Module Code & Module Title

CC6001NI - Advanced Database System Development

Assessment Weightage & Type

40% Individual Coursework

Year and Semester

2020-21 Autumn

Student Name: Suyogya Luitel

Group: C3

London Met ID: 19031784

College ID: NP01CP4A190035

Assignment Due Date: 09th March 2022

Assignment Submission Date: 09th March 2022

Introduction

This is a report of the coursework involving analysis, design, and implementation of a web-based database application in accordance with the given business case study with the help of tools such as Oracle SQL Developer Data Modeler, Oracle SQL Developer and ASP.NET. According to the presented business scenario, the database should contain information regarding college departments; students, including fee status and attendance; assignments, and their results. The scenario also elaborates that a teacher may be associated wit multiple modules and that a student may become a teacher after graduation.

Normalization

Figure 1: Example of Teacher allocation list

S.N.	Teacher Name	Address	Email	Module Code	Module Name	Credit Hours
1	Saul Goodman	595 Green Lake Road Black Lake 9115 Lake Street Harrietsfield	Saulthegood man@ abc.edu.np	CC12	Data Structureand Algorithm	**30**
2	Walter White	696 Madison St. Pierrefonds	whitywalker @abc.e du.np	CC12	Data Structureand Algorithm	**30**
3	Santana Lopez	6 Valley View Street Griffintown	Santanalopez @abc.edu.np	CC49	Engineering Thermodynamic	**60**
4	Rust Cohle	89 Coffee Dr. Plaster Rock	rustycohle @abc.ed u.np	SG101	Softwareengineer	**30**
				TG405	**Data Analysis**	**50**

Table 1: Example of Teacher allocation list

Normalization of teacher allocation:

UNF

Identifying the repeating groups among the given figure fields, we get the following UNF:

Teacher (Teacher_ID, Teacher_Name, {Address}, Email, {Module_Code, Module_Name, Credit_Hours,})

1NF

Separating the repeating groups identified in the UNF above and assigning composite keys, we get the following entities:

Teacher-1 (Teacher_ID, Teacher_Name, Email)
Address-1 (Address_ID, Teacher_ID*, Address)
Module-1 (Module_Code, Teacher_ID*, Module_Name, Credit_Hours)

2NF

After the repeating groups were separated into different entities and assigned new composite keys, they needed to be checked for partial dependencies.

For Teacher-1 :

- No composite keys were present and hence there were no partial dependencies.

Teacher_ID → Teacher_Name, Email

Teacher-2 (Teacher_ID, Teacher_Name, Email)

For Address-1:

- Address_ID key determines Address
- Address_ID, Teacher_ID* composite key determines nothing
- Teacher_ID* foreign key determines nothing

Address_ID → (Address)
Address_ID, Teacher_ID* → ()
Teacher_ID → ()

The partial dependencies are separated into new entities as:

Address-2 (Address_ID→ Address)
Address-Teacher-2 (Address_ID, Teacher_ID*)

For Module-1:

- Module_Code key determines Module_Name, Credit_Hours
- Module_Code, Teacher_ID* composite key determines nothing

Module_Code → (Module_Name, Credit_Hours)
Module_Code, Teacher_ID* → ()

Module -2 (Module_Code, Module_Name, Credit_Hours)
Module-Teacher -2 (Module_Code, Teacher_ID*)

Hence, the results of 2NF are the following entities:

Teacher-2	(Teacher_ID, Teacher_Name, Email)
Address-2	(Address_ID→ Address)
Address-Teacher-2	(Address_ID, Teacher_ID*)
Module -2	(Module_Code, Module_Name, Credit_Hours)
Module-Teacher -2	(Module_Code, Teacher_ID*)

3NF

After checking and removing any partial dependencies, transitive dependencies are required to be checked and removed.

For Teacher-2 :

- Teacher_ID determines Teacher_Name and Email

- Teacher_Name determines nothing
- Email determines nothing

There are no transitive dependencies.

Teacher-3 (<u>Teacher_ID</u>, Teacher_Name, Email)

For Address-2:

- Address_ID determines Address
- Address determines nothing

There are no transitive dependencies.

Address-3 (<u>Address_ID</u>, Address)

For Address-Teacher 2:

There are no transitive dependencies, the entity contains only a composite primary key.

Address-Teacher-3 (<u>Address_ID, Teacher_ID*</u>)

For Module-Teacher 2:

There are no transitive dependencies, the entity contains only a composite primary key.

Module-Teacher-3 (<u>Module_ID, Teacher_ID*</u>)

For Module-2:

- Module_Code determines Module_Name, Credit_Hours
- Module_Name determines nothing
- Credit_Hours determines nothing

Module_Code → (Module_Name, Credit_Hours)

The transitive dependencies are separated into entities as follows:

Module-3 → (Module_Code, Module_Name, Credit_Hours)

Hence, the results of 3NF are the following entities :

Teacher-3 (Teacher_ID, Teacher_Name, Email)

Address-3 (Address_ID, Address)

Address-Teacher-3 (Address_ID, Teacher_ID*)

Module-3 (Module_Code, Module_Name, Credit_Hours)

Module-Teacher-3 (Module_ID, Teacher_ID*)

Final entities from figure 1

After normalizing figure 1 up to third normal form, the following entities are obtained:

Teacher (Teacher_ID, Teacher_Name, Email)

Address (Address_ID, Address)

Address-Teacher (Address_ID, Teacher_ID*)

Module (Module_Code, Module_Name, Credit_Hours)

Module-Teacher (Module_ID, Teacher_ID*)

Fig 2: Example of Assignment and Examination Results

Student ID: 149893

Student Name: Mr. William Ishee

Student Address: 2508 Shinn Street New York

Module Code	Module Name	Assignment Type	Grade	Status
CC12	Data Structure and Algorithm	Coursework	A	Pass
CC49	Engineering Thermodynamic	Coursework	B	Pass
CC49	Engineering Thermodynamic	Written Exam	F	Fail
SG101	Software engineer	Individual Assignment	B+	Pass
SG101	Software engineer	Group Assignment	B	Pass
SG101	Software engineer	Unseen Examination	A	Pass

While the given example is very informative, it does not contain information regarding semester and semester fees. Additionally, student attendance as well as student fee payment information also seems missing. Hence, additional fields "semester", "semester fees", "student attendance", and "fee status" seem to be a necessary addition.

Assumptions:

- Attendance of students is recorded for each semester in percentage value.
- Each semester has a fee associated with it.

- Semester fees of the students are common regardless of the modules they study. This implies that students A and B studying module sets C and D have the same semester fees if they both are in semester E.
- When a student pays semester fees, their fee status is updated to paid. The paid amount is not recorded.

UNF

Adding the above-mentioned fields to the given Figure 2 fields, and identifying the repeating groups among them, we get the following UNF:

Student (<u>Student_ID</u>, Sudent_Name, Student_Address, {Module_Code, Module_Name {Assignment_Type, Grade, Status}})

1NF

Separating the repeating groups identified in the UNF above and assigning composite keys, we get the following entities:

Student-1 (<u>Student_ID</u>, Student_Name, Student_Address)
Module-1 (<u>Module_Code, Student_ID*</u>, Module_Name)
Assignment-1 (<u>Assignment_ID, Student_ID*, Module_Code*</u>, Assignment_Type, Grade, Status)

2NF

After the repeating groups were separated into different entities and assigned new composite keys, they needed to be checked for partial dependencies.

For Student-1 :

- No composite keys were present and hence there were no partial dependencies.

  Student_ID → (Student_Name, Student_Address)

Student-2 (Student_ID, Student_Name, Student_Address, Semester_No, Semester_Fees Semester_Attendance, Student_Attendance, Fee_Status)

For Module-1:

- Module_Code key determines Module_Name
- Module_Code, Student_ID* composite key determines nothing
- Student _ID* foreign key determines nothing

Student_ID, Module_Code → ()
Student_ID → ()
Module_Code → (Module_Name)

Module-Student-2 (Student_ID*, Module_Code*)
Module-2 (Module_Code, Module_Name)

For Assignment-1:

- Assignment_ID key determines Assigment_Type
- Assignment_ID, Student_ID*, Module_Code* composite key determines Grade, Status
- Student _ID* foreign key determines nothing
- Module _Code* foreign key determines nothing

Assignment_ID → (Assignment_Type)
Assignment_ID, Student_ID, Module_Code → (Grade, Status)
Student_ID → ()
Module_Code → ()

The partial dependencies are separated into new entities as:
Assignment-2 (Assignment_ID, Assignment_Type)

Student-Assignment-2 (<u>Assignment_ID*, Student_ID*, Module_Code*</u>, Grade, Status)

Hence, the results of 2NF are the following entities:

Student-2 (Student_ID, Student_Name, Student_Address)

Module-Student-2 (Student_ID*, Module_Code*)

Module-2 (Module_Code, Module_Name)

Assignment-2 (Assignment_ID, Assignment_Type)

Student-Assignment-2 (Assignment_ID*, Student_ID*, Module_Code*, Grade, Status)

3NF

After checking and removing any partial dependencies, transitive dependencies are required to be checked and removed.

For Student-2 :

- Student_ID determines Student_Name, Student_Address
- Student_Name determines nothing
- Student_Address determines nothing

Student_ID → (Student_Name, Student_Address)

There are no transitive dependencies.

Student-3 (Student_ID, Student_Name, Student_Address)

For Module-Student-2:

There are no transitive dependencies, the entity contains only a composite primary key.

Module-Student-3 (Student_ID*, Module_Code*)

For Module-2:

- Module_Code determines Module_Name
- Module_Name determines nothing

There are no transitive dependencies.

Module-3 (<u>Module_Code</u>, Module_Name)

For Assignment-2:

- Assignment_ID determines Assigment_Type
- Assignment_Type determines nothing
- Module_Name determines nothing

There are no transitive dependencies.

Assignment-3 (<u>Assignment_ID</u>, Assignment_Type)

For Student-Assignment-2:

- Assignment_ID, Student_ID, Module_Code composite key determines Grade
- Grade determines Status
- Status determines nothing

Assignment_ID*, Student_ID*, Module_Code* → Grade
Grade → Status

The transitive dependencies are separated as:

Student-Assignment-3 (<u>Assignment_ID*, Student_ID*, Module_Code*</u>, Grade_ID*)
Grade-3 (<u>Grade_ID</u>, Grade, Status)

Hence, the results of 3NF are the following entities :

Student-3 (Student_ID, Student_Name, Student_Address)

Module-Student-3 (Student_ID*, Module_Code*)

Module-3 (Module_Code, Module_Name)

Assignment-3 (Assignment_ID, Assignment_Type)

Student-Assignment-3 (Assignment_ID*, Student_ID*, Module_Code*, Grade_ID*)

Grade-3 (Grade_ID, Grade, Status)

Final entities from figure 2

After normalizing figure 2 up to third normal form, the following entities are obtained:

Student (Student_ID, Student_Name, Student_Address)

Module-Student (Student_ID*, Module_Code*)

Module (Module_Code, Module_Name)

Assignment (Assignment_ID, Assignment_Type)

Student-Assignment (Assignment_ID*, Student_ID*, Module_Code*, Grade_ID*)

Grade (Grade_ID, Grade, Status)

Integration

Assumptions:

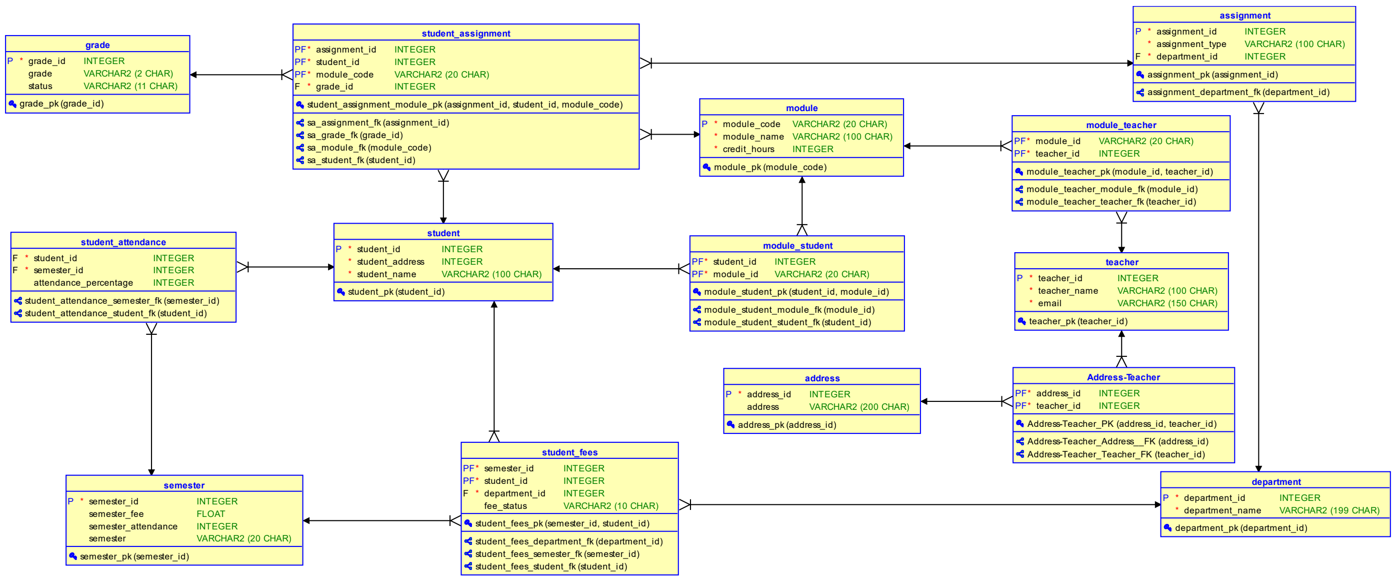
- A teacher may have multiple addresses but a student can only have one address recorded.
- Grade is evaluated in a single alphabet optionally followed by a sign (Ex : A, A+).
- College has additional departments for managing examinations, assessments, results; as well as fees.
- The students also have their attendance recorded as a percentage at the end of each semester before conducting examinations.
- Each semester, students are required to pay the allocated semester fees.
- When a student pays semester fees, their fee status is updated to paid. The paid amount is not recorded.

While the given example tables are very informative, they do not contain information regarding college departments. They also do not contain information regarding semester, semester fees, attendance, and fee status. Thus, the entities "Department", and "Semester", "Student_Fees" and "Student_Attendance" are added to the result of normalizing figures one and two with appropriate relations to existing entities.

Teacher (Teacher_ID, Teacher_Name, Email)
Address (Address_ID, Address)
Address-Teacher (Address_ID, Teacher_ID*)
Module (Module_Code, Module_Name, Credit_Hours)
Module-Teacher (Module_ID*, Teacher_ID*)
Student (Student_ID, Student_Name, Student_Address)
Module-Student (Student_ID*, Module_Code*)
Department (Department_ID, Department_Name)
Assignment (Assignment_ID, Assignment_Type, Department_ID*)
Student-Assignment (Assignment_ID*, Student_ID*, Module_Code*, Grade_ID*)
Grade (Grade_ID, Grade, Status)
Semester (Semester_ID, Semester, Semester_Fees)

Student_Fees (Semester_ID*, Student_ID*, Fee_Status, Department_ID*)

Student_Attendance (Semester_ID*, Student_ID*, Attendance_Percentage)



grade
P * grade_id INTEGER
grade VARCHAR2 (2 CHAR)
status VARCHAR2 (11 CHAR)
grade_pk (grade_id)
student_assignment
PF* assignment_id INTEGER
PF* student_id INTEGER
PF* module_code VARCHAR2 (20 CHAR)
F * grade_id INTEGER
student_assignment_module_pk (assignment_id, student_id, module_code)
sa_assignment_fk (assignment_id)
sa_grade_fk (grade_id)
sa_module_fk (module_code)
sa_student_fk (student_id)
assignment
P * assignment_id INTEGER
* assignment_type VARCHAR2 (100 CHAR)
F * department_id INTEGER
assignment_pk (assignment_id)
assignment_department_fk (department_id)
module
P * module_code VARCHAR2 (20 CHAR)
* module_name VARCHAR2 (100 CHAR)
* credit_hours INTEGER
module_pk (module_code)
module_teacher
PF* module_id VARCHAR2 (20 CHAR)
PF* teacher_id INTEGER
module_teacher_pk (module_id, teacher_id)
module_teacher_module_fk (module_id)
module_teacher_teacher_fk (teacher_id)
student_attendance
F * student_id INTEGER
F * semester_id INTEGER
attendance_percentage INTEGER
student_attendance_semester_fk (semester_id)
student_attendance_student_fk (student_id)
student
P * student_id INTEGER
* student_address INTEGER
* student_name VARCHAR2 (100 CHAR)
student_pk (student_id)
module_student
PF* student_id INTEGER
PF* module_id VARCHAR2 (20 CHAR)
module_student_pk (student_id, module_id)
module_student_module_fk (module_id)
module_student_student_fk (student_id)
teacher
P * teacher_id INTEGER
* teacher_name VARCHAR2 (100 CHAR)
* email VARCHAR2 (150 CHAR)
teacher_pk (teacher_id)
address
P * address_id INTEGER
address VARCHAR2 (200 CHAR)
address_pk (address_id)
Address-Teacher
PF* address_id INTEGER
PF* teacher_id INTEGER
Address-Teacher_PK (address_id, teacher_id)
Address-Teacher_Address__FK (address_id)
Address-Teacher_Teacher_FK (teacher_id)
student_fees
PF* semester_id INTEGER
PF* student_id INTEGER
F * department_id INTEGER
fee_status VARCHAR2 (10 CHAR)
student_fees_pk (semester_id, student_id)
student_fees_department_fk (department_id)
student_fees_semester_fk (semester_id)
student_fees_student_fk (student_id)
semester
P * semester_id INTEGER
semester_fee FLOAT
semester_attendance INTEGER
semester VARCHAR2 (20 CHAR)
semester_pk (semester_id)
department
P * department_id INTEGER
* department_name VARCHAR2 (199 CHAR)
department_pk (department_id)

Figure 1: Final ERD After Integration