Database Project

Ruth Anne Brennan

17329846

—

CS2014

—

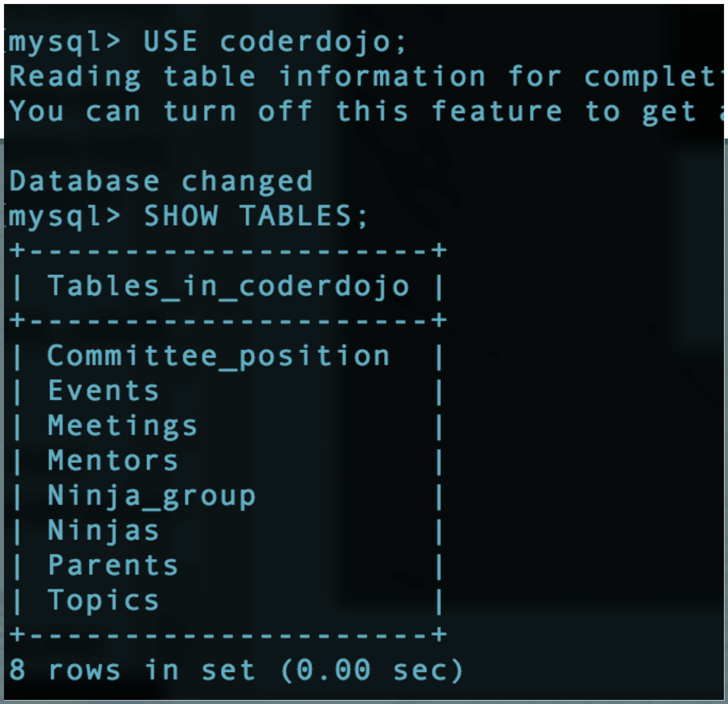
Information Management II

INSTRUCTIONS

For this project it was required to design and implement a database based on the following requirements:

- A minimum of 6 relational tables
- Appropriate implicit constraints
- Explicit semantic constraints
- At least one view
- A minimum of 5 tuples per table

It was then required to implement that database using MySQL. Finally, we were to design a functional dependency diagram and to write this report.



```
mysql> USE coderdojo;
Reading table information for complet
You can turn off this feature to get a

Database changed
mysql> SHOW TABLES;
+----------------------+
| Tables_in_coderdojo |
+----------------------+
| Committee_position   |
| Events               |
| Meetings             |
| Mentors              |
| Ninja_group          |
| Ninjas               |
| Parents              |
| Topics               |
+----------------------+
8 rows in set (0.00 sec)
```

Coderdojo Database

Coderdojo

I designed a database for the Trinity Branch of CoderDojo. CoderDojo is a global network of free, volunteer led, community-based programming clubs for young people. Anyone aged 6 to 16 can join a Dojo where they can learn to code, build a website, create an app or a game, and explore technology in an informal, creative, and social environment.

The Trinity CoderDojo is run and operated by Trinity students. Who act as the technical mentors at events as well as make up the committee that organizes the Dojo. The Committee consists of a Chairperson, an Information Officer (IO), a Child Protection Officer (CPO), a Promotional Officer (PRO) and a treasurer. They each handle the aspects of the Dojo that their title identifies, and Champion is the CoderDojo organization dubbed term for the head of the dojo. There are also 5 available Ordinary Committee Member Positions these are awarded to mentors who have contributed greatly to the Dojo.

The kids that attend the events that the Dojo runs are referred to as Ninja, a play on the name dojo. The ninjas are separated into groups based on their level and their main topic of interest. This allows the Dojo to run targeted events to help the ninjas in that group develop their skills. Ninjas are more than welcome to transfer between groups.

The group levels are divided up as follows 1-Creator, 2-Builder, 3-Developer and 4-Maker.

The Dojo also runs regular sessions which are events but the group that attends and the topic of the session are unspecified. At sessions we practice self-directed learning this means that the ninjas can decide what they want to work on and will be assisted a technical mentor versed in that area. Always open to new opportunities, occasionally a mentor and ninja preach a new language or tool together. We do however try to have rough syllabuses for the most common topics that our Ninjas cover. A syllabus simply lists resources and their difficulty in a manner that allows the Ninja to develop that skill.

For example with Scratch (A block-based visual programming language and online community targeted primarily at children) the syllabus would recommend they start with the basic tutorial and then proceed to follow tutorials on raspberry pi in order of difficulty but this would not stop a child who wanted to start developing a project of their own without following the syllabus.

Such topics have a Syllabus Director, this is a mentor who is in charge of updating the syllabus, gathering resources for the topic and if needed training new mentors in the area.

Syllabus directors are usually more advanced mentors. The Dojos mentor hierarchy guide is the following; a White Belt is a relative beginner mentor, a Yellow Belt is a mentor who has been to a few events, a senior mentor would be a Green Belt followed by a Purple Belt who is an experienced mentor who has taken on responsibilities at events (such as presentations) and then there is a Black Belt, an experienced mentor who has taken on responsibilities outside of events e.g. a committee position.

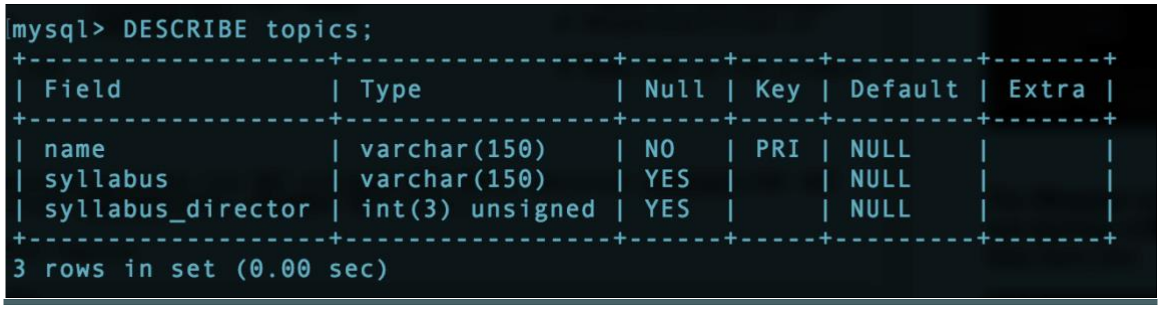
The committee members hold meetings at which they plan, organize and restructure the dojo. The agenda and minutes are available to other mentors. The former to view and subsequently attend the meeting if they wish and the latter to inform them what the committee was about.

Our dojos is very stringent about the safety of it’s ninjas, so our mentor applicants must be garda vetted before they are considered mentors and allowed to attend dojo. All mentors must also take an online child protection course so that they are familiar with the rules of our dojo. In keeping with these guidelines the dojo cannot contact the Ninjas directly and instead must contact the parents. So the dojo keeps a record of one parent for every child: name, email address and phone number.

Database

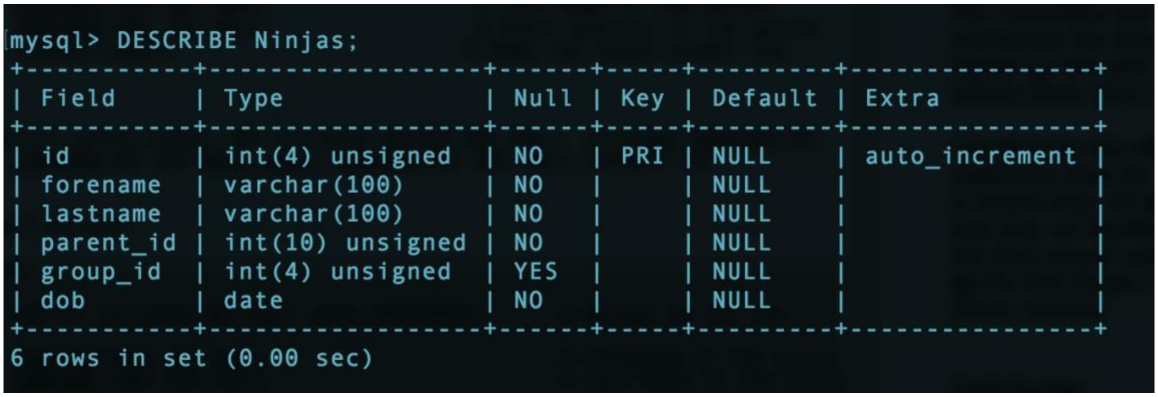
I modelled the database to contain 8 tables; Topics, Ninjas, Parents, Mentors, Ninja Group, Committee Positions, Events and Meetings.

The **Topic** table keeps track of the name of the topic (which is unique to each topic), a link to the syllabus which is hosted on google drive and the id of the mentor who acts as the syllabus director for the topic.



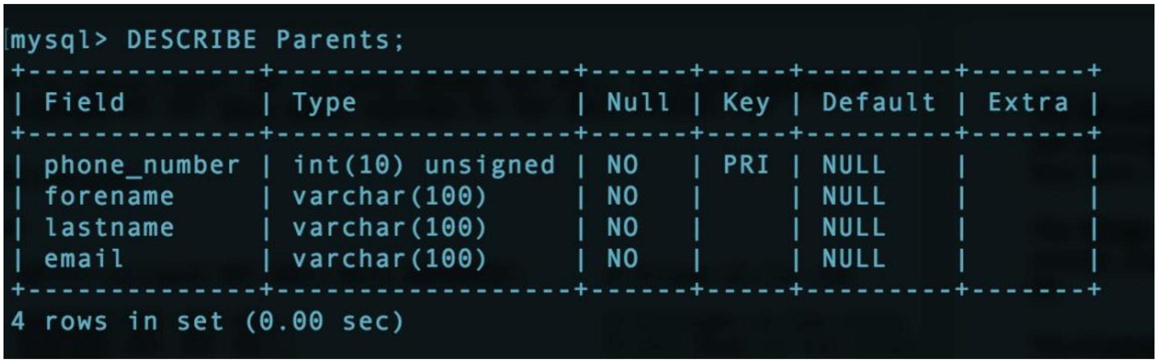
```
[mysql> DESCRIBE topics;
+-------------------+------------------+------+-----+---------+-------+
| Field             | Type             | Null | Key | Default | Extra |
+-------------------+------------------+------+-----+---------+-------+
| name              | varchar(150)     | NO   | PRI | NULL    |       |
| syllabus          | varchar(150)     | YES  |     | NULL    |       |
| syllabus_director | int(3) unsigned  | YES  |     | NULL    |       |
+-------------------+------------------+------+-----+---------+-------+
3 rows in set (0.00 sec)
```

The **Ninja** table documents the unique id of a Ninja (a four digit number not starting with 0), the id of their parent in the parents table, the id of the group of ninjas of similar interest and ability to which they are a part and their date of birth.



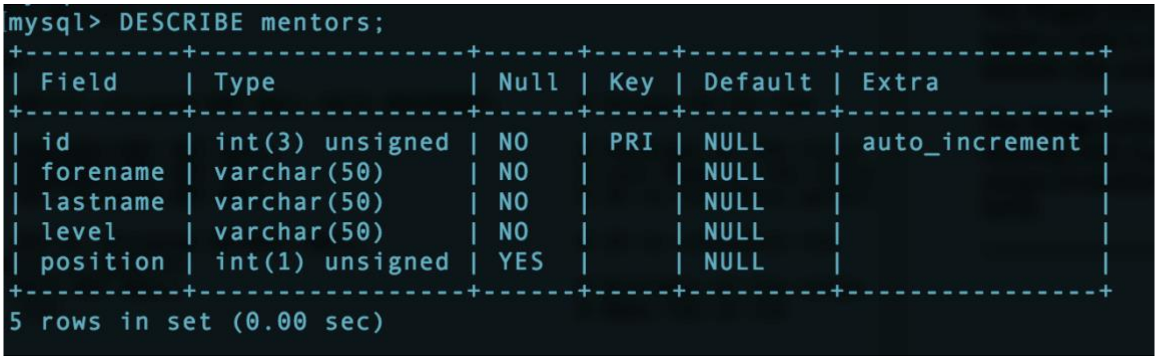
```
[mysql> DESCRIBE Ninjas;
+-----------+------------------+------+-----+---------+----------------+
| Field     | Type             | Null | Key | Default | Extra          |
+-----------+------------------+------+-----+---------+----------------+
| id        | int(4) unsigned  | NO   | PRI | NULL    | auto_increment |
| forename  | varchar(100)     | NO   |     | NULL    |                |
| lastname  | varchar(100)     | NO   |     | NULL    |                |
| parent_id | int(10) unsigned | NO   |     | NULL    |                |
| group_id  | int(4) unsigned  | YES  |     | NULL    |                |
| dob       | date             | NO   |     | NULL    |                |
+-----------+------------------+------+-----+---------+----------------+
6 rows in set (0.00 sec)
```

The **Parent** table as mentioned above keeps a record of a parents name, email address and phone number.



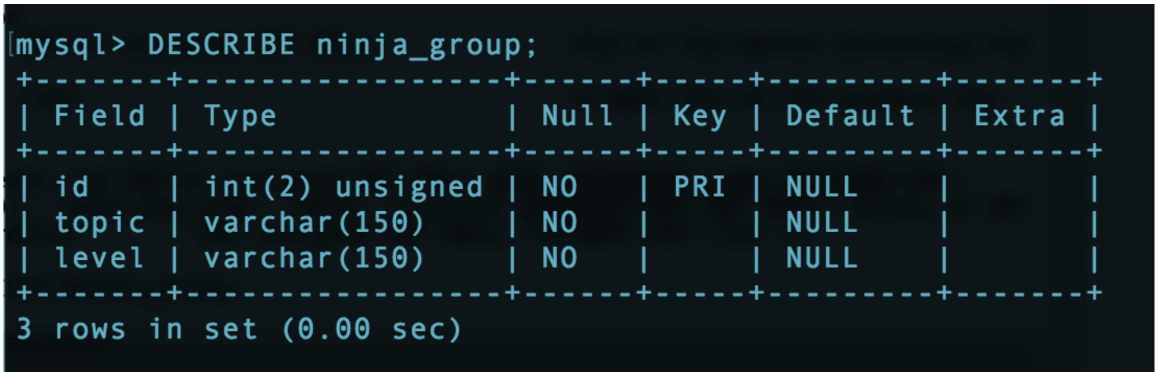
```
[mysql> DESCRIBE Parents;
+--------------+------------------+------+-----+---------+-------+
| Field        | Type             | Null | Key | Default | Extra |
+--------------+------------------+------+-----+---------+-------+
| phone_number | int(10) unsigned | NO   | PRI | NULL    |       |
| forename     | varchar(100)     | NO   |     | NULL    |       |
| lastname     | varchar(100)     | NO   |     | NULL    |       |
| email        | varchar(100)     | NO   |     | NULL    |       |
+--------------+------------------+------+-----+---------+-------+
4 rows in set (0.00 sec)
```

The **Mentor** table keeps track of the mentor id (for this we a three digit number not starting with 0), the mentor name, their level and their committee position if they have one.



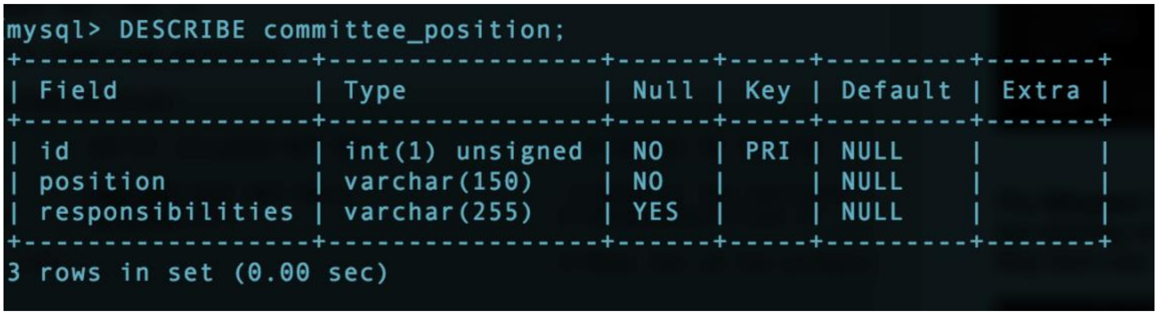
```
[mysql> DESCRIBE mentors;
+----------+-----------------+------+-----+---------+----------------+
| Field    | Type            | Null | Key | Default | Extra          |
+----------+-----------------+------+-----+---------+----------------+
| id       | int(3) unsigned | NO   | PRI | NULL    | auto_increment |
| forename | varchar(50)     | NO   |     | NULL    |                |
| lastname | varchar(50)     | NO   |     | NULL    |                |
| level    | varchar(50)     | NO   |     | NULL    |                |
| position | int(1) unsigned | YES  |     | NULL    |                |
+----------+-----------------+------+-----+---------+----------------+
5 rows in set (0.00 sec)
```

The **Ninja group** keeps a record of the level of experience it's ninjas have overall, along with their main topic of interest and it also has a unique two digit id.



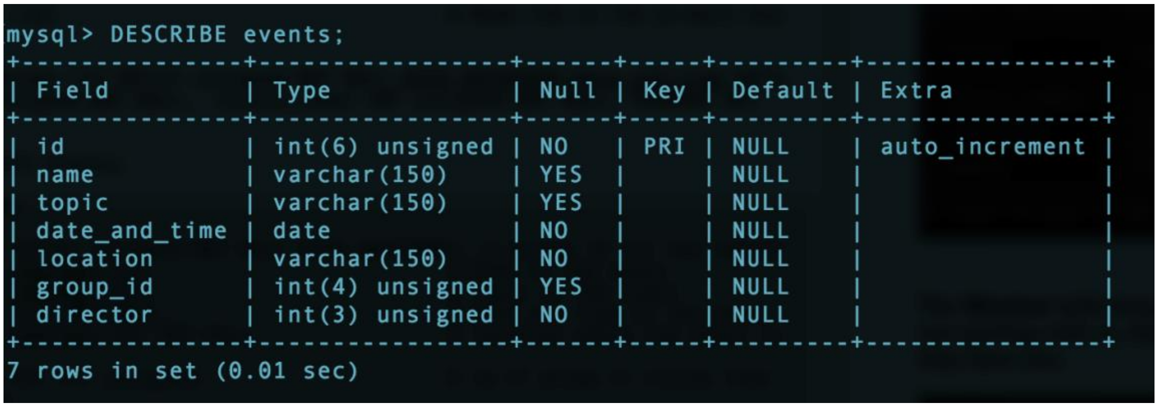
```
[mysql> DESCRIBE ninja_group;
+-------+-----------------+------+-----+---------+-------+
| Field | Type            | Null | Key | Default | Extra |
+-------+-----------------+------+-----+---------+-------+
| id    | int(2) unsigned | NO   | PRI | NULL    |       |
| topic | varchar(150)    | NO   |     | NULL    |       |
| level | varchar(150)    | NO   |     | NULL    |       |
+-------+-----------------+------+-----+---------+-------+
3 rows in set (0.00 sec)
```

The **Committee Positions** table stories the responsibilities of each position and has a unique one digit id to identify them.



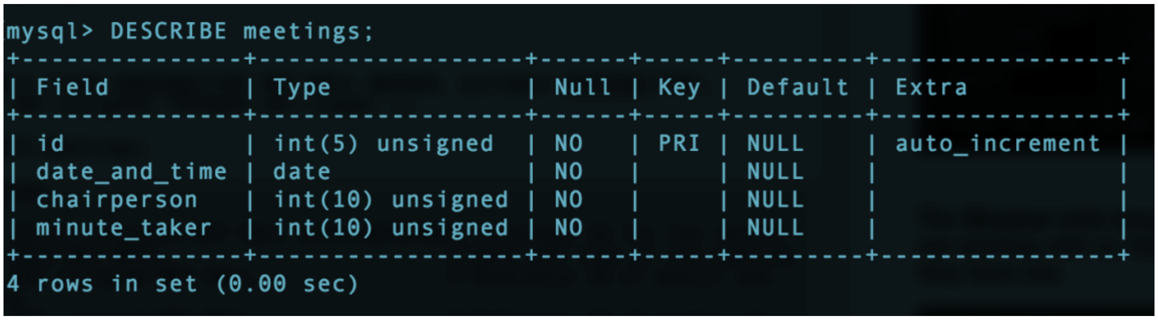
```
mysql> DESCRIBE committee_position;
+------------------+-----------------+------+-----+---------+-------+
| Field            | Type            | Null | Key | Default | Extra |
+------------------+-----------------+------+-----+---------+-------+
| id               | int(1) unsigned | NO   | PRI | NULL    |       |
| position         | varchar(150)    | NO   |     | NULL    |       |
| responsibilities | varchar(255)    | YES  |     | NULL    |       |
+------------------+-----------------+------+-----+---------+-------+
3 rows in set (0.00 sec)
```

The **Events** table keeps a record of the name of an event or a session, the topic covered (if only one), the date, the location, the id of the group that attended (if it was restricted to a single group) and the director of the event. There is also a unique 6 figure id where the first digit is not 0 used to uniquely identify the event.



```
mysql> DESCRIBE events;
+---------------+-----------------+------+-----+---------+----------------+
| Field         | Type            | Null | Key | Default | Extra          |
+---------------+-----------------+------+-----+---------+----------------+
| id            | int(6) unsigned | NO   | PRI | NULL    | auto_increment |
| name          | varchar(150)    | YES  |     | NULL    |                |
| topic         | varchar(150)    | YES  |     | NULL    |                |
| date_and_time | date            | NO   |     | NULL    |                |
| location      | varchar(150)    | NO   |     | NULL    |                |
| group_id      | int(4) unsigned | YES  |     | NULL    |                |
| director      | int(3) unsigned | NO   |     | NULL    |                |
+---------------+-----------------+------+-----+---------+----------------+
7 rows in set (0.01 sec)
```

Finally, the **Meetings** table details the date, the chairperson mentor id, the minute taker mentor id and a unique five digit id to identify the meeting.



```
mysql> DESCRIBE meetings;
+---------------+-----------------+------+-----+---------+----------------+
| Field         | Type            | Null | Key | Default | Extra          |
+---------------+-----------------+------+-----+---------+----------------+
| id            | int(5) unsigned | NO   | PRI | NULL    | auto_increment |
| date_and_time | date            | NO   |     | NULL    |                |
| chairperson   | int(10) unsigned | NO  |     | NULL    |                |
| minute_taker  | int(10) unsigned | NO  |     | NULL    |                |
+---------------+-----------------+------+-----+---------+----------------+
4 rows in set (0.00 sec)
```

Diagrams

Normalization

Database normalization is the process of restructuring a relational database in accordance with a series of normal forms in order to reduce data redundancy and improve data integrity.

- First Normal Form

A relation is in first normal form if and only if the domain of each attribute contains only atomic (indivisible) values, and the value of each attribute contains only a single value from that domain.

- Second Normal Form

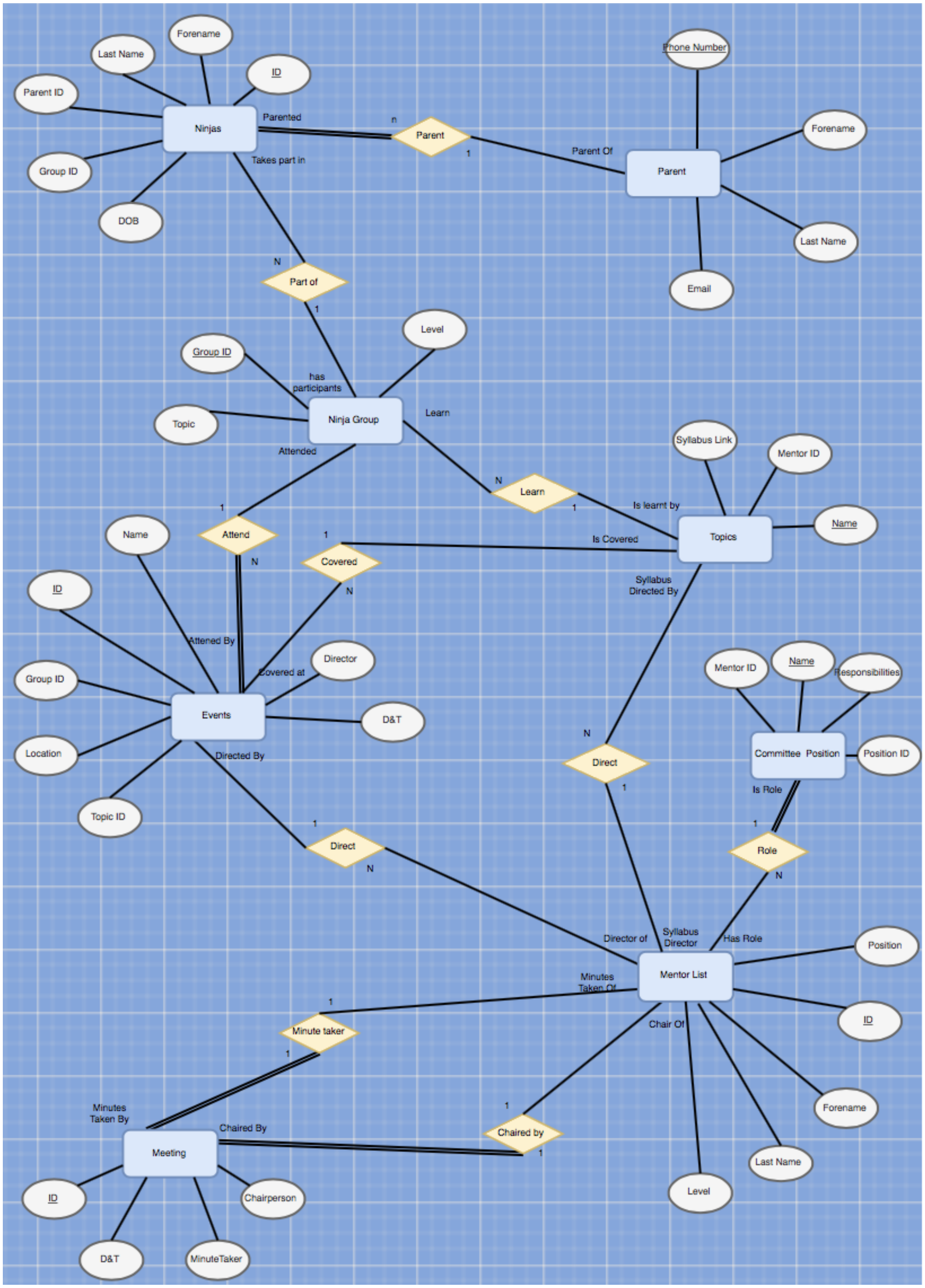
The table is in first normal form and all the columns depend on the table's primary key.

- Third Normal Form

The table is in second normal form and all of its columns are not transitively dependent on the primary key.

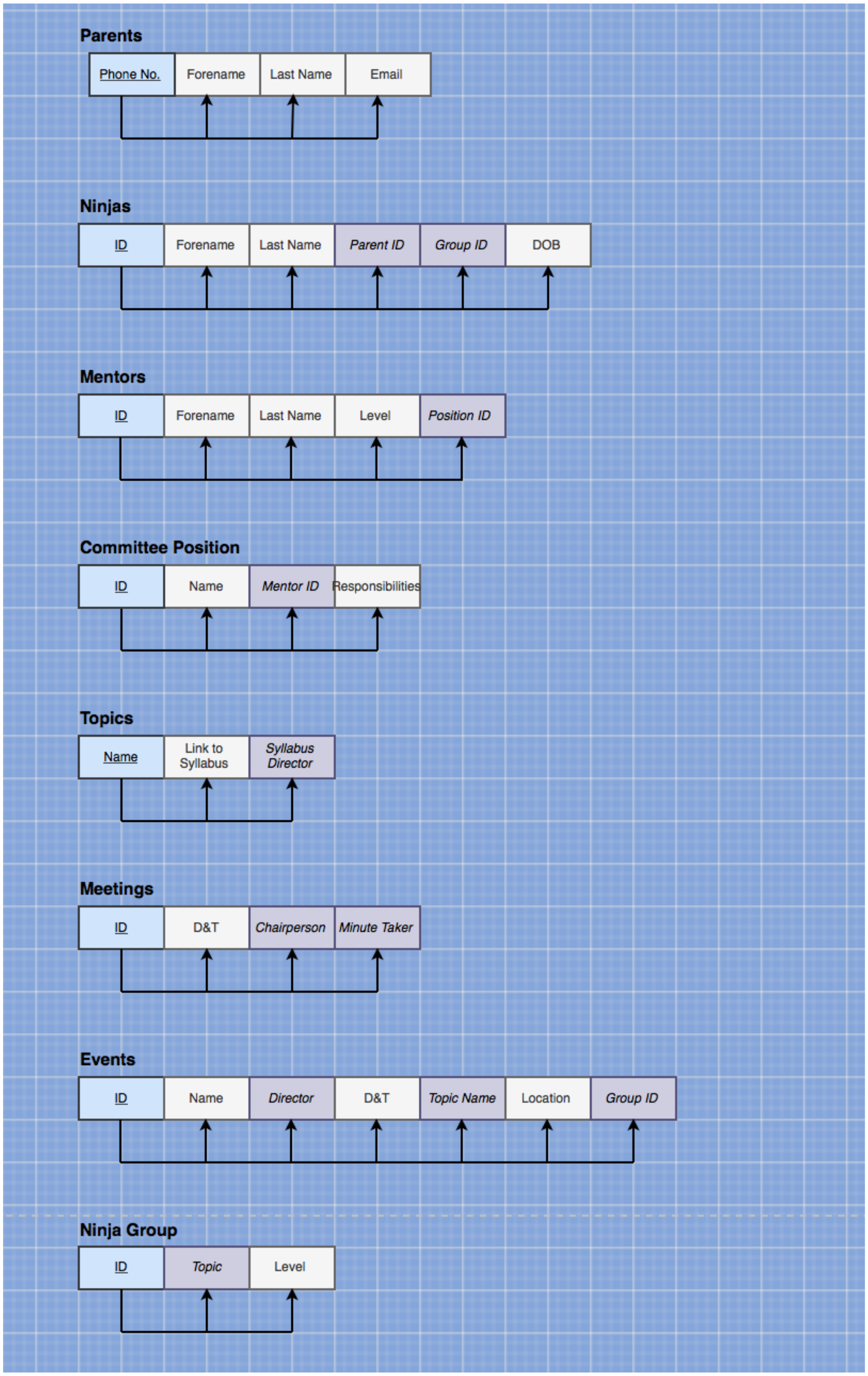
My relational schema was analysed based in order to minimise redundancy. For example, in my initial design, I planned to keep track of parents and ninjas in the same table. To have the parent name number and email as attributes in the ninja table. However, one parent can have multiple children so to reduce repeat data and redundancy I split them into two table and use the parent ID to uniquely identity the parent in the Ninja table.

Relational Schema



*Underlined Attribute indicates a Primary Key

Functional Dependency



Parents

Phone No.	Forename	Last Name	Email

Ninjas

ID	Forename	Last Name	*Parent ID*	*Group ID*	DOB

Mentors

ID	Forename	Last Name	Level	*Position ID*

Committee Position

ID	Name	*Mentor ID*	Responsibilities

Topics

Name	Link to Syllabus	*Syllabus Director*

Meetings

ID	D&T	*Chairperson*	*Minute Taker*

Events

ID	Name	*Director*	D&T	*Topic Name*	Location	*Group ID*

Ninja Group

ID	*Topic*	Level

*Underlined attribute indicates a primary key and the italic attributes indicate a foreign key

Semantic Constraints

There are a few semantic constraints that apply to this database.

Entity Integrity Constraints

The Entity integrity constraint states that primary key value can’t be null. This is because the primary key value is used to identify individual rows in relation and if the primary key has a null value, then we can’t identify these rows. A table can contain a null value other than the primary key field.

1. UNIQUE

Entity values such as that of an ID have to be unique so that they are easily identifiable.

```
CREATE TABLE Parents
(
        phone_number    INT unsigned UNIQUE NOT NULL,
        forename        VARCHAR(50) NOT NULL
        lastname        VARCHAR(50) NOT NULL
        email        VARCHAR(50) NOT NULL
);
```

2. NOT NULL

Again, we look to ensure that some values are not null to maintain the integrity of the database for example the value for forename and last name for all parents, ninjas and mentors and all the titled committee positions must have a valid mentor ID.

```
CREATE TABLE Parents
(
        phone_number    INT unsigned UNIQUE NOT NULL,
        forename        VARCHAR(50) NOT NULL
        lastname        VARCHAR(50) NOT NULL
        email        VARCHAR(50) NOT NULL
);
```

Referential Integrity Constraints

These are defined as part of an association between two entity types. The definition specifies: the principal end of the constraint: the foreign key.

Code

```
ALTER TABLE Ninjas ADD CONSTRAINT parent_id FOREIGN KEY (parent_id) REFERENCES
Parents(phone_number);
ALTER TABLE Ninjas ADD CONSTRAINT ninja_group_id FOREIGN KEY (group_id) REFERENCES
Ninja_group(id);
ALTER TABLE Committee_position ADD CONSTRAINT mentor_position FOREIGN KEY (position) REFERENCES
Committee_position(id);
ALTER TABLE Topics ADD CONSTRAINT topic_director FOREIGN KEY (syllabus_director) REFERENCES
Mentors(id);
ALTER TABLE Meetings ADD CONSTRAINT meeting_chair FOREIGN KEY (chairperson) REFERENCES
Mentors(id);
ALTER TABLE Meetings ADD CONSTRAINT meeting_minutes FOREIGN KEY (minute_taker) REFERENCES
Mentors(id);
ALTER TABLE Events ADD CONSTRAINT event_director FOREIGN KEY (director) REFERENCES Mentors(id);
ALTER TABLE Events ADD CONSTRAINT event_topic FOREIGN KEY (topic) REFERENCES Topics(name);
ALTER TABLE Events ADD CONSTRAINT event_group FOREIGN KEY (group_id) REFERENCES
Ninja_group(id);
ALTER TABLE Ninja_group ADD CONSTRAINT group_topic FOREIGN KEY (topic) REFERENCES Topics(name);
```

Table Constraints

We can use CHECK to maintain correctness in our number (int) attributes.

```
ALTER TABLE Committee_position ADD CONSTRAINT committee_id CHECK (id<7 AND id>0);
ALTER TABLE Mentors ADD CONSTRAINT mentor_id CHECK (id<999 AND id>99);
ALTER TABLE Mentors ADD CONSTRAINT position_id CHECK (position<7 AND position>0);
ALTER TABLE Committee_position ADD CONSTRAINT committee_id CHECK (id<7 AND id>0);
ALTER TABLE Ninjas ADD CONSTRAINT ninja_age CHECK (TIMESTAMPDIFF(YEAR, dob, '2019-11-06') > 5
AND TIMESTAMPDIFF(YEAR, dob, '2019-11-06') < 17);
ALTER TABLE Ninjas ADD CONSTRAINT ninja_id CHECK (id>999 && id<9999);
ALTER TABLE Ninja_group ADD CONSTRAINT ninja_group_id CHECK (id>9 && id<99);
ALTER TABLE Events ADD CONSTRAINT event_id CHECK (id>99999 && id<999999);
ALTER TABLE Meetings ADD CONSTRAINT meeting_id CHECK (id>9999 && id<99999);
```

There are also a number of constraints on String attributes.

```
ALTER TABLE Mentors ADD CONSTRAINT mentor_level CHECK(level IN ('White', 'Yellow', 'Green', 'Purple',
'Black'));
```

```
ALTER TABLE Committee_position ADD CONSTRAINT position_name CHECK(position IN ('Champion', 'IO',
'CPO', 'PRO', 'PRO', 'Treasurer', 'OCM'));
ALTER TABLE Topics ADD CONSTRAINT topic_director CHECK(level IN ('Creator', 'Builder', 'Maker',
'Developer'));
```

Database Security Commands for Access and Security Policy

We now take a look at database security. Integrity is concerned with accidental corruption. Security is concerned with deliberate corruption. To deal with this we look to some examples and to describe the intended security policy.

We will first look at relation level privileges. We do this by allowing only each committee member to have a user in order to prevent unauthorized access to the data. We would restrict read privileges to committee members, restrict read and modification and reference privileges to the three senior committee positions: Champion, Information Officer and the Child Protection Officer. Read Privileges give an account the ability to use SELECT to retrieve rows from this relation. Modification Privileges give an account the ability to use INSERT, UPDATE and DELETE to modify rows in this relation. Reference Privileges give an account the ability to refer to this relation when specifying integrity constraints.

The Database administrator positions (DBA) would be taken on by the information officer and the champion. They would create users for each of the committee members -Champion, IO, CPO, PRO, treasurer and 5 OCMs.

```
GRANT SELECT on Mentors TO treasurer;

GRANT CREATE TABLE TO champion WITH GRANT OPTION;
```

From this the treasurer has the ability to access and filter the mentor table and champion has the ability to create a new table and to share that privilege with other users, the treasurer cannot share their privilege with other users.

If we wanted to take away a privilege, we would use the following code.

```
REVOKE SELECT on Mentors FROM treasurer;
```

Adopting this type of approach and allowing different users to have different privileges tailored to their job within the dojo will ensure greater security for the data stored in the database. The creation of views from tables further enhances the security on the information stored in the database as it allows the DBA (Database Administrator) to restrict the attributed that can be seen by the different staff members.

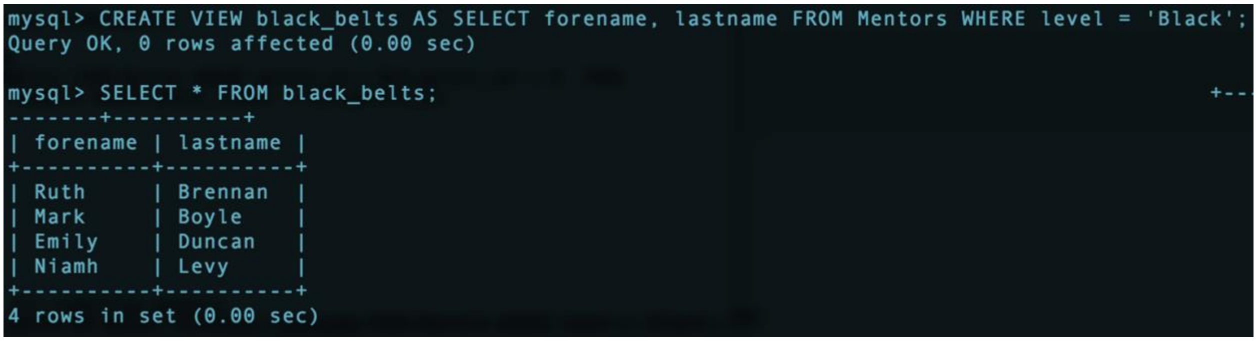
Examples

View Creation

The IO is able to use views to grant partial access to information contained in the relation as was described above. The views that were created are:

- A view to show the names of the mentors with a black belt:

  CREATE VIEW black_belts AS SELECT forename, lastname FROM Mentors WHERE level = 'Black';

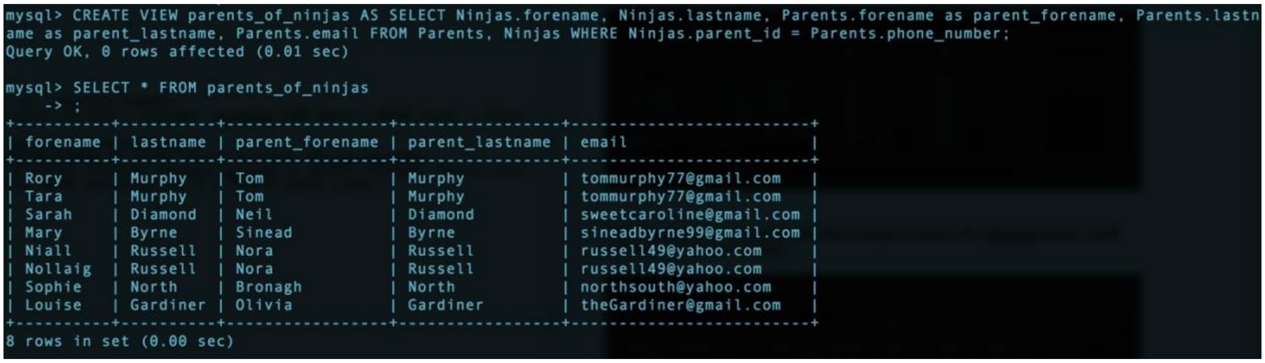


```
mysql> CREATE VIEW black_belts AS SELECT forename, lastname FROM Mentors WHERE level = 'Black';
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT * FROM black_belts;                                                          +--
-------+----------+
| forename | lastname |
+----------+----------+
| Ruth     | Brennan  |
| Mark     | Boyle    |
| Emily    | Duncan   |
| Niamh    | Levy     |
+----------+----------+
4 rows in set (0.00 sec)
```

- A view to show the ninjas, parents and parents emails:

  CREATE VIEW parents_of_ninjas AS SELECT Ninjas.forename, Ninjas.lastname, Parents.forename as parent_forename, Parents.lastname as parent_lastname, Parents.email FROM Parents, Ninjas WHERE Ninjas.parent_id = Parents.phone_number;

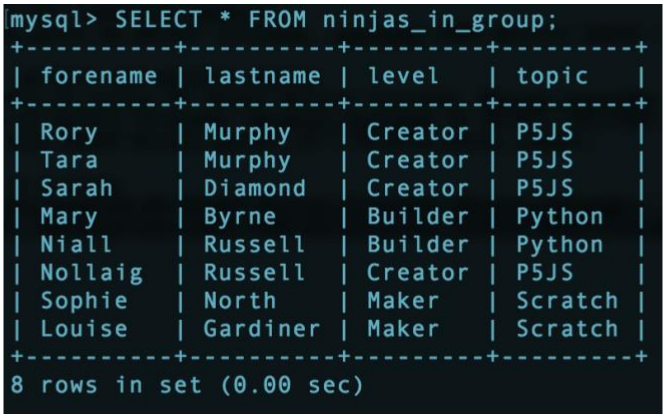


```
mysql> CREATE VIEW parents_of_ninjas AS SELECT Ninjas.forename, Ninjas.lastname, Parents.forename as parent_forename, Parents.lastn
ame as parent_lastname, Parents.email FROM Parents, Ninjas WHERE Ninjas.parent_id = Parents.phone_number;
Query OK, 0 rows affected (0.01 sec)

mysql> SELECT * FROM parents_of_ninjas
    -> ;
+----------+----------+-----------------+-----------------+-------------------------+
| forename | lastname | parent_forename | parent_lastname | email                   |
+----------+----------+-----------------+-----------------+-------------------------+
| Rory     | Murphy   | Tom             | Murphy          | tommurphy77@gmail.com   |
| Tara     | Murphy   | Tom             | Murphy          | tommurphy77@gmail.com   |
| Sarah    | Diamond  | Neil            | Diamond         | sweetcaroline@gmail.com |
| Mary     | Byrne    | Sinead          | Byrne           | sineadbyrne99@gmail.com |
| Niall    | Russell  | Nora            | Russell         | russell49@yahoo.com     |
| Nollaig  | Russell  | Nora            | Russell         | russell49@yahoo.com     |
| Sophie   | North    | Bronagh         | North           | northsouth@yahoo.com    |
| Louise   | Gardiner | Olivia          | Gardiner        | theGardiner@gmail.com   |
+----------+----------+-----------------+-----------------+-------------------------+
8 rows in set (0.00 sec)
```

- A view to show the level and main topic interest of all ninjas:

CREATE VIEW ninjas_in_group AS SELECT Ninjas.forename, Ninjas.lastname, Ninja_group.level, Ninja_group.topic FROM Ninja_group, Ninjas WHERE Ninjas.group_id = Ninja_group.id;

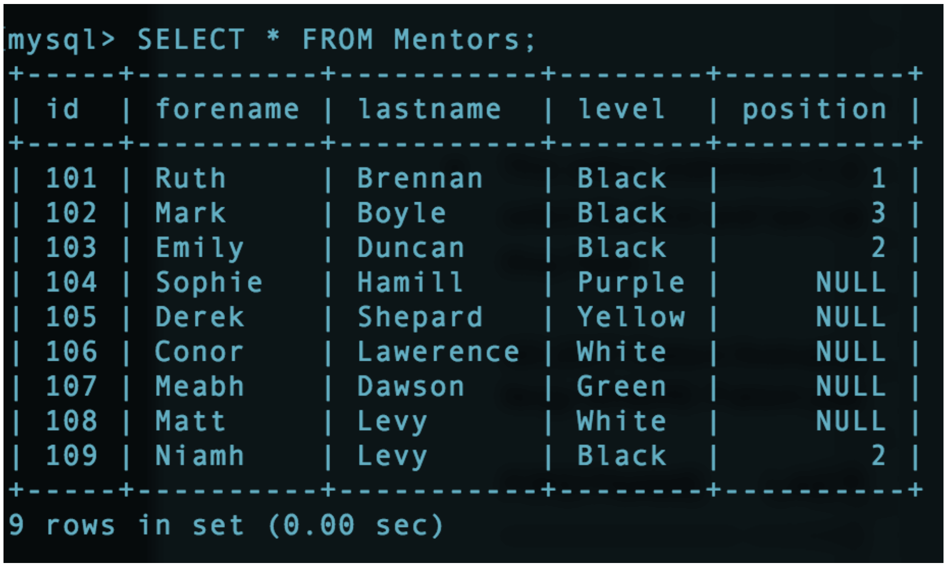


```
[mysql> SELECT * FROM ninjas_in_group;
+----------+----------+---------+---------+
| forename | lastname | level   | topic   |
+----------+----------+---------+---------+
| Rory     | Murphy   | Creator | P5JS    |
| Tara     | Murphy   | Creator | P5JS    |
| Sarah    | Diamond  | Creator | P5JS    |
| Mary     | Byrne    | Builder | Python  |
| Niall    | Russell  | Builder | Python  |
| Nollaig  | Russell  | Creator | P5JS    |
| Sophie   | North    | Maker   | Scratch |
| Louise   | Gardiner | Maker   | Scratch |
+----------+----------+---------+---------+
8 rows in set (0.00 sec)
```

Relational Select and Table Join Operations

- Select all from Mentors

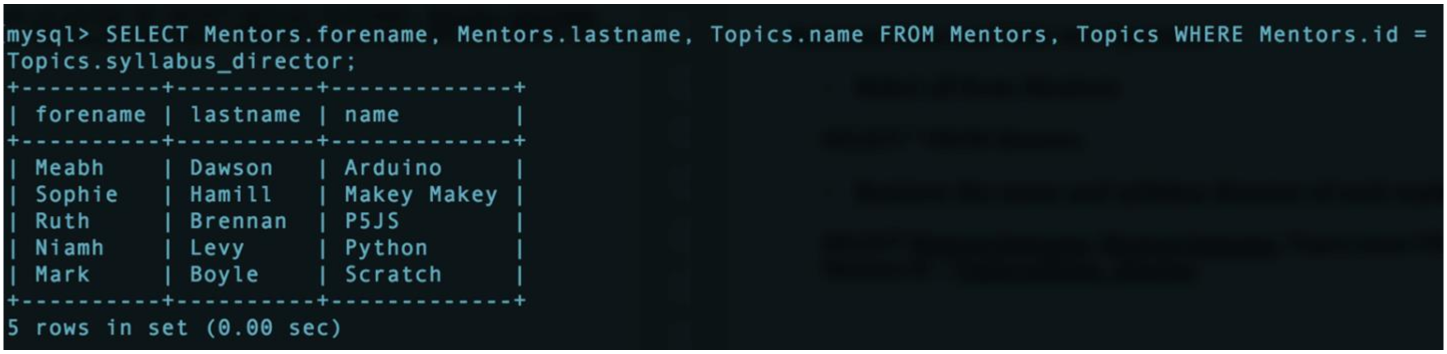
SELECT * FROM Mentors;



```
mysql> SELECT * FROM Mentors;
+-----+----------+-----------+--------+----------+
| id  | forename | lastname  | level  | position |
+-----+----------+-----------+--------+----------+
| 101 | Ruth     | Brennan   | Black  |        1 |
| 102 | Mark     | Boyle     | Black  |        3 |
| 103 | Emily    | Duncan    | Black  |        2 |
| 104 | Sophie   | Hamill    | Purple |     NULL |
| 105 | Derek    | Shepard   | Yellow |     NULL |
| 106 | Conor    | Lawerence | White  |     NULL |
| 107 | Meabh    | Dawson    | Green  |     NULL |
| 108 | Matt     | Levy      | White  |     NULL |
| 109 | Niamh    | Levy      | Black  |        2 |
+-----+----------+-----------+--------+----------+
9 rows in set (0.00 sec)
```

- Retrieve the name and syllabus director of each topic

```
SELECT Mentors.forename, Mentors.lastname, Topics.name FROM Mentors, Topics WHERE Mentors.id =
Topics.syllabus_director;
```



```
mysql> SELECT Mentors.forename, Mentors.lastname, Topics.name FROM Mentors, Topics WHERE Mentors.id =
Topics.syllabus_director;
+----------+----------+-------------+
| forename | lastname | name        |
+----------+----------+-------------+
| Meabh    | Dawson   | Arduino     |
| Sophie   | Hamill   | Makey Makey |
| Ruth     | Brennan  | P5JS        |
| Niamh    | Levy     | Python      |
| Mark     | Boyle    | Scratch     |
+----------+----------+-------------+
5 rows in set (0.00 sec)
```

Trigger Command

To delete parents when ninja is deleted if they don't have other children

```
DELIMITER $$
CREATE
TRIGGER delete_parent AFTER DELETE
  ON Ninjas
FOR EACH ROW BEGIN
    IF (SELECT COUNT(*) FROM Ninjas WHERE parent_id = OLD.parent_id) = 0  THEN
      DELETE FROM PARENTS WHERE phone_number = OLD.parent_id;
    END IF;
  END$$
DELIMITER;
```

Appendix

Appendix A – Table Creation

```
#
#       Create the Tables the Database
#

#DROP TABLE IF EXISTS Parents;

CREATE TABLE Parents
(
  phone_number      INT unsigned UNIQUE NOT NULL,       # Unique ID for the record and phone
number for contact
  forename          VARCHAR(100) NOT NULL,              # Forename of the parent
  lastname           VARCHAR(100) NOT NULL,             # Last Name of the parent
  email              VARCHAR(100) NOT NULL,              # Email address of the parent
  PRIMARY KEY        (phone_number)                  # Make the phone number the primary key
);


#DROP TABLE IF EXISTS Ninjas;

CREATE TABLE Ninjas
(
  id                 INT(4) unsigned NOT NULL,           # Unique ID for the record
  forename           VARCHAR(100) NOT NULL,              # Forename of the ninja
  lastname           VARCHAR(100) NOT NULL,              # Last Name of the ninja
  parent_id          INT unsigned NOT NULL,              # ID to reference parent of the child
  group_id           INT(4) unsigned DEFAULT NULL,        # ID to reference the group which the
child is part of
  dob                DATE NOT NULL,                      # Birthday of the ninja
  PRIMARY KEY         (id)                               # Make the id the primary key
);

#DROP TABLE IF EXISTS Mentors;

CREATE TABLE Mentors
(
  id                 INT(3) unsigned NOT NULL,           # Unique ID for the record
  forename           VARCHAR(50) NOT NULL,               # Forename of the ninja
  lastname           VARCHAR(50) NOT NULL,               # Last Name of the ninja
  level              VARCHAR(50) NOT NULL,               # Level of experience of mentor
  position           INT(1) unsigned DEFAULT NULL,       # committee position that a mentor may
hold
  PRIMARY KEY        (id)                                # Make the id the primary key
);

#DROP TABLE IF EXISTS Committee_position;

CREATE TABLE Committee_position
(
  id                 INT(1) unsigned NOT NULL,           # Unique ID for the record
  position           VARCHAR(150) NOT NULL,              # Name of the position
  responsibilities   VARCHAR(255),                       # Responsibilities of position
  PRIMARY KEY        (id)                                # Make the id the primary key

);

#DROP TABLE IF EXISTS Topics;

CREATE TABLE Topics
(
  name               VARCHAR(150) NOT NULL UNIQUE,   # Unique ID for the record
  syllabus           VARCHAR(150) DEFAULT NULL,          # Link to the syllabus
```

```
 syllabus_director  INT(3) unsigned,                        # Mentor in charge of syllabus
 PRIMARY KEY        (name)                                  # Make the name the primary key
);

#DROP TABLE IF EXISTS Meetings;

CREATE TABLE Meetings
(
 id                 INT(5) unsigned NOT NULL,      # Unique ID for the record
 date               DATE NOT NULL,                 # Date and time of the meeting
 chairperson        INT unsigned NOT NULL,         # ReferenceID of mentor who chaired meeting
 minute_taker       INT unsigned NOT NULL,         # Reference ID of mentor who did minutes
 PRIMARY KEY        (id)                           # Make the id the primary key
);

#DROP TABLE IF EXISTS Events;

CREATE TABLE Events
(
 id                 INT(6) unsigned NOT NULL,      # Unique ID for the record
 name               VARCHAR(150),                  # Name of the event
 topic              VARCHAR(150),                  # topic of the event
 date               DATE NOT NULL,                 # date and time of the event
 location           VARCHAR(150) NOT NULL,         # location where the event is held
 group_id           INT (4) unsigned,              # id of group of ninjas that attend the session
 director           INT(3) unsigned NOT NULL,      #id of the mentor directing the session
 PRIMARY KEY         (id)                          # Make the id the primary key
);

#DROP TABLE IF EXISTS Ninja_group;

CREATE TABLE Ninja_group
(
 id                 INT(2) unsigned NOT NULL,      # Unique ID for the record
 topic              VARCHAR(150) NOT NULL,         # Topic that the group covers
 level              VARCHAR(150) NOT NULL,         # Level of the group
 PRIMARY KEY        (id)                           # Make the id the primary key
);
```

Appendix B – Constraints

```
#
#       Constraints (Checks)
#

ALTER TABLE Committee_position ADD CONSTRAINT committee_id CHECK (id<7 AND id>0);

ALTER TABLE Mentors ADD CONSTRAINT mentor_id CHECK (id<999 AND id>99);
ALTER TABLE Mentors ADD CONSTRAINT position_id CHECK (position<7 AND position>0);
ALTER TABLE Mentors ADD CONSTRAINT mentor_level CHECK(level IN ('White', 'Yellow', 'Green', 'Purple', 'Black'));

ALTER TABLE Committee_position ADD CONSTRAINT position_name CHECK(position IN ('Champion', 'IO', 'CPO', 'PRO', 'PRO', 'Treasurer', 'OCM'));
ALTER TABLE Committee_position ADD CONSTRAINT committee_id CHECK (id<7 AND id>0);

ALTER TABLE Topics ADD CONSTRAINT topic_director CHECK(level IN ('Creator', 'Builder', 'Maker', 'Developer'));

ALTER TABLE Ninjas ADD CONSTRAINT ninja_age CHECK (TIMESTAMPDIFF(YEAR, dob, '2019-11-06') > 5 AND TIMESTAMPDIFF(YEAR, dob, '2019-11-06') < 17);
ALTER TABLE Ninjas ADD CONSTRAINT ninja_id CHECK (id>999 && id<9999);

ALTER TABLE Ninja_group ADD CONSTRAINT ninja_group_id CHECK (id>9 && id<99);

ALTER TABLE Events ADD CONSTRAINT event_id CHECK (id>99999 && id<999999);

ALTER TABLE Meetings ADD CONSTRAINT meeting_id CHECK (id>9999 && id<99999);


#
#	Constraints (References)
#

ALTER TABLE Ninjas ADD CONSTRAINT parent_id FOREIGN KEY (parent_id) REFERENCES Parents(phone_number);
ALTER TABLE Ninjas ADD CONSTRAINT ninja_group_id FOREIGN KEY (group_id) REFERENCES Ninja_group(id);

ALTER TABLE Committee_position ADD CONSTRAINT mentor_position FOREIGN KEY (position) REFERENCES Committee_position(id);

ALTER TABLE Topics ADD CONSTRAINT topic_director FOREIGN KEY (syllabus_director) REFERENCES Mentors(id);

ALTER TABLE Meetings ADD CONSTRAINT meeting_chair FOREIGN KEY (chairperson) REFERENCES Mentors(id);
ALTER TABLE Meetings ADD CONSTRAINT meeting_minutes FOREIGN KEY (minute_taker) REFERENCES Mentors(id);

ALTER TABLE Events ADD CONSTRAINT event_director FOREIGN KEY (director) REFERENCES Mentors(id);
ALTER TABLE Events ADD CONSTRAINT event_topic FOREIGN KEY (topic) REFERENCES Topics(name);
ALTER TABLE Events ADD CONSTRAINT event_group FOREIGN KEY (group_id) REFERENCES Ninja_group(id);
ALTER TABLE Ninja_group ADD CONSTRAINT group_topic FOREIGN KEY (topic) REFERENCES
Topics(name);
```

Appendix C – Populating the Database

```
#Populate table Parents
INSERT INTO Parents Values(0879619937, 'Sinead', 'Byrne', 'sineadbyrne99@gmail.com');
INSERT INTO Parents Values(0879248761, 'Tom', 'Murphy', 'tommurphy77@gmail.com');
INSERT INTO Parents Values(0859318145, 'Neil', 'Diamond', 'sweetcaroline@gmail.com');
INSERT INTO Parents Values(0864919123, 'Nora', 'Russell', 'russell49@yahoo.com');
INSERT INTO Parents Values(0860854150, 'Bronagh', 'North', 'northsouth@yahoo.com');
INSERT INTO Parents Values(0870835715, 'Olivia', 'Gardiner', 'theGardiner@gmail.com');

#Populate table Ninjas
INSERT INTO Ninjas Values(1001, 'Rory', 'Murphy', 0879248761, 10, '2006-01-03');
INSERT INTO Ninjas Values(1002, 'Tara', 'Murphy', 0879248761, 10, '2008-05-09');
INSERT INTO Ninjas Values(1003, 'Sarah', 'Diamond', 0859318145, 10, '2010-08-02');
INSERT INTO Ninjas Values(1004, 'Mary', 'Byrne', 0879619937, 12, '2011-03-03');
INSERT INTO Ninjas Values(1005, 'Niall', 'Russell', 0864919123, 12, '2012-12-12');
INSERT INTO Ninjas Values(1006, 'Nollaig', 'Russell', 0864919123, 10, '2006-10-02');
INSERT INTO Ninjas Values(1007, 'Sophie', 'North', 0860854150, 13, '2009-04-08');
INSERT INTO Ninjas Values(1008, 'Louise', 'Gardiner', 0870835715, 13, '2009-06-12');

#Populate table Mentors
INSERT INTO Mentors Values(101, 'Ruth', 'Brennan','Black',1);
INSERT INTO Mentors Values(102, 'Mark', 'Boyle','Black',3);
INSERT INTO Mentors Values(103, 'Emily','Duncan','Black',4);
INSERT INTO Mentors Values(104, 'Sophie','Hamill','Purple',NULL);
INSERT INTO Mentors Values(105, 'Derek','Shepard','Yellow',NULL);
INSERT INTO Mentors Values(106, 'Conor','Lawerence','White',NULL);
INSERT INTO Mentors Values(107, 'Meabh','Dawson','Green',NULL);
INSERT INTO Mentors Values(108, 'Matt','Levy','White',NULL);
INSERT INTO Mentors Values(109, 'Niamh','Levy','Black',2);

#Populate table Committee_position
INSERT INTO Committee_position Values(1, 'Champion', 'Organise Mentors, Events and Meetings');
INSERT INTO Committee_position Values(2, 'IO', 'Organise Parents and Ninjas');
INSERT INTO Committee_position Values(3, 'CPO', 'Ensure that child protection rules are followed
at events and to ensure all of the mentors are Garda vetted');
INSERT INTO Committee_position Values(4, 'PRO', 'Promotes the dojo to potential mentors and
parents of ninjas');
INSERT INTO Committee_position Values(5, 'Treasurer', 'Handles finances, equipment and
sponsorship');
INSERT INTO Committee_position Values(6, 'OCM', 'Attends meetings and contributes to running of
the dojo');
```

```
#Populate table Topics
INSERT INTO Topics Values('Scratch', Null, 102 );
INSERT INTO Topics Values('P5JS', Null, 101 );
INSERT INTO Topics Values('Arduino', Null, 107 );
INSERT INTO Topics Values('Makey Makey', Null, 104 );
INSERT INTO Topics Values('Python', Null, 109 );

#Populate table Meetings
INSERT INTO Meetings Values(10001, '2019-10-10 ', 101, 102);
INSERT INTO Meetings Values(10002, '2019-11-8', 101, 107);
INSERT INTO Meetings Values(10003, '2019-12-3', 101, 102);

#Populate table Events
INSERT INTO Events Values(100001, 'GirlCode','Scratch', '2019-08-10','Room 1.07 Lloyd Institute',13, 101);
INSERT INTO Events Values(100002, 'GirlCode','Scratch', '2018-08-07','Room 1.07 Lloyd Institute',13,103);
INSERT INTO Events Values(100003, 'Intro to Coderdojo','P5JS', '2019-11-03','Mac Lab, Hamilton Building',10, 109);
#INSERT INTO Events Values(100004, 'Intro to Coderdojo','P5JS', '2019-11-03','Mac Lab, Hamilton Building',13, 109);

#Populate Ninja Group
INSERT INTO Ninja_group Values(10, 'P5JS','Creator');
INSERT INTO Ninja_group Values(11, 'Arduino', 'Developer');
INSERT INTO Ninja_group Values(12, 'Python', 'Builder');
INSERT INTO Ninja_group Values(13, 'Scratch', 'Maker');
```