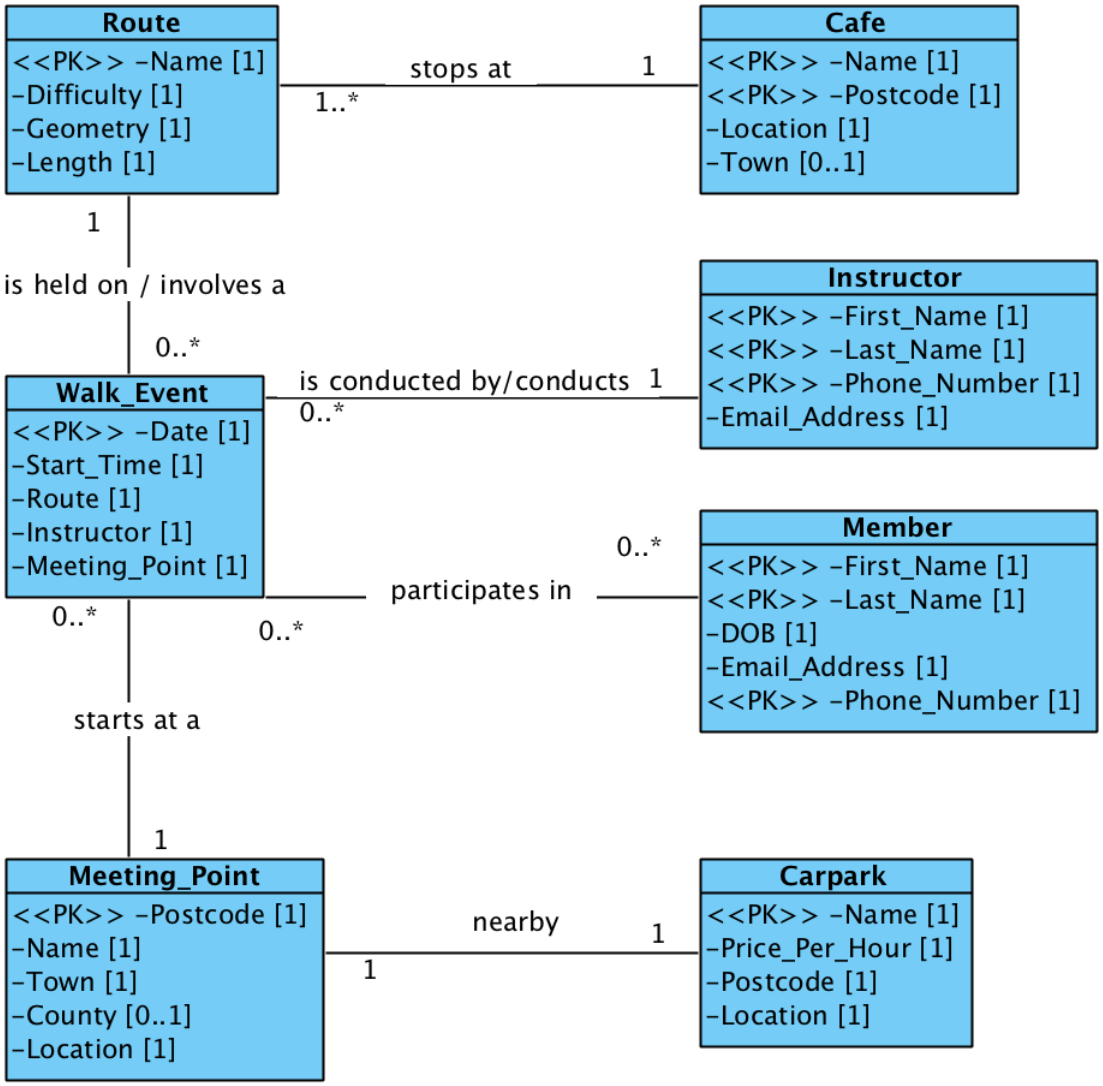
Assignment 2 – Spatial Data Management

Part A – Database Creation

1 Conceptual UML Diagram

The image below shows the conceptual UML diagram as submitted in assignment 1.

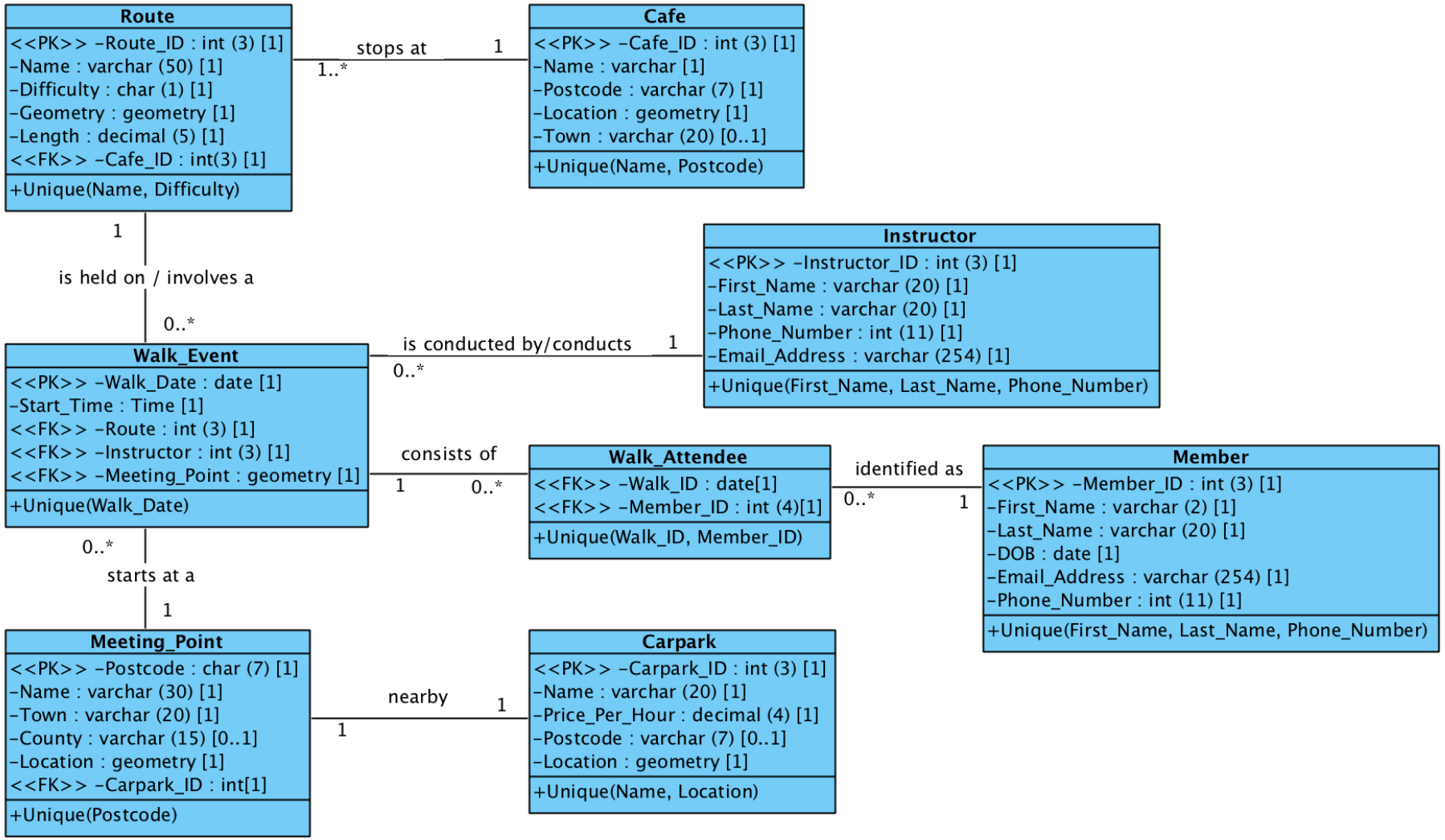


2 Logical UML Diagram

In order to create the logical UML diagram for the system, as shown on the next page, a few changes were made to the conceptual diagram. These changes are as follows:

- The addition of a link table named *Walk_Attendee,* which was created in order to resolve the many-to-many relationship between the *Member* and *Walk_Event* entities.
- A number of entities have been given an ID attribute as their primary key. The unique identifiers specified in the conceptual diagram can be seen as unique constraints in the logical diagram.
- The identification or addition of foreign key attributes.
- The cardinality of the *postcode* attribute within the *carpark* entity has been modified as it has been noted that some carparks may not have a postcode, so this is no longer a requirement.

The logical UML diagram for the system:



3 SQL Scripts Used to Create the Tables

3.1 Schema Creation

The tables for this database have been created within a new schema named *walkdb.* This new schema was created in order to organise the following tables away from those located in the public schema.

This was created using the following script:

```
create schema walkdb;
```

3.2 Table Creation

The following tables present the SQL script used to create the 8 entities identified by the Logical UML diagram (see Section 2). The script for creating both primary and foreign keys, and unique constraints for each entity is also detailed.

In some cases, a check constraint has been implemented to ensure information is entered correctly and abiding by the integrity constraints of the database. Additionally, trigger functions have been used to calculate derived information necessary for the database. The scripts used in these cases has been listed.

The entities *Carpark*, *Café*, *Meeting_Point*, and *Route* are all spatial, therefore the script was used to ensure the geometry of these could be held has also been listed.

Entity: Member

Create Table Script	Primary Key Constraint	Foreign Key Constraints	Unique Constraints
`create table walkdb.member ( member_id int not null, first_name varchar(20) not null, last_name varchar(20) not null, dob date not null, email_address varchar(254) not null, phone_number varchar(11) not null );`	`alter table walkdb.member add constraint member_pkey primary key (member_id);`	N/A	`alter table walkdb.member add constraint member_unique unique(first_name, last_name, phone_number);`

A check constraint was added for this entity to ensure the *Member's* age was over 18. A *Member* being over 18 was identified as an integrity constraint in assignment 1. The script used can be seen below:

```
alter table walkdb.member add constraint member_age check (dob < (current_date - interval '18' year));
```

Entity: Instructor

Create Table Script	Primary Key Constraint	Foreign Key Constraints	Unique Constraints
`create table walkdb.instructor ( instructor_id int not null, first_name varchar(20) not null, last_name varchar(20) not null, email_address varchar(254) not null, phone_number varchar(11) not null );`	`alter table walkdb.instructor add constraint instructor_pkey primary key (instructor_id);`	N/A	`alter table walkdb.instructor add constraint instructor_unique unique(first_name, last_name, phone_number);`

Entity: Carpark

Create Table Script	Primary Key Constraint	Foreign Key Constraints	Unique Constraints
`create table walkdb.carpark ( carpark_id int not null, name varchar(50) not null, price_per_hour double precision not null, postcode varchar(7) );`	`alter table walkdb.carpark add constraint carpark_pkey primary key (carpark_id);`	N/A	`alter table walkdb.carpark add constraint carpark_unique unique(name, location);`

In order to add the geometry for the *Carpark* entity to be stored as a polygon, the following script was used:

```
alter table walkdb.carpark add column location geometry(Polygon, 27700);
```

Entity: Meeting_Point

Create Table Script	Primary Key Constraint	Foreign Key Constraints	Unique Constraints
`create table walkdb.meeting_point ( postcode char(7) not null, name varchar(50) not null, town varchar(20) not null, county varchar(15) carpark_id int not null );`	`alter table walkdb.meeting_point add constraint meeting_point_pkey primary key (postcode);`	`alter table walkdb.meeting_point add constraint meeting_point_carpark_fkey foreign key (carpark_id) references walkdb.carpark (carpark_id);`	N/A as the primary key for this entity is *postcode* and therefore is unique.

In order to add the geometry for the *Meeting Point* entity to be stored as a point, the following script was used:

```
alter table walkdb.meeting_point add column location geometry(Point, 27700);
```

Entity: Cafe

Create Table Script	Primary Key Constraint	Foreign Key Constraints	Unique Constraints
`create table walkdb.cafe ( cafe_id int not null, name varchar(50) not null, postcode varchar(7) not null, town varchar(20) );`	`alter table walkdb.cafe add constraint cafe_pkey primary key (cafe_id);`	N/A	`alter table walkdb.cafe add constraint cafe_unique unique(name, postcode);`

In order to add the geometry for the *Café* entity to be stored as a polyhedral surface, the following script was used:

```
alter table walkdb.cafe add column location geometry(POLYHEDRALSURFACE, 27700, 3,1);
```

Additionally, as this entity is 3D, the following script was used to force 3D coordinates to be used:

```
alter table walkdb.cafe
  alter column location type geometry(POLYHEDRALSURFACEZ)
    using st_force_3d(location);
```

Entity: Route

Create Table Script	Primary Key Constraint	Foreign Key Constraints	Unique Constraints
`create table walkdb.route ( route_id int not null, name varchar(50) not null, difficulty char(1) not null, length double precision, cafe_id int not null );`	`alter table walkdb.route add constraint route_pkey primary key (route_id);`	`alter table walkdb.route add constraint route_cafe_fkey foreign key (cafe_id) references walkdb.cafe (cafe_id);`	`alter table walkdb.route add constraint route_unique unique(name, difficulty);`

In order to add the geometry for the *Route* entity to be stored as a line, the following script was used:

```
alter table walkdb.route add column location geometry(Linestring, 27700);
```

A type constraint was used to ensure the attribute 'difficulty' was entered correctly. This attribute can only have three possible entries: E, M, or S. These letters represent either an Easy, Moderate, or Strenuous walk. The following script was used:

```
alter table walkdb.route add constraint difficulty_types check (difficulty = 'E' OR difficulty = 'M' OR difficulty = 'S');
```

A trigger function was used in order to generate the value for the 'length' attribute. The following script was used:

```
create trigger route_length_update
  after insert or update of location on walkdb.route
  for each row
  execute procedure update_length();


create or replace function update_length() returns trigger as $body$
  begin
  update walkdb.route set length = st_length(location)
    where route_id = new.route_id;
    return null;
    end;
    $body$ language 'plpgsql';
```

Entity: Walk_Event

Create Table Script	Primary Key Constraint	Foreign Key Constraints	Unique Constraints

`create table walkdb.walk_event ( walk_date date not null, start_time time not null, route int, instructor int, meeting_point char(7) );`	`alter table walkdb.walk_event add constraint walk_event_pkey primary key (walk_date);`	`alter table walkdb.walk_event add constraint walk_event_route_fkey foreign key (route) references walkdb.route (route_id);`
		`alter table walkdb.walk_event add constraint walk_event_instrctor_fkey foreign key (instructor) references walkdb.instructor (instructor_id);`
		`alter table walkdb.walk_event add constraint walk_event_meeting_point_fkey foreign key (meeting_point) references walkdb.meeting_point (postcode);`
			N/A as the primary key for this entity is *date* and therefore is unique.

Entity: Walk_Attendee

Create Table Script	Primary Key Constraint	Foreign Key Constraints	Unique Constraints
`create table walkdb.walk_attendee ( walk_id date not null, member_id int not null );`	N/A as the entity consists of two foreign key attributes, which combined make up the unique constraint.	`alter table walkdb.walk_attendee add constraint walk_attendee_walk_id_fkey foreign key (walk_id) references walkdb.walk_event (walk_date);` `alter table walkdb.walk_attendee add constraint walk_attendee_member_id_fkey foreign key (member_id) references walkdb.member (member_id);`	`alter table walkdb.walk_attendee add constraint walk_attendee_unique unique(walk_id, member_id);`

4 SQL Insert Statements

This section details the SQL insert statements necessary to populate each table with the required sample information.

Entity: Member

```
insert into walkdb.member values
  (1, 'Nicola', 'Critten', '1995-12-20', 'nicolacritten@hotmail.com', 07756233454),
  (2, 'Kate', 'Halpin', '1987-04-17', 'k.halpin@gmail.co.uk', 02086547855),
  (3, 'Derek', 'Harvey', '1965-06-26', 'derek_harvey@hotmail.co.uk', 07645234980),
  (4, 'Laura', 'Andrews', '1990-08-19', 'lauraandrews@hotmail.co.uk', 07358922080),
  (5, 'Sam', 'Andrews', '1987-06-12', 'sam.andrewsss@sky.co.uk', 07745336729);
```

Entity: Instructor

```
insert into walkdb.instructor values
  (1, 'Gary', 'Critten', 'gicritten@sky.com', 07756445520),
  (2, 'Philippa', 'Smith', 'pip_smith@ghotmail.co.uk', 07789345562),
  (3, 'Rachel', 'Hassan', 'rachas22@hotmail.com', 01054356689);
```

Entity: Carpark

```
insert into walkdb.carpark values
  ('001', 'High Elms Carpark - Shire Lane', 0.00, null,
    st_geomfromtext('POLYGON((544567.5 163518.3, 544617.6 163562.7, 544589.0
163584.2, 544541.8 163539.8, 544567.5 163518.3))',27700));

insert into walkdb.carpark values
  ('002', 'Keston Ponds Carpark', 1.30, 'BR26BA',
    st_geomfromtext('POLYGON((541902.9 163956.7, 541938.9 164001.5, 541912.0
164019.3, 541876.5 163975.3, 541902.9 163956.7))',27700));

insert into walkdb.carpark values
  ('003', 'Foots Cray Place Carpark', 0.00, null,
    st_geomfromtext('POLYGON((547481.1 171034.7, 547581.1 171013.7, 547531.2
171037.0, 547502.2 171057.5, 547481.1 171034.7))',27700));

insert into walkdb.carpark values
  ('004', 'Chartwell National Trust Carpark', 3.00, 'TN161PS',
    st_geomfromtext('POLYGON((545387.5 151280.9, 545513.6 151360.8, 545527.6
151399.6, 545473.0 151412.6, 545369.3 151325.3, 545387.5 151280.9))',27700));
```

Entity: Meeting_Point

```
insert into walkdb.meeting_point values ('DA144RU', 'Foots Cray Meadows', 'Sidcup',
'Kent', st_geomfromtext('POINT(547524.657 171130.575)',27700), 3);

insert into walkdb.meeting_point values ('BR26HA', 'Keston Ponds', 'Keston',
'Kent', st_geomfromtext('POINT(541915.8 163997.0)',27700), 2);

insert into walkdb.meeting_point values ('BR67JH', 'High Elms', 'Orpington',
'Kent', st_geomfromtext('POINT(544592.9 163453.2)',27700), 1);

insert into walkdb.meeting_point values ('TN161QD', 'Chartwell and Toys Hill',
'Westerham', 'Kent', st_geomfromtext('POINT(545482.6 151669.9)',27700), 4);
```

Entity: Café

```
insert into walkdb.cafe values
  ('001', 'Costa coffee', 'BR26BB', 'Keston',
    st_geomfromtext('POLYHEDRALSURFACE(
      ((541557.0 164136.9 0,
      541569.5 164114.0 0,
      541587.9 164127.8 0,
      541574.5 164150.5 0,
      541557.0 164136.9 0)),

      ((541557.0 164136.9 0,
      541569.5 164114.0 0,
      541569.5 164114.0 10,
      541557.0 164136.9 10,
      541557.0 164136.9 0)),

      ((541569.5 164114.0 0,
      541587.9 164127.8 0,
      541587.9 164127.8 10,
      541569.5 164114.0 10,
      541569.5 164114.0 0)),

      ((541587.9 164127.8 0,
      541574.5 164150.5 0,
      541574.5 164150.5 10,
      541587.9 164127.8 10,
      541587.9 164127.8 0)),

      ((541574.5 164150.5 0,
      541557.0 164136.9 0,
      541557.0 164136.9 10,
      541574.5 164150.5 10,
      541574.5 164150.5 0)),

      ((541557.0 164136.9 10,
      541569.5 164114.0 10,
      541587.9 164127.8 10,
      541574.5 164150.5 10,
      541557.0 164136.9 10))
      )',27700));

insert into walkdb.cafe values
  ('002', 'Crockham Tea Rooms', 'TN86PW', 'Edenbridge',
    st_geomfromtext('POLYHEDRALSURFACE(
      ((544921.0 150022.2 0,
      544949.8 150023.6 0,
      544950.6 150042.7 0,
      544920.7 150041.9 0,
      544921.0 150022.2 0)),

      ((544921.0 150022.2 0,
      544949.8 150023.6 0,
      544949.8 150023.6 30,
      544921.0 150022.2 30,
      544921.0 150022.2 0)),

      ((544949.8 150023.6 0,
      544950.6 150042.7 0,
      544950.6 150042.7 30,
      544949.8 150023.6 30,
      544949.8 150023.6 0)),

      ((544950.6 150042.7 0,
      544920.7 150041.9 0,
      544920.7 150041.9 30,
      544950.6 150042.7 30,
```

```
      544950.6 150042.7 0)),

      ((544920.7 150041.9 0,
      544921.0 150022.2 0,
      544921.0 150022.2 30,
      544920.7 150041.9 30,
      544920.7 150041.9 0)),

      ((544921.0 150022.2 30,
      544949.8 150023.6 30,
      544950.6 150042.7 30,
      544920.7 150041.9 30,
      544921.0 150022.2 30))
      )',27700));

insert into walkdb.cafe values
  ('003', 'Green Roof Cafe', 'BR67JH', 'Orpington',
    st_geomfromtext('POLYHEDRALSURFACE(
      ((544669.4 163412.5 0,
      544717.3 163435.0 0,
      544695.2 163480.5 0,
      544645.2 163458.6 0,
      544669.4 163412.5 0)),

      ((544669.4 163412.5 0,
      544717.3 163435.0 0,
      544717.3 163435.0 15,
      544669.4 163412.5 15,
      544669.4 163412.5 0)),

      ((544717.3 163435.0 0,
      544695.2 163480.5 0,
      544695.2 163480.5 15,
      544717.3 163435.0 15,
      544717.3 163435.0 0)),

      ((544695.2 163480.5 0,
      544645.2 163458.6 0,
      544645.2 163458.6 15,
      544695.2 163480.5 15,
      544695.2 163480.5 0)),

      ((544645.2 163458.6 0,
      544669.4 163412.5 0,
      544669.4 163412.5 15,
      544645.2 163458.6 15,
      544645.2 163458.6 0)),

      ((544669.4 163412.5 15,
      544717.3 163435.0 15,
      544695.2 163480.5 15,
      544645.2 163458.6 15,
      544669.4 163412.5 15))
      )',27700));

insert into walkdb.cafe values
  ('004', 'Costa coffee', 'DA144RU', 'Sidcup',
    st_geomfromtext('POLYHEDRALSURFACE(
      ((547427.9 171723.5 0,
      547458.3 171710.7 0,
      547463.9 171724.9 0,
      547433.0 171737.6 0,
      547427.9 171723.5 0)),

      ((547427.9 171723.5 0,
      547458.3 171710.7 0,
      547458.3 171710.7 20,
      547427.9 171723.5 20,
```

```
      547427.9 171723.5 0)),

      ((547458.3 171710.7 0,
      547463.9 171724.9 0,
      547463.9 171724.9 20,
      547458.3 171710.7 20,
      547458.3 171710.7 0)),

      ((547463.9 171724.9 0,
      547433.0 171737.6 0,
      547433.0 171737.6 20,
      547463.9 171724.9 20,
      547463.9 171724.9 0)),

      ((547433.0 171737.6 0,
      547427.9 171723.5 0,
      547427.9 171723.5 20,
      547433.0 171737.6 20,
      547433.0 171737.6 0)),

      ((547427.9 171723.5 20,
      547458.3 171710.7 20,
      547463.9 171724.9 20,
      547433.0 171737.6 20,
      547427.9 171723.5 20))
      )',27700));
```

Entity: Route

```
insert into walkdb.route (route_id, name, difficulty, location, cafe_id) values
  ('001', 'Keston Ponds', 'E',
    st_geomfromtext('LINESTRING(
      541884 164020,
      541809 164337,
      541833 164488,
      541740 165004,
      541584 165042,
      541575 164687,
      541698 164431,
      541582 164077,
      541672 163708,
      541882 164006
      )',27700), 1);

insert into walkdb.route (route_id, name, difficulty, location, cafe_id) values
  ('002', 'High Elms short trail', 'E',
    st_geomfromtext('LINESTRING(
      544590 163434,
      544659 163289,
      544744 163348,
      544860 163109,
      544572 162666,
      544665 162271,
      544929 162824,
      544909 162894,
      545115 163173,
      545080 163241,
      544999 163190,
      544690 163516,
      544605 163457
      )',27700), 3);

insert into walkdb.route (route_id, name, difficulty, location, cafe_id) values
  ('003', 'High Elms long trail', 'M',
    st_geomfromtext('LINESTRING(
      544590 163434,
      544648 163278,
```

```
      544335 162976,
      544517 162877,
      544565 162695,
      544453 162449,
      544324 162323,
      544531 162020,
      544906 162830,
      544896 162901,
      545122 163195,
      545080 163261,
      544966 163194,
      544699 163508,
      544605 163457
      )',27700), 3);

insert into walkdb.route (route_id, name, difficulty, location, cafe_id) values
  ('004', 'Foots Cray Meadows', 'M',
    st_geomfromtext('LINESTRING(
      547535 171132,
      547791 171266,
      548227 171734,
      548551 172527,
      548434 172551,
      548270 172174,
      548122 172217,
      547949 171875,
      547621 172014,
      547430 171541,
      547623 171327,
      547524 171147
      )',27700), 4);

insert into walkdb.route (route_id, name, difficulty, location, cafe_id) values
  ('005', 'Chartwell and Toys Hill', 'S',
    st_geomfromtext('LINESTRING(
      545461 151636,
      545315 151386,
      545133 151495,
      545020 150973,
      544891 150961,
      544915 150011,
      546641 150108,
      546968 151289,
      545905 152679,
      545485 151639
      )',27700), 2);
```

Entity: Walk_Event

```
insert into walkdb.walk_event values
  ('2018-05-05', '12:00', 002, 0003, 'BR67JH'),
  ('2018-05-27', '12:00', 003, 0002, 'DA144RU'),
  ('2018-05-30', '11:30', 001, 0003, 'BR26HA'),
  ('2018-06-09', '10:00', 005, 0001, 'TN161QD'),
  ('2018-06-23', '12:00', 004, 0001, 'DA144RU'),
  ('2018-07-14', '13:00', 005, 0001, 'TN161QD'),
  ('2018-07-27', '10:00', 004, 0002, 'DA144RU'),
  ('2018-08-12', '12:45', 001, 0001, 'BR26HA'),
  ('2018-09-03', '11:00', 005, 0002, 'TN161QD');
```

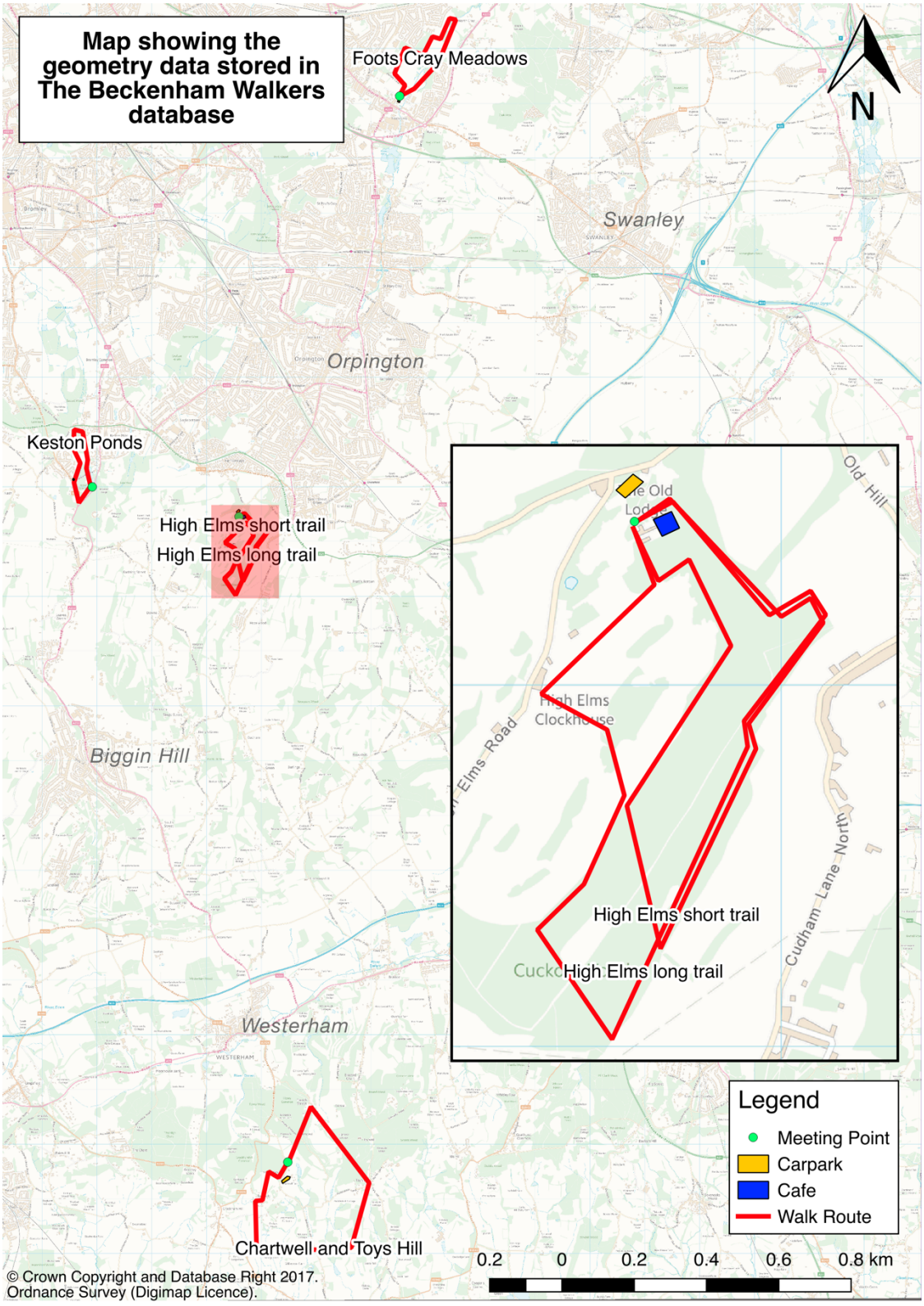
Entity: Walk_Attendee

```
insert into walkdb.walk_attendee values
  ('2018-05-05', 1),
  ('2018-05-05', 3),
```

```
  ('2018-05-05', 4),
  ('2018-05-05', 5);

insert into walkdb.walk_attendee values
  ('2018-05-27', 1),
  ('2018-05-27', 2);

insert into walkdb.walk_attendee values
  ('2018-05-30', 4),
  ('2018-05-30', 5),
  ('2018-05-30', 2);

insert into walkdb.walk_attendee values
  ('2018-06-9', 3),
  ('2018-06-9', 4),
  ('2018-06-9', 5);

insert into walkdb.walk_attendee values
  ('2018-06-23', 2),
  ('2018-06-23', 3);

insert into walkdb.walk_attendee values
  ('2018-07-14', 2),
  ('2018-07-14', 3),
  ('2018-07-14', 5);

insert into walkdb.walk_attendee values
  ('2018-07-27', 3),
  ('2018-07-27', 4),
  ('2018-07-27', 5);

insert into walkdb.walk_attendee values
  ('2018-08-12', 2);

insert into walkdb.walk_attendee values
  ('2018-09-03', 1),
  ('2018-09-03', 2),
  ('2018-09-03', 4),
  ('2018-09-03', 5);
```

5 QGIS Map

The following map displays the spatial entities stored in the database. These entities are *Meeting Point, Carpark, Café* and *Route.* The inset map shows the High Elms site zoomed in to show further detail and improve clarity. Additionally, High Elms is the only site to have two walking routes.

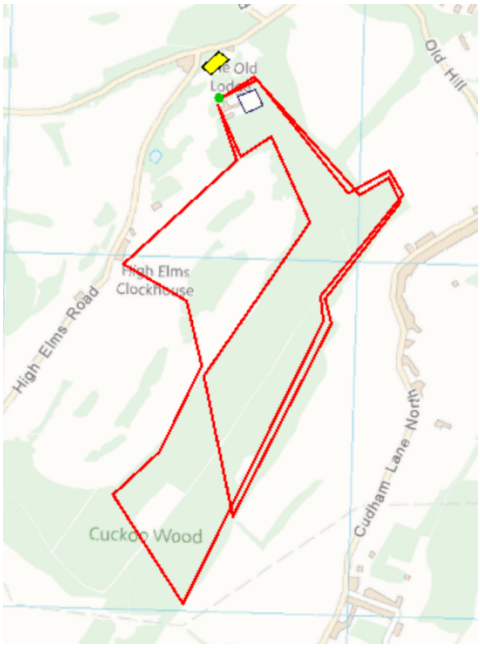
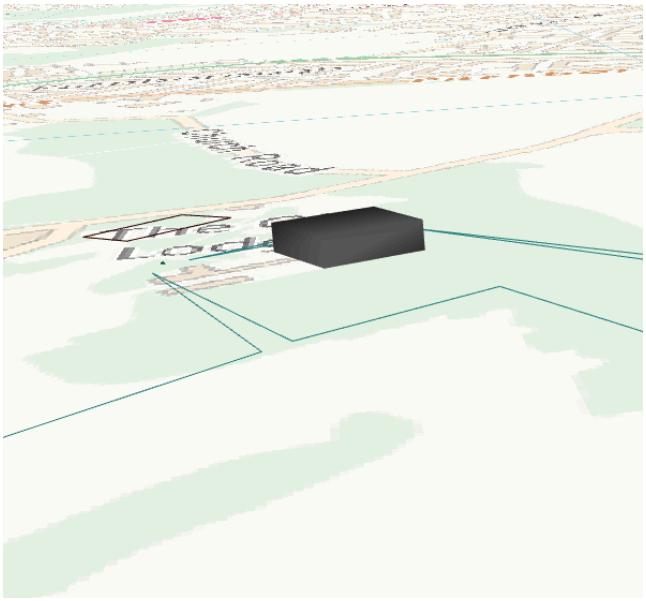


6 Screenshots from FME

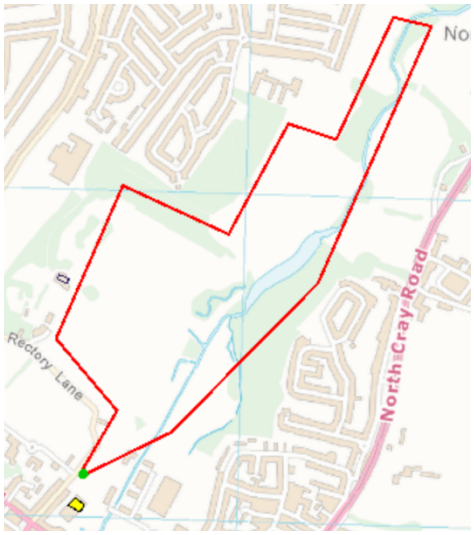
The table below displays the spatial data held in the database, as shown in FME whilst in the 2D and 3D viewing modes. The same background mapping from Ordnance survey used in the QGIS map has been again used, along with a similar colour scheme (the colour of the 3D café could not be changed, however).

It was not possible to show all of the data within one image in any clarity, due to a large geographic extent. For this reason, screenshots were taken of each site individually, in both a 2D and 3D view. The 3D view focuses on the *Café* entity, as that is the only 3D spatial entity in the database.

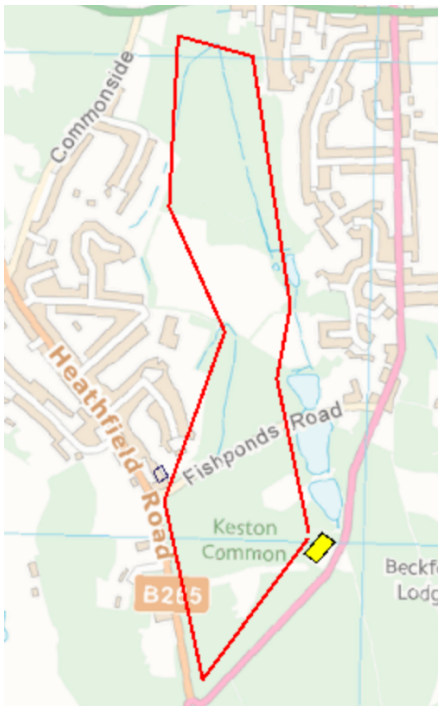
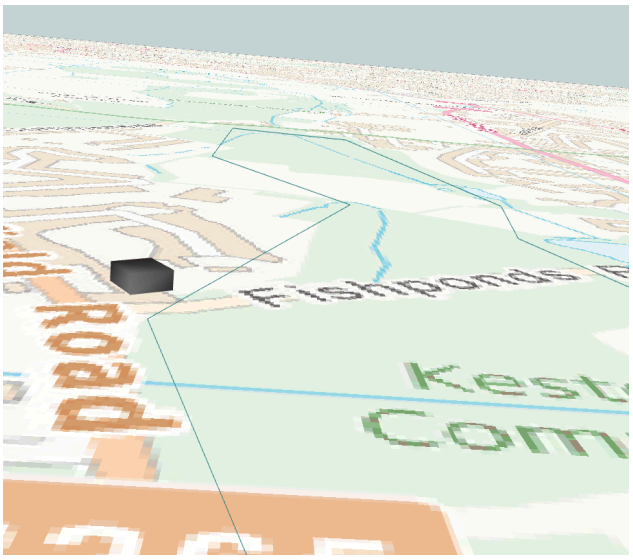
Site: High Elms

The 2D viewing mode	**The 3D viewing mode showing the café**
	

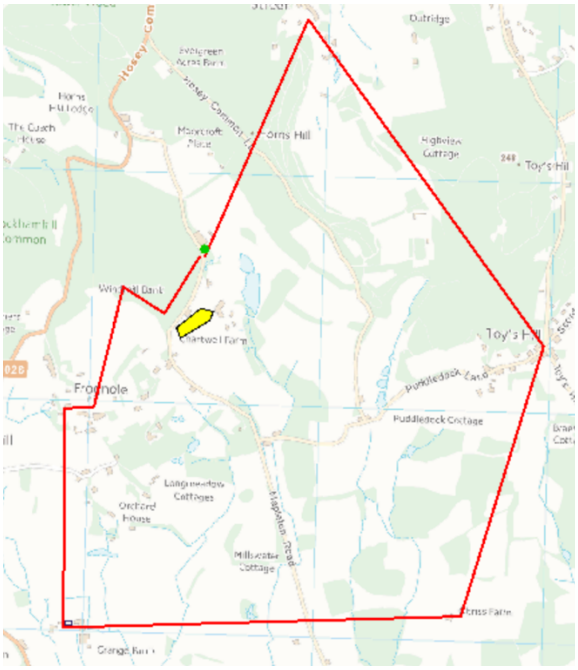
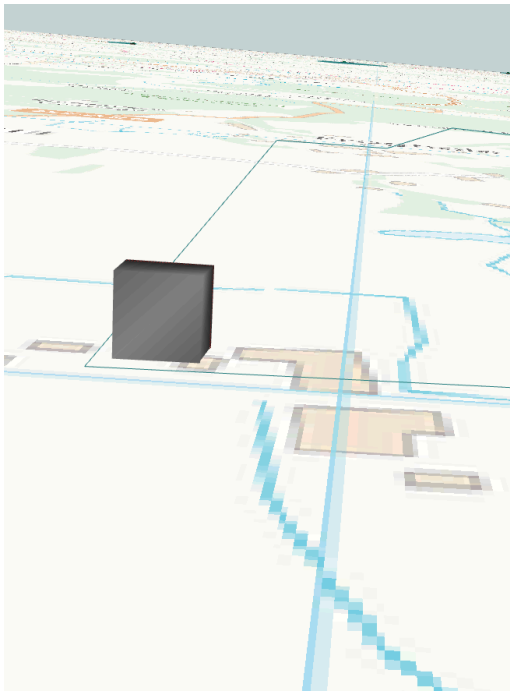
Site: Foots Cray Meadows

The 2D viewing mode	The 3D viewing mode showing the café
	

Site: Keston Ponds

The 2D viewing mode	The 3D viewing mode showing the café
	

Site: Chartwell and Toys Hill

The 2D viewing mode	The 3D viewing mode showing the café
	

7 Functional Requirements

The following functional requirements table has been taken from Assignment 1. Two of the requirement questions have been slightly modified in order for the query output to be more useful for the system user. These changes can be seen in the table.

For example, the requirement question originally posed as *'Which instructor is in charge of a given walk event?'* has been modified to *'Which instructor is in charge of each planned walk event and what is their phone number?'.*

This change means the output will contain more useful information for the system user. Firstly, the instructor for every planned walk will be given rather than just a given walk only. Secondly, the instructor's phone number will be generated, which can then be given to the members attending each walk, in case they need to contact the instructor.

Additionally, the entities required for requirement 3 have been changed due to the formation of the link table *Walk_Attendee* when the logical UML diagram was created.

Requirement #	Requirement	Entity or Entities Required	Spatial Query	Join
1	What is the distance from the carpark to the meeting point?	Meeting point, Carpark	Yes	Yes
2	What is the length of each route?	Route	Yes	No
3	Which members are going on a given walk event?	~~Members, Walk event~~ Members, Walk attendee	No	Yes
4	~~Which instructor is in charge of a given walk event?~~ Which instructor is in charge of each planned walk event and what is their phone number?	Instructor, Walk event	No	Yes
5	What is the area of each café?	Café	Yes	No
6	What is the area of each carpark?	Carpark	Yes	No
7	How many walk events are scheduled for each difficultly?	Walk Event, Route	No	Yes
8	~~How many walk events are being held in any given month?~~ How many walk events are being held each month?	Walk Event	No	No
9	Do any routes intersect?	Route	Yes	No
10	Which meeting points are located within a carpark?	Carpark, Meeting point	Yes	Yes
Totals	**11**	**N/A**	**6**	**5**

Each of the 10 functional requirement questions have been successfully answered. The scripts used, and the results obtained can be seen for each requirement in the following tables. Additionally, the purpose of completing each requirement is stated.

Requirement 1: What is the distance from the carpark to the meeting point?

This requirement would be necessary in order to ensure any *Carparks* that are a considerable distance from a *Meeting Point* can be identified and the *Members* told in advanced, so they are aware.

<table>
<tr><th>Entity or Entities Required</th><th>Spatial Query</th><th>Join</th><th>SQL Query</th><th>Screenshots of results from PG Admin</th></tr>
<tr><td>Meeting point, Carpark</td><td>Yes</td><td>Yes</td><td>

```
select
  a.name as "Meeting Point Name",
  b.name as "Carpark Name",
  round(st_distance(a.location,
b.location)::DECIMAL,2) as "Distance (m)"
from
  walkdb.meeting_point a left join
  walkdb.carpark b
on a.carpark_id = b.carpark_id;
```

</td><td>

Data Output | Explain | Messages | Query History

	Meeting Point Name character varying (50)	Carpark Name character varying (50)	Distance (m) numeric
1	Foots cray meadows	Foots Cray Place Carpark	76.45
2	Keston ponds	Keston Ponds Carpark	0.00
3	High elms	High Elms Carpark - Shire Lane	69.88
4	Chartwell and Toys Hill	Chartwell National Trust Carpark	257.48

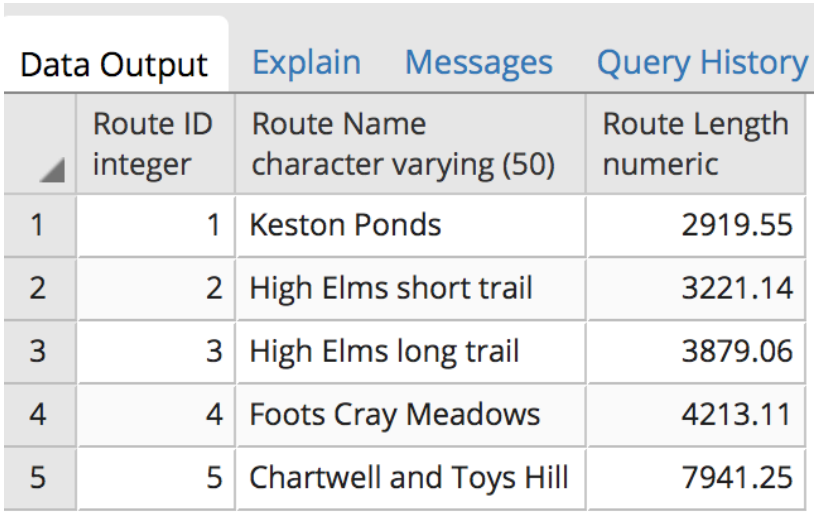
N.B. There is no distance between the Keston Ponds meeting point and carpark as this meeting point is located within the carpark. This is shown in Requirement 10.

</td></tr>
</table>

Requirement 2: What is the length of each route?

Although the length of the route has been added to the *Route* entity by using a trigger function, this requirement would still be necessary to clearly show the *Route* length information in order to inform the *Members* prior to the *Walk Event.*

Entity or Entities Required	Spatial Query	Join	SQL Query	Screenshots of results from PG Admin

Route	Yes	No	`select`
			`  a.route_id as "Route ID",`
			`  a.name as "Route Name",`
			`  round(st_length(a.location)::DECIMAL,2)`
			`  as "Route Length"`
			`from walkdb.route a;`
				

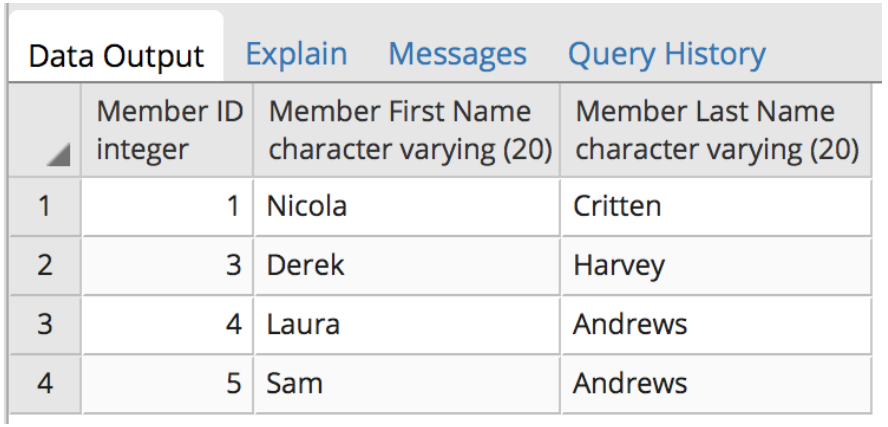
Data Output | Explain | Messages | Query History

	Route ID integer	Route Name character varying (50)	Route Length numeric
1	1	Keston Ponds	2919.55
2	2	High Elms short trail	3221.14
3	3	High Elms long trail	3879.06
4	4	Foots Cray Meadows	4213.11
5	5	Chartwell and Toys Hill	7941.25

Requirement 3: Which members are going on a given walk event?

This requirement would be necessary to provide the *Instructor* with an attendance list, so they can know which *Members* to expect. This could then be used in conjunction with Requirement 4 (see page 25) to identify which *Instructor* the list needs to be given to.

For this query, the *Walk Event* being held on 5/5/2018 was selected.

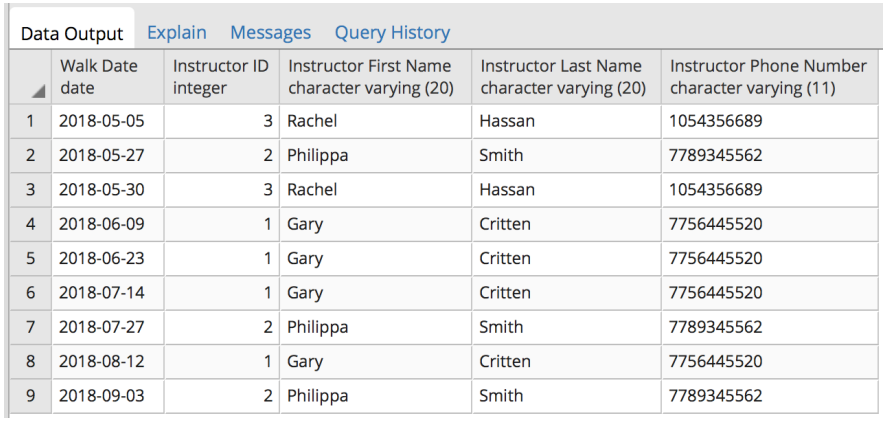
Entity or Entities Required	Spatial Query	Join	SQL Query	Screenshots of results from PG Admin
Members, Walk attendee	No	Yes	`select a.member_id as "Member ID", b.first_name as "Member First Name", b.last_name as "Member Last Name" from walkdb.walk_attendee a left join walkdb.member b on a.member_id = b.member_id where a.walk_id = '2018-05-05';`	

Data Output | Explain | Messages | Query History

	Member ID integer	Member First Name character varying (20)	Member Last Name character varying (20)
1	1	Nicola	Critten
2	3	Derek	Harvey
3	4	Laura	Andrews
4	5	Sam	Andrews

Requirement 4: Which instructor is in charge of each planned walk event and what is their phone number?

This requirement would be necessary for knowing which *Instructor* to give each attendance sheet to (see Requirement 3), giving the *Members* a phone number for the *Instructor*, and ensuring no *Instructor* is leading a dominant portion of *Walk Events.*

Entity or Entities Required	Spatial Query	Join	SQL Query	Screenshots of results from PG Admin
Instructor, Walk event	No	Yes	`select a.walk_date as "Walk Date", a.instructor as "Instructor ID", b.first_name as "Instructor First Name", b.last_name as "Instructor Last Name", b.phone_number as "Instructor Phone Number" from walkdb.walk_event a left join walkdb.instructor b on a.instructor = b.instructor_id group by a.walk_date, b.first_name, b.last_name, b.instructor_id order by "Walk Date";`	

Data Output | Explain | Messages | Query History

	Walk Date date	Instructor ID integer	Instructor First Name character varying (20)	Instructor Last Name character varying (20)	Instructor Phone Number character varying (11)
1	2018-05-05	3	Rachel	Hassan	1054356689
2	2018-05-27	2	Philippa	Smith	7789345562
3	2018-05-30	3	Rachel	Hassan	1054356689
4	2018-06-09	1	Gary	Critten	7756445520
5	2018-06-23	1	Gary	Critten	7756445520
6	2018-07-14	1	Gary	Critten	7756445520
7	2018-07-27	2	Philippa	Smith	7789345562
8	2018-08-12	1	Gary	Critten	7756445520
9	2018-09-03	2	Philippa	Smith	7789345562

Requirement 5: What is the area of each café?

This requirement would be necessary in order to estimate how many people could be seated in each *Café* and therefore help with deciding an upper limit of *Members* that can attend each *Walk Event.*

Entity or Entities Required	Spatial Query	Join	SQL Query	Screenshots of results from PG Admin

Café	Yes	No	`select`
			`  a.cafe_id as "Cafe ID",`
			`  a.name as "Cafe Name",`
			`  round(st_area(a.location)::DECIMAL,2) as "Cafe Area (m2)"`
			`from`
			`  walkdb.cafe a`
			`order by`
			`  "Cafe Area (m2)";`
				Data Output | Explain | Messages | Query History

	Cafe ID integer	Cafe Name character varying (50)	Cafe Area (m2) numeric
1	4	Costa coffee	1003.82
2	2	Crockham Tea Rooms	1138.23
3	1	Costa coffee	1173.35
4	3	Green Roof Cafe	5511.68

Requirement 6: What is the area of each carpark?

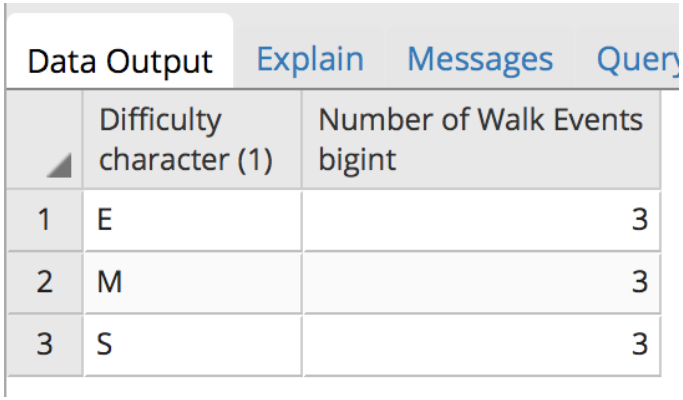
This requirement would be necessary in order to estimate how many cars could be parked in each *Carpark.* The *Members* could then be informed prior to the *Walk Event* if it will be necessary to car share, look into another parking solution, or avoid driving if possible.

Entity or Entities Required	Spatial Query	Join	SQL Query	Screenshots of results from PG Admin

Carpark	Yes	No	`select`
			`  a.carpark_id as "Carpark ID",`
			`  a.name as "Carpark Name",`
			`  round(st_area(a.location)::DECIMAL,2) as "Carpark Area (m2)"`
			`from`
			`  walkdb.carpark a`
			`order by`
			`  "Carpark Area (m2)";`
				Data Output | Explain | Messages | Query History
				Carpark ID integer | Carpark Name character varying (50) | Carpark Area (m2) numeric
				1 | 3 | Foots Cray Place Carpark | 1116.16
				2 | 2 | Keston Ponds Carpark | 1833.91
				3 | 1 | High Elms Carpark - Shire Lane | 2251.43
				4 | 4 | Chartwell National Trust Carpark | 9134.77

Requirement 7: How many walk events are scheduled for each difficulty?

This requirement would be necessary to ensure there are a close-to-equal amount of *Walk Events* currently scheduled for each difficultly level, in order to allow all *Members* opportunity to attend.

Entity or Entities Required	Spatial Query	Join	SQL Query	Screenshots of results from PG Admin
Walk Event, Route	No	Yes	`select b.difficulty as "Difficulty", count(*) as "Number of Walk Events" from walkdb.walk_event a left join walkdb.route b on a.route = b.route_id group by b.difficulty order by b.difficulty;`	

Requirement 8: How many walk events are being held each month?

This requirement would be necessary to help ensure there are enough *Walk Events* planned in the coming months, as The Beckenham Walkers aim to ensure *Walk Events* are regularly scheduled. Consequently, additional *Walk Events* could be scheduled if needed.

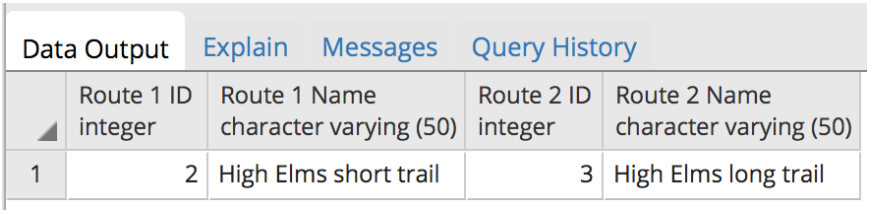
Entity or Entities Required	Spatial Query	Join	SQL Query	Screenshots of results from PG Admin

Walk Event	No	No	`select`
			`  count(*) as "Number of Walk Events",`
			`  to_char(a.walk_date, 'Mon') as "Month",`
			`  extract(year from a.walk_date) as "Year"`
			`from walkdb.walk_event a`
			`group by "Month", "Year"`
			`order by "Number of Walk Events";`
				Data Output | Explain | Messages | Query History
				Number of Walk Events bigint | Month text | Year double precision
				1 | 1 | Aug | 2018
				2 | 1 | Sep | 2018
				3 | 2 | Jun | 2018
				4 | 2 | Jul | 2018
				5 | 3 | May | 2018

Requirement 9: Do any routes intersect?

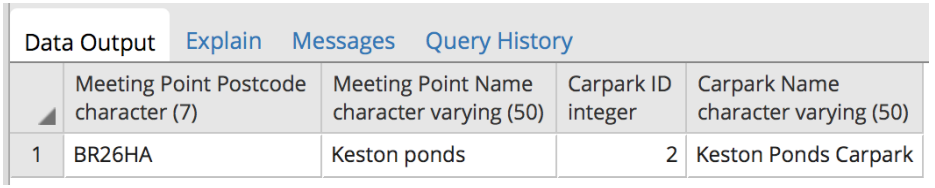
This requirement would be necessary in order to ensure the correct *Route* directions are known by the *Instructor.* For example, if an *Instructor* is familiar with conducting walks using the 'High Elms short trail', they may not be aware that the 'High Elms long trail' intersects but then follows a different direction.

Entity or Entities Required	Spatial Query	Join	SQL Query	Screenshots of results from PG Admin

Route	Yes	No	`select`
			`  a.route_id as "Route 1 ID",`
			`  a.name as "Route 1 Name",`
			`  b.route_id as "Route 2 ID",`
			`  b.name as "Route 2 Name"`
			`from`
			`  walkdb.route a,`
			`  walkdb.route b`
			`where`
			`  a.route_id < b.route_id and`
			`  ST_Intersects(a.location, b.location);`
				

Requirement 10: Which meeting points are located within a carpark?

This requirement would be necessary for safety reasons as the *Instructor* should be aware that the *Members* may start congregating where there are moving cars.

Entity or Entities Required	Spatial Query	Join	SQL Query	Screenshots of results from PG Admin
Carpark, Meeting point	Yes	Yes	`select a.postcode as "Meeting Point Postcode", a.name as "Meeting Point Name", b.carpark_id as "Carpark ID", b.name as "Carpark Name" from walkdb.meeting_point a, walkdb.carpark b where st_contains(b.location, a.location) = 't';`	

Data Output | Explain | Messages | Query History

	Meeting Point Postcode character (7)	Meeting Point Name character varying (50)	Carpark ID integer	Carpark Name character varying (50)
1	BR26HA	Keston ponds	2	Keston Ponds Carpark

Requirement 11: How many members are going on each walk event, what is the route, and who is the instructor?

This requirement has been added as after reviewing the topic description, it was felt it would be a necessary question for the system to answer.

This is firstly because before each *Walk Event* the *Instructor* in charge can be told how many *Members* they are to expect. Secondly, in order to track the popularity of the *Walk Events,* and the *Route* that will be taken.

Entity or Entities Required	Spatial Query	Join	SQL Query	Screenshots of results from PG Admin
Walk Attendee, Walk Event	No	Yes	See below	See below

```
select
  a.walk_id as "Walk date",
  count(a.member_id) as "Number
Attending",
  c.route_id as "Route",
  c.name as "Route Name",
  b.instructor as "Instructor ID"
from
  walkdb.walk_attendee a left join
  walkdb.walk_event b
on
  a.walk_id = b.walk_date left join
  walkdb.route c
on b.route = c.route_id
group by
  a.walk_id,
  b.instructor,
  c.route_id;
```

Data Output | Explain | Messages | Query History

	Walk date date	Number Attending bigint	Route integer	Route Name character varying (50)	Instructor ID integer
1	2018-06-23	2	4	Foots Cray Meadows	1
2	2018-05-05	4	2	High Elms short trail	3
3	2018-08-12	1	1	Keston Ponds	1
4	2018-06-09	3	5	Chartwell and Toys Hill	1
5	2018-05-30	3	1	Keston Ponds	3
6	2018-07-14	3	5	Chartwell and Toys Hill	1
7	2018-09-03	4	5	Chartwell and Toys Hill	2
8	2018-05-27	2	3	High Elms long trail	2
9	2018-07-27	3	4	Foots Cray Meadows	2

Part B – Support for Spatial Functionality in NoSQL Databases

Neo4j is an open source NoSQL graph database system launched in 2010 by NeoTechnology. It is the most popular graph-based system with Neo Technologies having more than 200 enterprise customers, including Cisco Systems, eBay, and Walmart (Yuhanna et al., 2016). Data is stored in the form of nodes and the relationships that connect them, each of which can have any number of properties (Figure 1). This format prioritises relationships, meaning handling large amounts of highly connected data is more efficient, compared to using a relational database (Neo4j, n.d.).

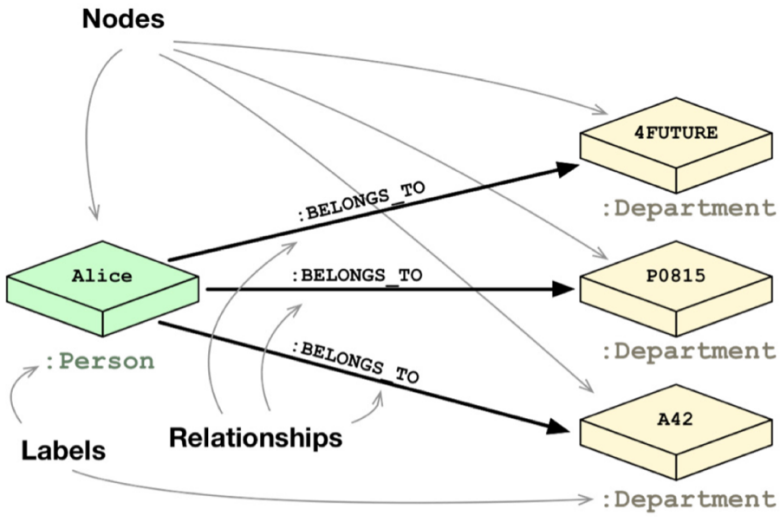


Figure 1: *An example of two connected data named 'Person' and 'Department' shown using the graph data model with Nodes and Relationships. In a relational database this would require the use of tables, foreign keys and a JOIN table (image from Neo4j, 2016).*

Due to its efficiency with handling linked data, Neo4j is usually found in applications such as telecommunications, logistics, social networks, and fraud detection (Baptista et al., 2014). This also makes it an ideal option for handling spatial data, which is done using the 'Neo4j-Spatial' extension, enabling the import, storage, and querying of spatial data. At the time of research, Baas (2012) and Baptista *et al* (2014) identified Neo4j as the most complete NoSQL spatial system. This essay aims to discuss the support for spatial functionality given by Neo4j and compare it with that of a spatially enabled relational database.

Spatial data is loaded into Neo4j using either a SimplePointLayer or EditableLayer type. The former, when only Points are required, and the latter, for any simple geometry type, including Point, LineString, Polygon, Multi-Polygon, and Multi-LineString. The storage format for these is 'Well Known Binary', a format specific to geographic geometries and is also used by PostGIS. Additionally, there is an option to import data in ESRI Shapefile and Open Street Map format. This list of types is fairly comprehensive, containing most of the simple geometry types expected from a spatially enabled relational database and similar or better than provided by other NoSQL databases.

Lyon (2016) used a series of complex Multi-polygons in Neo4j to represent the congressional district boundaries in the USA. A relationship was then created between each district polygons and its representing legislator, so the database could be queried to find the legislator representing each district when given a latitude and longitude point (Lyon, 2016) (Figure 2).

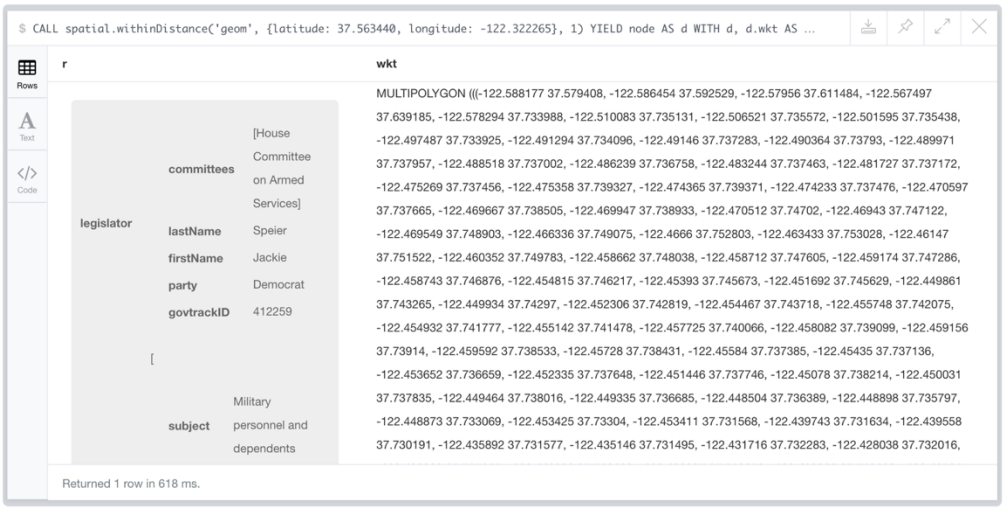


Figure 2: *The result of the query to find the legislator representing the district that contains the point 37.563440, -122.322265. The complex multi-polygon can be seen to be stored in the virtually the same way as it would be in PostGIS, a spatially enabled relational database.*

Unfortunately, there is no support provided for handling 3D entities in Neo4j. Relational database spatial extensions, such as OracleSpatial and PostGIS, can currently handle 3D. Also, once a node is created in Neo4j the geometry type must be

maintained, meaning any given node cannot be made up of Points and Linestrings, for example. In PostGIS, the type 'Geometry Collection' allows for this. Consequently, Neo4j lacks functionality in terms of the data type available.

Two coordinate reference systems (CRS) are supported in Neo4j: WGS84, and Cartesian 2D. Alternately, PostGIS includes 3,000 known spatial reference systems and the ability to transform or reproject between them (PostGIS, n.d.). OracleSpatial has similar capabilities (Oracle, n.d.). MongoDB, on the other hand, another NoSQL system, only uses WGS84. As stated by Agoub *et al* (2016), the lack of CRS options provided by Neo4j would make coordinate transformations mandatory, which in turn would generate an overhead for imports and exports. Additionally, it can lead to inaccuracies when calculating distance and length, which may be a fundamental issue for many spatial applications.

The queries that can be done on spatial data in Neo4j are reasonably extensive and similar to what would be expected from a spatially enabled relational database. These include: contain, cover, covered by, cross, equals, disjoint, intersect, intersect window, overlap, touch, within, and within distance. Lyon (2015) uses the within query to successfully find businesses, stored as points, within a user defined polygon.

The following analysis functions are also provided: area, bbox, boundary, distance, buffer, centroid, convexhull, and envelope. DeMarzi (2018) utilises a distance query to find food outlets within 10km of a given city (Figure 3).

```
MATCH (c:City)-[:IN_LOCATION]->(s:State), (c2:City)<-[:IN]-
(v:Vendor)-[:SELLS]->(f:Food {name:"Burrito"})
WHERE c.name = "Union City"
  AND s.name = "California"
  AND distance(c.location, c2.location) <= 10000
RETURN c2.name, v.name
```

Figure 3: *The final section of Cypher script used by DeMarzi to locate the food outlets within 10km of a given city (2018). In this case any food outlet selling burritos 10km from Union City in California were located. The Cypher synax is similar to that of SQL used by PostGIS in PostgreSQL.*

Again, the list of functions within PostGIS is much greater than Neo4j, however the main functions that would be expected are available. As previously mentioned, there is currently no support for 3D spatial objects, so there are no 3D queries.

To conclude, it is clear that the support provided by Neo4j successfully enables the import, storage, and querying of spatial data. It does this to a relatively high standard with the functionality being demonstrated by various projects (e.g. Lyon, 2015, 2016; DeMarzi, 2018). However, in many ways its functionality falls short of that currently provided by spatially enabled relational databases, such as PostGIS in PostgreSQL. Therefore, the use of Neo4j for handling spatial databases could be suitable for many applications, but its weaknesses and lacking functionality should be considered.

References

Agoub, A., Kunde, F. and Kada, M. (2016) 'Potential of Graph Databases in Representing and Enriching Standardized Geodata', Dreiländertagung der DGPF, der OVG und der SGPF, Bern, Switzerland.

Baas, B. (2012) 'NoSQL spatial – Neo4j versus PostGIS', M.S. Thesis. Geographic Information Management and Applications (GIMA), Delft University of Technology, Delft, Netherlands.

Baptista, C., Pires, C., Leite, D., Oliveira, M. and Lima, O. (2014) 'NoSQL Geographic Databases: An Overview', In Pourabbas, E. (ed.), *Geographical Information Systems: Trends and Technologies*, CRC Press Inc, Florida, pp. 73–103.

DeMarzi, M. (2018) 'Neo4j Geospatial Queries', Max De Marzi, [online] Available from: https://maxdemarzi.com/2018/03/21/neo4j-geospatial-queries/ (Accessed 14/04/18).

Lyon, W. (2015) 'Using Neo4j Spatial and Mapbox to search for businesses by location', [online] Available from: http://www.lyonwj.com/using-neo4j-spatial-and-leaflet-js-with-mapbox (Accessed 15/04/18).

Lyon, W. (2016) 'Using Neo4j Spatial Procedures in legis-graph-spatial', [online] Available from: http://www.lyonwj.com/2016/08/09/neo4j-spatial-procedures-congressional-boundaries/ (Accesses 15/04/18).

Neo4j (n.d.) 'Why Graph Databases?', Neo4j Graph Database Platform, [online] Available from: https://neo4j.com/why-graph-databases/ (Accessed 18/04/18).

Neo4j (2016) The Definitive Guide to Graph Databases, Neo Technologies.

Oracle (n.d.) 'Spatial Developer's Guide', [online] Available from: https://docs.oracle.com/cd/E11882_01/appdev.112/e11830/sdo_cs_concepts.htm#SPATL050 (Accessed 18/04/18).

PostGIS (n.d.) 'Chapter 4. Using PostGIS: Data Management and Queries', [online] Available from: https://postgis.net/docs/using_postgis_dbmanagement.html (Accessed 15/04/18).

Yuhanna, N., Leganza, G. and Austin, C. (2016) The Forrester WaveTM: Big Data NoSQL, Forrester Analyst Report.