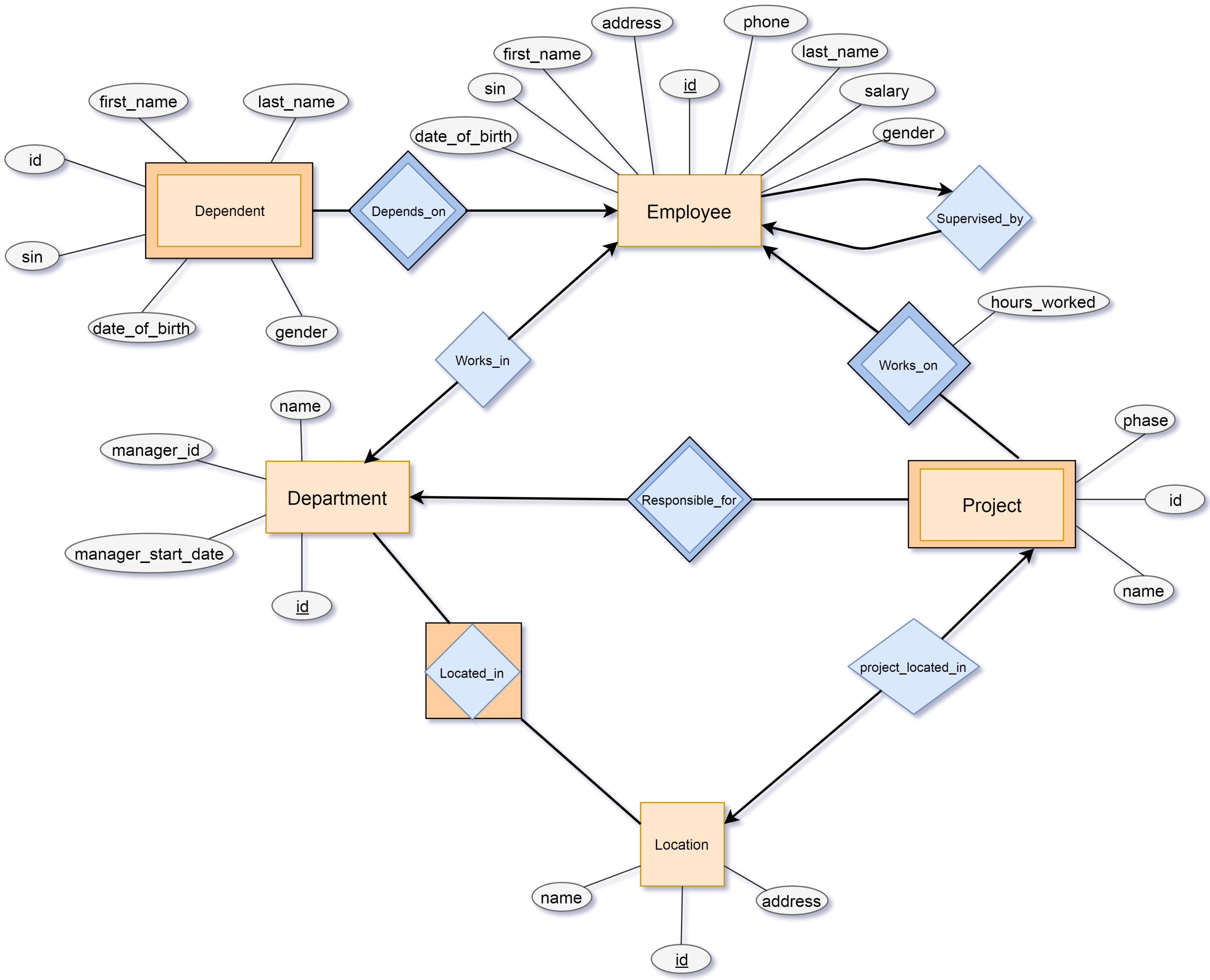
Comp353 Project Report

Kai Nicoll-Griffith[40012407], Stephen Prizio[40001739],
Giovanni Gebran[40018637], Nizar Belhassan[27519443]

Team kzc353_4

April 11, 2018

1 Entity Relationship Diagram



2 Reasonable Assumptions

2.1 General cases

An assumption has been made that all identification numbers are unsigned integers. An identification key will never have a sign so the database restricts this.

2.2 department table

In the case of the ‘department‘ table, both the manager_id and manager_start_date are given the opportunity to be null since it is not always true that a ‘department‘ needs a manager. Small groups could potentially self manage if that is the policy of the company.

2.3 employee table

To ensure there will always be relevant ‘employee‘ data, there are no optional or null possible parameters possible within the ‘employee‘ table. It is assumed that a company needs to keep accurate track of everyone within it and null values would encourage poor data management practice of the company. A salary(a 5,2 decimal datatype) is given to each employee in dollars per hour to make certain queries easier to process. Due to legislation, gender attribute is defined by one ambiguous character. An ‘employee‘ must work for a single ‘department‘.

2.4 project table

It is assumed that a ‘project‘ can not be assigned to multiple ‘departments‘. Also a ‘project‘ has a varchar phase attribute which keeps track of the progress of each individual project within the COMPANY database.

2.5 dependent table

The ‘dependent‘ table holds vital information that has potential legal importance so none of these fields may be null. A dependent is linked to an ‘employee‘ by a foreign key holding employee_id and has the multiplicity of one to many. An ‘employee‘ may have many ‘dependents‘.

2.6 location table

In order to specify where a ‘project‘ or ‘department‘ is situated, a ‘location‘ table keeps track of all of the possible locations where departments and projects operate. An entity table will therefore use a relation table holding an unsigned location_id to specify where the department or project is located in both address and an optional name. The name is assumed to be used for employee convenience to identify a location while a mandatory address is used for more direct positioning and referencing(as would be used by a post office). The name is a varchar, while the address is medium text since it is assumed that the address could be as specific as country down to room number and limitations on varchar size could be problematic.

2.7 supervised_by table

The ‘supervised_by table‘ defines a role of being a subordinate to someone and helps to give information about the status of an employee in the business hierarchy. Supervision does not imply that an employee is a manager and it could be that an employee both supervises and manages a 'department'. It is assumed that this relation is solely used to show the hierarchy of employees within the company. To recognize the 'employee' who is supervised, each employee is given a single supervisor_id, but a supervisor can supervise many. Our assumption is that an employee should only be supervised by one person or none at all therefore employee _id is a primary key enforcing uniqueness on being supervised while supervisor_id is a default null value, where null implies an 'employee' is unsupervised.

2.8 depends_on relation

The weak relation 'depends_on' creates the assumption an 'employee' can have many 'dependants' in a 1:many relationship.

2.9 works_on relation

The weak relation 'works_on' creates the assumption that an 'employee' can work on many 'projects' in a 1:many relationship

2.10 works_in relation

The strong relation 'works_in' creates the assumption that an 'employee' can only work in one 'department' in a 1:1 relationship.

2.11 responsible_for relation

The weak relation 'responsible_for' creates the assumption that a 'department' can be responsible for many 'projects' in a 1:many relationship

2.12 project_located_in relation

The strong relation 'project_located_in' creates the assumption that a project has to be tied to one location in a 1:1 relationship.

2.13 department_located_in relation

The associative entity 'project_located_in' creates the assumption that a 'department' can be positioned in many 'locations' while at the same time a 'location' can be assigned to many 'departments' in a many:many relationship.

3 ER to Relation conversion

- Department(id:uint, name:varchar, manager_id:uint, manager_start_date:date)
- Dependent(id:uint, first_name:varchar, last_name:varchar, sin:uint, date_of_birth:date, gender:char, employee_id:uint)
- Employee(id:uint, first_name:varchar, last_name:varchar, sin:uint, date_of_birth:date, address:varchar, phone:char, salary:unsigned-decimal, gender:char, department_id:uint)
- Project(id:uint, name:varchar, location_id, phase:varchar)
- Location(id:uint, name:varchar, address:mediumtext)
- Role(employee_id:uint, supervisor_id:uint)
- Works_on(project_id:uint, employee_id:uint, hours_worked:uint)
- Located_in(location_id:uint, department_id:uint)
- Responsible_for(department_id:uint, project_id:uint)

4 Implemented Functionalities

4.1 Database design

In the COMPANY database There are three primary categories of entity from which more complex entities are defined. These are:

1. departments,
2. employees,
3. projects,

Each of these tables specifies information that defines the three main entities in the database. These three main entity sets are also enhanced by the entity sets of:

1. dependent
2. location

And also the role relation:

1. supervised_by

Which specifies an employees role against other employees as a supervisor.
While the entity-relation diagram specifies multiple that multiple possible relations can be made, in order to reduce the complexity of the design(and therefore the queries) only the following relations are used

1. works_on
2. responsible_for
3. located_in

These three relations were deemed most important and the other relations seen on the E/R diagram have been omitted.

4.2 Language and tools

The application makes use of the PHP 5.5.9 language due to it's reliable and simple functions for connecting with a MySQL database. In order to more easily input queries on the database and build a modern looking front end system, Laravel has been used to make development easier which adds additional functionality to and shortcuts to front-end design.

4.3 Query Functionalities

Queries allow the system to select, update, modify and add to the company database whilst also providing key information. All of these queries can be found in /laravel/app/http/routes.php while some can also be found in the .php files found in /queries - forms. The *?* and *:id* fields are instances where a dynamic value would be inserted by the Laravel controllers that have been implemented. These dynamic values are captured from a user's input.

4.3.1 Department

1. Select a single department
   SELECT *
   FROM department
   WHERE id = :id;

2. Select all departments
   SELECT *
   FROM department
   ORDER BY id;

3. Select a department's locations
   SELECT *
   FROM located_in, location
   WHERE location_id = id AND department_id = :id;

4. Select a department's projects
   SELECT *
   FROM responsible_for, project
   WHERE project_id = id AND department_id = :id
   ORDER BY project_id;

5. Select all employees for a department
   SELECT *
   FROM employee
   WHERE department_id = :id
   ORDER BY last_name;

6. Select a department's total pay as a function of employee's salary and hours worked
   SELECT SUM(salary * hours_worked) as 'Pay', department_id
   FROM works_on, employee, project
   WHERE employee_id = employee.id AND project_id = project.id AND department_id=:id
   GROUP BY department_id;

7. Select locations that a department is not in
   SELECT *
   FROM location
   WHERE id NOT IN (SELECT location_id FROM located_in WHERE department_id = :id);

8. Add location to a department
   INSERT INTO located_in(location_id, department_id) VALUES (?, ?);

9. Delete department location
   DELETE FROM located_in WHERE department_id = ? AND location_id = ?;

10. Delete a project from department
    DELETE FROM responsible_for WHERE department_id = ? AND project_id = ?;

11. Select projects without a department
    SELECT *
    FROM project
    WHERE id NOT IN (SELECT project_id FROM responsible_for);

12. Add project to a department
    INSERT INTO responsible_for(department_id, project_id) VALUES (?, ?);

13. Create a deaprtment
    INSERT INTO department(name, manager_id, manager_start_date) VALUES (?, ?, ?);

14. Edit a department
    UPDATE department SET name = ?, manager_id = ?, manager_start_date = ? WHERE id = ?;

15. Delete a department
    DELETE FROM department WHERE id = :id;

4.3.2 Employee

1. Select a single employee
   SELECT *
   FROM employee
   WHERE id = :id;

2. Select all employees
   SELECT *
   FROM employee
   ORDER BY id;

3. Select an employee's dependents
   SELECT *
   FROM dependent
   WHERE employee_id = :id
   ORDER BY last_name;

4. Select projects that an employee works on
   SELECT *
   FROM project, works_on
   WHERE id = works_on.project_id AND works_on.employee_id = :id;

5. Create an employee
   INSERT INTO employee (first_name, last_name, sin, date_of_birth, address, phone, salary, gender, department_id) VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?);

6. Edit an employee
   UPDATE employee SET first_name = ?, last_name = ?, sin = ?, date_of_birth = ?, address = ?, phone = ?, salary = ?, gender = ?, department_id = ? WHERE id = ?;

7. Delete an employee
   DELETE FROM employee WHERE id = :id;

8. Select all dependents
   SELECT * FROM dependent WHERE id = :id;

9. Create a dependent
   INSERT INTO dependent(first_name, last_name, sin, date_of_birth, gender, employee_id) VALUES (?, ?, ?, ?, ?, ?);

10. Edit a dependent
    UPDATE dependent SET first_name = ?, last_name = ?, sin = ?, date_of_birth = ?, gender = ? WHERE id = ?;

11. Delete a dependent
    DELETE FROM dependent WHERE id = :id;

4.3.3 Supervisor

1. Select an employee's supervisor
   SELECT *
   FROM role, employee
   WHERE employee.id = supervisor_id AND employee_id = :id;

2. Select a supervisor's subordinates
   SELECT *
   FROM employee
   WHERE id IN (SELECT employee_id FROM role WHERE supervisor_id = :id);

3. Select employees that are not supervisors
   SELECT *
   FROM employee
   WHERE id NOT IN (SELECT supervisor_id FROM role)
   ORDER BY last_name;

4. Select all supervisors
   SELECT *
   FROM employee
   WHERE id IN (SELECT supervisor_id FROM role);

5. Select a supervisor
   SELECT *
   FROM employee
   WHERE id = (SELECT DISTINCT supervisor_id FROM role WHERE supervisor_id = :id);

6. Create a supervisor
   INSERT INTO role(employee_id, supervisor_id) VALUES (?, ?);

7. Select employees without supervisors SELECT * FROM employee WHERE id NOT IN (SELECT employee_id FROM role) AND id <> :id;

8. Delete a subordinate
   DELETE FROM role WHERE employee_id = ? AND supervisor_id = ?;

9. Delete a supervisor
   DELETE FROM role WHERE supervisor_id = :id;

4.3.4 Projects

1. Select all projects
   SELECT *
   FROM project
   ORDER BY id;

2. Select a single project
   SELECT *
   FROM project
   WHERE id = :id;

3. Select a project's department
   SELECT *
   FROM responsible_for, department
   WHERE department_id = id AND project_id = :id;

4. Select all employees for a project
   SELECT *
   FROM works_on, employee
   WHERE id = employee_id AND project_id = :id
   ORDER BY id;

5. Select number of employees for a given project
   SELECT COUNT(id)
   FROM works_on, employee
   WHERE id = employee_id AND project_id = :id;

6. Select total number of hours worked on a project
   SELECT SUM(hours_worked)
   FROM works_on, employee
   WHERE id = employee_id AND project_id = :id;

7. Select a project's total pay
   SELECT SUM(Pay)
   FROM (SELECT works_on.hours_worked, works_on.employee_id, employee.salary, (hours_worked * salary) AS Pay
   FROM works_on, employee
   WHERE works_on.project_id=:id AND employee.id=works_on.employee_id) as Payed;

8. Create a project
   INSERT INTO project(name, location_id, phase) VALUES (?, ?, ?);

9. Edit a project
   UPDATE project SET name = ?, location_id = ?, phase = ? WHERE id = ?;

10. Delete a project
    DELETE FROM project WHERE id = :id;

11. Select employees not assigned to a project
    SELECT *
    FROM employee
    WHERE id NOT IN (SELECT employee_id FROM works_on);

12. Add an employee to a project
    INSERT INTO works_on(project_id, employee_id, hours_worked) VALUES (?, ?, ?);

13. Select an employee working on a project
    SELECT *
    FROM works_on
    WHERE employee_id = :eid AND project_id = :id;

14. Edit an employee who is working on a project
    UPDATE works_on SET hours_worked = ? WHERE employee_id = ? AND project_id = ?;

15. Delete an employee from a project
    DELETE FROM works_on WHERE employee_id = :eid AND project_id = :id;

4.3.5 Location

1. Select a single location
   SELECT *
   FROM location
   WHERE id = :id;

2. Select all locations
SELECT *
FROM location
ORDER BY id;

3. Select a location's departments
SELECT *
FROM department
WHERE id IN (SELECT department_id FROM located_in WHERE location_id = :id);

4. Select a location's projects
SELECT *
FROM project
WHERE id IN (SELECT project_id FROM responsible_for WHERE department_id IN (SELECT department_id FROM located_in WHERE location_id = :id)) AND location_id = :id2;

5. Create a location
INSERT INTO location(name, address) VALUES (?, ?);

6. Edit a location
UPDATE location SET name = ?, address = ? WHERE id = ?;

7. Delete a location
DELETE FROM location WHERE id = :id;

4.3.6 Statistics

1. Select count of departments
SELECT COUNT(id)
FROM department;

2. Select count of employees
SELECT COUNT(id)
FROM employee;

3. Select count of projects
SELECT COUNT(id)
FROM project;

4. Select count of locations
SELECT COUNT(id)
FROM location;

5. Select department with the most employees
SELECT COUNT(department_id) as 'Count', department_id
FROM employee
GROUP BY department_id
ORDER BY COUNT(department_id) DESC LIMIT 1;

6. Select department with the least employees
SELECT COUNT(department_id) as 'Count', department_id
FROM employee
GROUP BY department_id
ORDER BY COUNT(department_id) ASC LIMIT 1;

7. Select department with the most projects
   SELECT COUNT(project_id) as 'Count', department_id
   FROM responsible_for
   GROUP BY department_id
   ORDER BY COUNT(department_id) DESC LIMIT 1;

8. Select department with the least projects
   SELECT COUNT(project_id) as 'Count', department_id
   FROM responsible_for
   GROUP BY department_id
   ORDER BY COUNT(department_id) ASC LIMIT 1;

9. Select department with the highest pay
   SELECT SUM(salary * hours_worked) as 'Pay', department_id
   FROM works_on, employee, project
   WHERE employee_id = employee.id AND project_id = project.id
   GROUP BY department_id
   ORDER BY SUM(salary * hours_worked) DESC LIMIT 1;

10. Select department with the lowest pay
   SELECT SUM(salary * hours_worked) as 'Pay', department_id
   FROM works_on, employee, project
   WHERE employee_id = employee.id AND project_id = project.id
   GROUP BY department_id
   ORDER BY SUM(salary * hours_worked) ASC LIMIT 1;

11. Select project with the highest pay
   SELECT project_id, project.name, SUM(salary * hours_worked) as 'Pay'
   FROM works_on, employee, project
   WHERE employee_id = employee.id AND project_id = project.id
   GROUP BY project_id
   ORDER BY SUM(salary * hours_worked) DESC LIMIT 1;

12. Select project with the lowest pay
   SELECT project_id, project.name, SUM(salary * hours_worked) as 'Pay'
   FROM works_on, employee, project
   WHERE employee_id = employee.id AND project_id = project.id
   GROUP BY project_id
   ORDER BY SUM(salary * hours_worked) ASC LIMIT 1;

13. Select project with the most employees
   SELECT project_id, COUNT(employee_id) as 'Count', project.name
   FROM works_on, project
   WHERE project_id = project.id
   GROUP BY project_id
   ORDER BY COUNT(employee_id) DESC LIMIT 1;

14. Select project with the least employees
   SELECT project_id, COUNT(employee_id) as 'Count', project.name
   FROM works_on, project
   WHERE project_id = project.id
   GROUP BY project_id
   ORDER BY COUNT(employee_id) ASC LIMIT 1;

15. Select the total pay for the whole company
SELECT SUM(Pay)
FROM (SELECT project_id, project.name, SUM(salary * hours_worked) as Pay
FROM works_on, employee, project
WHERE employee_id = employee.id AND project_id = project.id
GROUP BY project_id) AS P;

16. Select the company's weekly pay
SELECT SUM(40*department_cost_per_hour) AS Pay
FROM department_cost;

** This query is based off a custom view built on the database **

View creation:
CREATE VIEW department_cost AS
SELECT department_id, SUM(salary) AS department_cost_per_hour
FROM department,employee
WHERE department.id=employee.department_id
GROUP BY department_id;

17. Select total project hours
SELECT SUM(hours_worked) as 'Count'
FROM works_on; total project hours;

18. Select employee with the most projects
SELECT COUNT(project_id) as 'Count', works_on.employee_id, employee.first_name, employee.last_name
FROM works_on
JOIN employee ON employee.id=works_on.employee_id
GROUP BY employee_id
ORDER BY COUNT(project_id) DESC LIMIT 1;

19. Select employee with the least projects
SELECT COUNT(project_id) as 'Count', works_on.employee_id, employee.first_name, employee.last_name
FROM works_on
JOIN employee ON employee.id=works_on.employee_id
GROUP BY employee_id
ORDER BY COUNT(project_id) ASC LIMIT 1;

20. Select supervisor with the most subordinates
SELECT COUNT(employee_id) as 'Count', supervisor_id, first_name, last_name
FROM role, employee
WHERE supervisor_id = id
GROUP BY supervisor_id
ORDER BY COUNT(employee_id) DESC LIMIT 1;

21. Select supervisor with the least subordinates
SELECT COUNT(employee_id) as 'Count', supervisor_id, first_name, last_name
FROM role, employee
WHERE supervisor_id = id
GROUP BY supervisor_id
ORDER BY COUNT(employee_id) ASC LIMIT 1;

22. Select total salary per hour
SELECT SUM(salary) as 'Count'
FROM employee;

23. Select location with the most projects
SELECT location_id, location.name, COUNT(project.id) as 'Count'
FROM project, location
WHERE project.id IN (SELECT project_id FROM responsible_for WHERE department_id IN (SELECT department_id FROM located_in)) AND location_id = location.id
GROUP BY location_id
ORDER BY COUNT(project.id) DESC LIMIT 1;

24. Select location with the least projects
SELECT location_id, location.name, COUNT(project.id) as 'Count'
FROM project, location
WHERE project.id IN (SELECT project_id FROM responsible_for WHERE department_id IN (SELECT department_id FROM located_in)) AND location_id = location.id
GROUP BY location_id
ORDER BY COUNT(project.id) ASC LIMIT 1;

5 Normalization Steps and Assumptions

5.1 Functional Dependencies

$$
\begin{array}{llll}
F(employee) & \equiv & employee.id, sin & \rightarrow & first\_name, last\_name, date\_of\_birth, address, phone, salary, gender \\
F(dependent) & \equiv & dependent.id, sin & \rightarrow & first\_name, last\_name, date\_of\_birth, gender \\
F(department) & \equiv & department.id, name, manager\_id & \rightarrow & manager\_start\_date \\
F(project) & \equiv & project.id, name & \rightarrow & location\_id, phase \\
F(supervised\_by) & \equiv & employee\_id & \rightarrow & supervisor\_id \\
F(location) & \equiv & location.id, name & \rightarrow & address
\end{array}
$$

5.2 Normalization Process

5.3 Assumptions

5.3.1 Of Functional Dependencies

6 Contributions

6.1 Giovanni Gebran

- Database Design
- Report: Functional Dependencies

6.2 Nizar Belhassan

- Database Design
- Query Implementations

6.3 Kai Nicoll-Griffith

- Database Design
- Database Attribute Refinements
- Report Setup and LateX entry
- Report: ER Diagram
- Report: Constraints and Assumptions
- Report: Query Functionalities
- Report: Functional Dependencies & Normalization

6.4 Stephen Prizio

- Database Design
- Laravel Application Development
- SQL sample data and Database
- Query Implementations
- Report: Query Functionalities

Faculty of Engineering and Computer Science
Expectations of Originality

This form sets out the requirements for originality for work submitted by students in the Faculty of Engineering and Computer Science. Submissions such as assignments, lab reports, project reports, computer programs and take-home exams must conform to the requirements stated on this form and to the Academic Code of Conduct. The course outline may stipulate additional requirements for the course.

1. Your submissions must be your own original work. Group submissions must be the original work of the students in the group.
2. Direct quotations must not exceed 5% of the content of a report, must be enclosed in quotation marks, and must be attributed to the source by a numerical reference citation[1]. Note that engineering reports rarely contain direct quotations.
3. Material paraphrased or taken from a source must be attributed to the source by a numerical reference citation.
4. Text that is inserted from a web site must be enclosed in quotation marks and attributed to the web site by numerical reference citation.
5. Drawings, diagrams, photos, maps or other visual material taken from a source must be attributed to that source by a numerical reference citation.
6. No part of any assignment, lab report or project report submitted for this course can be submitted for any other course.
7. In preparing your submissions, the work of other past or present students cannot be consulted, used, copied, paraphrased or relied upon in any manner whatsoever.
8. Your submissions must consist entirely of your own or your group's ideas, observations, calculations, information and conclusions, except for statements attributed to sources by numerical citation.
9. Your submissions cannot be edited or revised by any other student.
10. For lab reports, the data must be obtained from your own or your lab group's experimental work.
11. For software, the code must be composed by you or by the group submitting the work, except for code that is attributed to its sources by numerical reference.

You must write one of the following statements on each piece of work that you submit:
For individual work: **"I certify that this submission is my original work and meets the Faculty's Expectations of Originality"**, with your signature, I.D. #, and the date.
For group work: **"We certify that this submission is the original work of members of the group and meets the Faculty's Expectations of Originality"**, with the signatures and I.D. #s of all the team members and the date.

A signed copy of this form must be submitted to the instructor at the beginning of the semester in each course.

I certify that I have read the requirements set out on this form, and that I am aware of these requirements. I certify that all the work I will submit for this course will comply with these requirements and with additional requirements stated in the course outline.

Course Number: Comp 353-X   Instructor: N. Shiri
Name: Kai Nicoll-Griffith   I.D. #: 40012407
Signature: Kainicollgriffith   Date: Wednesday, February 14th 2017 [14-01-17]

[1] Rules for reference citation can be found in "Form and Style" by Patrich MacDonagh and Jack Bordan, fourth edition, May, 2000, available at http://www.encs.concordia.ca/scs/Forms/Form&Style.pdf.
Approved by the ENCS Faculty Council February 10, 2012

Kai Nicoll-Griffith Kainicollgriffith 40012407 | Nizar Belhassan 27519443
Stephen Prizio 40001739 | Giovanni Gebran 40018637 GG.