from state-level to local government areas
- The snowflake schema implementation connects Dim_LGA to Dim_State via `state_id`

Time Hierarchy

```
Year → Month → Day
```

- This hierarchy in Dim_Date enables time-based analysis at multiple levels
- Additional temporal attributes like `day_of_week` and `is_weekend`, while not fitting directly in the main heirachy, support specialised time analyses

Vehicle Classification Hierarchy

```
Vehicle Type (Bus, Heavy Rigid Truck, Articulated Truck)
```

Road Classification Hierarchy

```
Speed Limit ranges
National Road Type categories
```

Person Hierarchy

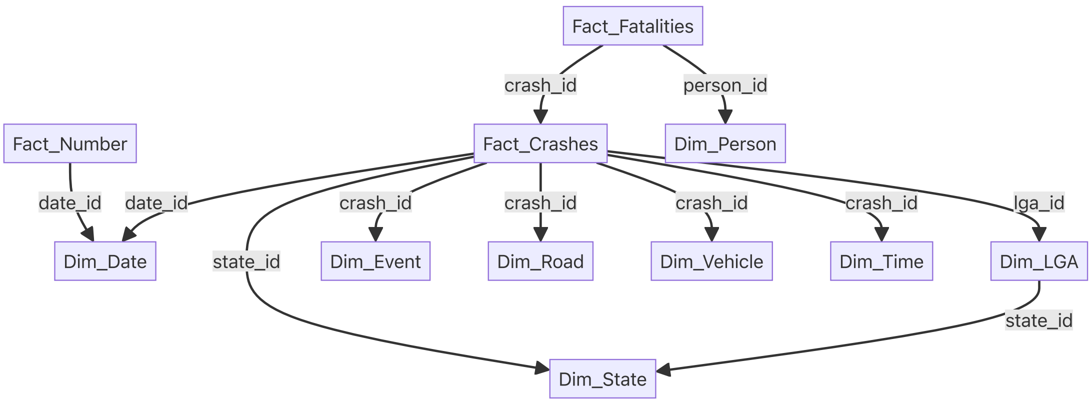
```
Age Group → Age
- Road User Type (categorical classification)
- Gender (categorical classification)
```

2.1.3 Schema and Justification

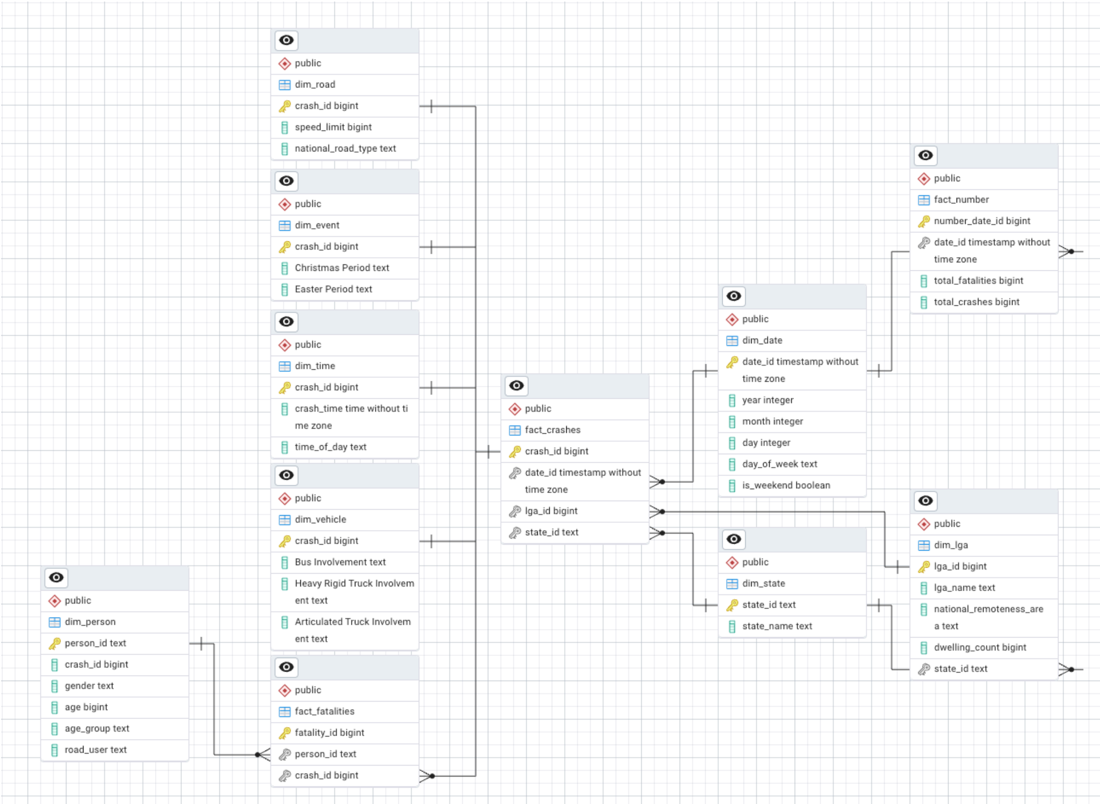
Schema Design

We have implemented a **partially normalised Snowflake** Schema design for the datawarehouse. This is outlined in the following figures

Visal Schema Diagram

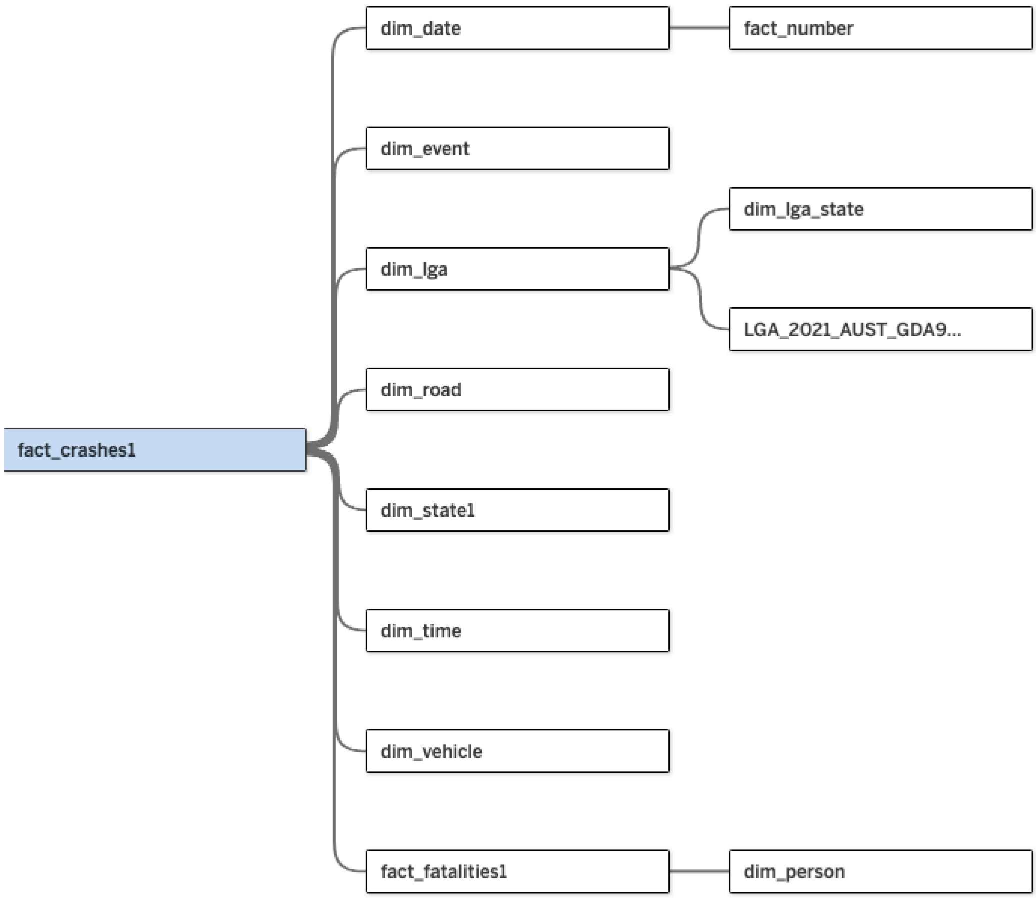


Schema Diagram from pgAdmin



Note: In Tableau implementation, we create two separate instances of Dim_State (one called Dim_LGA_State that connects to Dim_LGA and another connected directly to Fact_Crashes) because Tableau does not permit loops in relationship paths. In the actual database, these represent the same physical table.

Schema Diagram from Tablau with connected GeoJSON (Geospacial Data)



Note: Tableau implementation with 2 instances of Dim_State as mentioned in note above.

Characteristics of Schema Design

1. Normalised Components

The majority of our dimension tables are designed in accordance with normalisation principles. Specifically, the crash-dependent dimensions (Dim_Event, Dim_Road, Dim_Vehicle, and Dim_Time) maintain a one-to-one relationship with the Fact_Crashes table, ensuring data integrity and minimising redundancy. Each of these dimension tables directly relates to a unique crash event, avoiding the need to repeat crash-specific information.

2. Partial Normalisation in Dim_Person

However, the `Dim_Person` table deviates from full normalisation. It exhibits a transitive dependency between the `age_group` and `age` attributes. This violates the third normal form (3NF), as `age_group` is functionally dependent on `age` rather than the primary key `person_id`.

To adhere to 3NF, the table could be restructured as follows: `Dim_Person (person_id, crash_id, gender, age, road_user)` `Dim_Age_Groups (age, age_group)`

This separation would eliminate redundancy and potential update anomalies, where an age group change would require multiple updates if age groups were not in a separate table.

3. Snowflake Pattern in Geographic Dimensions

The snowflake pattern is distinctly observed in the geographic dimensions, specifically in the relationship between `Dim_State` and `Dim_LGA`. Each crash in the `Fact_Crashes` table is associated with a state via `state_id`. While all crashes are linked to a state, only a subset are further associated with a specific local government area (LGA) via `lga_id`.

To avoid redundant storage of state information at the LGA level, we've implemented a snowflake structure. `Dim_LGA` maintains a foreign key `state_id` linking it to `Dim_State`. This ensures that state information is stored only once, and can be joined to both crash level data and LGA level data. This structure optimises storage and simplifies updates to state-level information.

Justificaiton for Schema Design

The chosen design schema is grounded in several key principles:

1. Efficient Geographic Data Handling

The snowflake pattern is particularly evident in the geographic dimensions, specifically the relationship between `Dim_State` and `Dim_LGA`. This structure is crucial for our dataset due to the hierarchical nature of geographic data. Each crash is associated with a state, and optionally, a more granular Local Government Area (LGA). By normalising state information into `Dim_State` and linking it to `Dim_LGA` via `state_id`, we avoid redundant storage of state details. This approach is consistent with the principles outlined by Kimball and Ross [1] (#8-references), who advocate for snowflaking in scenarios with well-defined hierarchies. This design facilitates efficient querying and analysis at both state and LGA levels, crucial for spatial analysis and reporting.

There are efficiencies with linking the GeoJSON data to `Dim_LGA` via `lga_name` because then we only dig into that geospatial data table if queried. However, this does make these queries slower. Normalisation was preferred over query efficiency because this saved a lot of storage and we don't expect to use the geospatial data that much.

2. Storage Optimisation

Storing state information once in `Dim_State` significantly reduces storage requirements, especially considering the potential volume of crash data. Adamson [2] (#8-references) highlights the storage efficiency benefits of snowflaking, particularly for large dimension tables with low-cardinality attributes like state names.

3. Improved Data Maintenance

Separating `Dim_State` from `Dim_LGA` simplifies data maintenance. If state-level information changes, we only need to update it in one location, ensuring data consistency across the warehouse. Golfarelli and Rizzi [3] (#8-references) emphasize the ease of maintenance in normalised dimensions, especially when source data is subject to change.

4. Partial Normalisation and Practical Considerations

While most of the schema adheres to normalisation principles, the `Dim_Person` table exhibits a 3NF violation due to the transitive dependency between `age` and `age_group`. Ideally, we would separate these attributes into `Dim_Person` and `Dim_Age_Groups` tables. However, in practice, this deviation represents a balance between strict normalisation and performance considerations. In our specific use case, the potential performance overhead of additional joins may outweigh the benefits of full 3NF compliance. This decision aligns with the pragmatic approach discussed by Imhoff et al. [4] (#8-references), who acknowledge that data warehouse design often involves trade-offs.

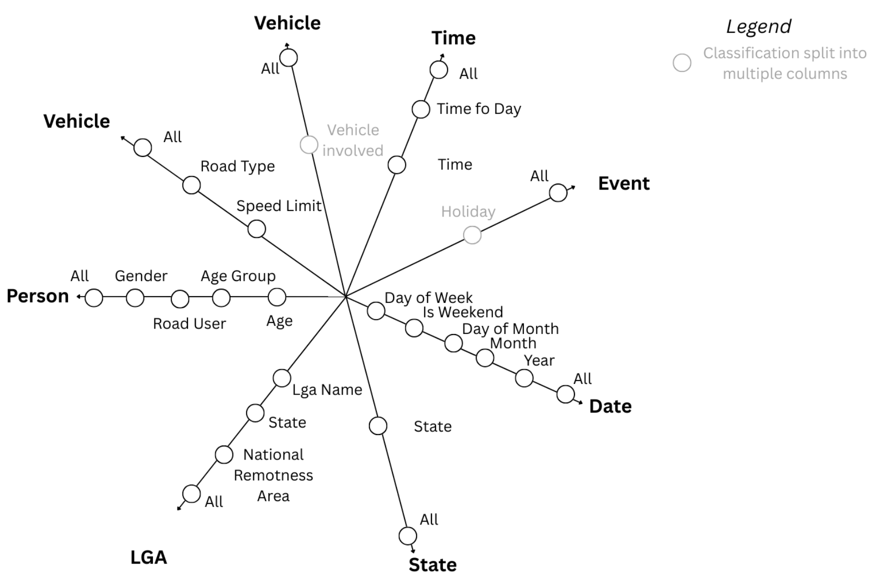
5. Support for Analytical Flexibility

The snowflake design, despite its partial normalisation, offers significant analytical flexibility. It allows users to query data at various levels of granularity, from state-level summaries to detailed LGA analyses. This flexibility is essential for comprehensive reporting and data exploration.

2.2 StarNet Diagram

2.2.1 Visualising the StarNet

StarNet Model



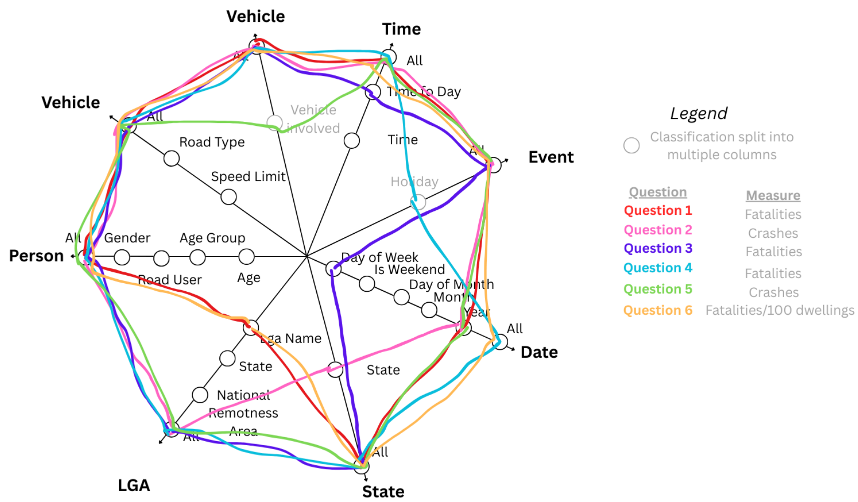
Note: Grey nodes are classification attributes that are split up into multiple columns in the datawarehouse. Upon reflection these should have been reduced to one multi-

valued column during ETL for simplifications during visualisations

2.2.2 Query Footprints

Note: the questions behind queries will be discussed in [5.1 Overview of Business Questions (#51-Overview-of-Business-Questions)]

StarNet Query Footprints



3. Data Preparation and ETL

In this section, we describe and reason through key components of the ELT process.

For a complete explanation of the ETL workflow implemented in `/scripts/etl_process.py` , refer to the working notebook at `/working_notebooks/ETL_Explained.ipynb` . This notebook provides a step-by-step walkthrough of the entire process.

To avoid redundancy, **we've chosen not to include code screenshots here**, as the full notebook is accessible and its use is encouraged for a deeper understanding.

3.1 Extract

The extraction phase involves reading data from various sources:

- `bitre_fatalities_dec2024.xlsx` : Contains fatality and fatality count data.
- `bitre_fatal_crashes_dec2024.xlsx` : Contains crash and crash count data.
- `LGA (count of dwellings).csv` : Contains local government area dwelling counts.

We use `pandas` to read these files into DataFrames, specifying sheet names and skipping unnecessary rows.

```
import pandas as pd
import os

data_dir = os.path.join("data", "raw")
fatalities_file = os.path.join(data_dir,
"bitre_fatalities_dec2024.xlsx")
crashes_file = os.path.join(data_dir,
"bitre_fatal_crashes_dec2024.xlsx")
dwellings_file = os.path.join(data_dir, "LGA (count of
dwellings).csv")

fatality_df = pd.read_excel(fatalities_file,
sheet_name="BITRE_Fatality", skiprows=4)
fatality_count_df = pd.read_excel(fatalities_file,
sheet_name="BITRE_Fatality_Count_By_Date", skiprows=2)
crash_df = pd.read_excel(crashes_file,
sheet_name="BITRE_Fatal_Crash", skiprows=4)
crash_count_df = pd.read_excel(crashes_file,
sheet_name="BITRE_Fatal_Crash_Count_By_Date", skiprows=2)
dwelling_df = pd.read_csv(dwellings_file, skiprows=7,
header=None, names=["lga_name", "dwelling_count", "extra"],
usecols=["lga_name",
"dwelling_count"]).iloc[2:-5].reset_index(drop=True)
```

3.2 Transform

The transformation phase includes several key steps:

Data Cleaning:

- *Merging DataFrames*: We merge fatality_df and crash_df on Crash ID to create a comprehensive DataFrame.
- Column Name Cleaning: We remove newline characters and handle duplicate column names.
- *Duplicate Column Handling*: We identify and remove or rename duplicate columns, accounting for _x and _y suffixes resulting from merges.
- *Data Type Adjustments*: We adjust data types as needed, such as converting date strings to datetime objects and ensuring numeric columns are properly formatted.
- *Handling Missing/Invalid Data*:
  - Invalid values (e.g., -9, "Undetermined") are converted to NaN using numpy.nan.
  - Missing data is considered when designing relationships and tables, ensuring that dimension and fact tables can handle NULL values where applicable.
  - We use dropna() to remove rows with critical missing values (e.g., lga_name).

```
import numpy as np

# Merge and clean crash and fatality dataframes
crashxfatality_df = fatality_df.merge(crash_df, on="Crash ID",
how="left").reset_index(drop=True)
cleaned_cols = crashxfatality_df.columns.str.replace("n", "",
regex=False)
seen = {}
final_cols = []

# ... (column cleaning logic) ...

crashxfatality_df = crashxfatality_df.loc[:,
crashxfatality_df.columns.notna()]

# ... (duplicate column handling logic) ...

crashxfatality_df.columns = [col[:-2] if col.endswith('_y') else
col for col in crashxfatality_df.columns]
crashxfatality_df = crashxfatality_df.drop(columns=['Time of
day'])

# Merge and clean crashxfatality_count_df
crashxfatality_count_df = fatality_count_df.merge(crash_count_df,
on="Date", how="left").loc[:,
~crashxfatality_count_df.columns.duplicated()].reset_index(drop=True

# Create Dimension Tables
def create_dim_date(df, date_col):
    df['date_id'] = pd.to_datetime(df[date_col])
    # ... (date dimension creation logic) ...
    return df[['date_id', 'year', 'month', 'day', 'day_of_week',
'is_weekend']].drop_duplicates()

dim_date = create_dim_date(crashxfatality_count_df,
"Date").reset_index(drop=True)
dim_state =
crashxfatality_df[["State"]].drop_duplicates().rename(columns=
{"State": "state_id"})
# ... (state dimension creation logic) ...
dim_lga = crashxfatality_df[["National LGA Name 2021", "State",
"National Remoteness Areas"]].drop_duplicates().merge(dim_state,
left_on="State", right_on="state_id", how="left").rename(columns=
{"National LGA Name 2021": "lga_name", "National Remoteness
Areas": "national_remoteness_area"})
# ... (lga dimension creation logic) ...
dim_time = crashxfatality_df[["Crash ID", "Time", "Time of
Day"]].rename(columns={"Crash ID": "crash_id", "Time":
"crash_time", "Time of Day":
"time_of_day"}).drop_duplicates().reset_index(drop=True)
dim_vehicle = crashxfatality_df[["Crash ID", "Bus Involvement",
"Heavy Rigid Truck Involvement", "Articulated Truck
Involvement"]].rename(columns={"Crash ID":
"crash_id"}).drop_duplicates().replace(-9,
np.nan).reset_index(drop=True)
dim_person = crashxfatality_df[["Crash ID", "Gender", "Age", "Age
```

```
Group", "Road User"]].rename(columns={"Crash ID": "crash_id",
"Age Group": "age_group", "Road User":
"road_user"}).drop_duplicates()
# ... (person dimension creation logic) ...
dim_event = crashxfatality_df[["Crash ID", "Christmas Period",
"Easter Period"]].rename(columns={"Crash ID":
"crash_id"}).drop_duplicates().reset_index(drop=True)
dim_road = crashxfatality_df[["Crash ID", "Speed Limit",
"National Road Type"]].rename(columns={"Crash ID": "crash_id",
"Speed Limit": "speed_limit", "National Road Type":
"national_road_type"}).drop_duplicates()
dim_road["speed_limit"] = pd.to_numeric(dim_road["speed_limit"],
errors='coerce').astype("Int64")
dim_road["national_road_type"] =
dim_road["national_road_type"].replace("Undetermined", pd.NA)
dim_road["speed_limit"] = dim_road["speed_limit"].replace(-9,
np.nan).reset_index(drop=True)

# Create Fact Tables
# ... (fact table creation logic) ...
```

3.3 Load

The load phase involves writing the transformed DataFrames into PostgreSQL tables:

- *Database Connection*: We establish a connection to the PostgreSQL database using sqlalchemy.
- *Table Creation*: We use df.to_sql() to create and populate tables, replacing existing tables if necessary.
- *Primary and Foreign Key Constraints*: We add primary and foreign key constraints to enforce data integrity and relationships.

```
from sqlalchemy import create_engine, text

DATABASE_URL =
"postgresql://postgres:postgres@pgdb:5432/datawarehouse"
engine = create_engine(DATABASE_URL)

tables = {
    "dim_date": dim_date,
    "dim_state": dim_state,
    "dim_lga": dim_lga,
    "dim_time": dim_time,
    "dim_vehicle": dim_vehicle,
    "dim_person": dim_person,
    "dim_event": dim_event,
    "dim_road": dim_road,
    "fact_fatalities": fact_fatalities,
    "fact_crashes": fact_crashes,
    "fact_number": fact_number
}

for table_name, df in tables.items():
```

```
    df.to_sql(table_name, engine, index=False,
if_exists='replace')

with engine.connect() as connection:
    # ... (primary and foreign key constraint logic) ...
```

4. Database Implementation

4.1 Containerised Setup

1. **PostgreSQL Container** - Hosts the data warehouse with automatic initialization using environment variables for credentials. Maps container port 5432 to host port 5433 to avoid conflicts.
2. **pgAdmin Container** - Provides web-based database management accessible at `http://localhost:5051` with preconfigured admin credentials.
3. **ETL Container** - Custom-built Python service that runs the transformation pipeline on startup.

4.2 Virtual Environment Isolation

The ETL container creates an isolated Python environment:

1. A dedicated virtual environment is created at `/app/venv`
2. All dependencies are installed from `requirements.txt` into this venv
3. The PATH is modified to ensure only venv packages are used
4. This guarantees identical package versions across all deployments

4.3 Database Access

The system provides multiple access methods:

pgAdmin Web Interface

- **URL**: http://localhost:5051 (http://localhost:5051)
- **Login**: admin@admin.com / root
- **Server Connection**:
  - Host: `pgdb` (container name)
  - Port: `5432`
  - Username: `postgres`
  - Password: `postgres`

External Applications (Tableau/Power BI)

- **Host**: localhost
- **Port**: 5433 (mapped from container's 5432)
- **Database**: datawarehouse
- **Credentials**: postgres / postgres

4.4 Reproducibilty Features

The one-command setup ensures identical environments:

```
docker-compose up --build
```

Note: Ensure you are in the `/Project1_Fatalities` directory

This Command:

1. Builds the ETL service image with locked dependencies
2. Creates the PostgreSQL container with empty database
3. Initialises pgAdmin with admin credentials
4. Automatically runs the ETL process to:
    - Create all database tables
    - Establish relationships
    - Load transformed data
5. Preserves data between runs via Docker volumes

5. Business Queries and Insights

5.1 Overview of Business Questions

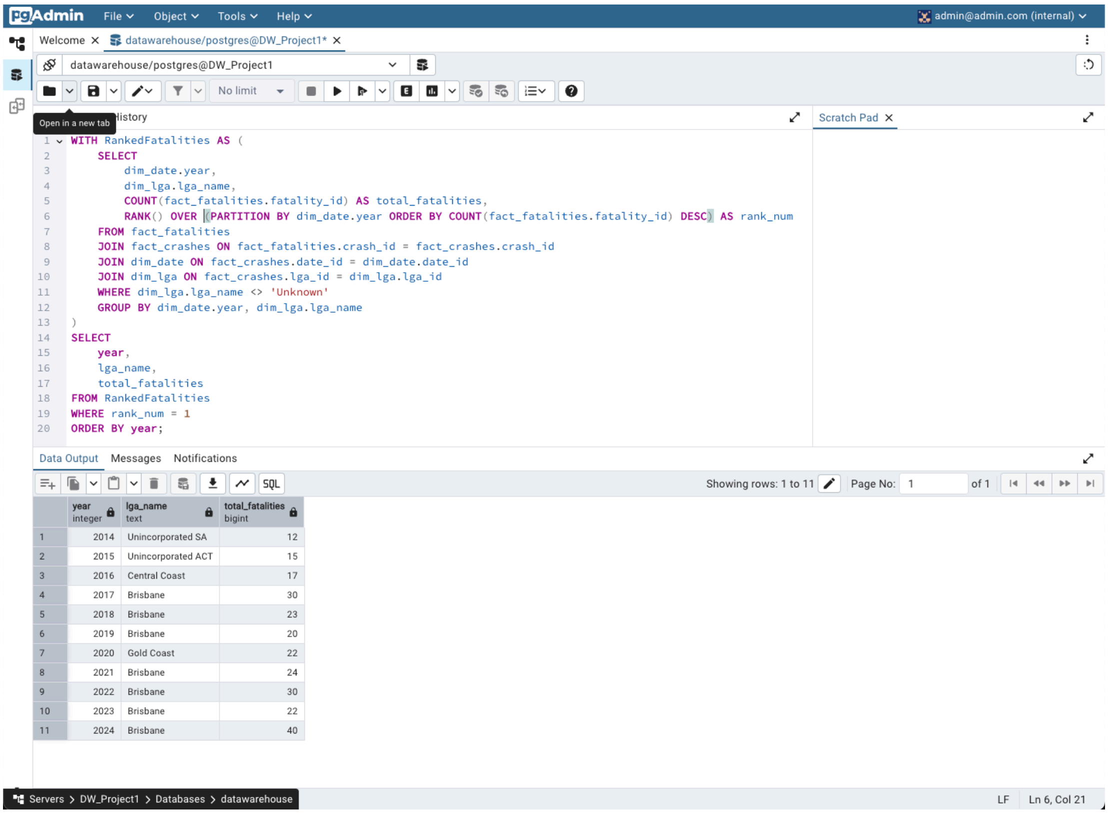
We developed a set of questions designed to explore various aspects of the data warehouse, leveraging different hierarchies across multiple dimensions:

1. Which local government areas had the most road fatalities each year?
2. Which state had the most crashes in 2023?
3. What time of day and days of the week are associated with the highest number of fatalities?
4. How many fatalities occurred during the Christmas and Easter holiday periods?
5. Which types of vehicles are most commonly involved in fatal crashes by year in years after 2010?
6. (Bonus) - Average fatalities per 1000 dwelling for local government areas

5.2 SQL Queries

A full list of queries can be found at `/scripts/queries.sql`. The following is those queries applied to the datawarehouse in pgAdmin.

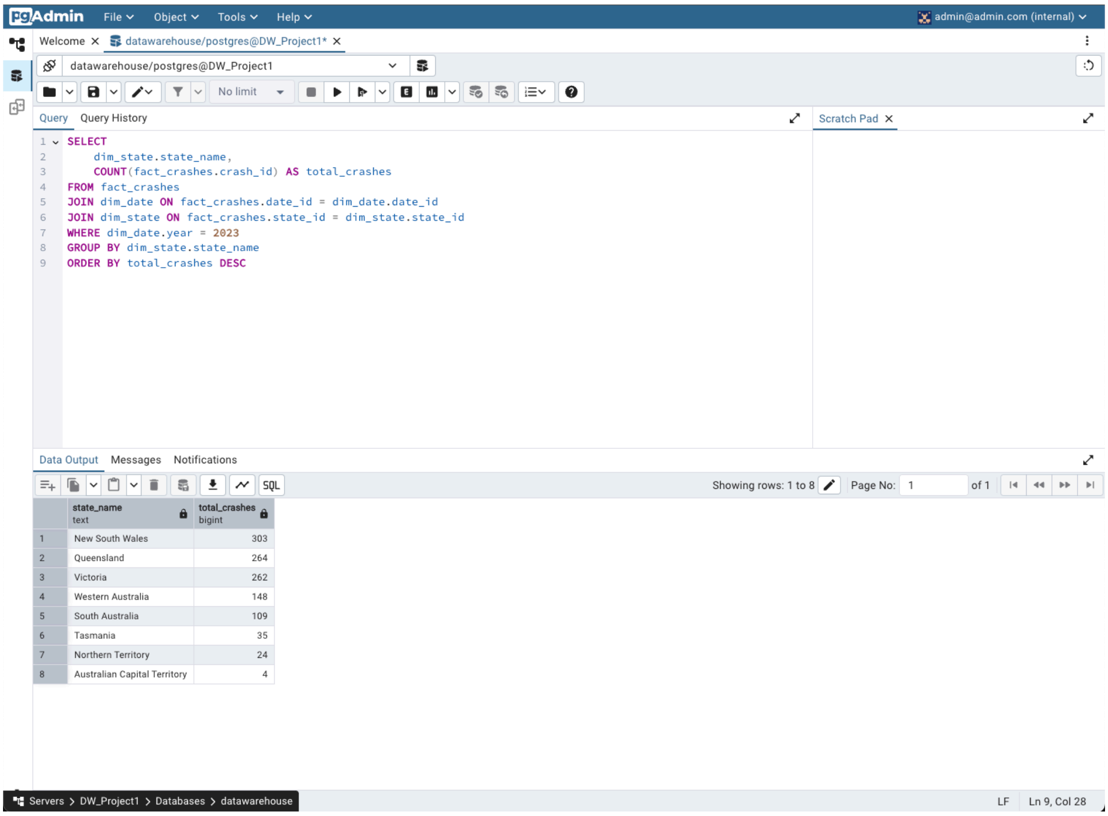
Question 1 : Which local government areas had the most road fatalities each year?



```sql
WITH RankedFatalities AS (
    SELECT
        dim_date.year,
        dim_lga.lga_name,
        COUNT(fact_fatalities.fatality_id) AS total_fatalities,
        RANK() OVER (PARTITION BY dim_date.year ORDER BY COUNT(fact_fatalities.fatality_id) DESC) AS rank_num
    FROM fact_fatalities
    JOIN fact_crashes ON fact_fatalities.crash_id = fact_crashes.crash_id
    JOIN dim_date ON fact_crashes.date_id = dim_date.date_id
    JOIN dim_lga ON fact_crashes.lga_id = dim_lga.lga_id
    WHERE dim_lga.lga_name <> 'Unknown'
    GROUP BY dim_date.year, dim_lga.lga_name
)
SELECT
    year,
    lga_name,
    total_fatalities
FROM RankedFatalities
WHERE rank_num = 1
ORDER BY year;
```

	year integer	lga_name text	total_fatalities bigint
1	2014	Unincorporated SA	12
2	2015	Unincorporated ACT	15
3	2016	Central Coast	17
4	2017	Brisbane	30
5	2018	Brisbane	23
6	2019	Brisbane	20
7	2020	Gold Coast	22
8	2021	Brisbane	24
9	2022	Brisbane	30
10	2023	Brisbane	22
11	2024	Brisbane	40

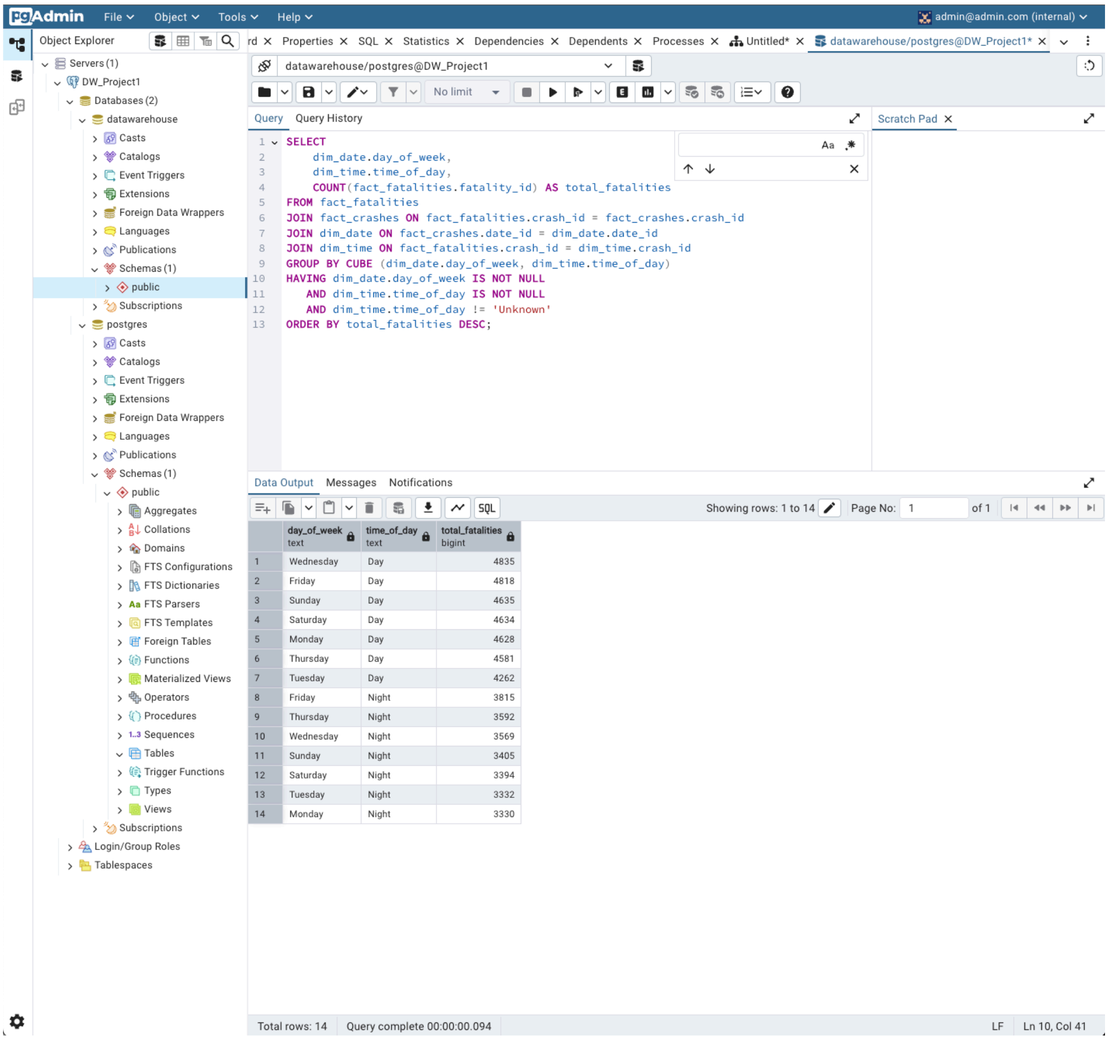
Question 2: Which state had the most crashes in 2023?



```sql
SELECT
    dim_state.state_name,
    COUNT(fact_crashes.crash_id) AS total_crashes
FROM fact_crashes
JOIN dim_date ON fact_crashes.date_id = dim_date.date_id
JOIN dim_state ON fact_crashes.state_id = dim_state.state_id
WHERE dim_date.year = 2023
GROUP BY dim_state.state_name
ORDER BY total_crashes DESC
```

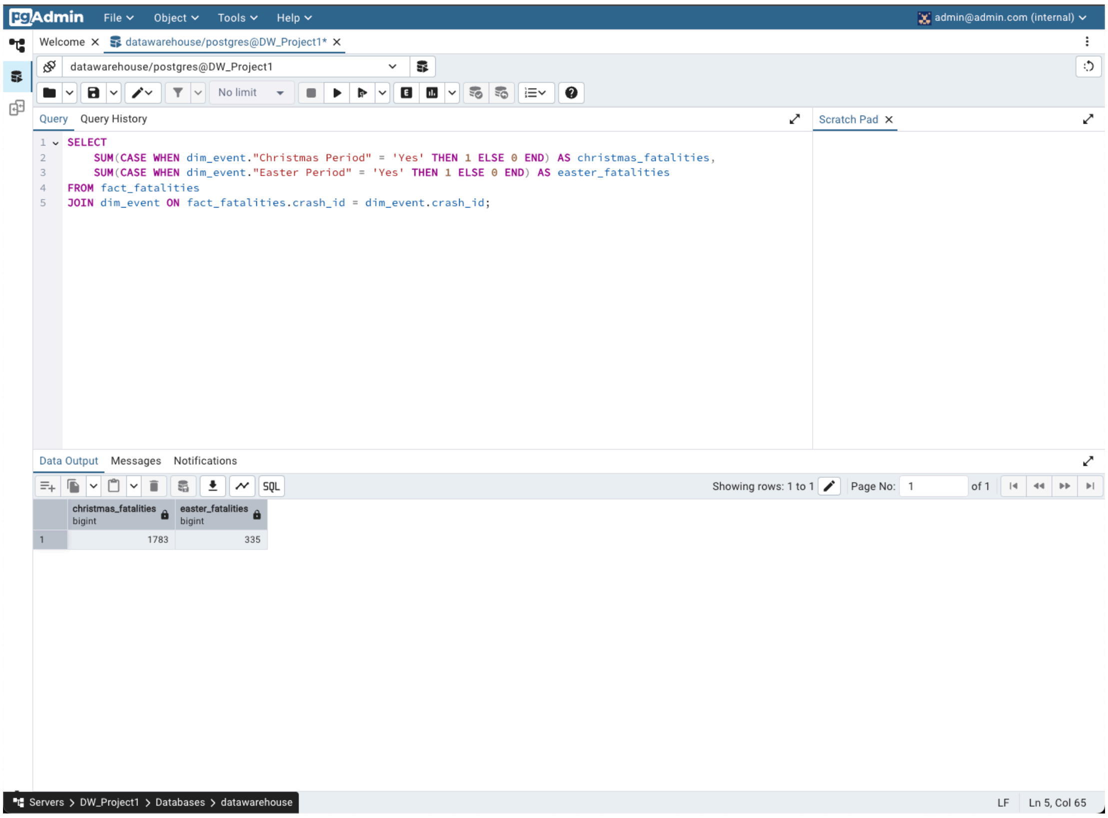
	state_name text	total_crashes bigint
1	New South Wales	303
2	Queensland	264
3	Victoria	262
4	Western Australia	148
5	South Australia	109
6	Tasmania	35
7	Northern Territory	24
8	Australian Capital Territory	4

Question 3: What time of day and days of the week are associated with the highest number of fatalities?

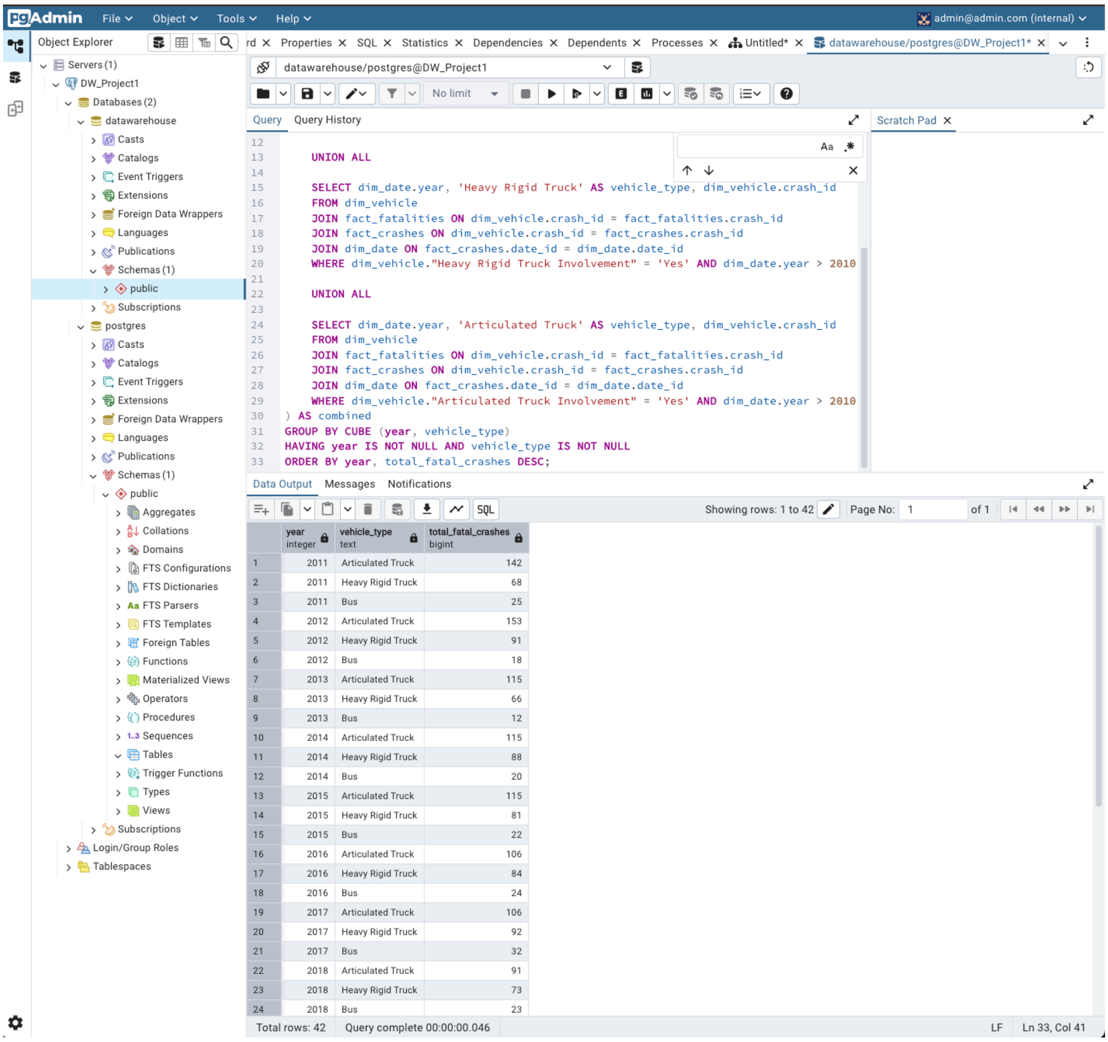


Note the use of the CUBE function to group by both day_of_week and time_of_day. This allows for calculating fatalities by each (day, time) pair, as well as totals for each day, each time, and a grand total, all in one query. This improves efficiency by eliminating the need for multiple GROUP BY queries and UNIONs.

Question 4: How many fatalities occurred during the Christmas and Easter holiday periods?

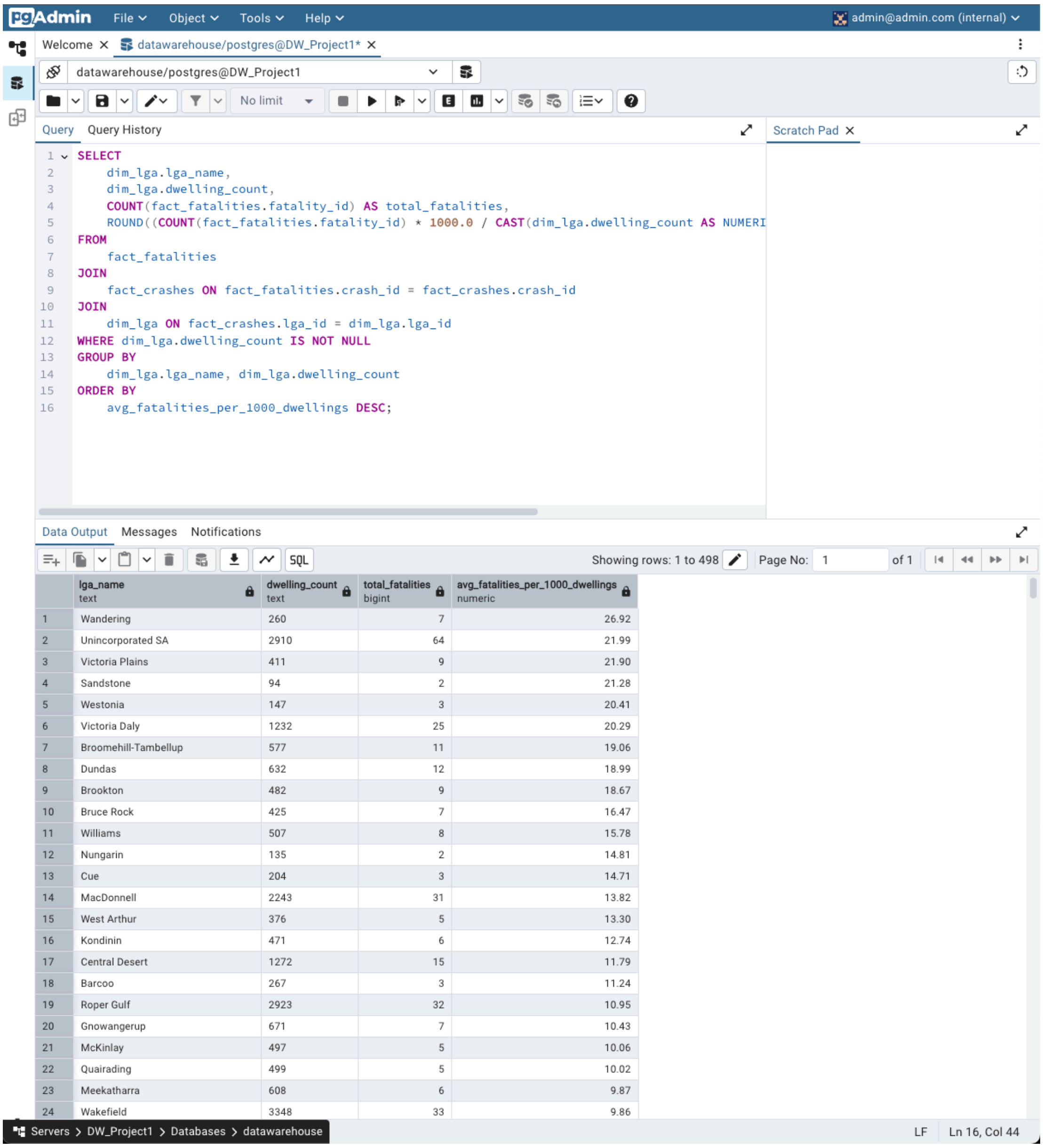


Question 5: Which types of vehicles are most commonly involved in fatal crashes by year in years after 2010?



The CUBE function is used to group by both year and vehicle_type, providing counts for each vehicle type per year, along with yearly and overall totals. This simplifies the process, avoiding multiple queries with UNIONs and offering a more scalable solution if additional vehicle types are added in the future.

Question 6: Average fatalities per 1000 dwelling for local government areas



```sql
SELECT
    dim_lga.lga_name,
    dim_lga.dwelling_count,
    COUNT(fact_fatalities.fatality_id) AS total_fatalities,
    ROUND((COUNT(fact_fatalities.fatality_id) * 1000.0 / CAST(dim_lga.dwelling_count AS NUMERI
FROM
    fact_fatalities
JOIN
    fact_crashes ON fact_fatalities.crash_id = fact_crashes.crash_id
JOIN
    dim_lga ON fact_crashes.lga_id = dim_lga.lga_id
WHERE dim_lga.dwelling_count IS NOT NULL
GROUP BY
    dim_lga.lga_name, dim_lga.dwelling_count
ORDER BY
    avg_fatalities_per_1000_dwellings DESC;
```

Showing rows: 1 to 498 Page No: 1 of 1

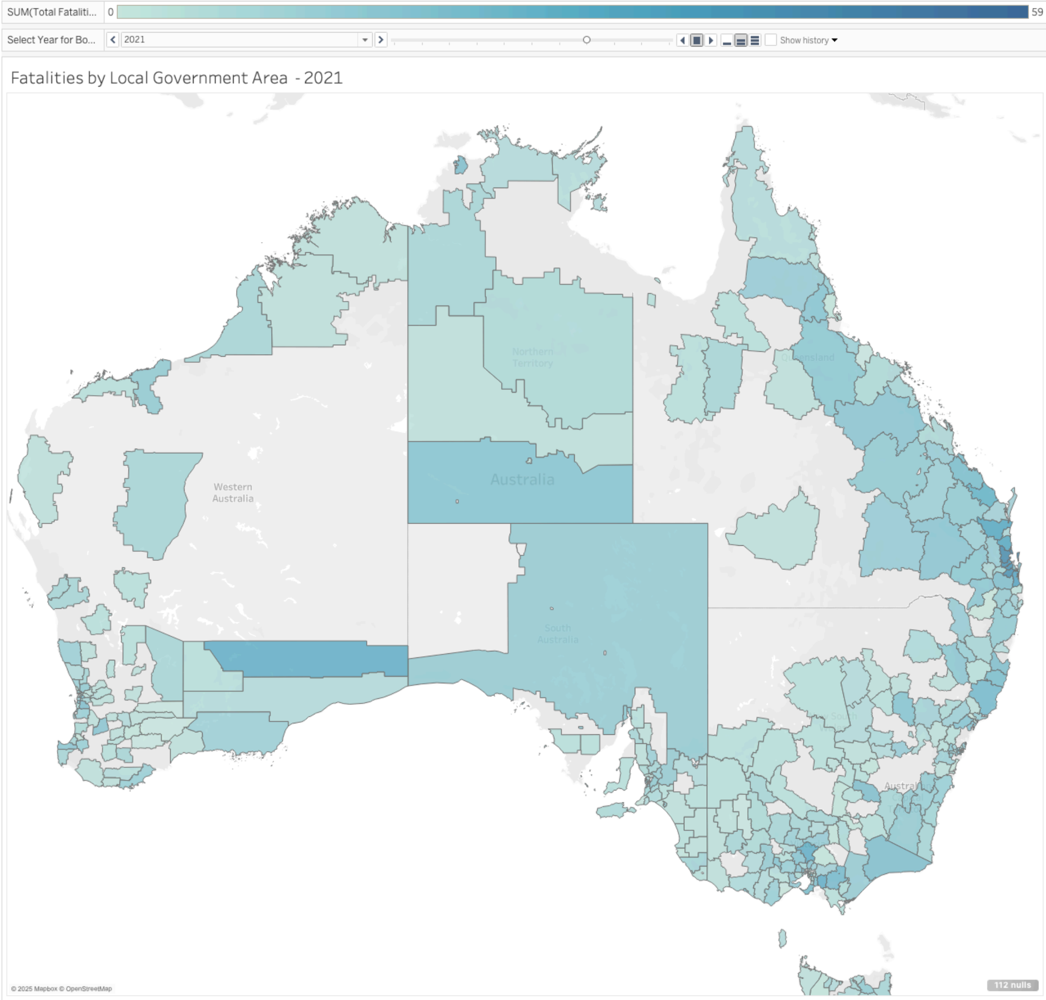
	lga_name text	dwelling_count text	total_fatalities bigint	avg_fatalities_per_1000_dwellings numeric
1	Wandering	260	7	26.92
2	Unincorporated SA	2910	64	21.99
3	Victoria Plains	411	9	21.90
4	Sandstone	94	2	21.28
5	Westonia	147	3	20.41
6	Victoria Daly	1232	25	20.29
7	Broomehill-Tambellup	577	11	19.06
8	Dundas	632	12	18.99
9	Brookton	482	9	18.67
10	Bruce Rock	425	7	16.47
11	Williams	507	8	15.78
12	Nungarin	135	2	14.81
13	Cue	204	3	14.71
14	MacDonnell	2243	31	13.82
15	West Arthur	376	5	13.30
16	Kondinin	471	6	12.74
17	Central Desert	1272	15	11.79
18	Barcoo	267	3	11.24
19	Roper Gulf	2923	32	10.95
20	Gnowangerup	671	7	10.43
21	McKinlay	497	5	10.06
22	Quairading	499	5	10.02
23	Meekatharra	608	6	9.87
24	Wakefield	3348	33	9.86

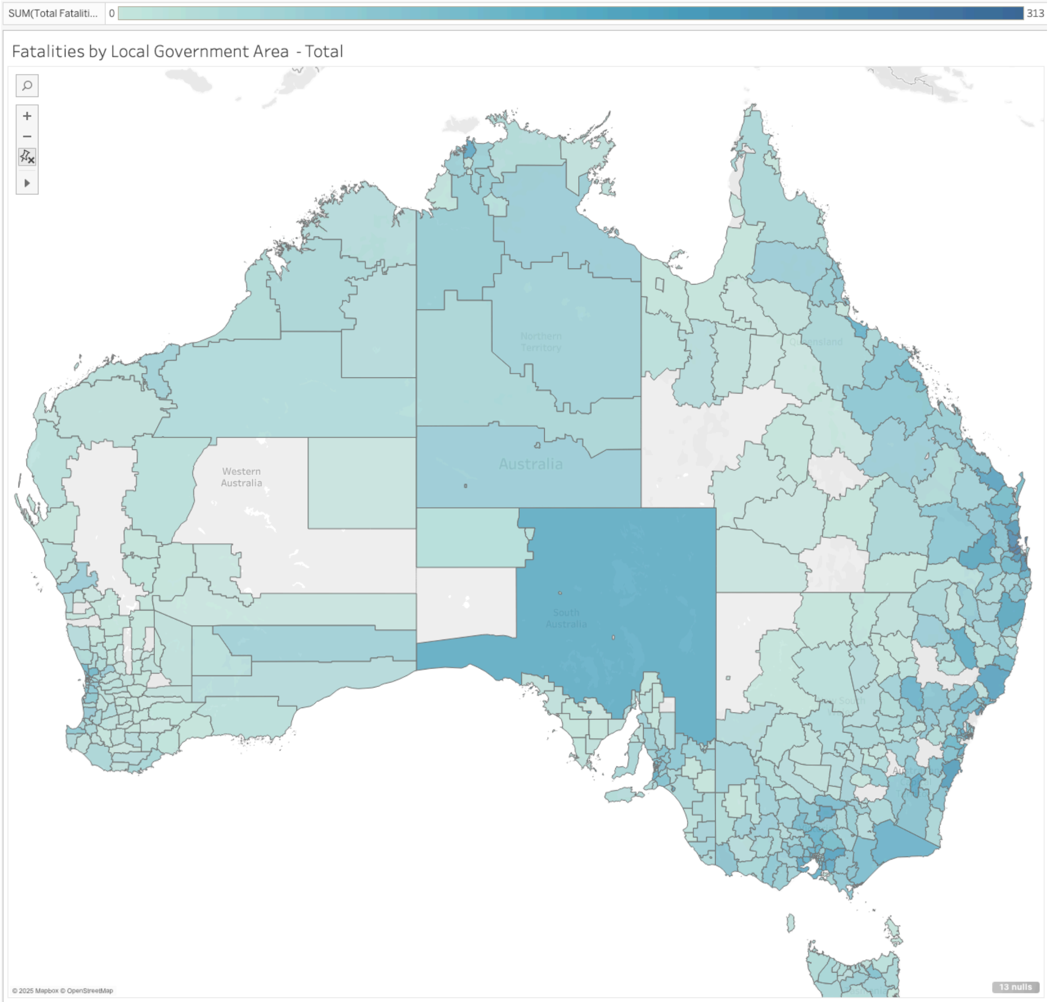
Note this query defines a new, calcualted measure field (average fatalities per 1000 dwellings) across lgs's.

5.3 Visualisation of Queries

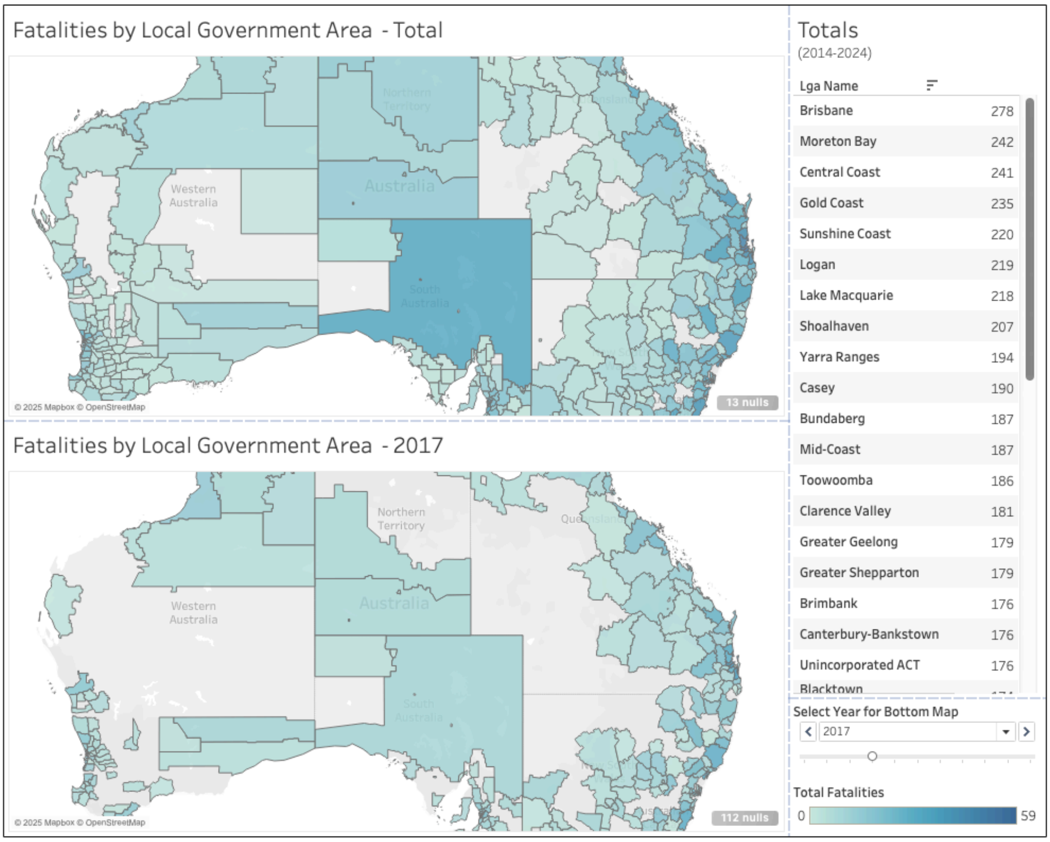
The following is a compilation of Tableau visualisations and dashboards designed to address the key queries. These tools enable us to visualise the data in innovative ways, generating valuable insights that are crucial for informed business decisions.

- **1. Which local government areas had the most road fatalities each year?**

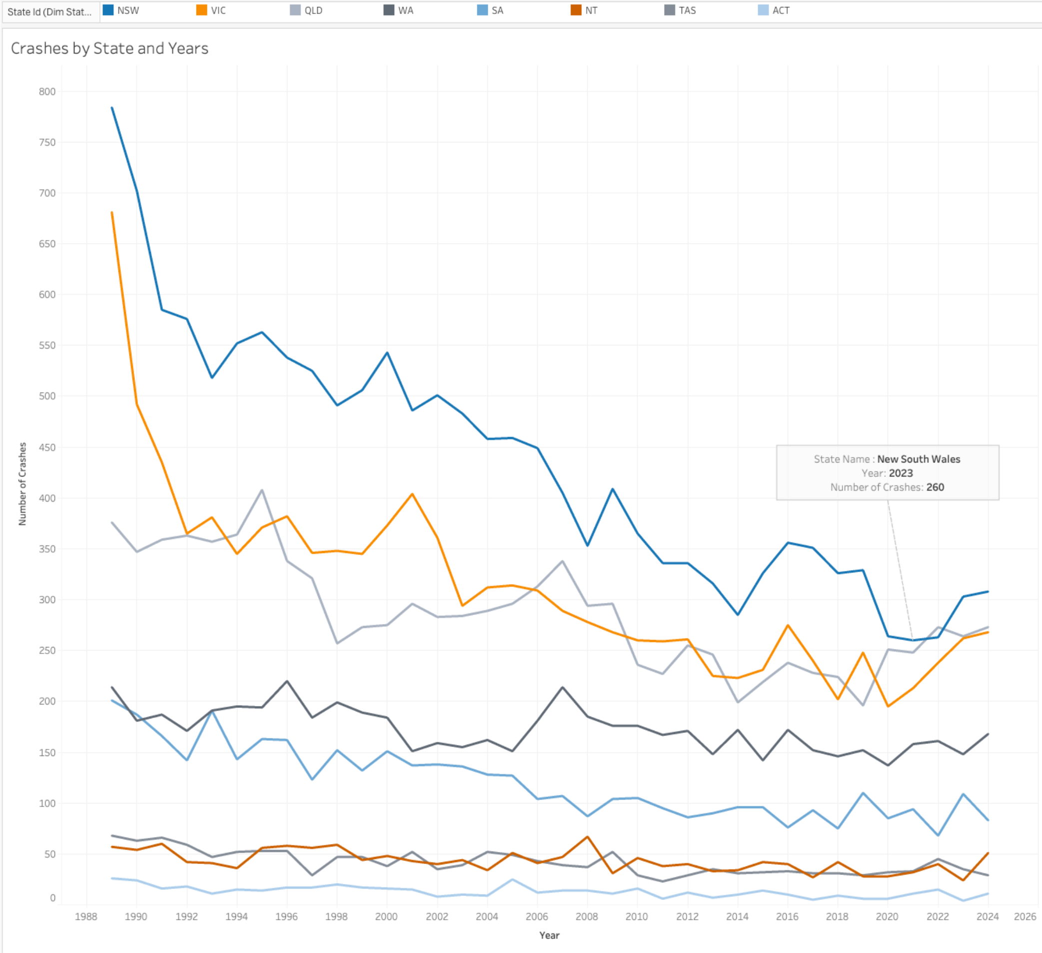



SUM(Total Fataliti...
0
313
Fatalities by Local Government Area - Total
Western Australia
Australia
© 2025 Mapbox © OpenStreetMap
13 nulls

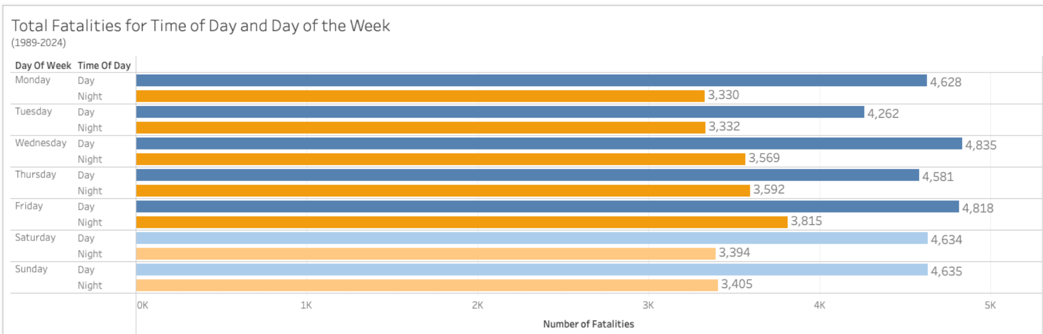
Dashboard:


Fatalities by Local Government Area - Total
Western Australia
Australia
© 2025 Mapbox © OpenStreetMap
13 nulls
Fatalities by Local Government Area - 2017
Northern Territory
Western Australia
Australia
© 2025 Mapbox © OpenStreetMap
112 nulls
Totals
(2014-2024)
Lga Name
Brisbane 278
Moreton Bay 242
Central Coast 241
Gold Coast 235
Sunshine Coast 220
Logan 219
Lake Macquarie 218
Shoalhaven 207
Yarra Ranges 194
Casey 190
Bundaberg 187
Mid-Coast 187
Toowoomba 186
Clarence Valley 181
Greater Geelong 179
Greater Shepparton 179
Brimbank 176
Canterbury-Bankstown 176
Unincorporated ACT 176
Blacktown
Select Year for Bottom Map
2017
Total Fatalities
0
59

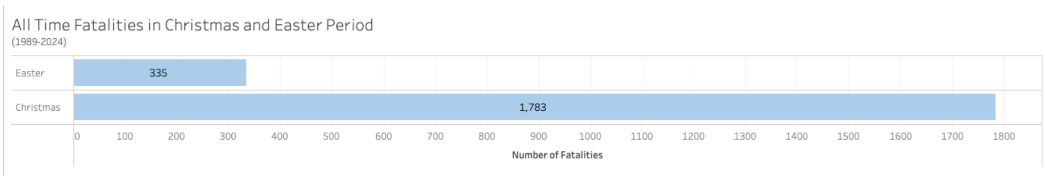
- **2. Which state had the most crashes in 2023?**



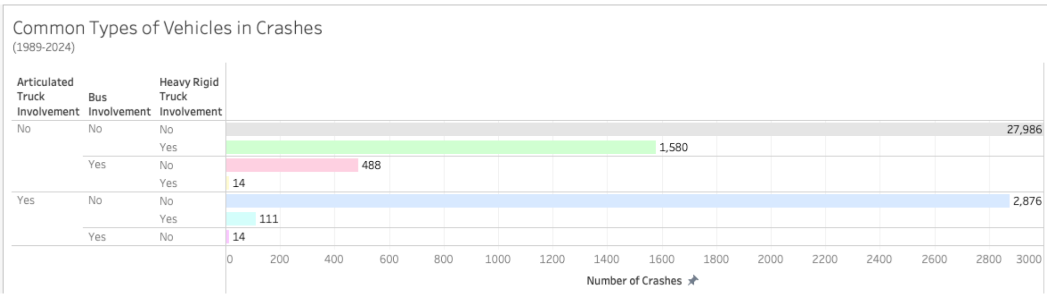
- **3. What time of day and days of the week are associated with the highest number of fatalities?**



- **4. How many fatalities occurred during the Christmas and Easter holiday periods?**



- **5. Which types of vehicles are most commonly involved in fatal crashes by year in years after 2010?**

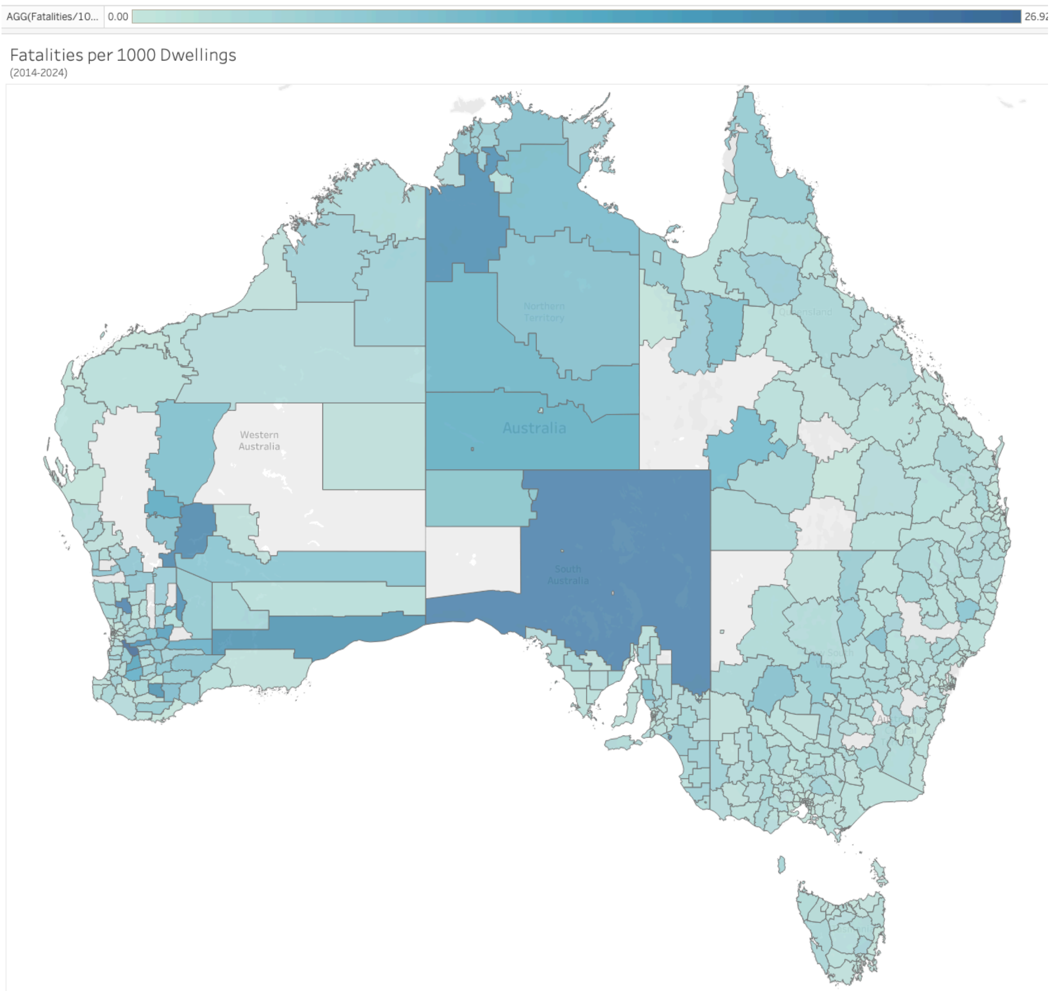

Common Types of Vehicles in Crashes
(1989-2024)
Articulated Truck Involvement
Bus Involvement
Heavy Rigid Truck Involvement
No
Yes
27,986
1,580
488
14
2,876
111
14
0
200
400
600
800
1000
1200
1400
1600
1800
2000
2200
2400
2600
2800
3000
Number of Crashes

- **6. (Bonus) - Average fatalities per 1000 dwelling for local government areas**

Fatalities per 1000 Dwellings
(2014-2024)

Lga Name	
Wandering	26.92
Unincorporated SA	21.99
Victoria Plains	21.90
Sandstone	21.28
Westonia	20.41
Victoria Daly	20.29
Broomehill-Tambellup	19.06
Dundas	18.99
Brookton	18.67
Bruce Rock	16.47
Williams	15.78
Nungarin	14.81
Cue	14.71
MacDonnell	13.82
West Arthur	13.30
Kondinin	12.74
Central Desert	11.79
Barcoo	11.24
Roper Gulf	10.95
Gnowangerup	10.43
McKinlay	10.06
Quairading	10.02
Meekatharra	9.87
Wakefield	9.86
Corrigin	9.84
Mount Magnet	9.76
Barkly	9.55
West Arnhem	9.44
Lake Grace	9.32
Balranald	9.17
Narromine	9.12
Anangu Pitjantjatjara Yun..	9.00
Dumbleyung	8.96
Wickepin	8.96
Lockhart River	8.77
Hope Vale	8.62

Walcha	8.55
Menzies	8.44
Dalwallinu	8.18
Carnamah	8.09



Note: Navigate to `visualisations/query_visualisations.pdf` *to view all visualisations in one document.*

Tableau Visualisations (../visualisations/query_visualisations.pdf)

Note: the third page of this PDF is an interactive dashboard (allowing visualisations of multiple years of fatalities totals per LGA) and can be opened in tableau from the file at `/visualisations/query_workbook.twb` *if the database has been set up as explained in* `README.md`

6. Association Rule Mining

6.1 Methodology

6.1.1 Algorithm Selection

We implemented the *Apriori algorithm* [5] (#8-references) using Python's mlxtend library [6] (#8-references) . The algorithm identifies frequent itemsets through iterative candidate generation and pruning, making it suitable for our dataset of ~50,000 transactions. This approach efficiently handles categorical data while maintaining interpretability for road safety analysis.

6.1.2 Data Preparation

Key preprocessing steps included:

- **Data Cleaning**: Replaced '-9', 'Unknown' and 'Undetermined' values with NaN
- **Feature Engineering**:
  - Binned numerical `Speed Limit` into 4 categories (Low: 0-60km/h, Medium: 61-80km/h, High: 81-100km/h, Very High: 100+km/h)
  - Formatted categorical values as `Feature=Value` pairs
- **Transaction Encoding**: Converted each fatality record into a transaction of co-occurring attributes
- **Column Selection**: Focused on 6 key attributes:
  ```
  ['Road User', 'Age Group', 'Gender', 'Speed Category',
   'Time of Day', 'National Road Type']
  ```

6.1.3 Parameter Selection

Final parameters were determined through iterative testing:

- **Minimum Support**: 0.05 (5% of transactions)
- **Maximum Itemset Length**: 3 items
- **Lift Threshold**: >1.0

These values were chosen to balance rule significance with computational efficiency, filtering out rare patterns while maintaining actionable insights.

6.2 Implementation

The implementation comprises:

- **Script Location**: `/association_rules/association_rule_mining.py`
- **Docker Configuration**:
  ```
  services:
    association_rules:
      build: .
      volumes:
        - ./data:/app/data
        -
  ./association_rules/results:/app/association_rules/results
  ```
- **Reproducibility Steps**:
  1. Clone repository
  2. Run `docker-compose build association_rules`
  3. Execute `docker-compose run --rm association_rules`
- **Output**: Generates `association_rules.csv` in `/association_rules/results`

6.3 Results and Insights

Top 3 Rules (by Lift):

1. **Rule 1**:
   `{Speed Category=Low} → {Time of Day=Day, Road User=Pedestrian}`

   - **Support**: 0.054 (5.4% of transactions)
   - **Confidence**: 0.172 (17.2% accuracy)
   - **Lift**: 2.573
   - **Interpretation**: Pedestrian incidents are 2.57 times more likely than average to occur in low-speed daytime areas.
2. **Rule 2**:
   `{Speed Category=Very High} → {National Road Type=National or State Highway, Road User=Driver}`

   - **Support**: 0.050 (5.0%)
   - **Confidence**: 0.378 (37.8%)
   - **Lift**: 2.334
   - **Interpretation**: High-speed driver incidents on highways occur 2.33 times more frequently than expected.
3. **Rule 3**:
   `{Time of Day=Day, Speed Category=Low} → {Road User=Pedestrian}`

   - **Support**: 0.054 (5.4%)
   - **Confidence**: 0.308 (30.8%)
   - **Lift**: 2.299
   - **Interpretation**: 30.8% of daytime low-speed crashes involve pedestrians, with 2.3x increased likelihood.

Key Insights:

- Pedestrian safety is strongly associated with low-speed zones
- Driver incidents correlate with high-speed highways
- Temporal patterns show daytime predominance for pedestrian incidents
- Lift values indicate non-random associations across all top rules

6.4 Recommendations

1. **Low-Speed Pedestrian Infrastructure**
   Install raised crossings and pedestrian-activated signals in urban areas with frequent low-speed incidents (supported by Rules 1 & 3's lift >2.29).

2. **National Highway Speed Management**
   Implement average-speed cameras and dynamic signage on high-speed corridors (aligned with Rule 2's 37.8% confidence).

3. **Daytime Pedestrian Awareness Programs**
   Develop targeted education campaigns for schools and senior communities (informed by Rules 1 & 3's daytime pattern).

7. Conclusion and Reflection

This project delivered a functional data warehouse for analysing road safety trends, enabling efficient querying and visualisation of crash and fatality data. The snowflake schema design successfully supported complex analytical questions while maintaining data integrity through partial normalisation. Integration with Tableau and Power BI demonstrated the warehouse's practical utility for spatial and temporal analysis, though several key improvements were identified for future iterations.

Several learnings emerged that could refine the design:

- **Classification attributes** like vehicle types (e.g., bus/truck involvement) were stored as separate boolean columns in `Dim_Vehicle`. Consolidating these into a single categorical column (e.g., "vehicle_type") would streamline visualisation workflows in BI tools.
- While most Yes/No fields were converted to booleans (e.g., holiday flags in `Dim_Event`), standardising this conversion earlier in the ETL process would improve consistency.
- The redundancy between `age` and `age_group` in `Dim_Person` could be resolved by fully normalising these into separate tables, trading minor query complexity for better storage efficiency.

These adjustments would better align the schema with dimensional modelling best practices while preserving the snowflake pattern's strengths in handling geographic hierarchies.

Back to top (#cits5504-project-1)

---

8. References

[1] R. Kimball and M. Ross, *The Data Warehouse Toolkit: The Definitive Guide to Dimensional Modeling*, 3rd ed. John Wiley & Sons, 2013.

[2] C. Adamson, *Star Schema: The Complete Reference*. McGraw-Hill, 2010.

[3] M. Golfarelli and S. Rizzi, *Data Warehouse Design: Modern Principles and Methodologies*. McGraw-Hill, 2009.

[4] C. Imhoff, N. Galemmo, and J. G. Geiger, *Mastering Data Warehouse Design: Relational and Dimensional Techniques*. John Wiley & Sons, 2003.

[5] R. Agrawal and R. Srikant, “Fast Algorithms for Mining Association Rules,” Proceedings of the 20th International Conference on Very Large Data Bases (VLDB), 1994.

[6] S. Raschka, “MLxtend: Providing Machine Learning Extensions,” Journal of Open Source Software, vol. 3, no. 24, p. 638, 2018.