# Introduction

## General Objective

To design and develop two small databases for a social networking of software developers using MySQL Workbench and OrientDB Studio.

## Specifications

The system should:

1. Allow developers to join and leave. The properties of developers should include their name, email address, software that they use and years of experience.

2. Allow developers to follow the postings of other developers. A post must include the date created title and details.

3. Allow developers to post questions to others with expertise in a particular area.

4. Suggest new connections to other developers based on their shared interests.

5. Allow developers to like the posts of other developers.

6. Enable ranking of members according to their number of followers, posts and answers.

# Discussion

## Relational Database

### 1. Modeling the Relational Database

**Entity Identification**

The following entities were identified for the database:

* Developer
* Field
* Follower
* Post
* Like
* Question
* Answer

**Relationship Identification**

The following relationships hold for the entities:

* A developer specializes in fields
* A developer has followers
* A developer posts posts
* A post belongs to a field
* A post has likes
* A developer posts questions
* A developer posts answers to questions

**Cardinality Identification**

The following cardinalities exist for the database:

* One developer may specialize in many fields. One field may be specialized in by many developers. This is a many-to-many cardinality. An entity *developer\_field* has to be introduced to normalize this relationship.
* One developer can have many followers. One follower can have many followed developers. This is a many-to-many cardinality. A twist however exists. A follower is a user. This is as such a *self-referential* relationship.
* One developer can post many posts. One post can only be post by one developer. The relationship between developer and post is one-to-many cardinality.
* One post belongs to one field. One field may be in many posts. The relationship between post and field is many-to-one cardinality.
* One post can have many likes. One like can only be for one post. The relationship between post and like is one-to-many cardinality.
* One developer can issue many likes. One like can only be issued by one developer. The relationship between developer and like is one-to-many relationship.
* One developer can post many questions. One question can only be post by one developer. The relationship between post and question is one-to-many relationship.
* One question can have many answers. One answer only belongs to one post. The relationship between question and answer is one-to-many relationship.
* One answer can be posted by one developer. One developer can post many answers. The relationship between answer and developer is many-to-one relationship.

**Attribute Identification**

The attributes of the entities are as follows:

|  |
| --- |
| **Developer** |
| **Attribute** |
| id |
| email |
| first name |
| last name |
| experience |
| date\_joined |
| date\_left |

Table 1: Developer entity attributes

|  |
| --- |
| **Field** |
| **Attribute** |
| id |
| name |

Table 2: Field entity attributes

|  |
| --- |
| **Developer\_Field** |
| **Attribute** |
| id |
| developer\_id |
| field\_id |

Table 3: Developer\_Field entity attributes

|  |
| --- |
| **Post** |
| **Attribute** |
| id |
| developer\_id |
| field\_id |
| title |
| description |
| time\_posted |

Table 4: Post entity attributes

|  |
| --- |
| **Like** |
| **Attribute** |
| post\_id |
| developer\_id |
| time\_liked |

Table 5: Like entity attributes

|  |
| --- |
| **Follower** |
| **Attribute** |
| follower\_id |
| followed\_id |
| time\_followed |

Table 6: Follower entity attributes

|  |
| --- |
| **Question** |
| **Attribute** |
| id |
| title |
| description |
| developer\_id |
| field\_id |
| time\_posted |

Table 7: Question entity attributes

|  |
| --- |
| **Answer** |
| **Attribute** |
| id |
| question\_id |
| developer\_id |
| time\_posted |
| answer |

Table 8: Answer entity attributes

**Entity Relationship Diagram**

The entity relationship diagram of the database is as shown

The ERD is included in the Appendix item 1.

### 2. Verifying normalization rules

**(a) Identifying candidate keys**

This section discusses the candidate keys in the database modeled by the ERD. The discussion of the candidate keys is based on each entity.

1. **Developer**: the candidate keys in the developer table are *id,* and *email.* The attribute *id* is the primary key, to enable faster and more efficient indexing and data access, being numeric and expectedly shorter-length than email.

2. **Field:** the candidate keys are *id* and *name.* The primary key is *id* to enable faster and more efficient indexing and data access, being numeric and shorter-length than the attribute *name.*

3. **Developer\_Field:** the candidate key is *id.* The attributes *developer\_id* and *field\_id* are foreign keys.

4. **Post:** the candidate key is *id.* The attributes *developer\_id* and *field\_id* are foreign keys.

5. **Like:** the candidate key is a composite key made of two attributes *post\_id* and *developer\_id.* The two make up the primary key because a developer may only like a post once. The two attributes are also foreign keys.

6. **Question:** the candidate key is *id*. The entity has two foreign keys: *developer\_id*  and *field\_id.*

7. **Answer:** the candidate key is *id.* The entity has two foreign keys: *question\_id* and *developer\_id*.

**(b) Verifying and verifying relations are in 3NF**

A relation is in 3NF if there is no transitive dependency for non-prime attributes and the relation is in 2NF.

No attributes in the *Developer,* and *Field* entities are reliant on a foreign key. No attribute in the *Post* entity is tied to the foreign keys: developer\_id and field\_id. No attribute in the *Like* entity is tied to the foreign keys: post\_id and developer\_id. No attribute in the *Follower* entity is tied to the foreign keys: follower\_id and followed\_id. No attribute on the *Question* entity is tied to the foreign keys: developer\_id and field\_id. No attribute in the entity *Answer* is tied to the foreign keys: question\_id and developer\_id.

As such, the relations are in 3NF.

### 3. Implementing Relational Database using MySQL

**(a) Create MySQL Database and SQL statements to create required tables and add data to them**

A MySQL database was created, and SQL statements written to create tables and add data to them. The SQL statements are shown in the appendix item 2.

**(b) Write SQL Statements in MySQL Workbench to:**

**(i) List the member with the most posts**

The SQL statement to list the member with most posts is shown in the appendix item 3(1).

The query result returned the member with most posts as expected from the test data entered as shown in the appendix item 3(2).

**(ii) Count the number of likes for posts, showing the “developer name”, “post”, “date posted”, and “likes” in the result set**

The SQL statement to count the number of likes for posts is shown in the appendix item 4(1).

The query result returned the count of the likes for a post alongside the developer’s name, post title and date of posting as shown in the appendix item 4(2).

## Graph Database

### 1. Design a NoSQL database

**(a) Identifying frequent queries**

The following are the frequent queries expected to be performed on the social network:

* Inserting a post into the database by a developer
* Liking a post by a developer
* Inserting a question into the database by a developer
* Posting answers to a question by a developer
* Following another developer by a developer

**(b) Identify Vertices (Entities)**

The following vertices are to be used in the database:

* Developer
* Post
* Question
* Field

**(c) Identify Edges (Relationships between vertices)**

The following edges are to exist in the database:

* *Posted –* a developer can post many posts. A post can only be posted by one developer. The relationship between developer and post is a one-to-many relationship
* *LikedBy –* a post can be liked by many developers. A like can only originate from one developer. The relationship between post and developer is a one-to-many relationship.
* *AskedBy –* a developer can ask many questions. A question can only be asked by one developer. The relationship between developer and question is a many-to-one relationship.
* *AnsweredBy –* a question can be answered by many developers. An answer can only come from one developer. The relationship between question and answer is a many-to-one relationship.
* *Followed –* a developer can follow many developers. A developer can be followed by many developers. This is a many-to-may relationship. Fortunately, NoSQL supports many-to-many relationships without the need to normalize.
* *BelongsIn –* a post belongs to one field. A field can be referenced in many posts. The relationship between post and field is many-to-one. Similarly, a question belongs to one field. A field can be reference in many questions. This is also a many-to-one relationship.

**(d) Eliminating Relations**

The vertices and edges identified in this model were sufficient, enough and realizable to server the application purpose of the social network website.

**Identifying Attributes**

Vertices and edges have attributes. This section identifies the attributes of each for this database.

Vertices

|  |
| --- |
| **Developer** |
| **Attribute** |
| email |
| first name |
| last name |
| experience |
| date\_joined |

Table 9: Developer vertice attributes

|  |
| --- |
| **Post** |
| **Attribute** |
| title |
| description |

Table 10: Post vertice attributes

|  |
| --- |
| **Question** |
| **Attribute** |
| title |
| description |

Table 11: Question vertice attributes

|  |
| --- |
| **Field** |
| **Attribute** |
| name |

Table 12: Field vertice attributes

Edges

|  |
| --- |
| **Posted** |
| **Attribute** |
| time\_posted |

Table 13: Posted edge attributes

|  |
| --- |
| **Followed** |
| **Attribute** |
| time\_followed |

Table 14: Followed edge attributes

|  |
| --- |
| **LikedBy** |
| **Attribute** |
| time\_liked |

Table 15: LikedBy edge attributes

|  |
| --- |
| **AskedBy** |
| **Attribute** |
| time\_posted |

Table 16: AskedBy edge attributes

**Graph Model**

The graph model of the database is shown in the appendix item 5.

### 2. Implementing Graph Database

**(a) Creating Database in OrientDB**

A graph database was created in OrientDB Studio. Classes were created extending the standard vertices V class to represent the classes for the four entities identified. Classes extending the standard edges E class were created to represent the classes for the six edges identified.

The NoSQL graph statements to achieve this are included in the appendix item 6.

**(b)**  **Populating created database schema**

Vertices and edges were created to populate the database with data. The statements to populate the test data were as shown in the appendix item 7(1). Three developers, two field, four posts and two questions were populated.

The NoSQL graph model at the end of this is as shown in the appendix item 7(2).

**(c) NoSQL Statement to list all developers who a particular developer listed in your database follows**

The statement to list all developers followed by a particular user is shown in appendix item 8(1). The test user #27:0 was considered in this case.

The result of the query returned the expected results as shown in appendix item 8(2).

# Conclusion

This assignment set out to design and develop two small databases for a social networking of software developers using MySQL Workbench and OrientDB Studio. A model of the relational database was created, the normalization rules verified and the relational database implemented using MySQL, recording the SQL queries needed to create, populate and query the database. The queries needed for the graph database were identified, after which the vertices, edges and subsequently the proposed graph model were identified. The graph database was created using OrientDB, recording the statements needed to create, populate and query the database.

Even though a lot of the specifications were achieved in the assignment, a few specifications were not met. Queries to rank developers according to their participation and make recommendations to developers on other developers to follow were not achieved.

The assignment, however, provided an excellent opportunity to practice relational and NoSQL database design and implementation.

# Appendix

**Item 1: ERD**

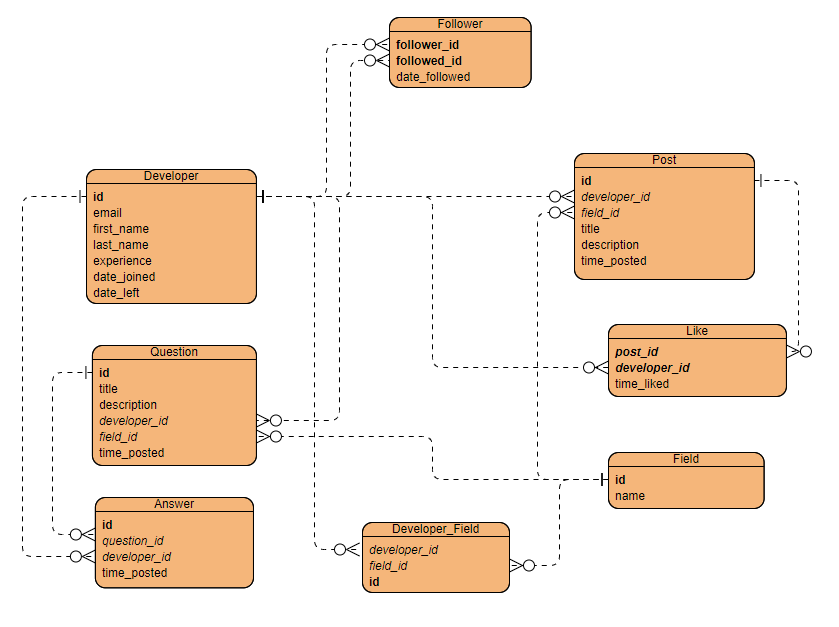


Figure 1: Entity Relationship Diagram (ERD) of the relational database

**Item 2: SQL Statements to Create Tables and Populate them with Data**

-- \*\*\*\*\*\*\* Create Database \*\*\*\*\*\*\* --

DROP DATABASE IF EXISTS mydeveloperdb;

CREATE DATABASE mydeveloperdb;

-- Use this database

USE mydeveloperdb;

-- \*\*\*\*\*\*\* Create Tables \*\*\*\*\*\*\* --

CREATE TABLE Developer (

id int NOT NULL AUTO\_INCREMENT,

email varchar(100) NOT NULL,

first\_name varchar(50),

last\_name varchar(50),

experience int,

date\_joined datetime DEFAULT CURRENT\_TIMESTAMP,

date\_left datetime,

PRIMARY KEY (id)

);

CREATE TABLE Field (

id int NOT NULL AUTO\_INCREMENT,

name varchar(100),

PRIMARY KEY (id)

);

CREATE TABLE Developer\_Field (

id int NOT NULL AUTO\_INCREMENT,

developer\_id int,

field\_id int,

PRIMARY KEY (id),

FOREIGN KEY (developer\_id) REFERENCES Developer(id),

FOREIGN KEY (field\_id) REFERENCES Field(id)

);

CREATE TABLE Post (

id int NOT NULL AUTO\_INCREMENT,

developer\_id int,

field\_id int,

title TEXT,

description TEXT,

time\_posted datetime DEFAULT CURRENT\_TIMESTAMP,

PRIMARY KEY (id),

FOREIGN KEY (developer\_id) REFERENCES Developer(id),

FOREIGN KEY (field\_id) REFERENCES Field(id)

);

CREATE TABLE Like\_ (

post\_id int NOT NULL,

developer\_id int NOT NULL,

time\_liked datetime DEFAULT CURRENT\_TIMESTAMP,

CONSTRAINT pk\_like PRIMARY KEY (post\_id,developer\_id),

FOREIGN KEY (post\_id) REFERENCES Post(id),

FOREIGN KEY (developer\_id) REFERENCES Developer(id)

);

CREATE TABLE Follower (

follower\_id int NOT NULL,

followed\_id int NOT NULL,

time\_followed datetime DEFAULT CURRENT\_TIMESTAMP,

CONSTRAINT pk\_follwer PRIMARY KEY (follower\_id, followed\_id),

FOREIGN KEY (follower\_id) REFERENCES Developer(id),

FOREIGN KEY (followed\_id) REFERENCES Developer(id)

);

CREATE TABLE Question (

id int NOT NULL AUTO\_INCREMENT,

title TEXT,

description TEXT,

developer\_id int,

field\_id int,

time\_posted datetime DEFAULT CURRENT\_TIMESTAMP,

PRIMARY KEY (id),

FOREIGN KEY (developer\_id) REFERENCES Developer(id),

FOREIGN KEY (field\_id) REFERENCES Field(id)

);

CREATE TABLE Answer (

id int NOT NULL AUTO\_INCREMENT,

question\_id int,

developer\_id int,

time\_posted datetime DEFAULT CURRENT\_TIMESTAMP,

answer TEXT,

PRIMARY KEY (id),

FOREIGN KEY (question\_id) REFERENCES Question(id),

FOREIGN KEY (developer\_id) REFERENCES Developer(id)

);

-- \*\*\*\*\*\*\* Creating Test Data \*\*\*\*\*\*\* --

-- Insert Test Fields

INSERT INTO Field (id, name)

VALUES

(1, "Python"),

(2, "Java");

-- Insert Test Developers

INSERT INTO Developer (id, email, first\_name, last\_name, experience)

VALUES

(1, "joe123@gmail.com", "joe", "justin", 4),

(2, "randomartin@gmail.com", "randy", "martin", 7),

(3, "danieljoe@gmail.com", "daniel", "joe", 4);

-- Insert fields of expertise of developers

INSERT INTO Developer\_Field (developer\_id, field\_id)

VALUES

(1, 1),

(2, 2),

(3, 1);

-- Insert posts

INSERT INTO Post (id, developer\_id, field\_id, title, description)

VALUES

(1, 1, 1, "Tuples", "Tuples provide a way to store data that is ..."),

(2, 1, 1, "TKinter", "TKinter allows you to create GUI in Python ..."),

(3, 1, 1, "The Good, the Bad, the Ugly: Python", "There has been a growing demand for Python developers...."),

(4, 2, 2, "Java EE", "Java EE provides a collection of tools that enable you create ...");

-- Insert likes

INSERT INTO Like\_ (post\_id, developer\_id)

VALUES

(1, 2),

(1, 3),

(2, 2),

(3, 1);

**Item 3 (1): List Member with Most Posts**

SELECT developer\_id, MAX(post\_count.count) AS number\_of\_posts

FROM

(SELECT developer\_id, COUNT(\*) AS count

FROM Post

GROUP BY developer\_id) post\_count;

**Item 3(2): Query Result**

The query result of the above query on the test data entered was as shown:



Figure 2: Query Result - Listing Member with Most Posts

**Item 4(1): Count Number of Likes for Posts**

SELECT Developer.first\_name, Developer.last\_name, x.Post, x.Date\_Posted, x.Likes

FROM Developer

INNER JOIN

(SELECT Like\_.developer\_id, COUNT(\*) AS Likes, Post.title AS Post, Post.time\_posted AS Date\_Posted

FROM Like\_

INNER JOIN Post ON Like\_.post\_id = Post.id

GROUP BY Like\_.post\_id) x

ON Developer.id = x.developer\_id;

**Item 4(2): Query Result**

The query result of the above query on the test data entered was as shown:

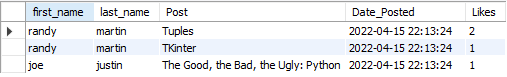


Figure 3: Query result - Counting Number of Likes for Posts

**Item 5: Proposed Graph Model**

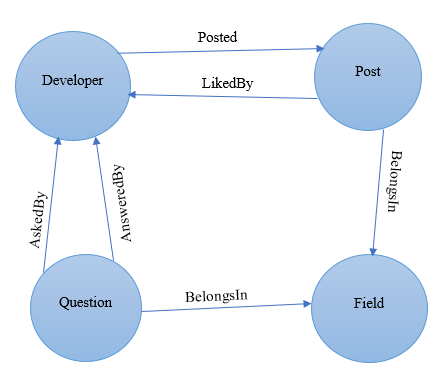
****

Figure 4: Proposed Graph Model

**Item 6: Creating Vertices and Edges Classes**

CREATE CLASS Developer EXTENDS V

CREATE CLASS Post EXTENDS V

CREATE CLASS Question EXTENDS V

CREATE CLASS Field EXTENDS V

CREATE CLASS BelongsIn EXTENDS E

CREATE CLASS Posted EXTENDS E

CREATE CLASS LikedBy EXTENDS E

CREATE CLASS AskedBy EXTENDS E

CREATE CLASS AnsweredBy EXTENDS E

CREATE CLASS Followed EXTENDS E

**Item 7(1): Statements to populate data**

CREATE VERTEX Developer SET first\_name="joe", second\_name="justin", email="joe123@gmail.com", experience=4, date\_joined=sysdate()

CREATE VERTEX Developer SET first\_name="martin", second\_name="randy", email="randomartin@gmail.com", experience=7, date\_joined=sysdate()

CREATE VERTEX Developer SET first\_name="daniel", second\_name="joe", email="danieljoe@gmail.com", experience=4, date\_joined=sysdate()

CREATE EDGE Followed FROM #27:0 TO #26:0 SET time\_followed=sysdate()

CREATE EDGE Followed FROM #28:0 TO 26:0 SET time\_followed=sysdate()

CREATE VERTEX Field SET name="Python"

CREATE VERTEX Field SET name="Java"

CREATE VERTEX Post SET title="Tuples", description="Tuples provide a way to store data that is ..."

CREATE VERTEX Post SET title="TKinter", description="TKinter allows you to create GUI in Python ..."

CREATE VERTEX Post SET title="The Good, the Bad, the Ugly: Python", description="There has been a growing demand for Python developers...."

CREATE VERTEX Post SET title="Java EE", description="Java EE provides a collection of tools that enable you create ..."

CREATE EDGE Posted FROM #26:0 TO #30:0 SET time\_posted=sysdate()

CREATE EDGE Posted FROM #26:0 TO #31:0 SET time\_posted=sysdate()

CREATE EDGE Posted FROM #26:0 TO #32:0 SET time\_posted=sysdate()

CREATE EDGE Posted FROM #27:0 TO #33:0 SET time\_posted=sysdate()

CREATE EDGE BelongsIn FROM #30:0 TO #58:0

CREATE EDGE BelongsIn FROM #31:0 TO #58:0

CREATE EDGE BelongsIn FROM #32:0 TO #58:0

CREATE EDGE BelongsIn FROM #33:0 TO #59:0

CREATE EDGE LikedBy FROM #30:0 TO #27:0 SET time\_liked=sysdate()

CREATE EDGE LikedBy FROM #30:0 TO #28:0 SET time\_liked=sysdate()

CREATE EDGE LikedBy FROM #31:0 TO #27:0 SET time\_liked=sysdate()

CREATE EDGE LikedBy FROM #32:0 TO #26:0 SET time\_liked=sysdate()

CREATE VERTEX Question SET title="Handling Streams", question="Am having a hard time flushing streams...."

CREATE VERTEX Question SET title="Passing parameters in Java EE", question="Am stuck trying to pass values from a JSF to the backing bean...."

CREATE EDGE AskedBy FROM #34:0 TO #26:0 SET time\_posted=sysdate()

CREATE EDGE AskedBy FROM #35:0 TO #27:0 SET time\_posted=sysdate()

CREATE EDGE BelongsIn FROM #34:0 TO #58:0

CREATE EDGE BelongsIn FROM #35:0 TO #59:0

CREATE EDGE AnsweredBy FROM #34:0 TO #28:0 SET time\_posted=sysdate(), answer="Hey xxxx. You could use the networking function ...."

**Item 7(2): Actualized NoSQL Graph Model Upon Population of Data**

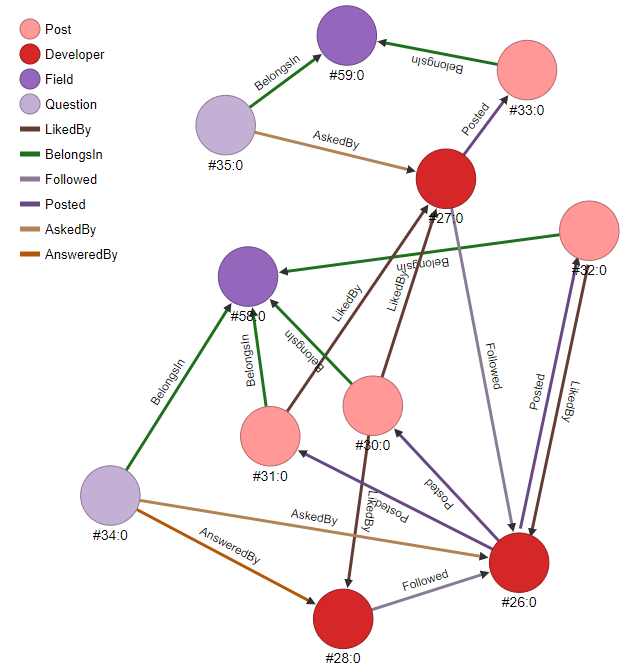
****

Figure 5: Actualized NoSQL Graph

**Item 8(1): Selecting Developers followed by a Particular Developer**

SELECT @rid, first\_name, second\_name FROM (TRAVERSE out("Followed") FROM #27:0 MAXDEPTH 1) WHERE @rid!="#27:0"

**Item 8(2): Query Result**

****

Figure : Selecting users followed by a user