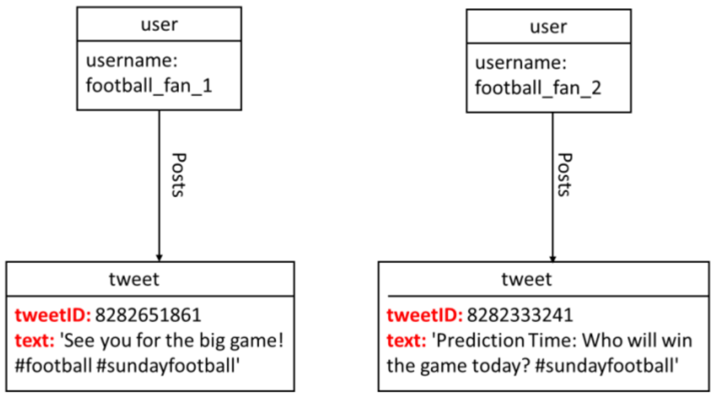
# COMP3008 - Big Data Analytics

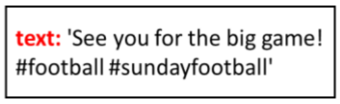
## Workshop 1 - Graph Database Introduction

### Introduction

A graph database stores data in a graph, one of the most generic types of data structures. Here is an example graph that represents tweets collected on a Sunday before a football game. We will approach this graph step by step in the following sections:


user
username: football_fan_1
Posts
tweet
tweetID: 8282651861
text: 'See you for the big game! #football #sundayfootball'
user
username: football_fan_2
Posts
tweet
tweetID: 8282333241
text: 'Prediction Time: Who will win the game today? #sundayfootball'

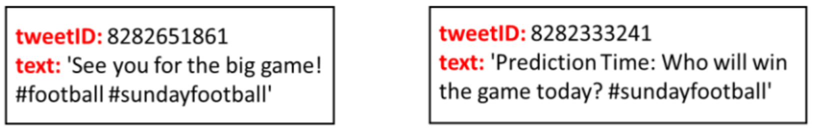
A graph records data in *nodes* and *relationships*. Indeed, the fundamental units that form a graph are nodes and relationships. The simplest possible graph is a single node. A node can have zero or more named values referred to as *properties*. Lets start out with a graph with only one node that has a single property named text.



Relationships provide directed, relevant connections between two nodes. A relationship always has a *direction*, a *type*, a *start node*, and an *end node*. Like nodes, relationships can have many properties. In most cases, relationships have quantitative properties, such as weights, costs, distances, ratings, time intervals, or strengths. As relationships are stored efficiently, two nodes can share any number or type of relationships without sacrificing performance. Note that although they are directed, relationships can always be navigated regardless of direction.

There is one core consistent rule in a graph database: **No broken links**. Since a relationship always has a start and end node, you cannot delete a node without also deleting its associated relationships. You can also always assume that an existing relationship will never point to a non-existing endpoint.

Lets add to the graph in the previous example another node, and one more property on the node (tweetID):

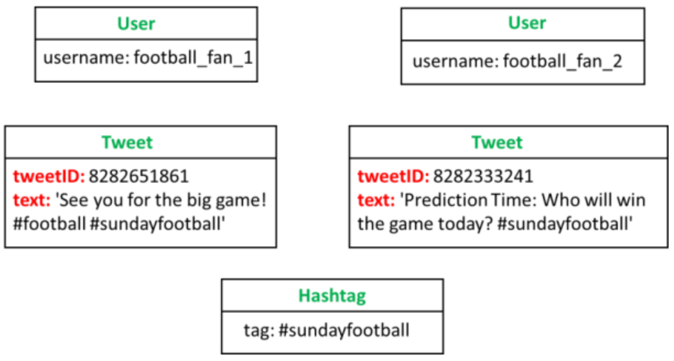

tweetID: 8282651861
text: 'See you for the big game! #football #sundayfootball'
tweetID: 8282333241
text: 'Prediction Time: Who will win the game today? #sundayfootball'

A *label* is a graph construct that is used to group nodes into sets; all nodes labelled with the same label belong to the same set. Many database queries can work with these sets instead of the whole graph, making queries easier to write and more efficient to execute. A node may be labelled with any number of labels, or none, making labels an optional addition to the graph.

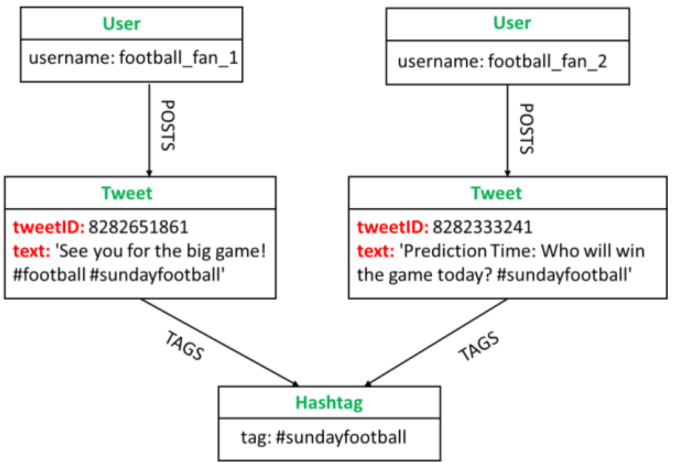
An example would be a label named User, which you can use to label all your nodes representing users. With that in place, you can ask Neo4j to perform operations only on your user nodes, such as finding all users with a given name.

You can also use labels to perform other tasks. For instance, since labels can be added and removed during runtime, they can be used to mark temporary states for your nodes- you might create an Offline label for phones that are offline.

In our example, we will add Tweet, User and Hashtag labels to our graph. Let's add two users-each with a property called username - and one hashtag to the graph. Then, label the nodes using the labels Tweet, User and Hashtag accordingly:



Relationships organize the nodes by connecting them. Relationships organize nodes into arbitrary structures, allowing a graph to resemble a list, a tree, a map, or a compound entity - any of which can be combined into yet more complex, richly inter-connected structures. Our example graph will make a lot more sense once we add relationships to it:



**Exercise - Neo4j: Nodes and Relationships**

The CREATE clause is used in Neo4j to create graph elements - nodes and relationships. Creating a single node is done by issuing the following query.

```
CREATE (n)
```

While this may be useful in some cases, we will typically want to create a node and associate it with a label. Lets create a node that represents a tweet. We can do so by adding the label Tweet to the node that we will create with the following command.

CREATE (n:Tweet)

When creating a new node with labels, you can add properties at the same time. Lets create a node that represents a tweet and has the tweetID and text displayed below:

```
tweetID: ‘8282651861’
text:  ‘See you for the big game!  #football #sundayfootball’
```

The command would be as follows,

```
CREATE (t:Tweet { tweetID: ‘8282651861’, text:  ‘See you for the big game!  #football #sundayfootball’
})
```

Now, lets create a User with the username `football_fan_1`

```
CREATE (u:User { username:  ’football_fan_1’ })
```

Then, we can create a second User, a second Tweet.

```
CREATE (u:User { username:  ‘football_fan_2’ })
```

```
CREATE (t:Tweet { tweetID: ‘8282333241’, text:  ‘Prediction Time:  Who will win the game today?
#sundayfootball’ })
```

Finally, remove all the nodes and relationships created thus far before starting the following section. This can be done with the command,

```
// Delete all nodes
MATCH (n)
DETACH DELETE n
```

Some useful commands and keyboard shortcuts for the Web console are listed below.

| **Shortcut** | **Purpose** |
|---|---|
| :help | Help System |
| :help commands | Useful Commands |
| :clear | Clear Frames |
| :style [reset] | Styling Popup & Reset |
| :help keys | Keyboard Help |
| Ctrl+Enter or<br>Cmd+Enter | Execute Statement |
| Ctrl+Up or Cmd+Up 7 Previous Statement | |
| Shift+Enter | Enter Multiline Mode |
| / | Move Focus to Editor |
| ESC | Toggle Editor to Full Screen |