

**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**

I confirm that I understand my coursework needs to be submitted online via Google Classroom under the relevant module page before the deadline for my assignment to be accepted and marked. I am fully aware that late submissions will be treated as non-submission and a mark of zero will be awarded.

# 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.

Initially the database is normalized based on given example data and integrated along with assumptions to better fit the given business scenario. The final entities and relations obtained from integration are visualized with Oracle Developer Data Modeler and a DDL script is generated. The generated DDL is then used to generate a database through Oracle SQL, which is then populated with sample data. Webforms are then created to perform CRUD operations on some tables and selective filtration operations on joined tables. Testing is then performed

# Textual Analysis

From the given case study, I derived the following relations between entities:

## Teacher-Module

A teacher is allowed to teach multiple modules.

A module can be taught by multiple teachers.

Hence, a many-to-many relation seems to be established.

Diagram

Description automatically generated

Figure : Relation between Module and Teacher

## Student-Module

A student can study multiple modules.

A module can be studied by multiple students.

Hence a many-to-many relation seems to be established.

Diagram

Description automatically generated

Figure : Relation between Module and Student

## Student-Assignment

A student can partake in multiple assignments.

An assignment is submitted by multiple students.

Hence, a many-to-many relation is established.

Diagram, box and whisker chart

Description automatically generated

Figure : Relation between Student and Assignment

## Department-Fees-Assignment-Result

A department manages multiple student fees.

A department manages multiple assignments.

A department manages multiple results.

Student fee is managed by a single department.

Assignment is managed by a single department.

Result is managed by a single department.

Hence, department establishes a one-to-many relation with fees, assignments, and results.

Diagram

Description automatically generated

Figure : Relation between Department with Fees, Assignments and Results

# ERD From Case Study

Based on the relations derived in the textual analysis of the given case study and some general assumptions, the following ERD is generated.

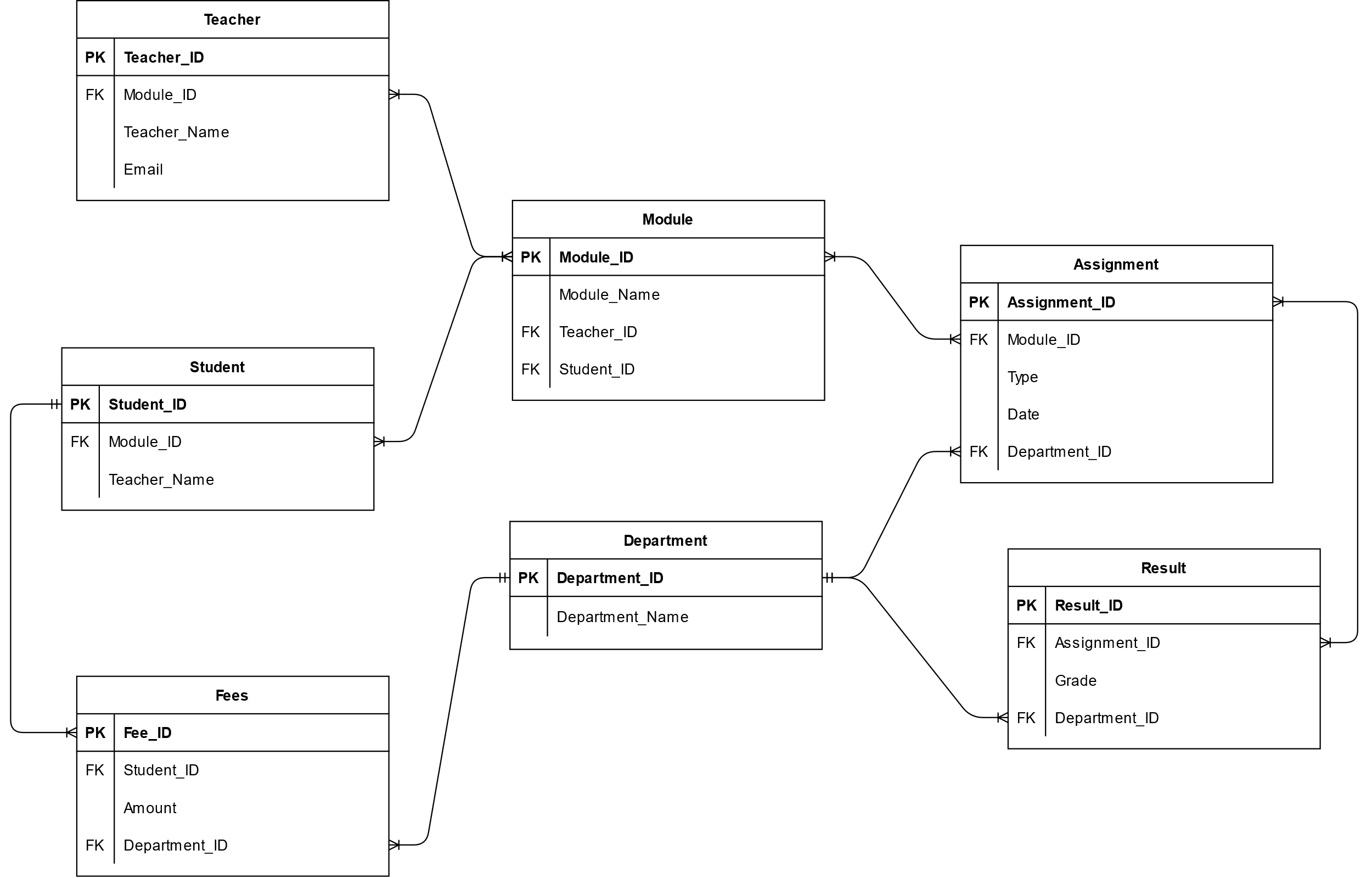


Figure : Initial ERD from Case Study

Assumptions:

* A student pays multiple fees throughout their stay in the college, and a fee belongs to only one student.
* Multiple results are published for the same assignment and a result relates to multiple assignments.
* A module may be included in multiple assignments and an assignment type may be implemented for multiple modules.

# 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 | Saulthegoodman@ abc.edu.np | CC12 | Data Structure and Algorithm | 30 |
| 2 | Walter White | 696 Madison St. Pierrefonds | [whitywalker@abc.e](mailto:whitywalker@abc.e) du.np | CC12 | Data Structure and 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](mailto:rustycohle@abc.ed) u.np | SG101 | Software engineer | 30 |
|  |  |  |  | TG405 | Data Analysis | 50 |

Table : 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, {Street\_No., Street\_Name, State\_Name}, 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\*, Street\_No., Street\_Name, State\_Name)

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 Street\_No., Street\_Name, State\_Name
* Address\_ID, Teacher\_ID\* composite key determines nothing
* Teacher\_ID\* foreign key determines nothing

Address\_ID → (Street\_No., Street\_Name, State\_Name)

Address\_ID, Teacher\_ID\* → ()

Teacher\_ID → ()

The partial dependencies are separated into new entities as:

Address-2 (Address\_ID\*, Street\_No., Street\_Name, State\_Name)

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→ Street\_No., Street\_Name, State\_Name)

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 (Teacher\_ID, Teacher\_Name, Email)

For Address-2:

* Address\_ID determines Street\_No., Street\_Name, State\_Name
* Address determines nothing

There are no transitive dependencies.

Address-3 (Address\_ID, Street\_No., Street\_Name, State\_Name)

For Address-Teacher 2:

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

Address-Teacher-3 (Address\_ID, Teacher\_ID\*)

For Module-Teacher 2:

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

Module-Teacher-3 (Module\_ID, Teacher\_ID\*)

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, Street\_No., Street\_Name, State\_Name)

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 |

**Assumptions**:

* A student cannot have multiple addresses.

### 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 (Student\_ID, Sudent\_Name, Street\_No., Street\_Name, State\_Name, {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(Student\_ID, Student\_Name, Street\_No., Street\_Name, State\_Name)

Module-1 (Module\_Code, Student\_ID\*, Module\_Name)

Assignment-1(Assignment\_ID, Student\_ID\*, Module\_Code\*, 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, Street\_No., Street\_Name, State\_Name)

Student-2 (Student\_ID, Student\_Name, Street\_No., Street\_Name, State\_Name)

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(Assignment\_ID\*, Student\_ID\*, Module\_Code\*, Grade, Status)

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

Student-2 (Student\_ID, Student\_Name, Street\_No., Street\_Name, State\_Name)

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, Street\_No., Street\_Name, State\_Name Student\_Name determines nothing
* Student\_Address determines nothing

Student\_ID → (Student\_Name, Street\_No., Street\_Name, State\_Name)

There are no transitive dependencies.

Student-3 (Student\_ID, Student\_Name, Street\_No., Street\_Name, State\_Name)

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(Module\_Code, 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 (Assignment\_ID, 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 (Assignment\_ID\*, Student\_ID\*, Module\_Code\*, Grade\_ID\*)

Grade-3 (Grade\_ID, Grade, Status)

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

Student-3 (Student\_ID, Student\_Name, Street\_No., Street\_Name, State\_Name)

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, Street\_No., Street\_Name, State\_Name)

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 and Assumptions

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.
* Attendance of students is managed by a different system and recorded for each semester in percentage value in this database at the end of each semester right before examinations are conducted.
* Each semester has a fee associated with it which may or may not be equal for all semesters.
* 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.
* Each semester, students are required to pay the allocated semester fees in a single installment.
* When a student pays semester fees, their fee status is updated to paid.
* To store the fees paid amount, the fee amount is redundantly added to the student\_fees table so it always remains the same even if the semester fees change over time.
* Assignments and their results, attendance, and fees are managed by departments
* A student can study any three module they want.
* A teacher can teach one or multiple modules.
* Different semesters may have different fees.
* Student assignments for each module are recorded only after their grades have been evaluated.
* A student, after graduation, may become a teacher. Hence, a reference for student\_id is added to the teacher table in case a student does become a teacher after graduation.

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, Student\_ID\*)

Address (Address\_ID, Street\_No., Street\_Name, State\_Name)

Address-Teacher (Address\_ID, Teacher\_ID\*)

Module (Module\_Code, Module\_Name, Credit\_Hours)

Module-Teacher (Module\_ID\*, Teacher\_ID\*)

Student (Student\_ID, Student\_Name, Street\_No., Street\_Name, State\_Name)

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\*, Fee\_Amount)

Student\_Attendance (Semester\_ID\*, Student\_ID\*, Attendance\_Percentage)

# Final ERD

After normalization and integration of the given two example figures, the obtained entities form the following ERD:

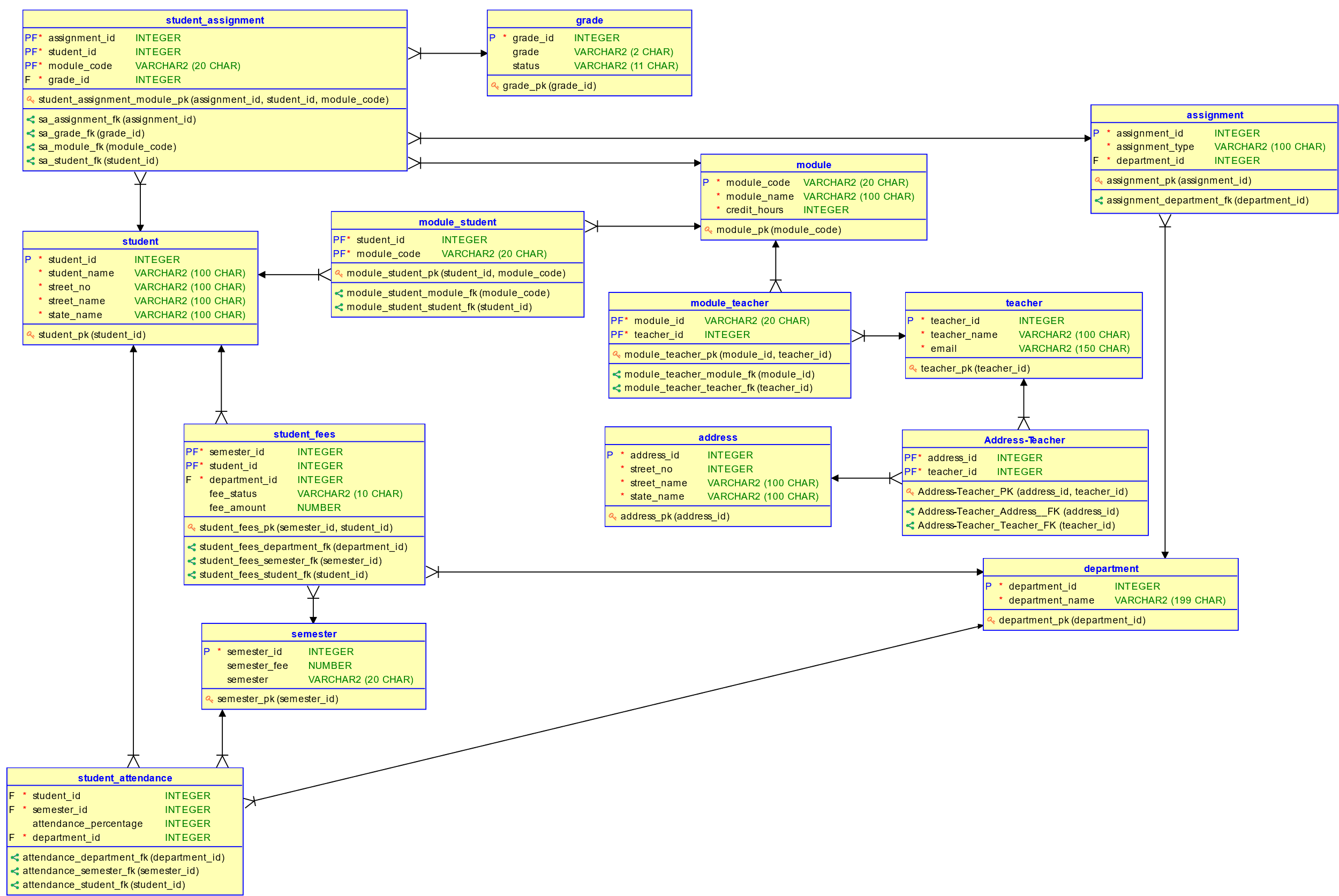


Figure : Final ERD After Integration

# Data Dictionary

## Table: Teacher

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Column Name | Data Type | Size | Constraint | Reference Table | Reference Column | Description | Sample Data |
| Teacher\_ID | INTEGER | 38 | Primary Key |  |  | Uniquely identifies each teacher | 9 |
| Teacher\_Name | VARCHAR | 100 | NOT NULL |  |  | Stores the name of the teacher | 4251 |
| Email | VARCHAR | 100 | NOT NULL, UNIQUE |  |  | Stores the active email address of the teacher | Ethan\_  Cunningham  2529@brety  .org |
| Student\_ID | INTEGER | 38 |  |  |  | Stores student id of the person if they were a graduated student | 6 |

Table : Teacher Table Dictionary

## Table: Student

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Column Name | Data Type | Size | Constraint | Reference Table | Reference Column | Description | Sample Data |
| Student\_ID | INTEGER | 38 | Primary Key |  |  | Uniquely identifies each student | 5 |
| Student\_Name | VARCHAR | 100 | NOT NULL |  |  | Stores the name of the student | Ethan Shrestha |
| Street\_No | INTEGER | 38 | NOT NULL |  |  | Stores street number of an address | 4251 |
| Street\_Name | VARCHAR | 100 | NOT NULL |  |  | Stores street name of an address | Sheffield Walk |
| State\_Name | VARCHAR | 100 | NOT NULL |  |  | Stores state name of an address | Hawaii |

Table : Student Table Dictionary

## Table: Address

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Column Name | Data Type | Size | Constraint | Reference Table | Reference Column | Description | Sample Data |
| Address\_ID | INTEGER | 38 | Primary Key |  |  | Uniquely identifies each address | 3 |
| Street\_No | INTEGER | 38 | NOT NULL |  |  | Stores street number of an address | 4251 |
| Street\_Name | VARCHAR | 100 | NOT NULL |  |  | Stores street name of an address | Sheffield Walk |
| State\_Name | VARCHAR | 100 | NOT NULL |  |  | Stores state name of an address | Hawaii |

Table : Address Table Dictionary

## Table: Department

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Column Name | Data Type | Size | Constraint | Reference Table | Reference Column | Description | Sample Data |
| Department\_ID | INTEGER | 38 | Primary Key |  |  | Uniquely identifies each department | 9 |
| Department\_Name | VARCHAR | 100 | NOT NULL, UNIQUE |  |  | Stores the name of the department | RTE |

Table : Department Table Dictionary

## Table: Assignment

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Column Name | Data Type | Size | Constraint | Reference Table | Reference Column | Description | Sample Data |
| Assignment\_ID | INTEGER | 38 | Primary Key |  |  | Uniquely identifies each assignment | 2 |
| Assignment\_Type | VARCHAR | 100 | NOT NULL, UNIQUE |  |  | Stores the type of current assignment | 4251 |
| Department\_ID | VARCHAR | 100 | NOT NULL | Department | Department  \_ID | Stores the department ID that manages assignments | Sheffield Walk |

Table : Assignment Table Dictionary

## Table: Grade

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Column Name | Data Type | Size | Constraint | Reference Table | Reference Column | Description | Sample Data |
| Grade\_ID | INTEGER | 38 | Primary Key |  |  | Uniquely identifies each grade as a number | 2 |
| Grade | VARCHAR | 100 | NOT NULL, UNIQUE |  |  | Stores the grade | D+ |
| Status | VARCHAR | 100 | NOT NULL |  |  | Stores the pass or fail status of the grade | Pass |

Table : Grade Table Dictionary

## Table: Module

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Column Name | Data Type | Size | Constraint | Reference Table | Reference Column | Description | Sample Data |
| Module\_Code | VARCHAR | 20 | Primary Key |  |  | Uniquely identifies each assignment | CC12 |
| Module\_Name | VARCHAR | 100 | NOT NULL, UNIQUE |  |  | Stores the name of the module | Data Structure and Algorithm |
| Credit\_Hours | INTEGER | 38 | NOT NULL |  |  | Stores the credit hours of the module | 30 |

Table : Module Table Dictionary

## Table: Module\_Student

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Column Name | Data Type | Size | Constraint | Reference Table | Reference Column | Description | Sample Data |
| Student\_ID | VARCHAR | 100 | FOREIGN KEY, PRIMARY KEY (composite primary key) | Student | Student\_ID | Stores the student id of the student associated with a module | 6 |
| Module\_Code | VARCHAR | 20 | FOREIGN KEY, PRIMARY KEY (composite primary key) | Module | Module\_Code | Stores the module code of a module associated with the student | CC12 |

Table : Module\_Student Table Dictionary

## Table: Module\_Teacher

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Column Name | Data Type | Size | Constraint | Reference Table | Reference Column | Description | Sample Data |
| Teacher\_ID | VARCHAR | 100 | FOREIGN KEY, PRIMARY KEY (composite primary key) | Teacher | Teacher\_ID | Stores the teacher id of the teacher associated with a module | 9 |
| Module\_ID | VARCHAR | 20 | FOREIGN KEY, PRIMARY KEY (composite primary key) | Module | Module\_Code | Stores the module code of a module associated with the teacher | CC12 |

Table : Module\_Teacher Table Dictionary

## Table: Semester

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Column Name | Data Type | Size | Constraint | Reference Table | Reference Column | Description | Sample Data |
| Semester\_ID | INTEGER | 38 | Primary Key |  |  | Uniquely identifies each semester | 6 |
| Semester | INTEGER | 38 | NOT NULL, UNIQUE |  |  | Stores the semester number | 9 |
| Semester\_Fees | INTEGER | 38 | NOT NULL |  |  | Stores the fee for the semester | 650 |

Table : Semester Table Dictionary

## Table: Student\_Assignment

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Column Name | Data Type | Size | Constraint | Reference Table | Reference Column | Description | Sample Data |
| Assignment\_ID | INTEGER | 38 | FOREIGN KEY, PRIMARY KEY (composite primary key) | Assignment | Assignment\_ID | Stores the assignment id of the assignment | 6 |
| Student\_ID | INTEGER | 38 | FOREIGN KEY, PRIMARY KEY (composite primary key) | Student | Student\_ID | Stores the student id of the student associated with the assignment | 9 |
| Module\_Code | VARCHAR | 20 | FOREIGN KEY, PRIMARY KEY (composite primary key) | Module | Module\_Code | Stores the module id of the module of which thew assignment is | CC12 |
| Grade\_ID | INTEGER | 38 | FOREIGN KEY | Grade | Grade\_ID | Stores the grade id of the grade obtained by the student | 3 |

Table : Student\_Assignment Table Dictionary

## Table: Student\_Attendance

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Column Name | Data Type | Size | Constraint | Reference Table | Reference Column | Description | Sample Data |
| Student\_ID | INTEGER | 38 | FOREIGN KEY, PRIMARY KEY (composite primary key) | Student | Student\_ID | Stores the student id of the student whose attendance is being recorded | 6 |
| Semester\_ID | INTEGER | 38 | FOREIGN KEY, PRIMARY KEY (composite primary key) | Semester | Semester\_ID | Stores the semester id of the semester in which the attendance was recorded | 3 |
| Department\_ID | INTEGER | 38 | FOREIGN KEY, NOT NULL | Department | Department\_ID | Stores the department id of the department responsible for managing attendance | 1 |

Table : Student\_Attendance Table Dictionary

## Table: Student\_Fees

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Column Name | Data Type | Size | Constraint | Reference Table | Reference Column | Description | Sample Data |
| Student\_ID | INTEGER | 38 | FOREIGN KEY, PRIMARY KEY (composite primary key) | Student | Student\_ID | Stores the student id of the student whose fee detail is being recorded | 6 |
| Semester\_ID | INTEGER | 38 | FOREIGN KEY, PRIMARY KEY (composite primary key) | Semester | Semester\_ID | Stores the semester id of the semester for which the fee detail was recorded | 3 |
| Department\_ID | INTEGER | 38 | FOREIGN KEY, NOT NULL | Department | Department\_ID | Stores the department id of the department responsible for managing fees | 2 |
| Fee\_Amount | NUMBER | (30,8) |  |  |  | Stores the amount of fee paid by the student | 700 |
| Fee\_Status | VARCHAR | 10 |  |  |  | Stores the status of fees determining if the fee has been paid or not | Due |
| Payment\_Date | DATE |  |  |  |  | Stores the date of payment of | TO\_DATE( '09, 01, 2017', 'MM, DD, YYYY' ) |

Table : Student\_Attendance Table Dictionary

## Table: Address\_Teacher

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Column Name | Data Type | Size | Constraint | Reference Table | Reference Column | Description | Sample Data |
| Teacher\_ID | INTEGER | 38 | FOREIGN KEY, PRIMARY KEY (composite primary key) | Teacher | Teacher\_ID | Stores the teacher id of the teacher associated with an address | 6 |
| Address\_ID | INTEGER | 38 | FOREIGN KEY, PRIMARY KEY (composite primary key) | Address | Address\_ID | Stores the address id of the address that is associated with the teacher | 9 |

Table : Address\_Teacher Table Dictionary

# DDL Script

## Script

*-- Generated by Oracle SQL Developer Data Modeler 21.4.1.349.1605*

*--   at:        2022-03-12 18:05:35 NPT*

*--   site:      Oracle Database 11g*

*--   type:      Oracle Database 11g*

*-- predefined type, no DDL - MDSYS.SDO\_GEOMETRY*

*-- predefined type, no DDL - XMLTYPE*

CREATE SEQUENCE address\_address\_id\_seq START WITH 1 NOCACHE ORDER;

CREATE SEQUENCE assignment\_assignment\_id\_seq START WITH 1 NOCACHE ORDER;

CREATE SEQUENCE department\_department\_id\_seq START WITH 1 NOCACHE ORDER;

CREATE SEQUENCE grade\_grade\_id\_seq START WITH 1 NOCACHE ORDER;

CREATE SEQUENCE semester\_semester\_id\_seq START WITH 1 NOCACHE ORDER;

CREATE SEQUENCE student\_student\_id\_seq START WITH 1 NOCACHE ORDER;

CREATE SEQUENCE teacher\_teacher\_id\_seq START WITH 1 NOCACHE ORDER;

CREATE TABLE address (

    address\_id  INTEGER NOT NULL,

    street\_no   INTEGER NOT NULL,

    street\_name VARCHAR2(100 CHAR) NOT NULL,

    state\_name  VARCHAR2(100 CHAR) NOT NULL

)

;

ALTER TABLE address ADD *CONSTRAINT* address\_pk PRIMARY KEY ( address\_id );

CREATE TABLE address\_teacher (

    address\_id INTEGER NOT NULL,

    teacher\_id INTEGER NOT NULL

)

;

ALTER TABLE address\_teacher ADD *CONSTRAINT* "Address\_Teacher\_PK" PRIMARY KEY ( address\_id,

                                                                                teacher\_id );

CREATE TABLE assignment (

    assignment\_id   INTEGER NOT NULL,

    assignment\_type VARCHAR2(100 CHAR) NOT NULL,

    department\_id   INTEGER DEFAULT 1 NOT NULL

)

;

ALTER TABLE assignment ADD *CONSTRAINT* assignment\_pk PRIMARY KEY ( assignment\_id );

ALTER TABLE assignment ADD *CONSTRAINT* assignment\_assignment\_type\_un UNIQUE ( assignment\_type );

CREATE TABLE department (

    department\_id   INTEGER NOT NULL,

    department\_name VARCHAR2(199 CHAR) NOT NULL

)

;

ALTER TABLE department ADD *CONSTRAINT* department\_pk PRIMARY KEY ( department\_id );

ALTER TABLE department ADD *CONSTRAINT* department\_department\_name\_un UNIQUE ( department\_name );

CREATE TABLE grade (

    grade\_id INTEGER NOT NULL,

    grade    VARCHAR2(2 CHAR),

    status   VARCHAR2(11 CHAR)

)

;

ALTER TABLE grade ADD *CONSTRAINT* grade\_pk PRIMARY KEY ( grade\_id );

ALTER TABLE grade ADD *CONSTRAINT* grade\_\_un UNIQUE ( grade );

CREATE TABLE module (

    module\_code  VARCHAR2(20 CHAR) NOT NULL,

    module\_name  VARCHAR2(100 CHAR) NOT NULL,

    credit\_hours INTEGER NOT NULL

)

;

ALTER TABLE module ADD *CONSTRAINT* module\_pk PRIMARY KEY ( module\_code );

ALTER TABLE module ADD *CONSTRAINT* module\_module\_name\_un UNIQUE ( module\_name );

CREATE TABLE module\_student (

    student\_id  INTEGER NOT NULL,

    module\_code VARCHAR2(20 CHAR) NOT NULL

)

;

ALTER TABLE module\_student ADD *CONSTRAINT* module\_student\_pk PRIMARY KEY ( student\_id,

                                                                          module\_code );

CREATE TABLE module\_teacher (

    module\_id  VARCHAR2(20 CHAR) NOT NULL,

    teacher\_id INTEGER NOT NULL

)

;

ALTER TABLE module\_teacher ADD *CONSTRAINT* module\_teacher\_pk PRIMARY KEY ( module\_id,

                                                                          teacher\_id );

CREATE TABLE semester (

    semester\_id  INTEGER NOT NULL,

    semester\_fee NUMBER,

    semester     VARCHAR2(20 CHAR)

)

;

ALTER TABLE semester ADD *CONSTRAINT* semester\_pk PRIMARY KEY ( semester\_id );

ALTER TABLE semester ADD *CONSTRAINT* semester\_semester\_un UNIQUE ( semester );

CREATE TABLE student (

    student\_id   INTEGER NOT NULL,

    student\_name VARCHAR2(100 CHAR) NOT NULL,

    street\_no    VARCHAR2(100 CHAR) NOT NULL,

    street\_name  VARCHAR2(100 CHAR) NOT NULL,

    state\_name   VARCHAR2(100 CHAR) NOT NULL

)

;

ALTER TABLE student ADD *CONSTRAINT* student\_pk PRIMARY KEY ( student\_id );

CREATE TABLE student\_assignment (

    assignment\_id INTEGER NOT NULL,

    student\_id    INTEGER NOT NULL,

    module\_code   VARCHAR2(20 CHAR) NOT NULL,

    grade\_id      INTEGER NOT NULL

)

;

ALTER TABLE student\_assignment

    ADD *CONSTRAINT* student\_assignment\_module\_pk PRIMARY KEY ( assignment\_id,

                                                              student\_id,

                                                              module\_code );

CREATE TABLE student\_attendance (

    student\_id            INTEGER NOT NULL,

    semester\_id           INTEGER NOT NULL,

    attendance\_percentage INTEGER,

    department\_id         INTEGER DEFAULT 1 NOT NULL

)

;

ALTER TABLE student\_attendance ADD *CONSTRAINT* student\_attendance\_pk PRIMARY KEY (student\_id,

                                                                                    semester\_id);

CREATE TABLE student\_fees (

    semester\_id   INTEGER NOT NULL,

    student\_id    INTEGER NOT NULL,

    department\_id INTEGER DEFAULT 2 NOT NULL,

    fee\_status    VARCHAR2(10 CHAR),

    fee\_amount    NUMBER(30,8),

    payment\_date  DATE

)

;

ALTER TABLE student\_fees ADD *CONSTRAINT* student\_fees\_pk PRIMARY KEY ( semester\_id,

                                                                      student\_id );

CREATE TABLE teacher (

    teacher\_id   INTEGER NOT NULL,

    teacher\_name VARCHAR2(100 CHAR) NOT NULL,

    email        VARCHAR2(150 CHAR) NOT NULL,

    student\_id   INTEGER DEFAULT NULL

)

;

ALTER TABLE teacher ADD *CONSTRAINT* teacher\_pk PRIMARY KEY ( teacher\_id );

ALTER TABLE teacher ADD *CONSTRAINT* teacher\_email\_un UNIQUE ( email );

ALTER TABLE address\_teacher

    ADD *CONSTRAINT* "address\_teacher\_Address\_\_FK" *FOREIGN KEY* ( address\_id )

*REFERENCES* address ( address\_id )

*ON DELETE CASCADE*

    NOT DEFERRABLE;

ALTER TABLE address\_teacher

    ADD *CONSTRAINT* "address\_teacher\_Teacher\_FK" *FOREIGN KEY* ( teacher\_id )

*REFERENCES* teacher ( teacher\_id )

*ON DELETE CASCADE*

    NOT DEFERRABLE;

ALTER TABLE assignment

    ADD *CONSTRAINT* assignment\_department\_fk *FOREIGN KEY* ( department\_id )

*REFERENCES* department ( department\_id )

*ON DELETE CASCADE*

    NOT DEFERRABLE;

ALTER TABLE student\_attendance

    ADD *CONSTRAINT* attendance\_department\_fk *FOREIGN KEY* ( department\_id )

*REFERENCES* department ( department\_id )

*ON DELETE CASCADE*

    NOT DEFERRABLE;

ALTER TABLE student\_attendance

    ADD *CONSTRAINT* attendance\_semester\_fk *FOREIGN KEY* ( semester\_id )

*REFERENCES* semester ( semester\_id )

*ON DELETE CASCADE*

    NOT DEFERRABLE;

ALTER TABLE student\_attendance

    ADD *CONSTRAINT* attendance\_student\_fk *FOREIGN KEY* ( student\_id )

*REFERENCES* student ( student\_id )

*ON DELETE CASCADE*

    NOT DEFERRABLE;

ALTER TABLE module\_student

    ADD *CONSTRAINT* module\_student\_module\_fk *FOREIGN KEY* ( module\_code )

*REFERENCES* module ( module\_code )

*ON DELETE CASCADE*

    NOT DEFERRABLE;

ALTER TABLE module\_student

    ADD *CONSTRAINT* module\_student\_student\_fk *FOREIGN KEY* ( student\_id )

*REFERENCES* student ( student\_id )

*ON DELETE CASCADE*

    NOT DEFERRABLE;

ALTER TABLE module\_teacher

    ADD *CONSTRAINT* module\_teacher\_module\_fk *FOREIGN KEY* ( module\_id )

*REFERENCES* module ( module\_code )

*ON DELETE CASCADE*

    NOT DEFERRABLE;

ALTER TABLE module\_teacher

    ADD *CONSTRAINT* module\_teacher\_teacher\_fk *FOREIGN KEY* ( teacher\_id )

*REFERENCES* teacher ( teacher\_id )

*ON DELETE CASCADE*

    NOT DEFERRABLE;

ALTER TABLE student\_assignment

    ADD *CONSTRAINT* sa\_assignment\_fk *FOREIGN KEY* ( assignment\_id )

*REFERENCES* assignment ( assignment\_id )

*ON DELETE CASCADE*

    NOT DEFERRABLE;

ALTER TABLE student\_assignment

    ADD *CONSTRAINT* sa\_grade\_fk *FOREIGN KEY* ( grade\_id )

*REFERENCES* grade ( grade\_id )

*ON DELETE CASCADE*

    NOT DEFERRABLE;

ALTER TABLE student\_assignment

    ADD *CONSTRAINT* sa\_module\_fk *FOREIGN KEY* ( module\_code )

*REFERENCES* module ( module\_code )

*ON DELETE CASCADE*

    NOT DEFERRABLE;

ALTER TABLE student\_assignment

    ADD *CONSTRAINT* sa\_student\_fk *FOREIGN KEY* ( student\_id )

*REFERENCES* student ( student\_id )

*ON DELETE CASCADE*

    NOT DEFERRABLE;

ALTER TABLE student\_fees

    ADD *CONSTRAINT* student\_fees\_department\_fk *FOREIGN KEY* ( department\_id )

*REFERENCES* department ( department\_id )

*ON DELETE CASCADE*

    NOT DEFERRABLE;

ALTER TABLE student\_fees

    ADD *CONSTRAINT* student\_fees\_semester\_fk *FOREIGN KEY* ( semester\_id )

*REFERENCES* semester ( semester\_id )

*ON DELETE CASCADE*

    NOT DEFERRABLE;

ALTER TABLE student\_fees

    ADD *CONSTRAINT* student\_fees\_student\_fk *FOREIGN KEY* ( student\_id )

*REFERENCES* student ( student\_id )

*ON DELETE CASCADE*

    NOT DEFERRABLE;

ALTER TABLE teacher

    ADD *CONSTRAINT* teacher\_student\_fk *FOREIGN KEY* ( student\_id )

*REFERENCES* student ( student\_id )

*ON DELETE CASCADE*

    NOT DEFERRABLE;

CREATE OR REPLACE TRIGGER address\_address\_id\_trg BEFORE

    INSERT ON address

    FOR EACH ROW

    WHEN ( new.address\_id IS NULL )

BEGIN

    :new.address\_id := address\_address\_id\_seq.nextval;

END;

/

CREATE OR REPLACE TRIGGER assignment\_assignment\_id\_trg BEFORE

    INSERT ON assignment

    FOR EACH ROW

    WHEN ( new.assignment\_id IS NULL )

BEGIN

    :new.assignment\_id := assignment\_assignment\_id\_seq.nextval;

END;

/

CREATE OR REPLACE TRIGGER department\_department\_id\_trg BEFORE

    INSERT ON department

    FOR EACH ROW

    WHEN ( new.department\_id IS NULL )

BEGIN

    :new.department\_id := department\_department\_id\_seq.nextval;

END;

/

CREATE OR REPLACE TRIGGER grade\_grade\_id\_trg BEFORE

    INSERT ON grade

    FOR EACH ROW

    WHEN ( new.grade\_id IS NULL )

BEGIN

    :new.grade\_id := grade\_grade\_id\_seq.nextval;

END;

/

CREATE OR REPLACE TRIGGER semester\_semester\_id\_trg BEFORE

    INSERT ON semester

    FOR EACH ROW

    WHEN ( new.semester\_id IS NULL )

BEGIN

    :new.semester\_id := semester\_semester\_id\_seq.nextval;

END;

/

CREATE OR REPLACE TRIGGER student\_student\_id\_trg BEFORE

    INSERT ON student

    FOR EACH ROW

    WHEN ( new.student\_id IS NULL )

BEGIN

    :new.student\_id := student\_student\_id\_seq.nextval;

END;

/

CREATE OR REPLACE TRIGGER teacher\_teacher\_id\_trg BEFORE

    INSERT ON teacher

    FOR EACH ROW

    WHEN ( new.teacher\_id IS NULL )

BEGIN

    :new.teacher\_id := teacher\_teacher\_id\_seq.nextval;

END;

/

*-- Oracle SQL Developer Data Modeler Summary Report:*

*--*

*-- CREATE TABLE                            14*

*-- CREATE INDEX                             0*

*-- ALTER TABLE                             30*

*-- CREATE VIEW                              0*

*-- ALTER VIEW                               0*

*-- CREATE PACKAGE                           0*

*-- CREATE PACKAGE BODY                      0*

*-- CREATE PROCEDURE                         0*

*-- CREATE FUNCTION                          0*

*-- CREATE TRIGGER                           7*

*-- ALTER TRIGGER                            0*

*-- CREATE COLLECTION TYPE                   0*

*-- CREATE STRUCTURED TYPE                   0*

*-- CREATE STRUCTURED TYPE BODY              0*

*-- CREATE CLUSTER                           0*

*-- CREATE CONTEXT                           0*

*-- CREATE DATABASE                          0*

*-- CREATE DIMENSION                         0*

*-- CREATE DIRECTORY                         0*

*-- CREATE DISK GROUP                        0*

*-- CREATE ROLE                              0*

*-- CREATE ROLLBACK SEGMENT                  0*

*-- CREATE SEQUENCE                          7*

*-- CREATE MATERIALIZED VIEW                 0*

*-- CREATE MATERIALIZED VIEW LOG             0*

*-- CREATE SYNONYM                           0*

*-- CREATE TABLESPACE                        0*

*-- CREATE USER                              0*

*--*

*-- DROP TABLESPACE                          0*

*-- DROP DATABASE                            0*

*--*

*-- REDACTION POLICY                         0*

*--*

*-- ORDS DROP SCHEMA                         0*

*-- ORDS ENABLE SCHEMA                       0*

*-- ORDS ENABLE OBJECT                       0*

*--*

*-- ERRORS                                   0*

*-- WARNINGS                                 0*

## Output

Graphical user interface, text, application

Description automatically generated

Figure :DDL Script Output 1

Graphical user interface, text, application

Description automatically generated

Figure :DDL Script Output 2

Graphical user interface, text, application

Description automatically generated

Figure :DDL Script Output 3

**Graphical user interface, text, application

Description automatically generated**

Figure :DDL Script Output 4

**Graphical user interface, text, application

Description automatically generated**

Figure :DDL Script Output 5

**Graphical user interface, text, application

Description automatically generated** Figure :DDL Script Output 6

# INSERT Statements

## Teacher

Graphical user interface, text, application

Description automatically generated

Figure : INSERT Teacher

## Student

Table

Description automatically generated Figure : INSERT Student

## Semester

Graphical user interface, text, application

Description automatically generated Figure : INSERT Semester

## Module

Graphical user interface, text

Description automatically generated with medium confidence

Figure : INSERT Module

## Grade

Graphical user interface, text

Description automatically generated

Figure : INSERT Grade

## Department

Graphical user interface, text, application

Description automatically generated

Figure : INSERT Department

## Address

Graphical user interface, text, application

Description automatically generated Figure : INSERT Address

## Assignment

Graphical user interface, text

Description automatically generated

Figure : INSERT Assignment

## Student\_Fees

Graphical user interface, application

Description automatically generated Figure : INSERT Student\_Fees

## Student\_Attendance

Graphical user interface, text

Description automatically generated Figure : INSERT Student\_Attendance

## Student\_Assignment

Graphical user interface, text, application

Description automatically generated Figure : INSERT Student\_Assignment

## Module\_Teacher

Graphical user interface, text, application

Description automatically generated Figure : INSERT Module\_Teacher

## Module\_Student

Graphical user interface, application

Description automatically generated Figure : INSERT Module\_Student

## Address\_Teacher

Graphical user interface, text

Description automatically generated

Figure : INSERT Address\_Teacher

# SELECT Statements

## Address

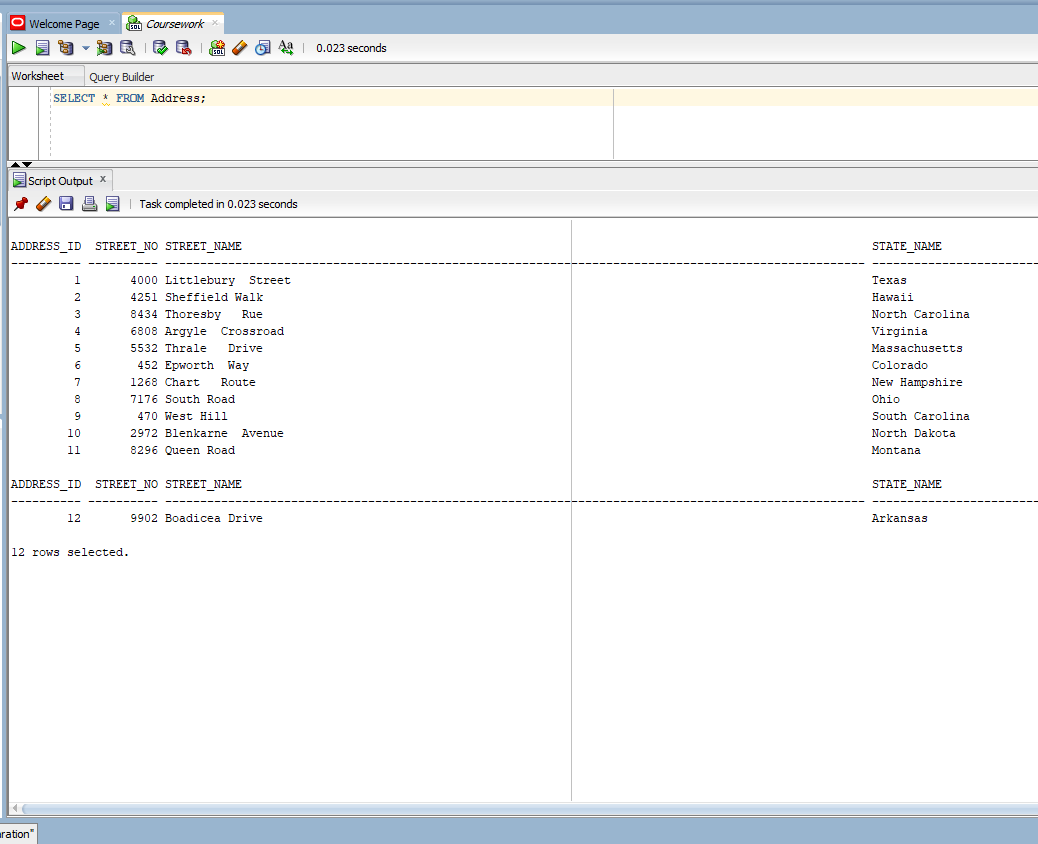


Figure :Select Address

## Address\_Teacher

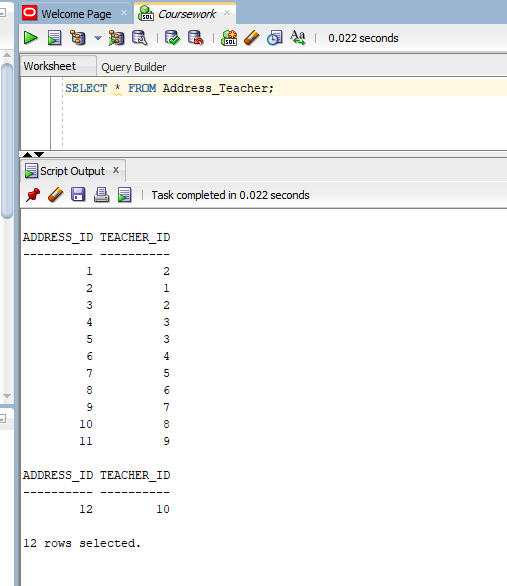


Figure :Select Address\_Teacher

## Assignment

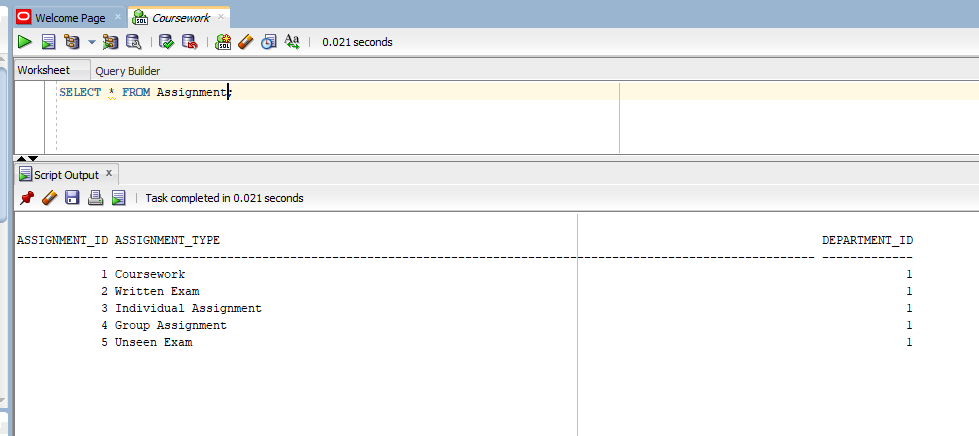


Figure :Select Assignment

## Grade

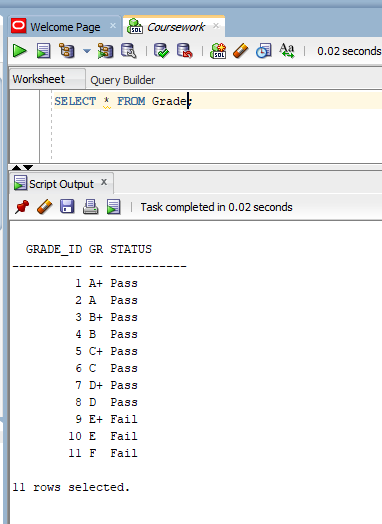


Figure :Select Grade

## Module

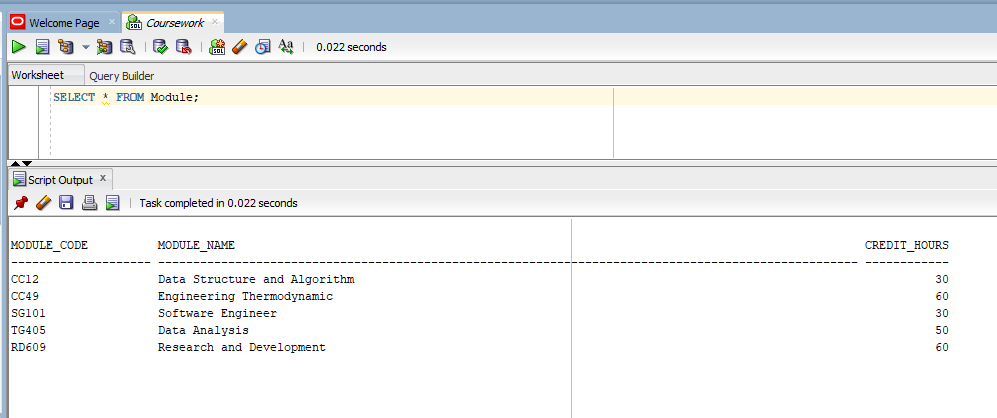


Figure :Select Module

## Module\_Student

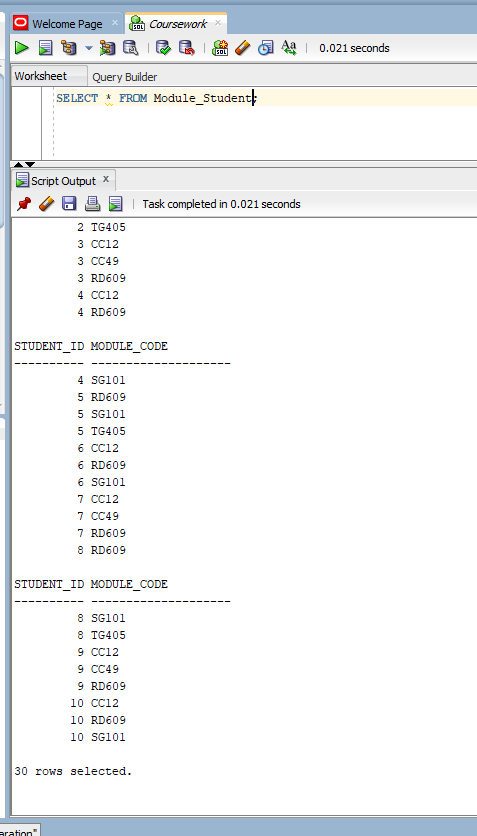


Figure :Select Module\_Student

## Module\_Teacher

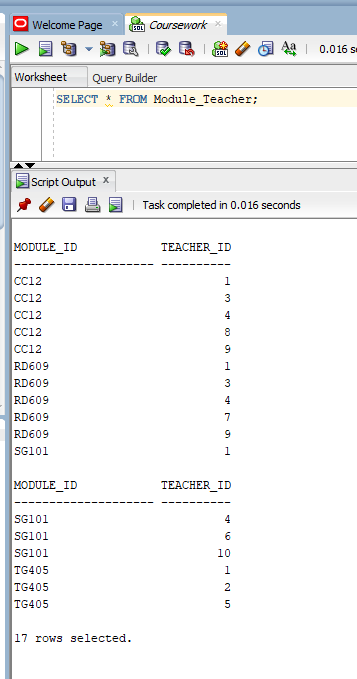


Figure :Select Module\_Teacher

## Semester

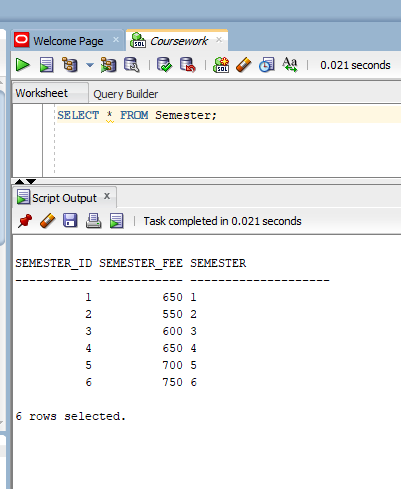


Figure :Select Semester

## Student

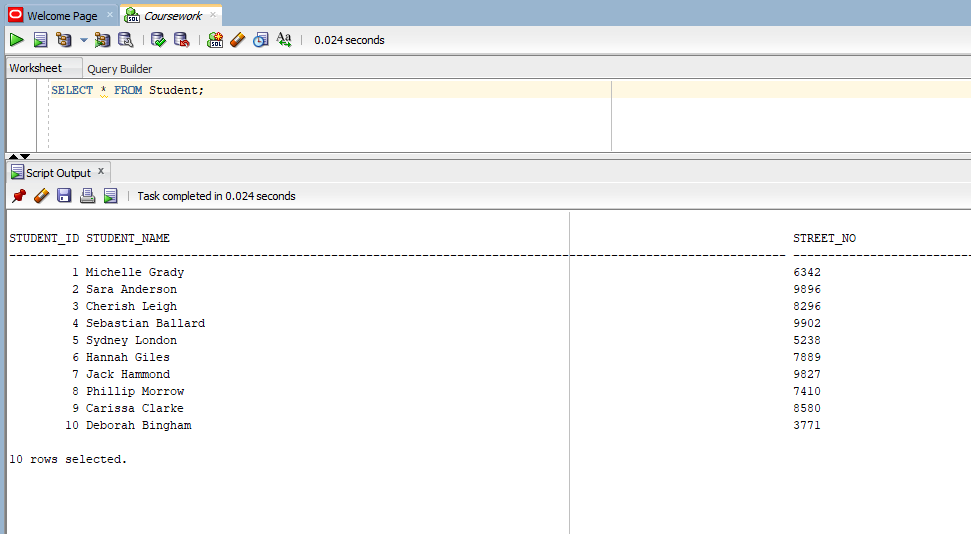


Figure :Select Student

## Student\_Assignment

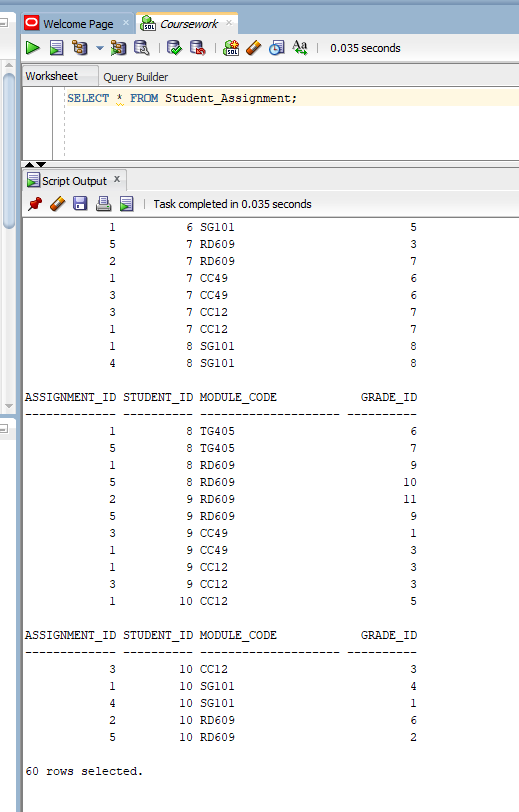


Figure :Select Student\_Assignment

## Student\_Attendance

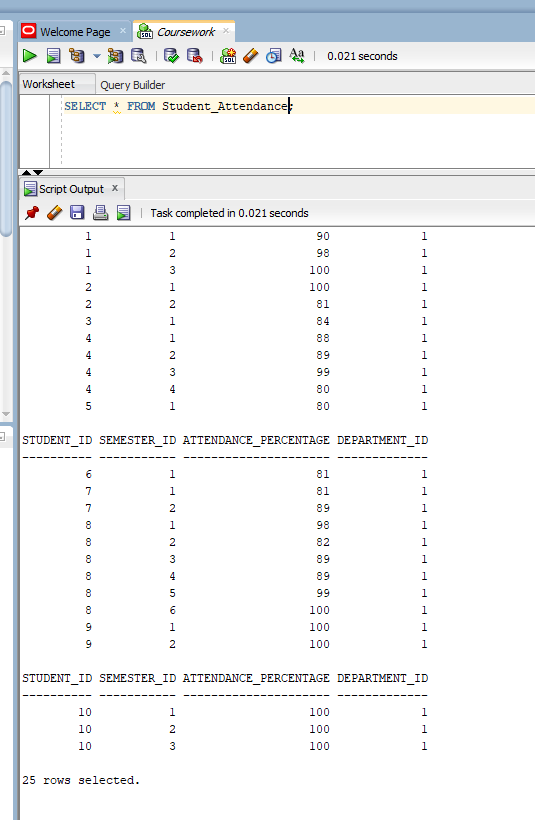


Figure :Select Attendance

## Student\_Fees

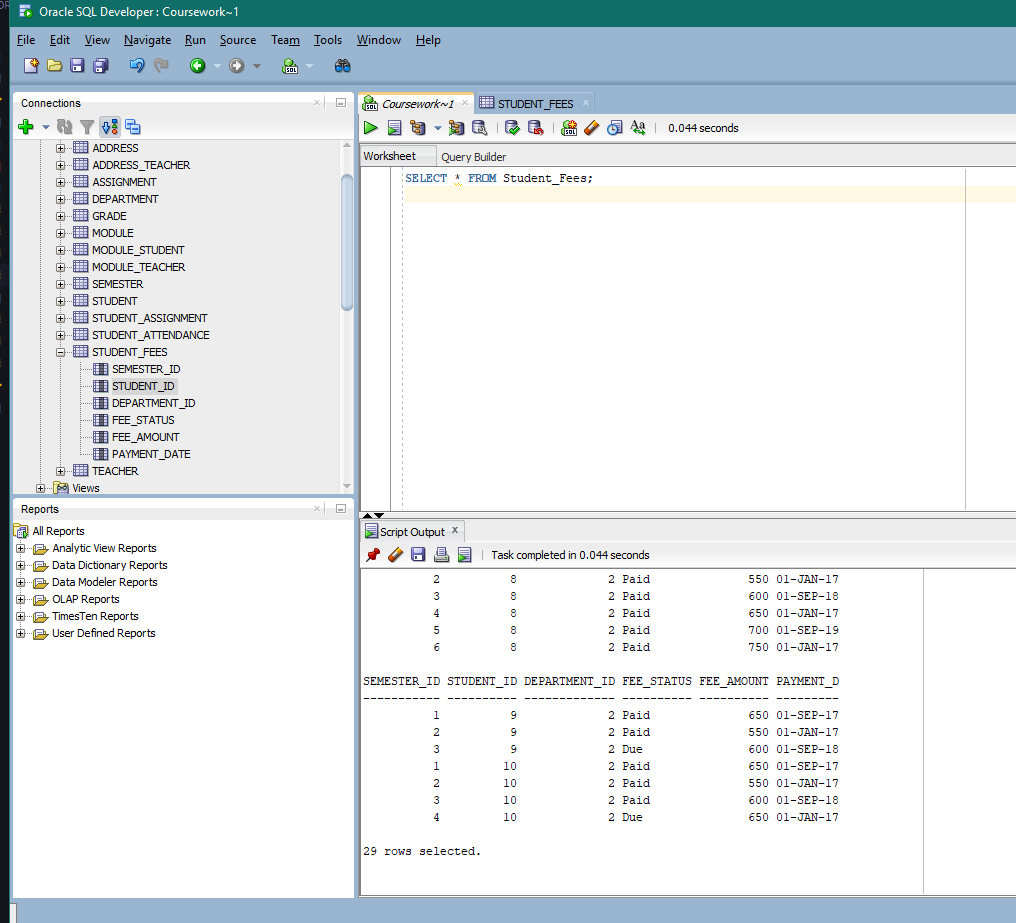


Figure :Select Fees

## Teacher

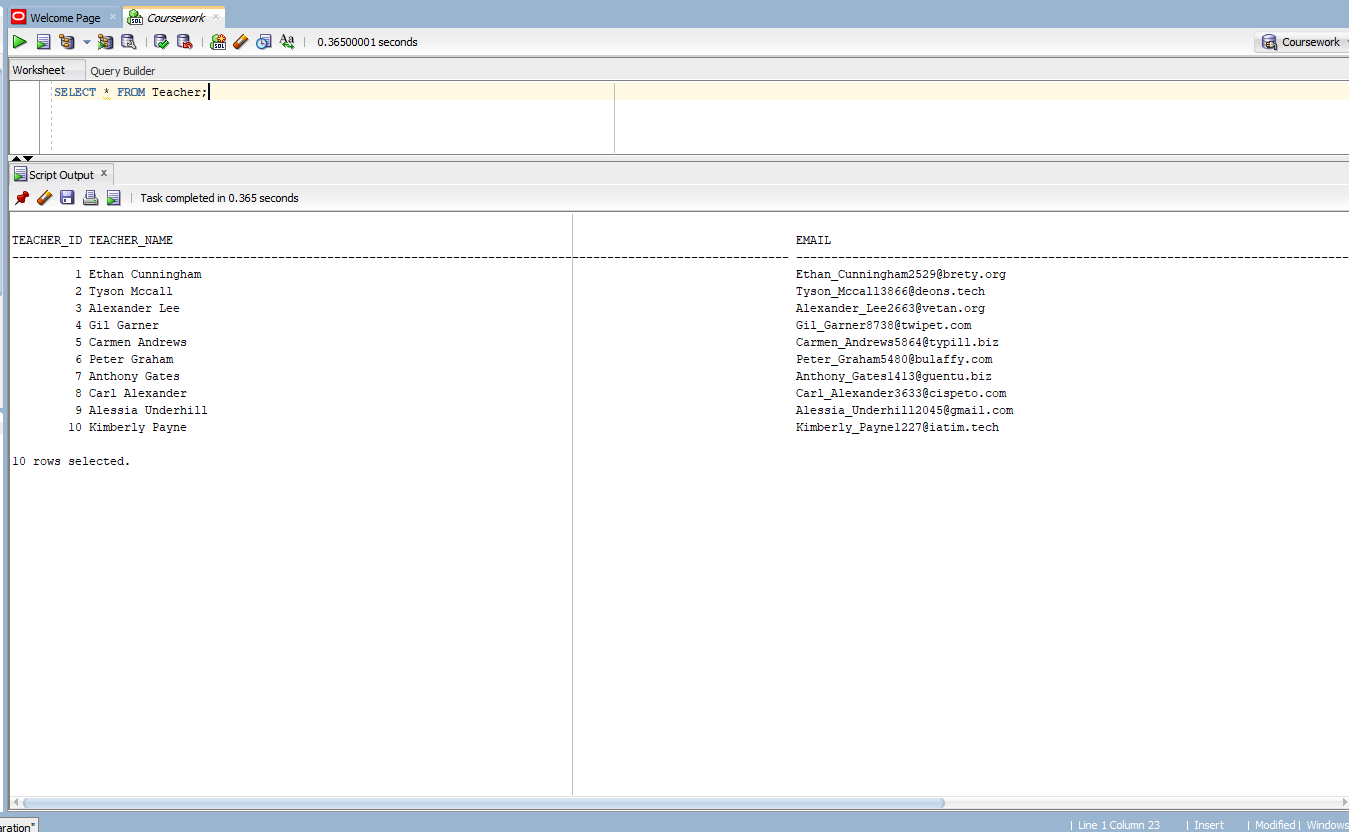


Figure :Select Teacher

# Forms

## Dashboard

Website

Description automatically generated with low confidence

Figure : WebForms Dashboard

## Basic Forms

### Teacher Form

Graphical user interface, text, application

Description automatically generated

Figure : Teacher Form

### Address Form

Graphical user interface, application

Description automatically generated

Figure : Teacher Form

### Module Form

Graphical user interface, text, application

Description automatically generated

Figure : Module Form

### Department Form

Graphical user interface, application

Description automatically generated

Figure : Department Form

### Student Form

Graphical user interface, application

Description automatically generated

Figure : Student Form

## Complex Forms

### Teacher-Module Mapping

#### Form

Graphical user interface, application, Teams

Description automatically generated

Figure : Teacher Module Mapping Form Before Selection

Graphical user interface, application

Description automatically generated Figure : Teacher Module Mapping Form After Selection

#### Query

Assuming the selected student has student\_id 1, the query is as following:

1. SELECT m.Module\_Code as "Module Code", mt.Teacher\_ID "Teacher ID", m.Module\_Name as "Module Name", m.Credit\_Hours "Credit Hours"
2. FROM Module m
3. INNER JOIN Module\_Teacher mt ON mt.Module\_ID = m.Module\_Code
4. WHERE mt.Teacher\_ID = 1
5. ;

In the webform the query string is written as:

1. $@"SELECT m.Module\_Code as ""Module Code"", mt.Teacher\_ID ""Teacher ID"", m.Module\_Name as ""Module Name"", m.Credit\_Hours ""Credit Hours""
2. FROM Module m
3. INNER JOIN Module\_Teacher mt ON mt.Module\_ID = m.Module\_Code
4. WHERE mt.Teacher\_ID = {teacherID}
5. ";

## Student-Fees Mapping

#### Form

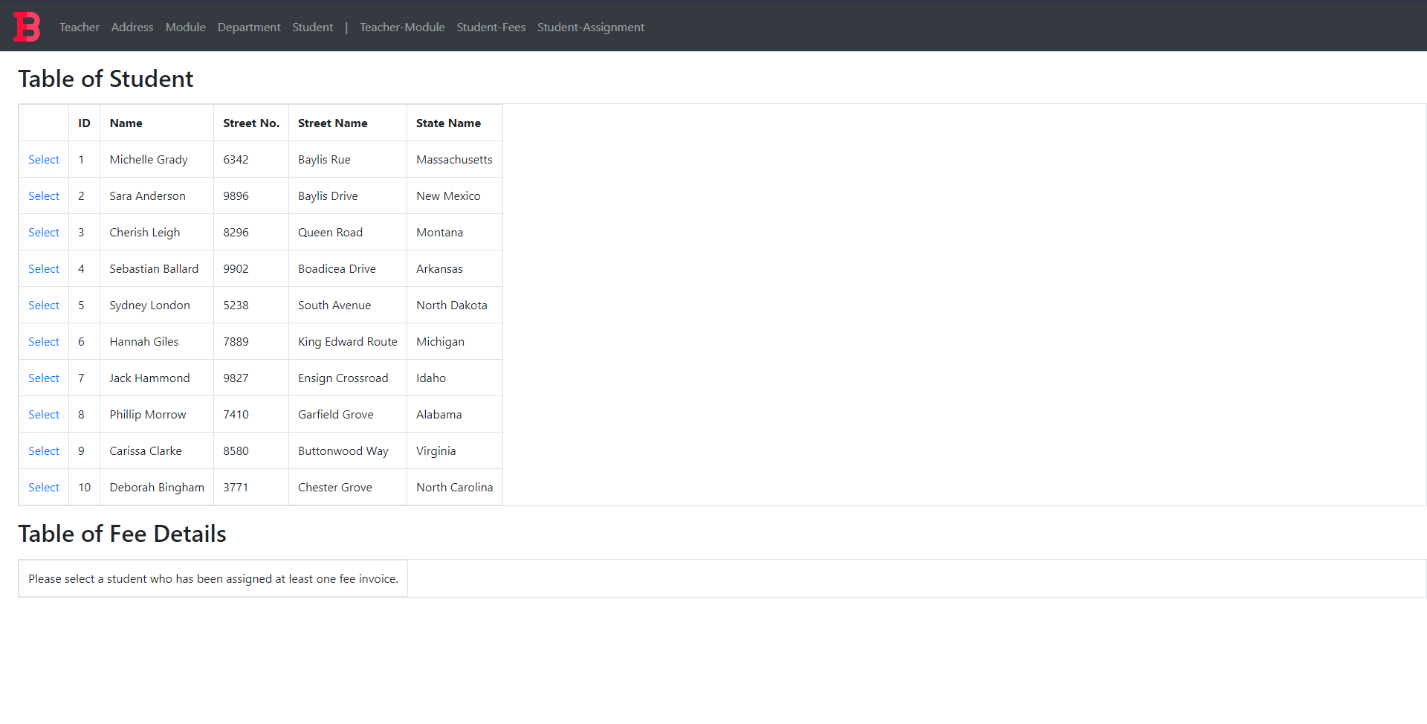
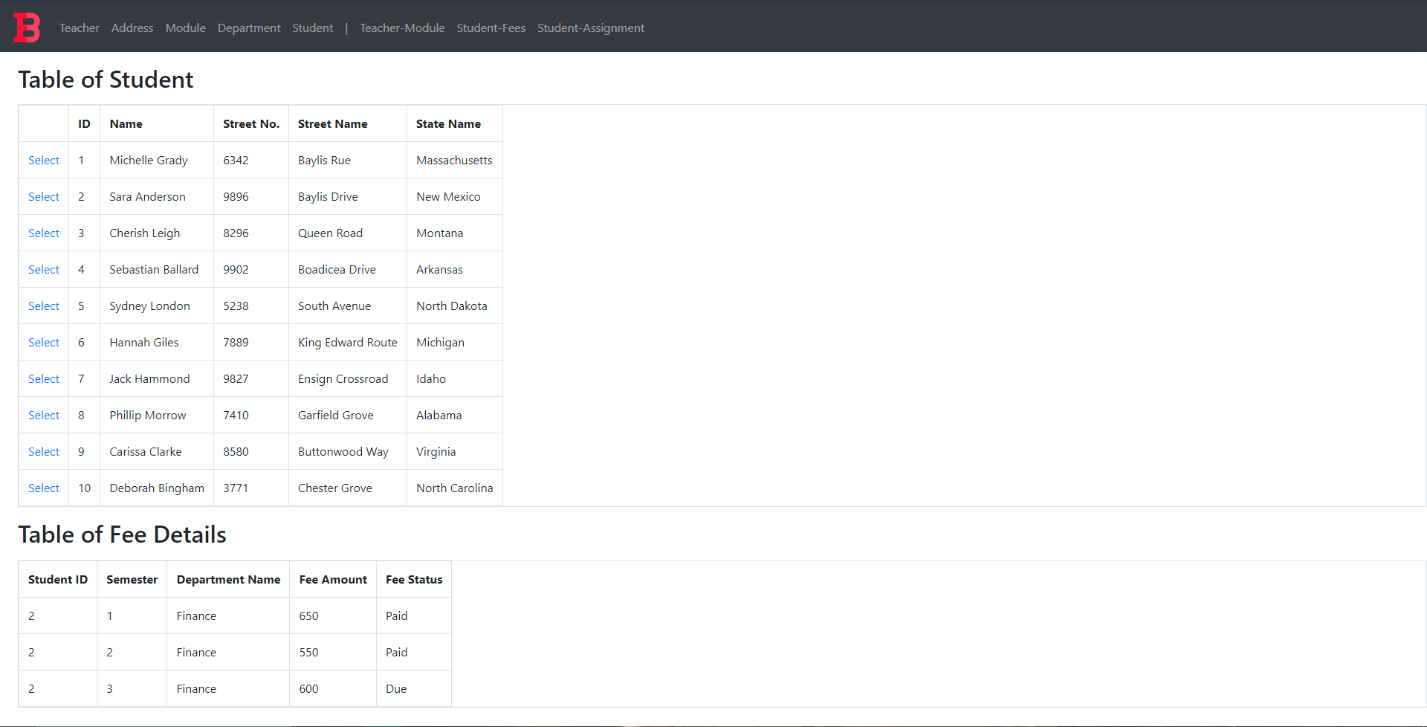


Figure : Student-Fees Mapping Form Before Selection

 Figure : Student-Fees Mapping Form After Selection

#### Query

Assuming the selected student has student\_id 1, the query is as following:

1. SELECT sf.Student\_ID "Student ID", se.Semester "Semester", d.Department\_Name as "Department Name", sf.Fee\_Amount "Fee Amount", sf.Fee\_Status "Fee Status"
2. FROM Student s
3. INNER JOIN Student\_Fees sf ON sf.Student\_ID = s.Student\_ID
4. INNER JOIN Semester se ON se.Semester\_ID = sf.Semester\_ID
5. INNER JOIN Department d ON d.Department\_ID = sf.Department\_ID
6. WHERE sf.Student\_ID = 1
7. ;

In the webform the query string is written as:

1. $@"SELECT sf.Student\_ID ""Student ID"", se.Semester ""Semester"", d.Department\_Name as ""Department Name"", sf.Fee\_Amount ""Fee Amount"", sf.Fee\_Status ""Fee Status""
2. FROM Student s
3. INNER JOIN Student\_Fees sf ON sf.Student\_ID = s.Student\_ID
4. INNER JOIN Semester se ON se.Semester\_ID = sf.Semester\_ID
5. INNER JOIN Department d ON d.Department\_ID = sf.Department\_ID
6. WHERE sf.Student\_ID = {studentID}
7. ";

## Student-Assignment Mapping

#### Form

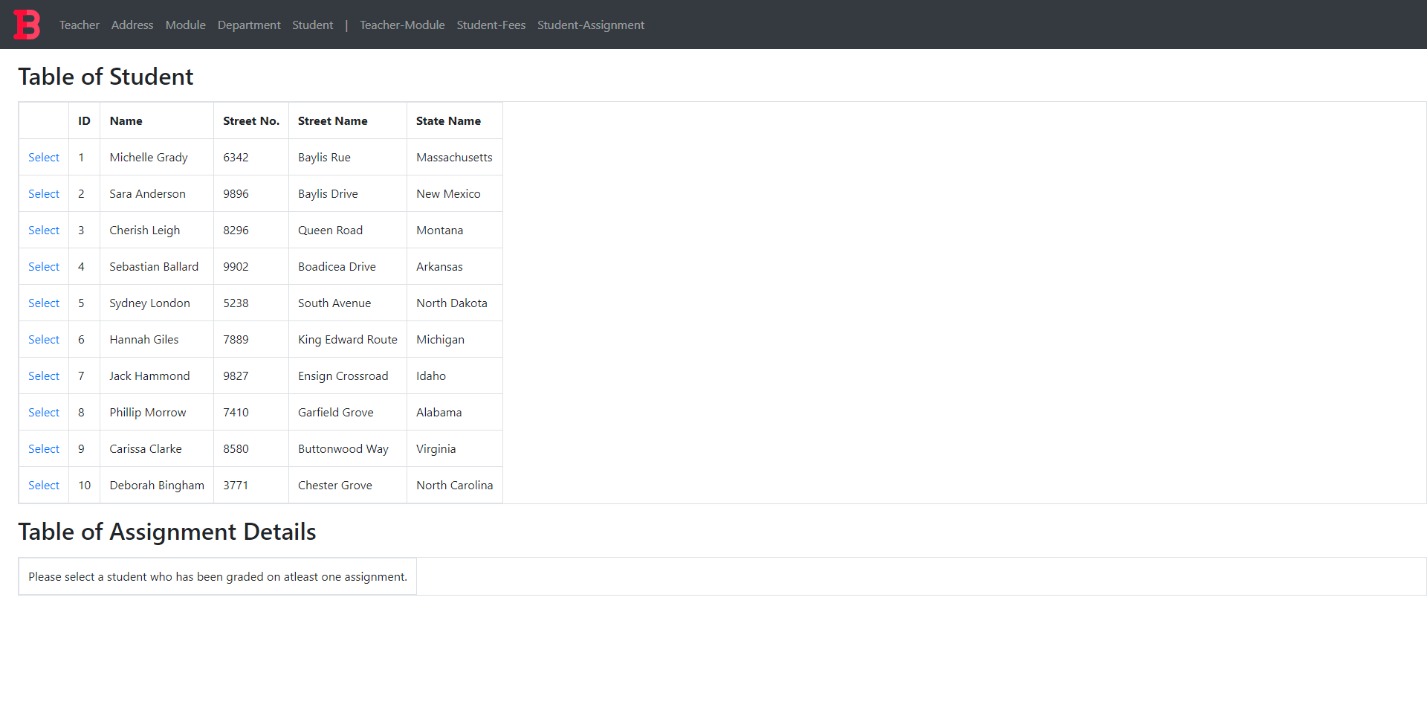
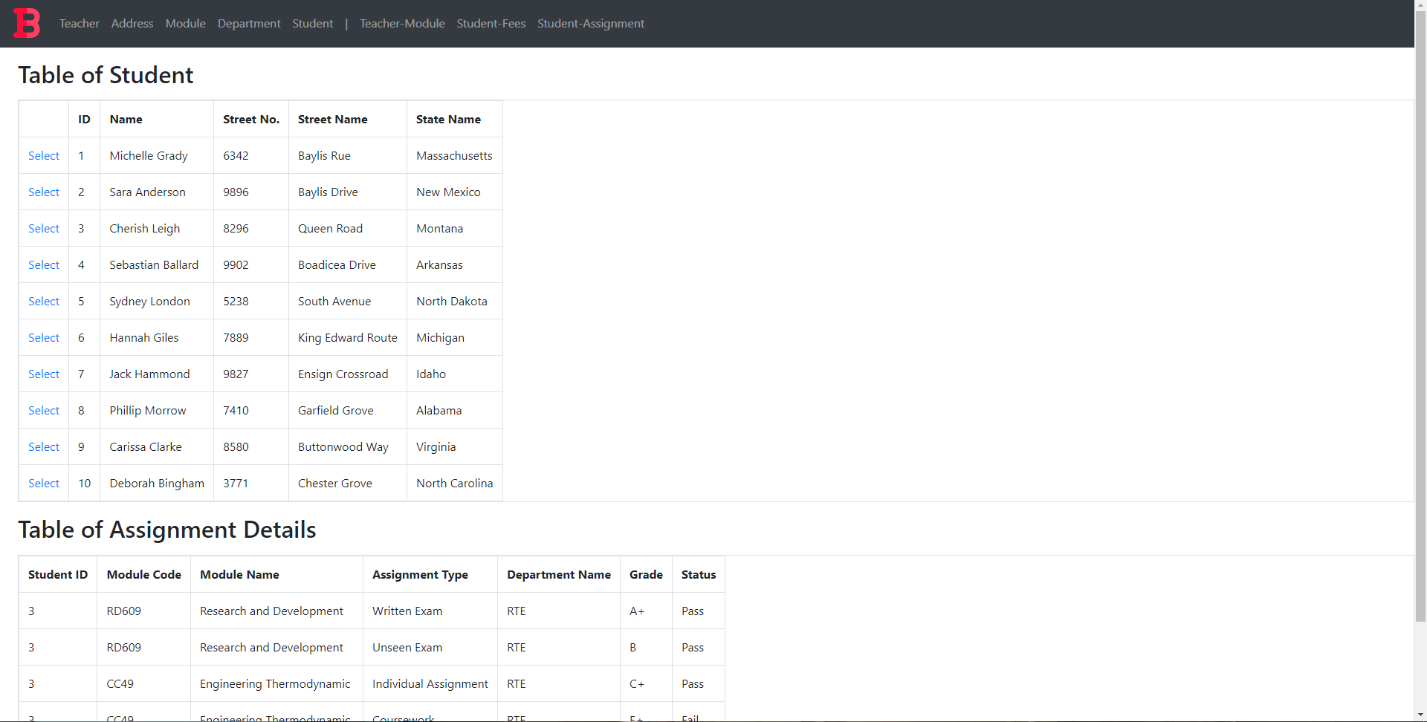


Figure : Student-Assignment Mapping Form Before Selection

 Figure : Student-Fees Mapping Form After Selection

#### Query

Assuming the selected student has student\_id 1, the query is as following:

1. SELECT sa.Student\_ID "Student ID", m.Module\_Code "Module Code" , m.Module\_Name "Module Name", a.Assignment\_Type "Assignment Type" , d.Department\_Name "Department Name" , g.Grade "Grade", g.Status "Status"
2. FROM Student s
3. INNER JOIN Student\_Assignment sa ON sa.Student\_ID = s.Student\_ID
4. INNER JOIN Grade g ON sa.Grade\_ID = g.Grade\_ID
5. INNER JOIN Module m ON sa.Module\_Code = m.Module\_Code
6. INNER JOIN Assignment a ON sa.Assignment\_ID = a.Assignment\_ID
7. INNER JOIN Department d ON a.Department\_ID = d.Department\_ID
8. WHERE sa.Student\_ID = 1;

In the webform the query string is written as:

1. $@"SELECT sa.Student\_ID ""Student ID"", m.Module\_Code ""Module Code"" , m.Module\_Name ""Module Name"", a.Assignment\_Type ""Assignment Type"" ,
2. d.Department\_Name ""Department Name"" , g.Grade ""Grade"", g.Status ""Status""
3. FROM Student s
4. INNER JOIN Student\_Assignment sa ON sa.Student\_ID = s.Student\_ID
5. INNER JOIN Grade g ON sa.Grade\_ID = g.Grade\_ID
6. INNER JOIN Module m ON sa.Module\_Code = m.Module\_Code
7. INNER JOIN Assignment a ON sa.Assignment\_ID = a.Assignment\_ID
8. INNER JOIN Department d ON a.Department\_ID = d.Department\_ID
9. WHERE sa.Student\_ID = {studentID}
10. ";

# User Manual

If the project is run without changing the port settings, then the url for the project is:

**https://localhost:44370/Index.aspx**

## Included Content



## Landing Page

Website

Description automatically generated with medium confidence

Figure : Landing Page

Shown above is the landing page of the project.

It contains a navigation bar description the webforms.

The user can always click on the logo at the top left to get back to the dashboard or continue navigating through the provided navigation bar.

Website

Description automatically generated with medium confidence

Figure : Form Types on Landing Page

Form types are shown on the top of each description for ease of access.

## Simple Forms

Graphical user interface, application

Description automatically generated

Figure : Simple Form

Shown above is the simple form for modules

### Adding Data

Graphical user interface, website

Description automatically generated

Figure Simple Form Add button

After filling out the required fields, simply press the Add button to add the entered data.

The data will be added to the form immediately as shown in the image below.

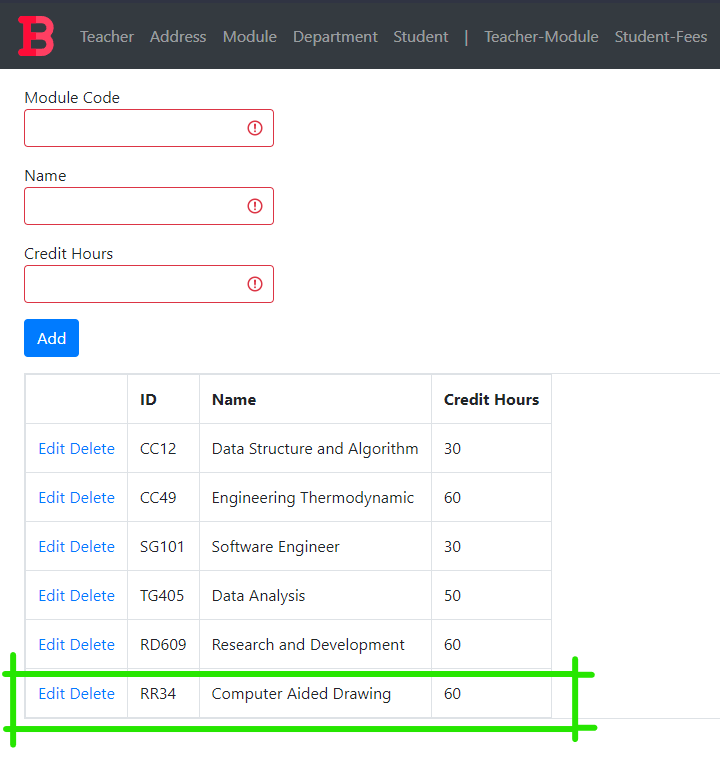


Figure : Data Added

### Deleting Data

To delete data, simply click the delete button next to the data you want to delete.

Graphical user interface, application

Description automatically generated

Figure : Delete Button

A confirmation prompt appears, and after clicking the OK button, the data gets deleted.

Graphical user interface, text, application

Description automatically generated

Figure : Deletion confirmation

Graphical user interface, application

Description automatically generated

Figure : Post Deletion Form

The data got removed as shown above.

### Editing Data

To edit or update data, simply click on Edit.

Graphical user interface, website

Description automatically generated

Figure : Edit button

After clicking the edit button, the Add button disappears and an update button appears as shown below.

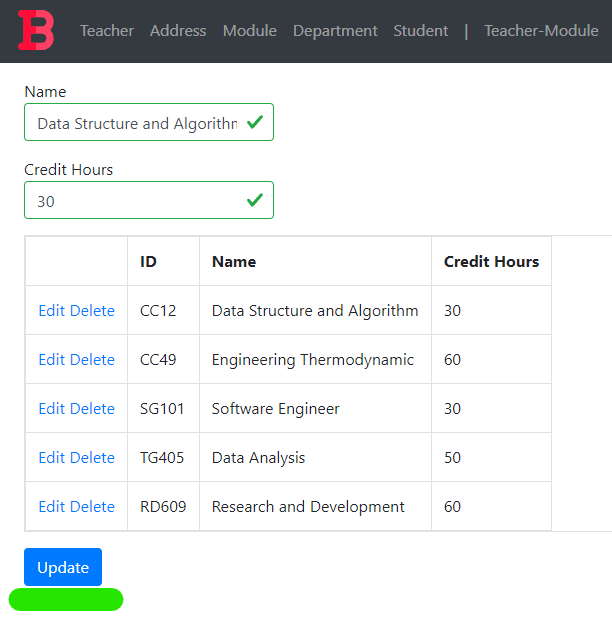


Figure : Update Button

Make any changes you may want to and press the update button

Graphical user interface, application

Description automatically generated

Here, the credit hours has been changed to 60 in place of 30.

Graphical user interface, application

Description automatically generated

As shown in the image above, the credit hour of Data Structure and Algorithm has been updated to 60, and the add button reappears.

## Complex Forms

Graphical user interface, text, application, email

Description automatically generated

Figure : Complex Form

Shown above is a complex form for teacher-module mapping.

### Selecting a Data

In order to view the mapping, a data row must be selected in the displayed grid.

Graphical user interface, application

Description automatically generated

Figure : Select Button

To select a data row, simply click on its corresponding select button as shown in the figure above.

Graphical user interface, text, application, email

Description automatically generated

Figure : Complex form Post Selection

In this complex form example, a teacher was selected and the details of the module they teach were displayed in another grid below.

# Testing

## Basic Forms

### Teacher Form

#### Test 1: View

|  |  |  |
| --- | --- | --- |
| **Test Condition** | **Expected Outcome** | **Obtained Outcome** |
| Load the teacher form and check if data is loaded in the grid. | The data is loaded on page load. | The data was loaded on page load. |

Table : Teacher Test 1

Graphical user interface, application

Description automatically generated

Figure : Teacher Test 1 evidence

#### Test 2: Add

|  |  |  |
| --- | --- | --- |
| **Test Condition** | **Expected Outcome** | **Obtained Outcome** |
| Fill the teacher form appropriately and press the add button. | The data is added, and the grid is refreshed with the new data. | The data was added without any errors and the grid was refreshed with the new data. |

Table : Teacher Test 2

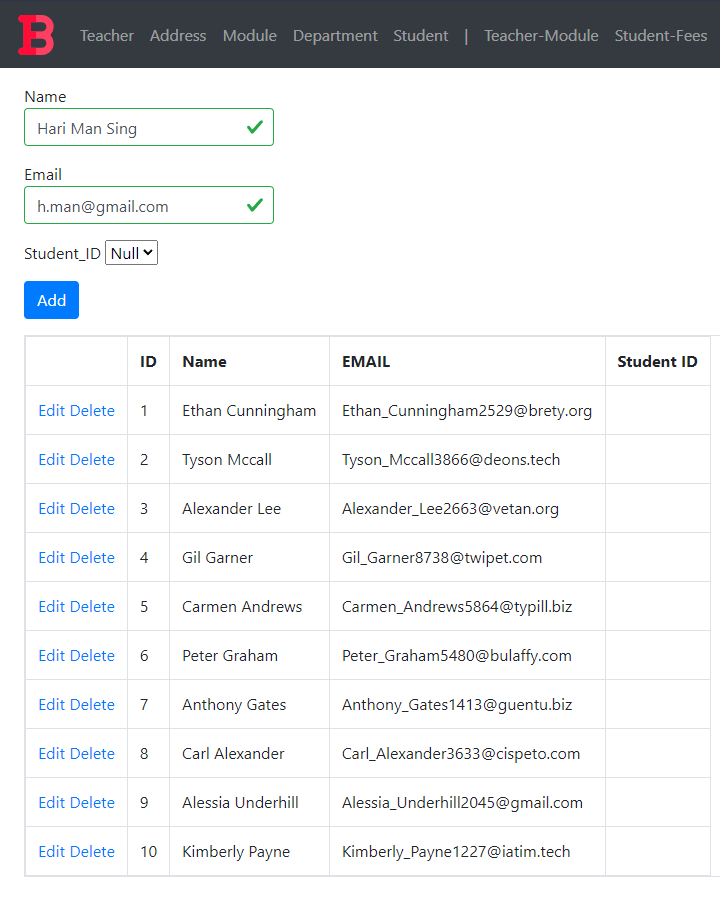


Figure : Teacher Test 2 Added Data

Graphical user interface, application

Description automatically generated

Figure : Teacher Test 2 Addition Result

#### Test 3: Edit

|  |  |  |
| --- | --- | --- |
| **Test Condition** | **Expected Outcome** | **Obtained Outcome** |
| Click the edit button | * The add button disappears and an update button appears. * The data in the row gets filled in the available textboxes. | * The add button disappeared and an update button appeared. * The data in the field got filled in the available textboxes. |
| Click the update button | * The update button disappears and the add button reappears * The data in the grid is refreshed with the edited data. | * The update button disappears and the add button reappears * The data in the grid is refreshed with the edited data. |

Table : Teacher Test 3

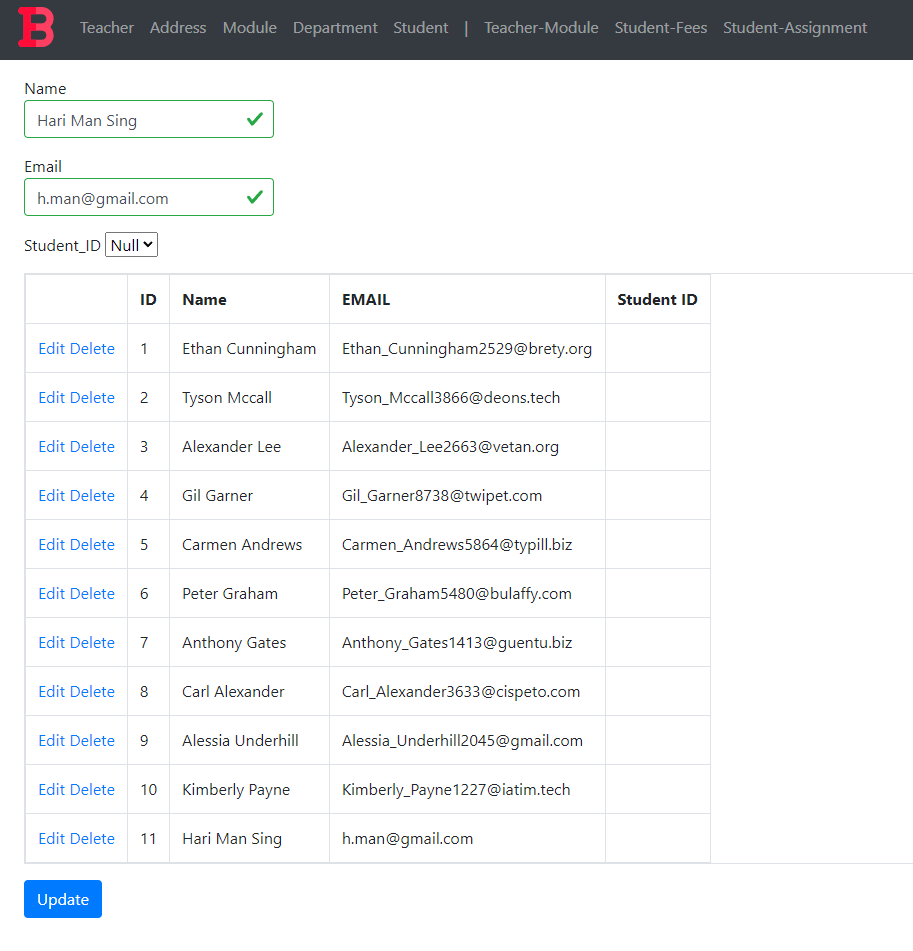


Figure : Teacher Test 3 condition 1

Graphical user interface, application

Description automatically generated

Figure : Edited value in Email Textbox

Graphical user interface, application

Description automatically generated

Figure : Teacher Test 3 Condition 2

#### Test 4: Delete

|  |  |  |
| --- | --- | --- |
| **Test Condition** | **Expected Outcome** | **Obtained Outcome** |
| Click the delete button. | The grid is refreshed with the data row deleted. | The grid was refreshed with the data row deleted. |

Table : Teacher Test 4

Graphical user interface, application

Description automatically generated

Figure : Teacher Grid Pre-Deletion

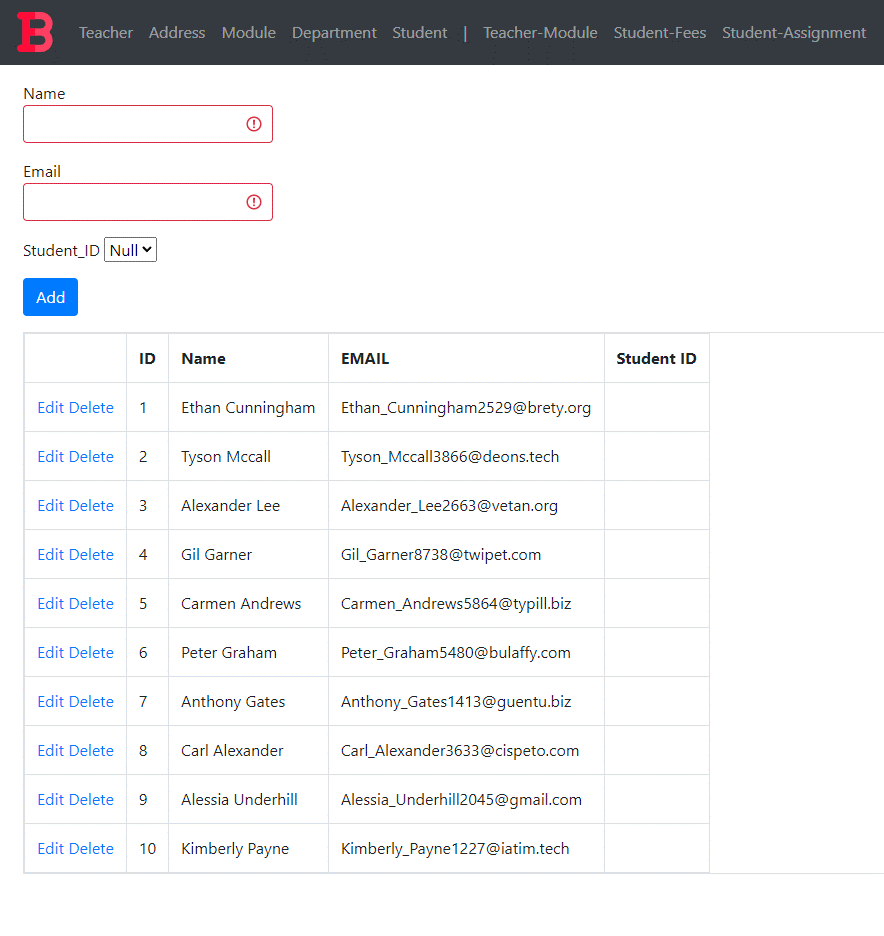


Figure : Teacher Grid Post-Deletion

### Address Form

#### Test 1: View

|  |  |  |
| --- | --- | --- |
| **Test Condition** | **Expected Outcome** | **Obtained Outcome** |
| Load the address form and check if data is loaded in the grid. | The data is loaded on page load. | The data was loaded on page load. |

Table : Address Test 1

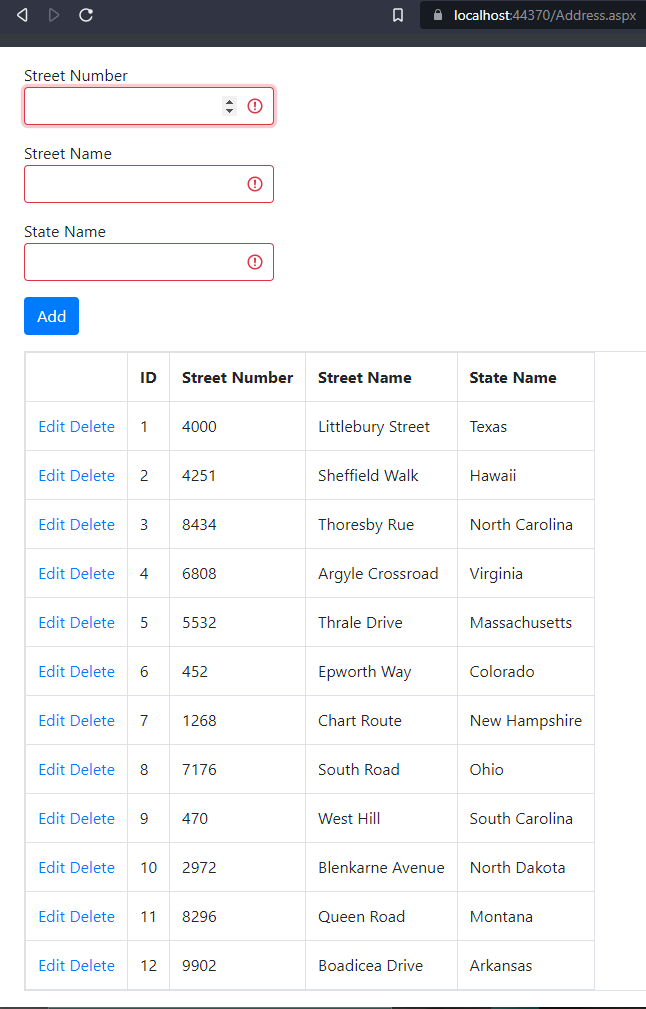


Figure : Address Test 1 evidence

#### Test 2: Add

|  |  |  |
| --- | --- | --- |
| **Test Condition** | **Expected Outcome** | **Obtained Outcome** |
| Fill the address form appropriately and press the add button. | The data is added, and the grid is refreshed with the new data. | The data was added without any errors and the grid was refreshed with the new data. |

Table : Address Test 2

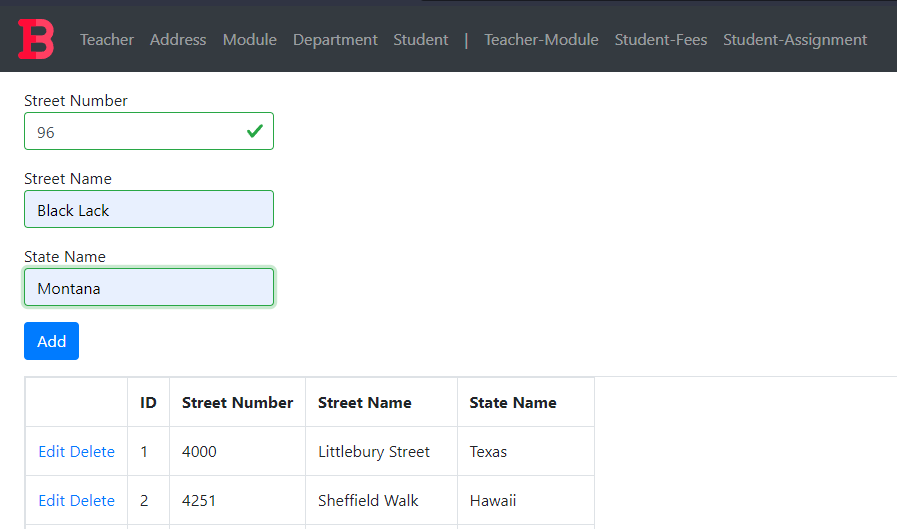


Figure : Address Test 2 Data to be Added

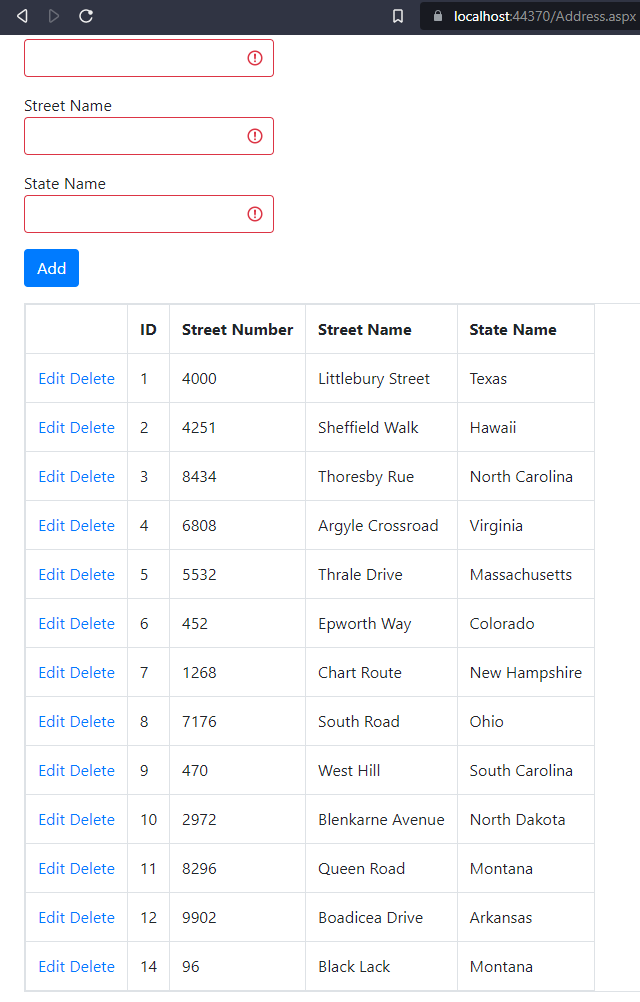


Figure : Address Test 2 Addition Result

#### Test 3: Edit

|  |  |  |
| --- | --- | --- |
| **Test Condition** | **Expected Outcome** | **Obtained Outcome** |
| Click the edit button | * The add button disappears and an update button appears. * The data in the row gets filled in the available textboxes. | * The add button disappeared and an update button appeared. * The data in the field got filled in the available textboxes. However, the street number got loaded into all textboxes |

Table : Address Test 3 Fail Case

A screenshot of a computer

Description automatically generated with medium confidence

Figure : Address Test 3 condition 1 fail case

##### What caused the issue and how it was rectified:

Text

Description automatically generated

Figure : Issue causing the fail case

The textboxes were updated with the grid row cell containing the street number value.

Once they were changed, relevant values got added to the respective textboxes.

Text

Description automatically generated

Figure : Rectification of the Issue causing the fail case

Graphical user interface

Description automatically generated

Figure : Test 3 Condition 1 post rectification

|  |  |  |
| --- | --- | --- |
| **Test Condition** | **Expected Outcome** | **Obtained Outcome** |
| Click the update button | * The update button disappears and the add button reappears * The data in the grid is refreshed with the edited data. | * The update button disappears and the add button reappears * The data in the grid is refreshed with the edited data. |

Table : Address Test 3 condition 2

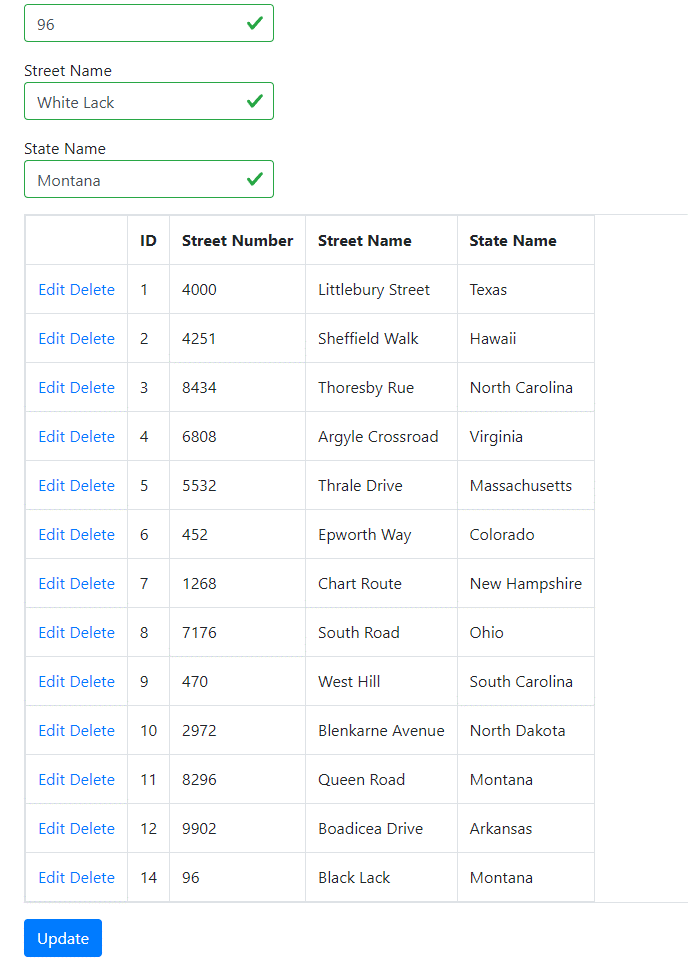


Figure : Edited value in Street Name Textbox

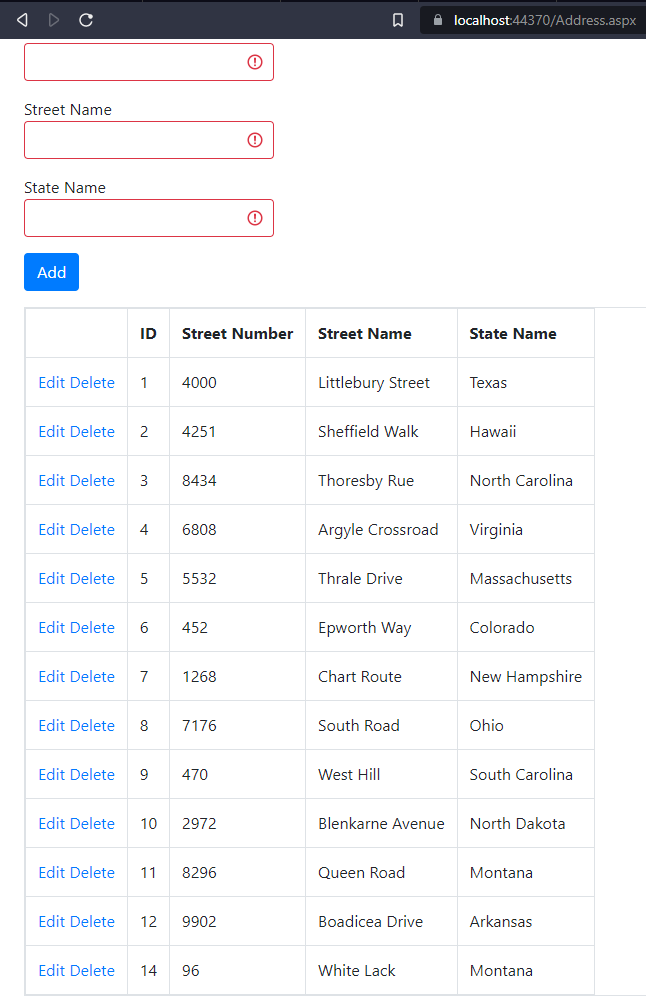


Figure : Address Test 3 Condition 2

#### Test 4: Delete

|  |  |  |
| --- | --- | --- |
| **Test Condition** | **Expected Outcome** | **Obtained Outcome** |
| Click the delete button. | The grid is refreshed with the data row deleted. | The grid was refreshed with the data row deleted. |

Table : Address Test 4

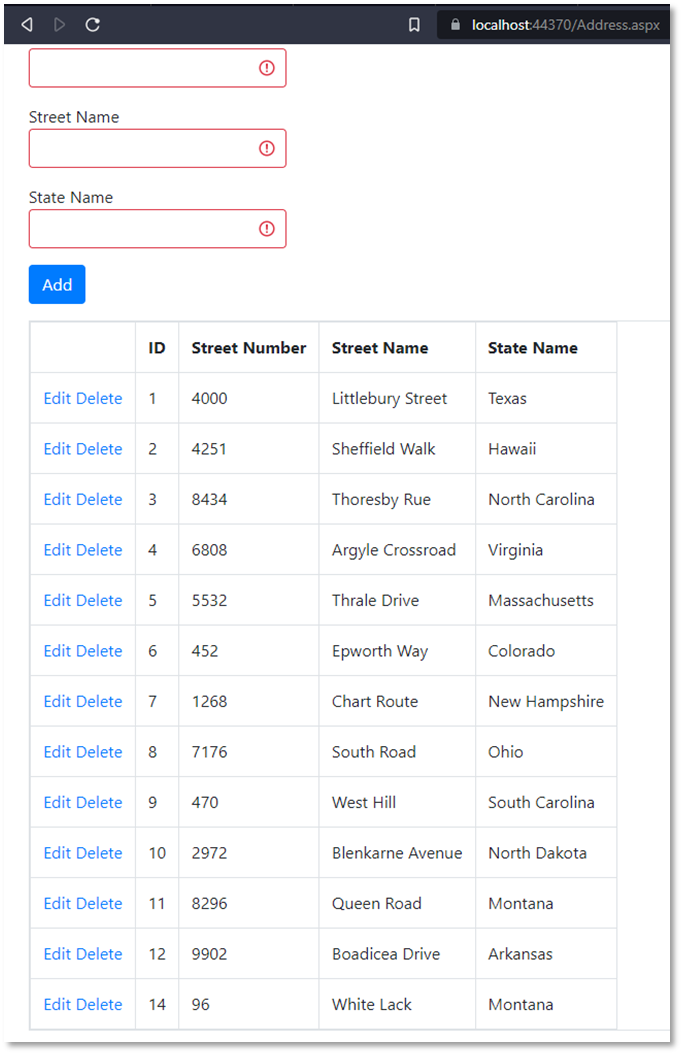


Figure : Address Grid Pre-Deletion

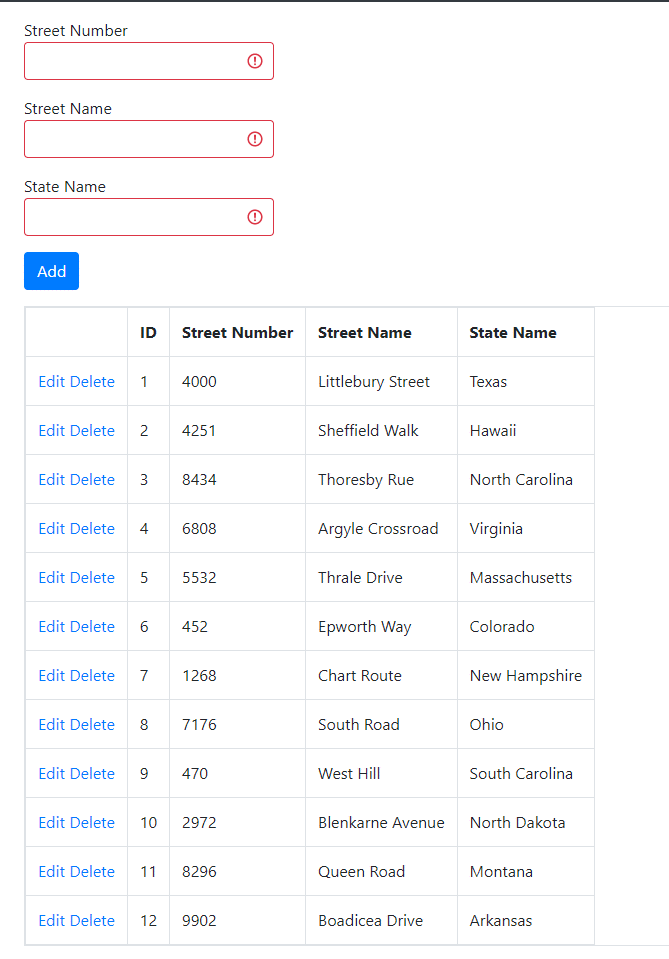


Figure : Address Grid Post-Deletion

### Module Form

#### Test 1: View

|  |  |  |
| --- | --- | --- |
| **Test Condition** | **Expected Outcome** | **Obtained Outcome** |
| Load the module form and check if data is loaded in the grid. | The data is loaded on page load. | The data was loaded on page load. |

Table : Module Test 1

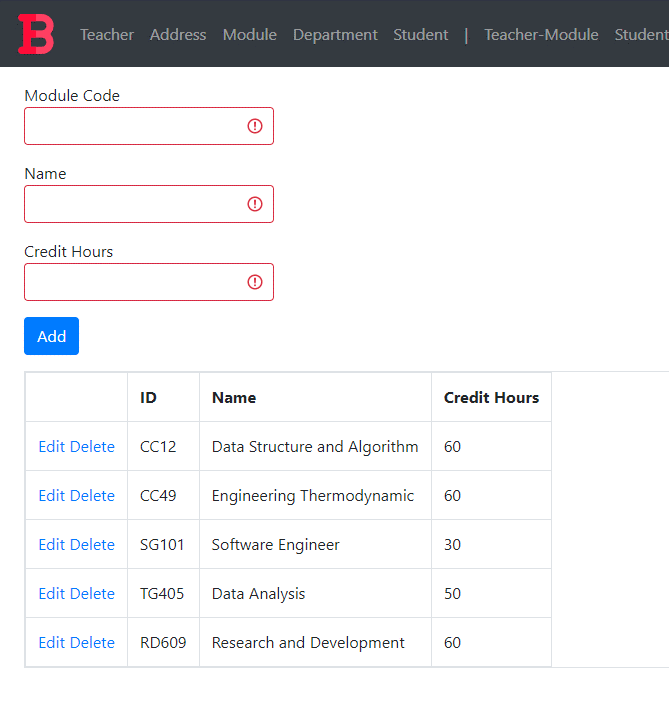


Figure : Module Test 1 evidence

#### Test 2: Add

|  |  |  |
| --- | --- | --- |
| **Test Condition** | **Expected Outcome** | **Obtained Outcome** |
| Fill the module form appropriately and press the add button. | The data is added, and the grid is refreshed with the new data. | The data was added without any errors and the grid was refreshed with the new data. |

Table : Module Test 2

Graphical user interface, application

Description automatically generated

Figure : Module Test 2 Data to be Added

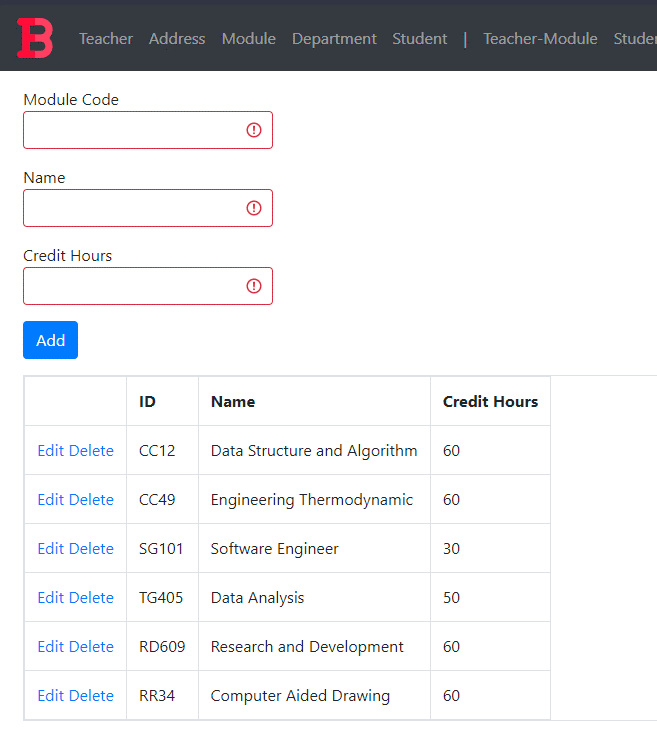


Figure : Module Test 2 Addition Result

#### Test 3: Edit

|  |  |  |
| --- | --- | --- |
| **Test Condition** | **Expected Outcome** | **Obtained Outcome** |
| Click the edit button | * The add button disappears and an update button appears. * The data in the row gets filled in the available textboxes. | * The add button disappeared and an update button appeared. * The data in the field got filled in the available textboxes. |
| Click the update button | * The update button disappears and the add button reappears * The data in the grid is refreshed with the edited data. | * The update button disappears and the add button reappears * The data in the grid is refreshed with the edited data. |

Table : Module Test 3

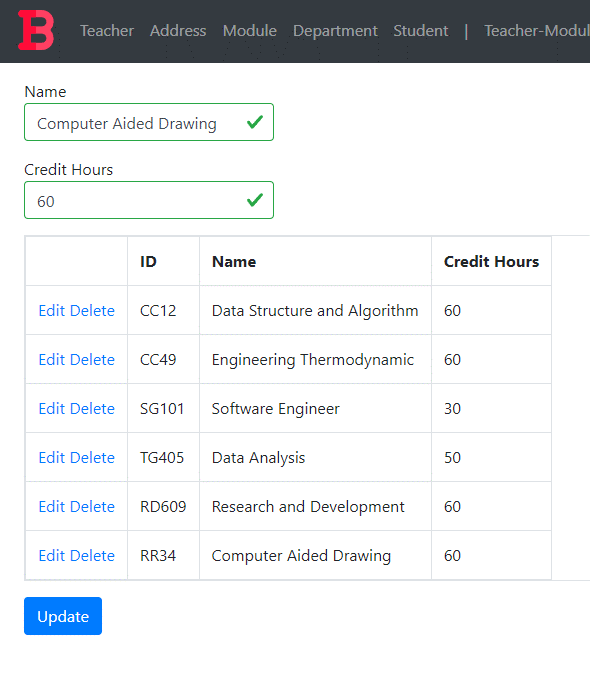


Figure : Module Test 3 condition 1

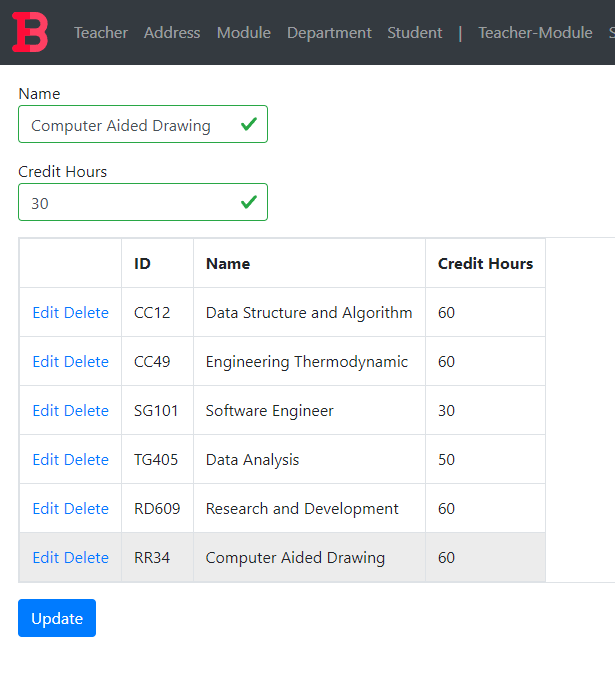


Figure : Edited value in Credit Hours Textbox

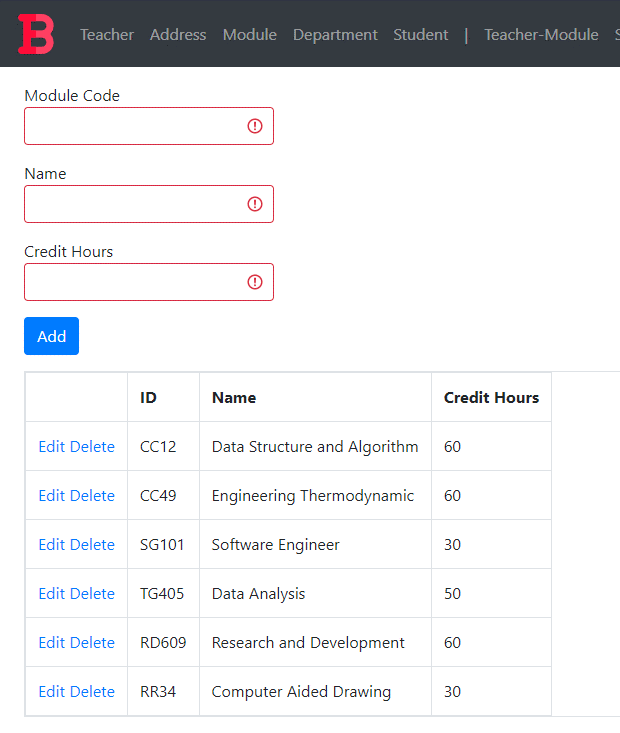


Figure : Module Test 3 Condition 2

#### Test 4: Delete

|  |  |  |
| --- | --- | --- |
| **Test Condition** | **Expected Outcome** | **Obtained Outcome** |
| Click the delete button. | The grid is refreshed with the data row deleted. | The grid was refreshed with the data row deleted. |

Table : Module Test 4

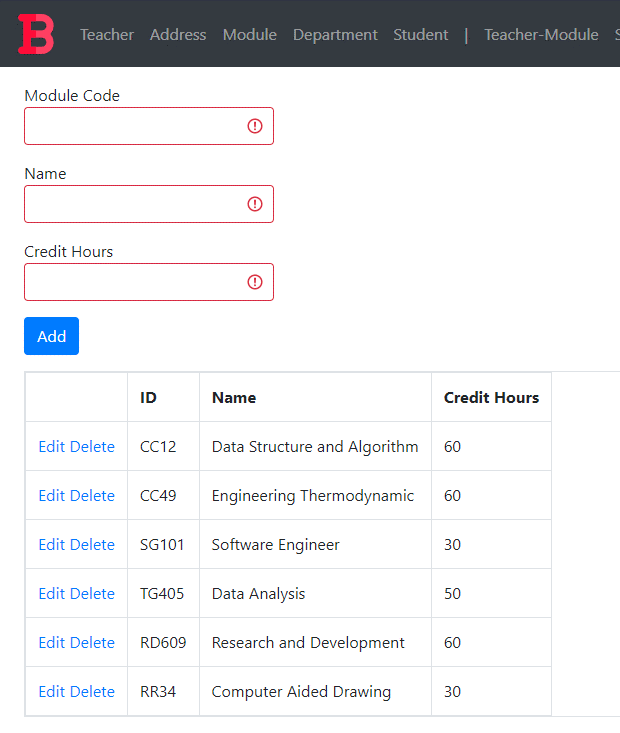


Figure : Address Grid Pre-Deletion

##### What caused the issue and how it was rectified:

A screenshot of a computer

Description automatically generated

Figure : Issue causing the fail case

Unlike the primary keys from other form table, module table had a VARCHAR primary key. Here, I was trying to convert the string into an integer in order to obtain the primary key for the row of data that needed to be deleted. Hence, it caused a format exception and did not carry out the delete procedure.

I set the variable to string and used the value obtained from the grid view row as a string and rectified the issue. This made the deletion procedure go smoothly and uninterrupted.

Text

Description automatically generated

Figure : Rectification of the Issue causing the fail case

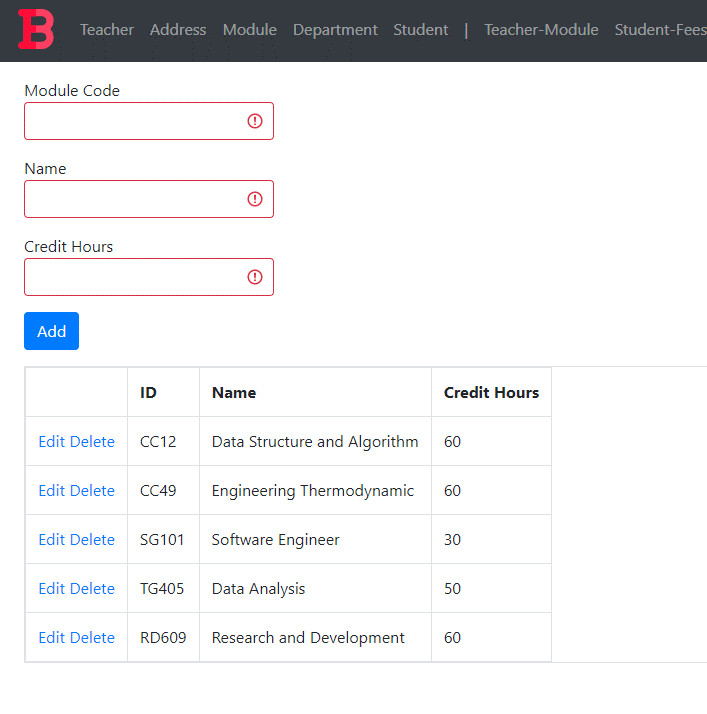


Figure : Test 4 post deletion

### Department Form

#### Test 1: View

|  |  |  |
| --- | --- | --- |
| **Test Condition** | **Expected Outcome** | **Obtained Outcome** |
| Load the department form and check if data is loaded in the grid. | The data is loaded on page load. | The data was loaded on page load. |

Table : Department Test 1



Figure : Department Test 1 evidence

#### Test 2: Add

|  |  |  |
| --- | --- | --- |
| **Test Condition** | **Expected Outcome** | **Obtained Outcome** |
| Fill the department form appropriately and press the add button. | The data is added, and the grid is refreshed with the new data. | The data was added without any errors and the grid was refreshed with the new data. |

Table : Department Test 2

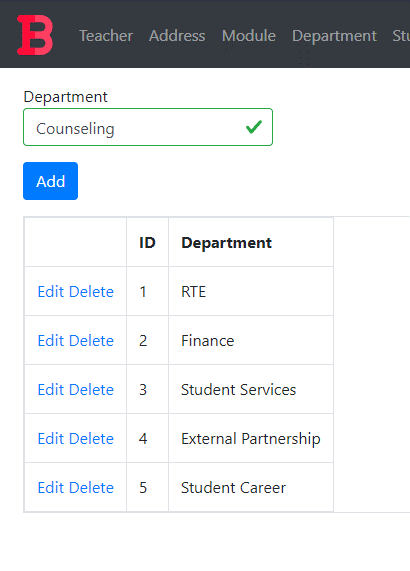


Figure : Department Test 2 Added Data

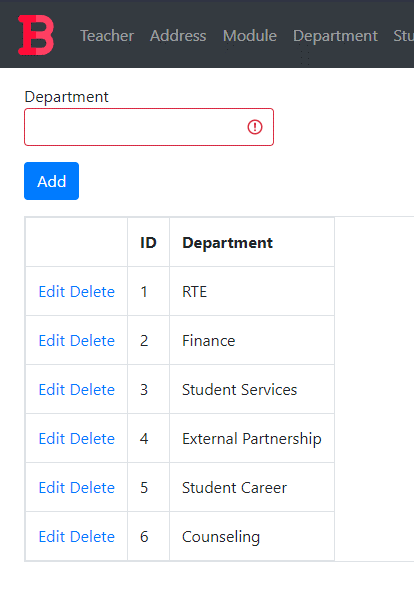


Figure : Department Test 2 Addition Result

#### Test 3: Edit

|  |  |  |
| --- | --- | --- |
| **Test Condition** | **Expected Outcome** | **Obtained Outcome** |
| Click the edit button | * The add button disappears and an update button appears. * The data in the row gets filled in the available textboxes. | * The add button disappeared and an update button appeared. * The data in the field got filled in the available textboxes. |
| Click the update button | * The update button disappears and the add button reappears * The data in the grid is refreshed with the edited data. | * The update button disappears and the add button reappears * The data in the grid is refreshed with the edited data. |

Table : Department Test 3

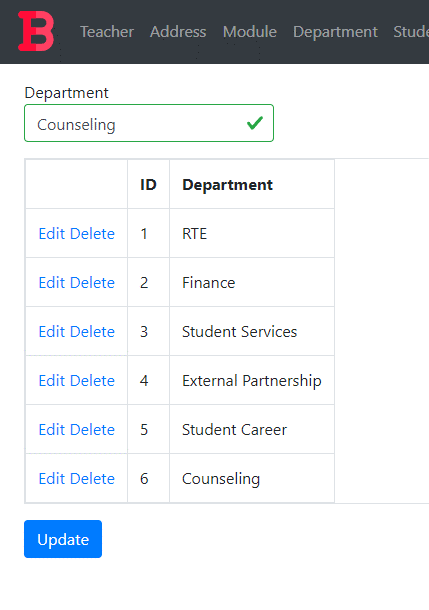


Figure : Department Test 3 condition 1

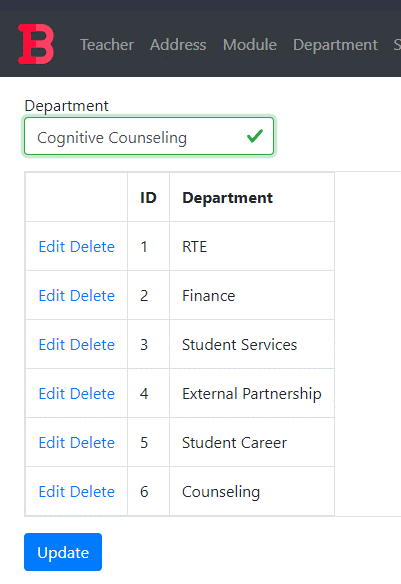


Figure : Edited value in Department Textbox

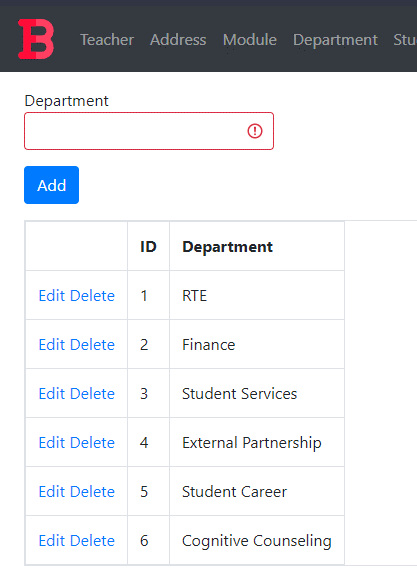


Figure : Department Test 3 Condition 2

#### Test 4: Delete

|  |  |  |
| --- | --- | --- |
| **Test Condition** | **Expected Outcome** | **Obtained Outcome** |
| Click the delete button. | The grid is refreshed with the data row deleted. | The grid was refreshed with the data row deleted. |

Table : Department Test 4

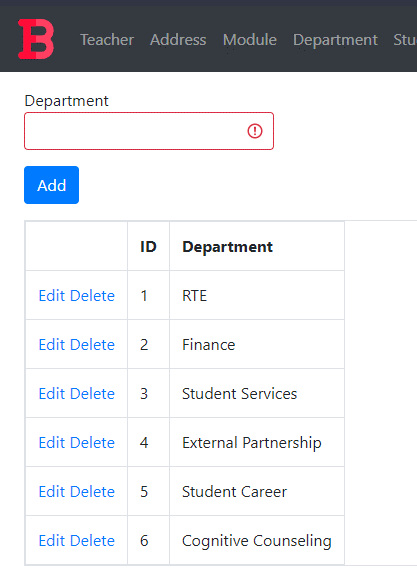


Figure : Department Grid Pre-Deletion

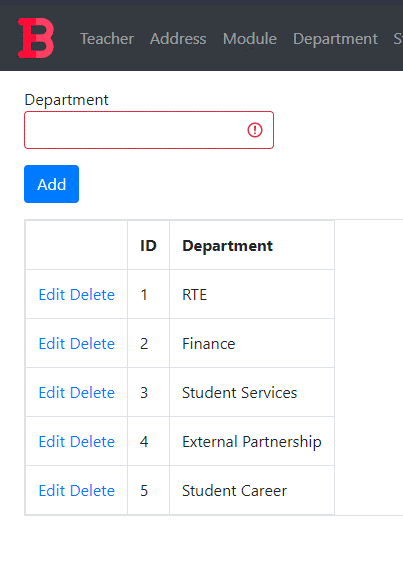


Figure : Department Grid Post-Deletion

### Student Form

#### Test 1: View

|  |  |  |
| --- | --- | --- |
| **Test Condition** | **Expected Outcome** | **Obtained Outcome** |
| Load the student form and check if data is loaded in the grid. | The data is loaded on page load. | The data was loaded on page load. |

Table : Student Test 1



Figure : Student Test 1 evidence

#### Test 2: Add

|  |  |  |
| --- | --- | --- |
| **Test Condition** | **Expected Outcome** | **Obtained Outcome** |
| Fill the student form appropriately and press the add button. | The data is added, and the grid is refreshed with the new data. | The data was added without any errors and the grid was refreshed with the new data. |

Table : Student Test 2

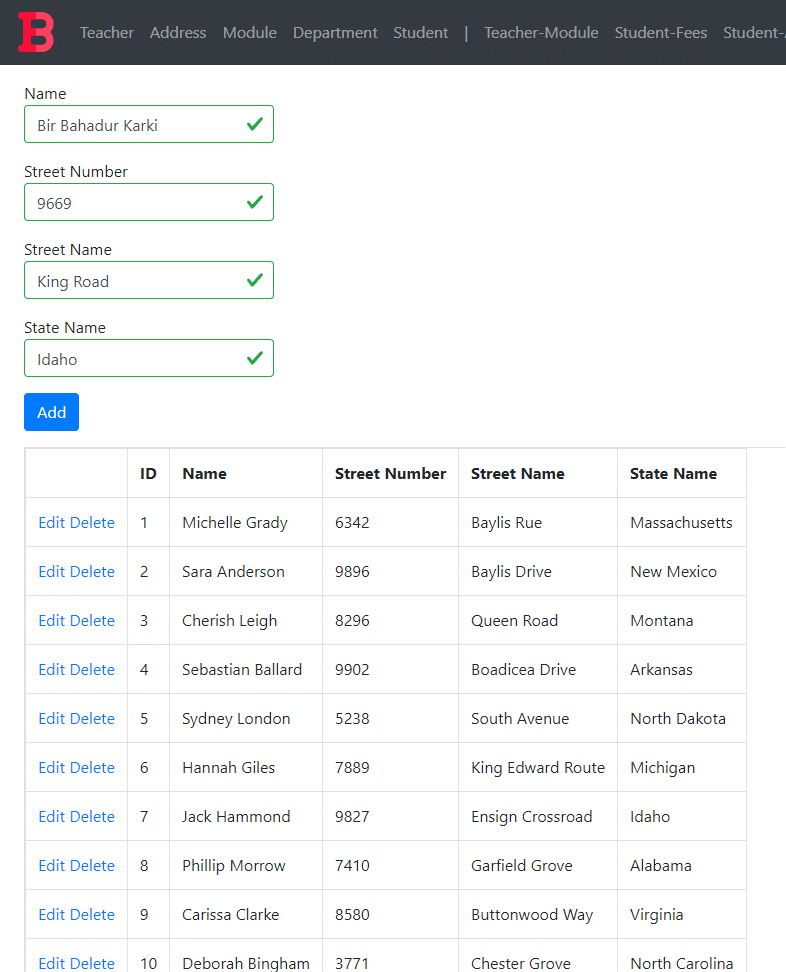


Figure : Student Test 2 Added Data

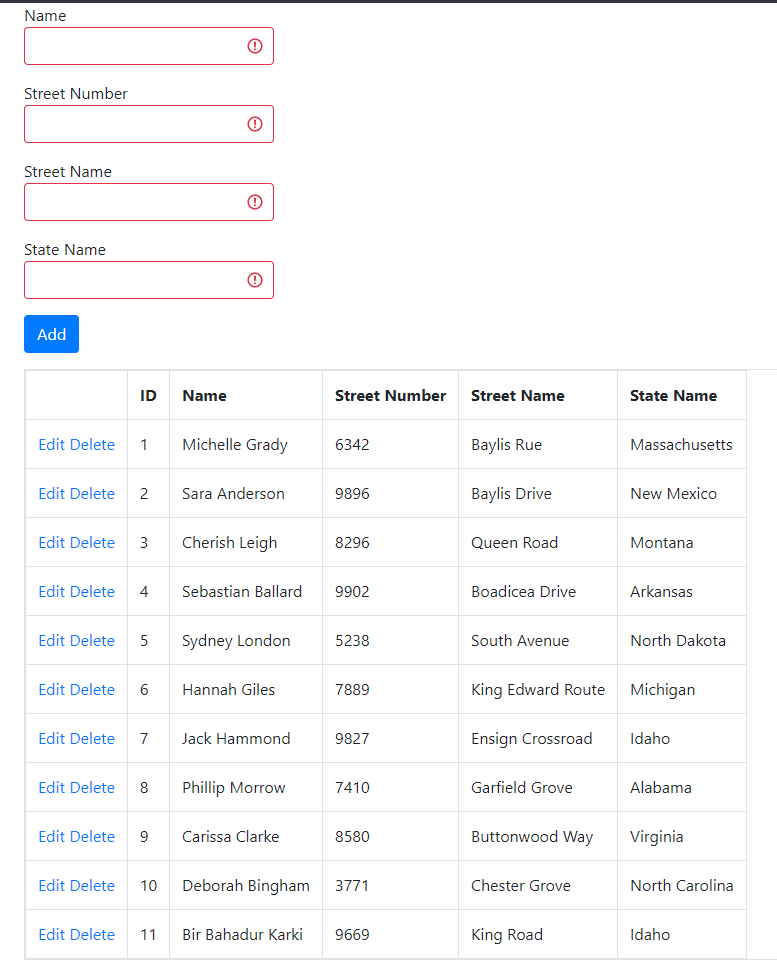


Figure : Student Test 2 Addition Result

#### Test 3: Edit

|  |  |  |
| --- | --- | --- |
| **Test Condition** | **Expected Outcome** | **Obtained Outcome** |
| Click the edit button | * The add button disappears and an update button appears. * The data in the row gets filled in the available textboxes. | * The add button disappeared and an update button appeared. * The data in the field got filled in the available textboxes. |
| Click the update button | * The update button disappears and the add button reappears * The data in the grid is refreshed with the edited data. | * The update button disappears and the add button reappears * The data in the grid is refreshed with the edited data. |

Table : Student Test 3

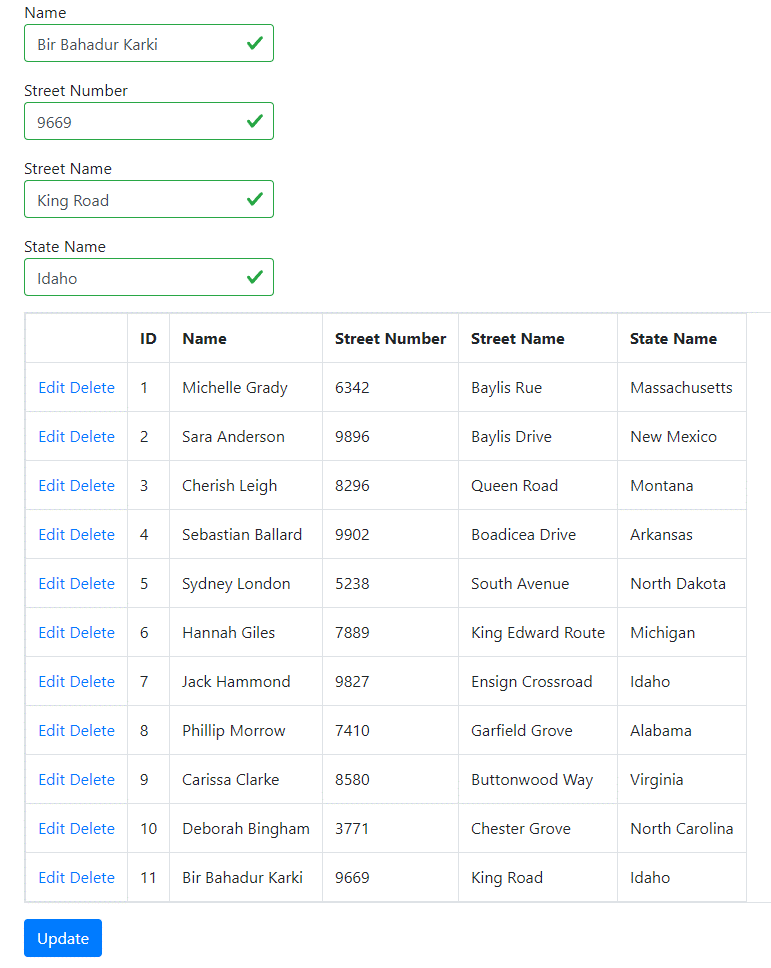


Figure : Student Test 3 condition 1

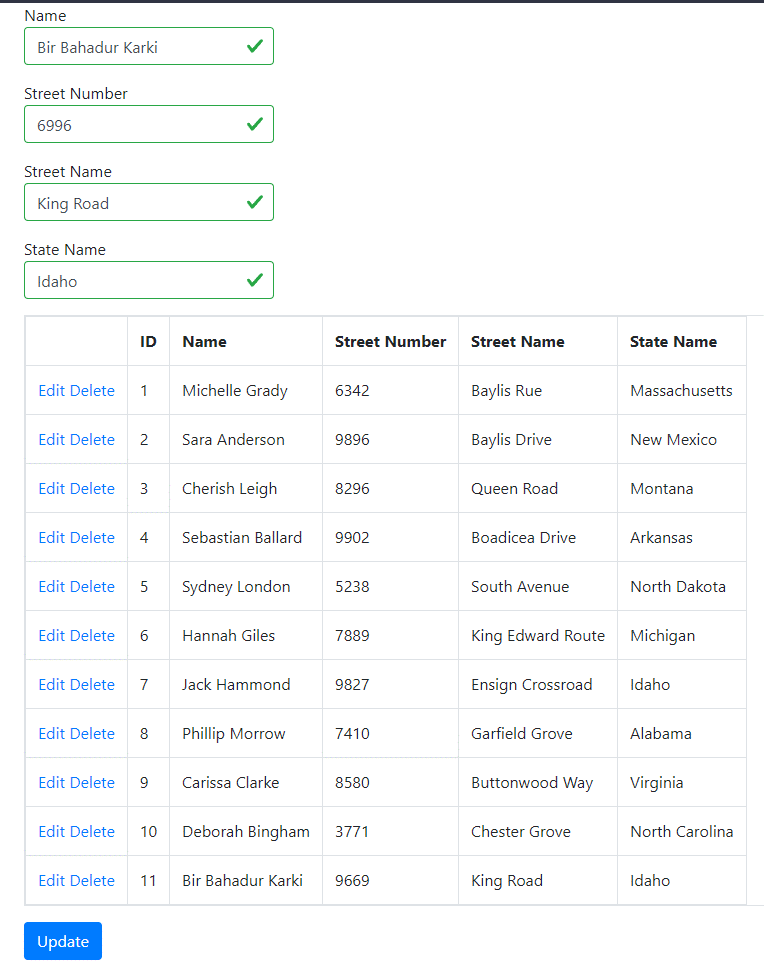


Figure : Edited value in Street Number Textbox

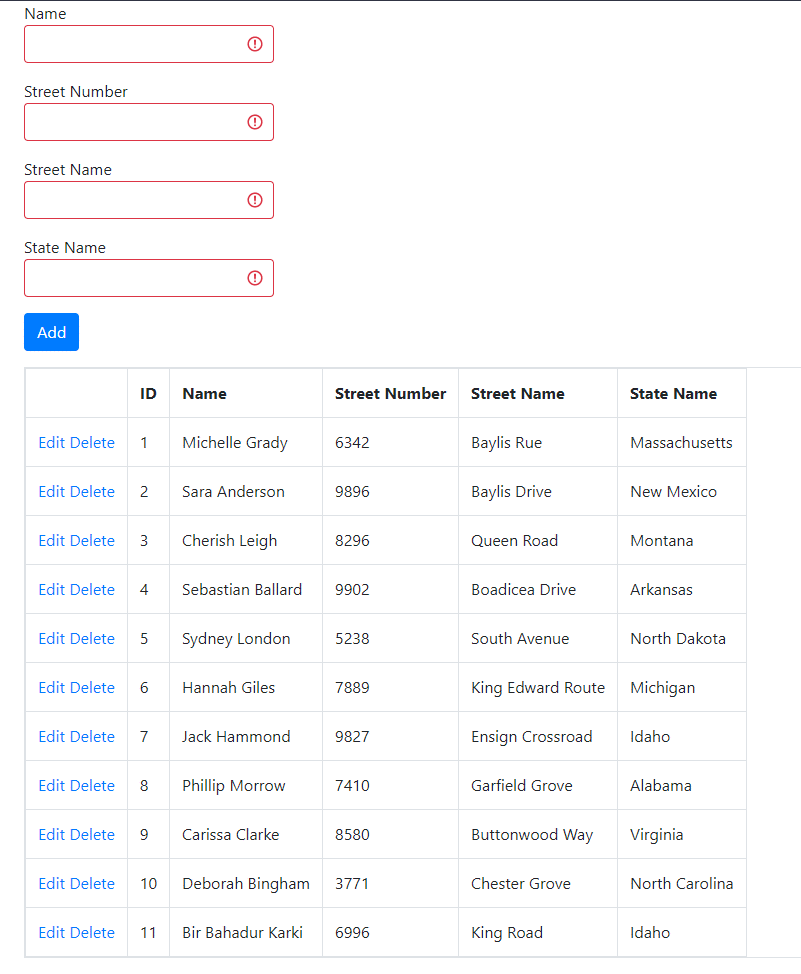


Figure : Student Test 3 Condition 2

#### Test 4: Delete

|  |  |  |
| --- | --- | --- |
| **Test Condition** | **Expected Outcome** | **Obtained Outcome** |
| Click the delete button. | The grid is refreshed with the data row deleted. | The grid was refreshed with the data row deleted. |

Table : Student Test 4

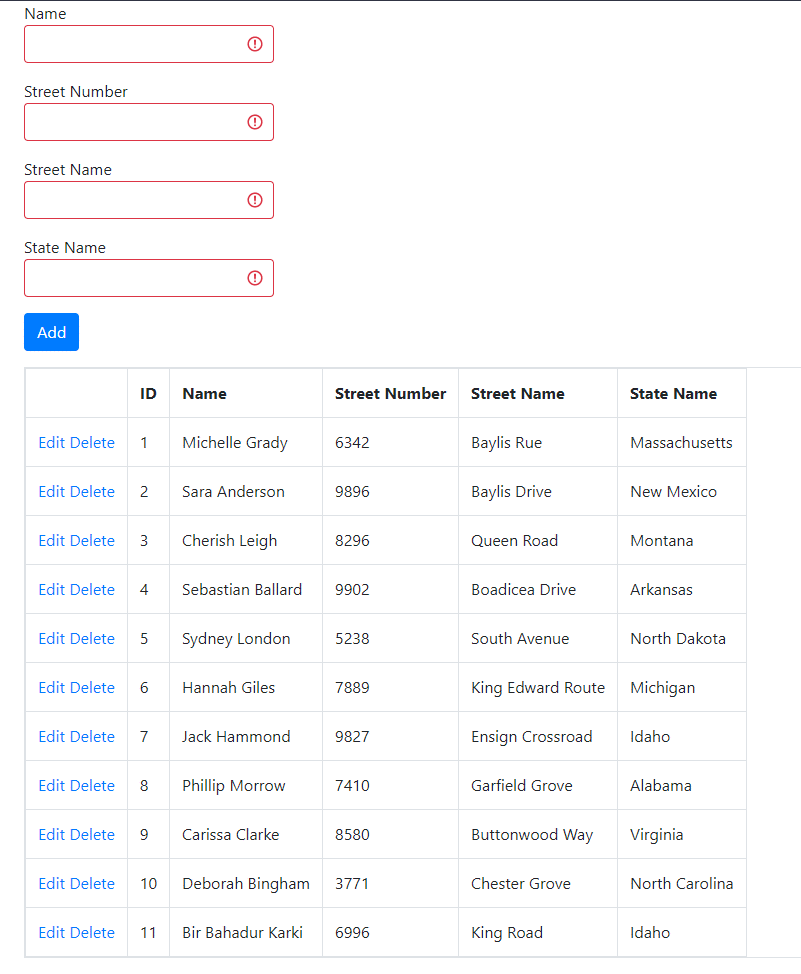


Figure : Student Grid Pre-Deletion

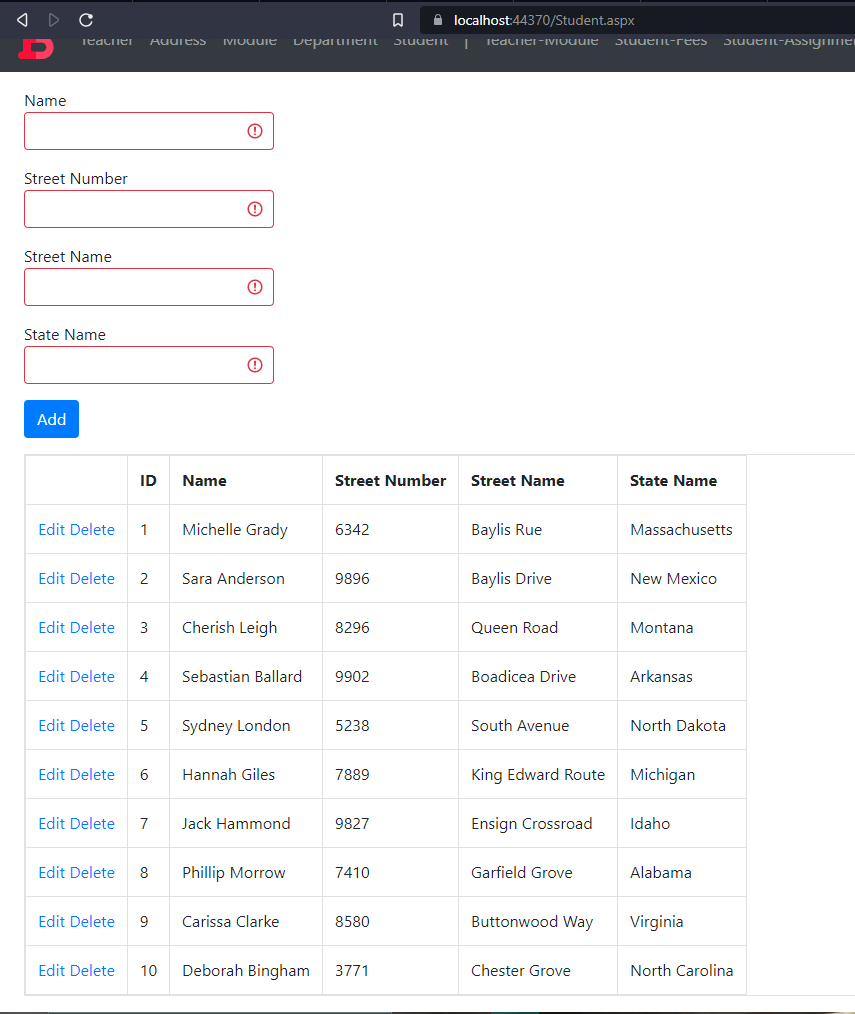


Figure : Student Grid Post-Deletion

## Complex Forms

### Teacher-Module Mapping Form

#### Test 1: Page Load

|  |  |  |
| --- | --- | --- |
| **Test Condition** | **Expected Outcome** | **Obtained Outcome** |
| Load the complex form and check if data is loaded in the initial grid. | The data is loaded to the initial grid on page load. | The data was loaded to the initial grid on page load. |

Table : Teacher-Module Test 1

Graphical user interface, application

Description automatically generated

Figure : Teacher-Module Test 1 evidence

#### Test 2: Data Load on Selection

|  |  |  |
| --- | --- | --- |
| **Test Condition** | **Expected Outcome** | **Obtained Outcome** |
| Select a data row from the initial grid view and check if the other grid gets loaded with relevant data. | The other grid is loaded with relevant data. | The other grid was loaded with relevant data. |

Table : Teacher-Module Test 2

Graphical user interface, text, application

Description automatically generated

Figure : Teacher-Module Test 2 evidence

### Student-Fees Mapping Form

#### Test 1: Page Load

|  |  |  |
| --- | --- | --- |
| **Test Condition** | **Expected Outcome** | **Obtained Outcome** |
| Load the complex form and check if data is loaded in the initial grid. | The data is loaded to the initial grid on page load. | The data was loaded to the initial grid on page load. |

Table : Student-Fees Test 1

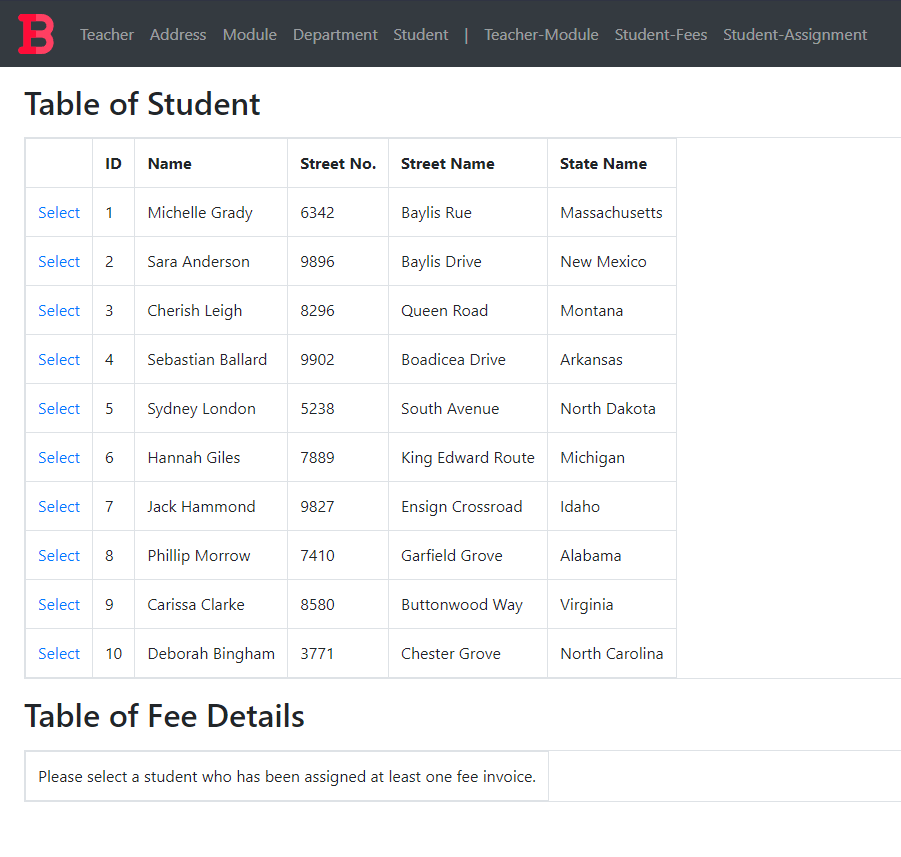


Figure : Student-Fees Test 1 evidence

#### Test 2: Data Load on Selection

|  |  |  |
| --- | --- | --- |
| **Test Condition** | **Expected Outcome** | **Obtained Outcome** |
| Select a data row from the initial grid view and check if the other grid gets loaded with relevant data. | The other grid is loaded with relevant data. | The other grid was loaded with relevant data. |

Table : Student-Fees Test 2

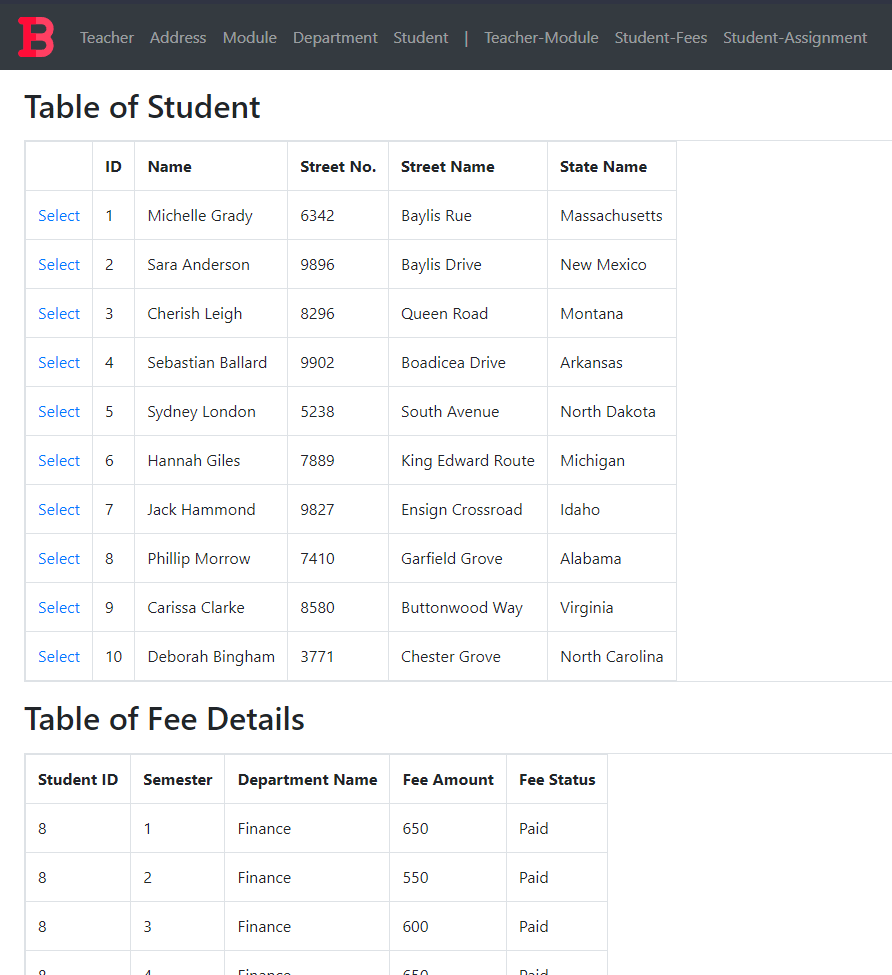


Figure : Student-Fees Test 2 evidence

### Student-Assignment Mapping Form

#### Test 1: Page Load

|  |  |  |
| --- | --- | --- |
| **Test Condition** | **Expected Outcome** | **Obtained Outcome** |
| Load the complex form and check if data is loaded in the initial grid. | The data is loaded to the initial grid on page load. | The data was loaded to the initial grid on page load. |

Table : Student-Assignment Test 1

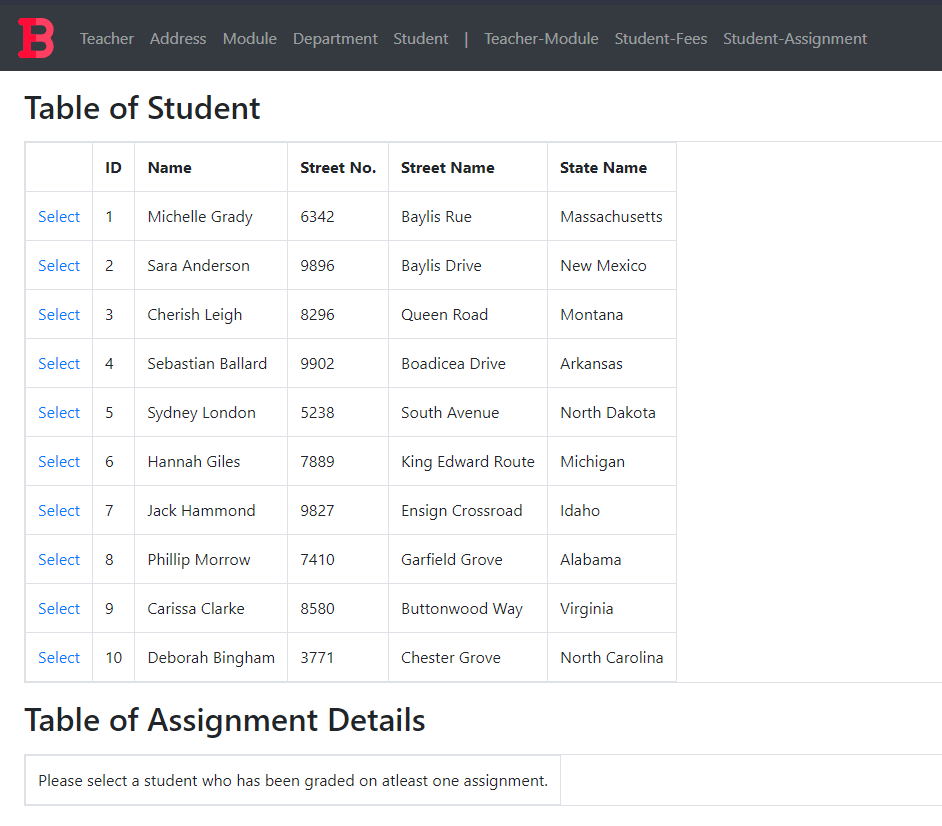


Figure : Student-Assignment Test 1 evidence

#### Test 2: Data Load on Selection

|  |  |  |
| --- | --- | --- |
| **Test Condition** | **Expected Outcome** | **Obtained Outcome** |
| Select a data row from the initial grid view and check if the other grid gets loaded with relevant data. | The other grid is loaded with relevant data. | The other grid was loaded with relevant data. |

Table : Student-Assignment Test 2

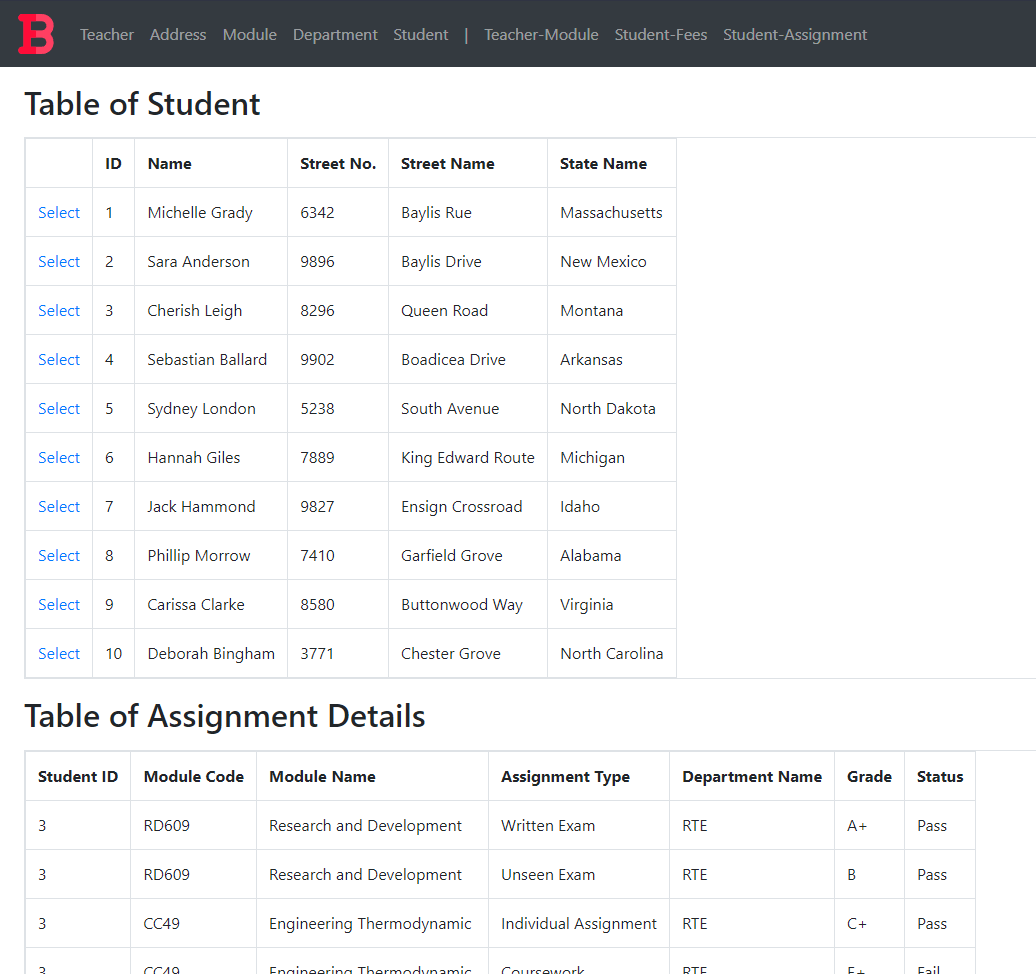


Figure : Student-Assignment Test 2 evidence

# Further Discussion

The project was a simple one, but it provided a lot of insight on oracle databases and dotnet webforms. While the database normalization, integration with assumptions, creation and data population portions were familiar to what we had already been taught, the dotnet webform creation was a unique experience in this project. As dotnet was used previously for another module to create a windows form application, I was already familiar with the dotnet syntax. Similarly, web development was also quite familiar since I had experience working with JavaScript frontend frameworks like React.

However, web development with dotnet was almost completely different than what I had experience with. The project taught me about real world business scenarios. I learned how normalization, being the useful tool that it is, sometimes does not give what we need, about how redundancy must be sometimes added artificially to keep record of things. The project also helped me better understand dotnet as a framework for web development, about how master classes can be used for page layouts, about how libraries like bootstrap can be used to visually enhance a webpage without too much effort and about HTML5 technologies such as input validation. The project also taught me about Oracle SQL sequences and triggers for auto generation of keys, about how Oracle Developer and Oracle Developer Data Modeler can be used to visualize databases, easily edit, or create tables, and generate DDL scripts. It also deepened my understanding about user permissions and Oracle database connection with third parties such as visual studio.

# Conclusion

The coursework completion fueled my feel confidence to create a database based on real world business scenario to store data in a professional manner. Since the coursework was about creating a database record system and a web application for a college, it has lots of applicability in the real world as most colleges and schools use a database to record their data. Not only schools but a lot of management systems are built the same way with different business scenarios.

All in all, the help from teachers and timely self-research turned the project from what I believe would be intimidating to a simple and fun project that taught me a ton of techniques. While the project was a simple one it taught me more about oracle, dotnet for web development and more about problem solving in general.