SQL Relational Database Demonstration – UK Road Accident Data

Content (text on slide):

•**Student:** Anthony Eddei Kwofie

Programme

MSc Artificial Intelligence and Data Science – University of Hull (London Campus)

•**Dataset:** accident_data_v1.0.0_2023.db

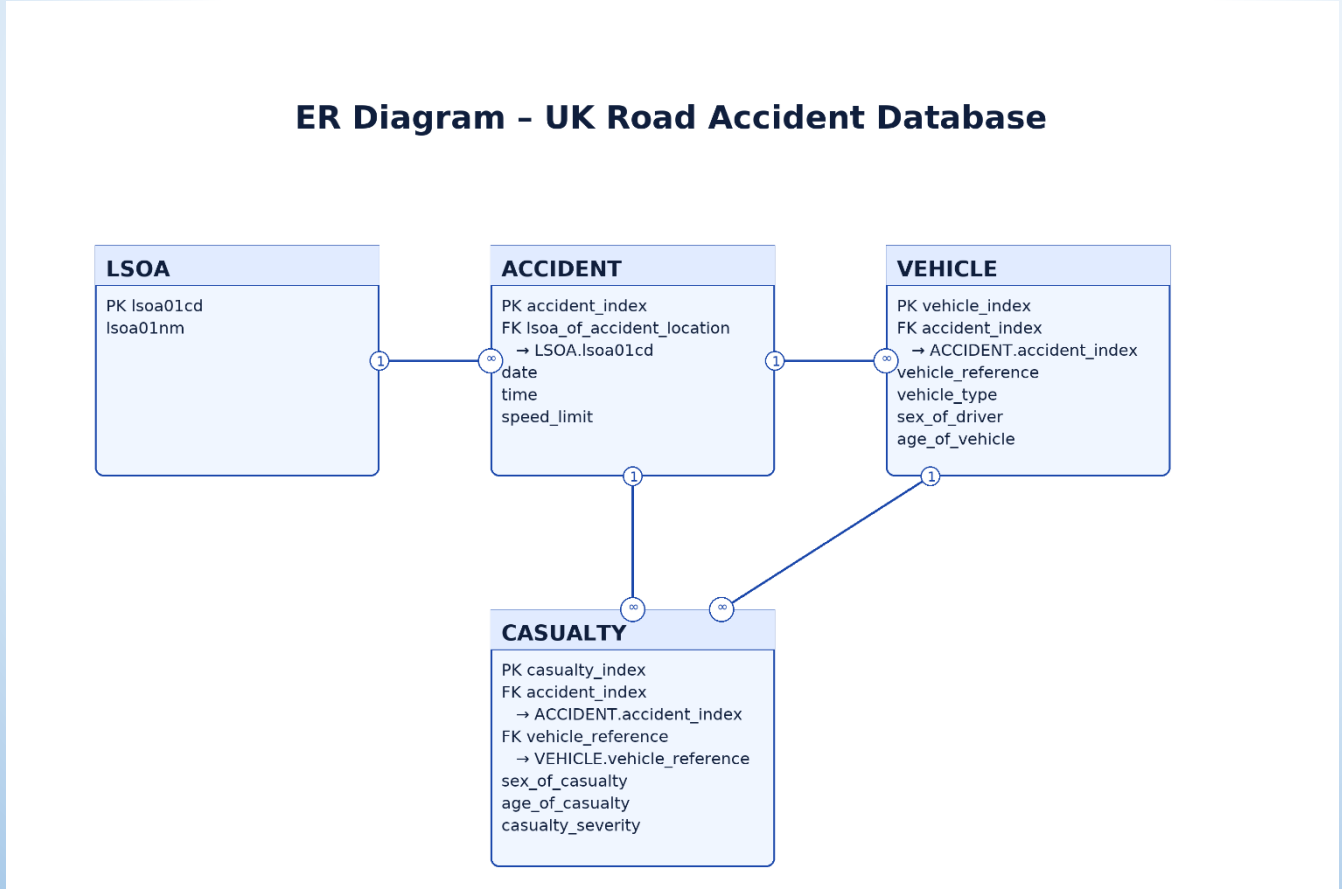
Scenario: Acting as a government data scientist building a secure SQL database for accident data.

Managing Data Responsibly

Technical	Ethical	Legal
Data integrity & security	Protecting individual privacy	GDPR (2018) compliance
Avoiding redundancy	Fair and unbiased data use	Lawful access & retention
Access control & audit trails	Responsible reporting	Data minimization

Entity–Relationship Model of the Accident Database

- Diagram of tables and relationships (simplified version below)
- Primary Keys (PK) and Foreign Keys (FK) clearly labelled



Finding the Oldest Driver or Rider

- Show query and output
- Explain use of aggregate function

Task A – Finding the Age of the Oldest Driver or Rider

I wrote a query to identify the oldest driver or rider recorded in the dataset.

This helped me test the use of an aggregate function in SQL and understand the upper age range of individuals involved in accidents.

[11]:
```
# I used the MAX() aggregate function to find the highest value in the 'age_of_driver' column
# from the 'vehicle' table.

query_a = """
SELECT MAX(age_of_driver) AS oldest_driver
FROM vehicle;
"""

# I executed the query and displayed the result in a pandas DataFrame.
result_a = pd.read_sql_query(query_a, conn)
print(result_a)
```

```
   oldest_driver
0            102
```

Counting Vehicles of Type 19

- Show total count result
- Explain filtering with WHERE

Task B – Counting All Vehicles of Type 19

Next, I wrote a query to count the total number of vehicles classified as type 19 in the database.
This demonstrated my ability to apply filtering conditions using the `WHERE` clause together with the `COUNT()` aggregate function.

[12]:
```
# I used the COUNT() function to calculate the number of records
# in the 'vehicle' table where the 'vehicle_type' equals 19.

query_b = """
SELECT COUNT(*) AS total_vehicle_type_19
FROM vehicle
WHERE vehicle_type = 19;
"""

# I executed the query and displayed the total count of type 19 vehicles.
result_b = pd.read_sql_query(query_b, conn)
print(result_b)
```

```
   total_vehicle_type_19
0                  47458
```

Joining Tables to Retrieve Local Accident Details

- Explain use of multiple joins
- List displayed fields:
- Sex of driver
- Sex of casualty
- Speed limit
- Age of vehicle
- Region name

Task C – Retrieving Accident Details for Kingston upon Hull

Finally, I wrote a more complex SQL query that joined several tables together.
The aim was to extract the sex of the driver, sex of the casualty, speed limit, and age of the vehicle for all accidents that occurred in **Kingston upon Hull**.
This demonstrated my ability to perform multi-table joins and filter data using a text condition.

[13]:
```
# I joined four related tables: accident, vehicle, casualty, and lsoa.
# The joins connected their primary and foreign keys to combine relevant information.

query_c = """
SELECT
    v.sex_of_driver,
    c.sex_of_casualty,
    a.speed_limit,
    v.age_of_vehicle,
    l.lsoa01nm AS region
FROM accident a
JOIN vehicle v ON a.accident_index = v.accident_index
JOIN casualty c ON a.accident_index = c.accident_index
JOIN lsoa l ON a.lsoa_of_accident_location = l.lsoa01cd
WHERE l.lsoa01nm LIKE '%Kingston upon Hull%';
"""

# I executed the query and viewed a sample of the results along with the total number of rows.
df = pd.read_sql_query(query_c, conn)
print(df.head())        # display first few rows
print("Total rows:", len(df))
```

```
   sex_of_driver  sex_of_casualty  speed_limit  age_of_vehicle  \
0              1                1           30              13
1              1                1           30               9
2              3                1           30               6
3              3                2           30               6
4              1                1           30              11

                   region
0  Kingston upon Hull 028C
1  Kingston upon Hull 028C
2  Kingston upon Hull 029A
3  Kingston upon Hull 029A
4  Kingston upon Hull 029A
Total rows: 6387
```

Summary of Findings and Good Practice

- SQL enabled structured and secure data access.
- Relational design reduced redundancy and improved data integrity.
- Queries provided actionable insights for public safety.
- Ethical and legal principles guided the data management process.