HWS Register Database

Mohammad Ismail Ashiq Aslam - 2023

Table of Contents

[Phase 1: Fact-Finding, Information Gathering, and Conceptual Database Design 3](#_Toc135648671)

[1.1 Fact-Finding Techniques and Information Gathering 3](#_Toc135648672)

[1.1.1 Introduction to Enterprise 3](#_Toc135648673)

[1.1.2 Description of Fact-Finding Techniques 3](#_Toc135648674)

[1.1.3 Database Design Focus 4](#_Toc135648675)

[1.1.4 Entity and Relationship Set Description 4](#_Toc135648676)

[1.2 Conceptual Database Design 5](#_Toc135648677)

[1.2.1 Entity Set Description 5](#_Toc135648678)

[1.2.2Relationship Set Description 11](#_Toc135648679)

[1.2.3Related Entity Set 13](#_Toc135648680)

[1.2.4E-R Diagram 13](#_Toc135648681)

[Phase 2: From E-R Model to Relational Model 15](#_Toc135648682)

[2.1 Conceptual Database and Logical Database 15](#_Toc135648683)

[2.1.1 E-R Model and Relational Model 15](#_Toc135648684)

[2.1.2 Comparison of Two Different Models 15](#_Toc135648685)

[2.2 Conversion from E-R Model to a Relational Database 17](#_Toc135648686)

[2.2.1 Converting Entity Types to Relations 17](#_Toc135648687)

[2.2.2 Converting Relationship Types to Relations 18](#_Toc135648688)

[2.2.3 Database Constraints 19](#_Toc135648689)

[2.3 Sample Queries for Database 22](#_Toc135648690)

[2.3.1 Samples Data 22](#_Toc135648691)

[2.3.2 Design of Queries 27](#_Toc135648692)

[Phase 3: Implementation of Relational Database 32](#_Toc135648693)

[3.1 Relation Normalization 33](#_Toc135648694)

[3.2 SQL Queries 36](#_Toc135648695)

[Phase 4: Stored Procedures, Packages, and Triggers 41](#_Toc135648696)

[4.1 Oracle PL/SQL 41](#_Toc135648697)

[4.2 MS SQL Server and MySQL Stored Procedures 45](#_Toc135648698)

[Phase 5: Graphical User Interface 47](#_Toc135648699)

[5.1 SQL Tables 47](#_Toc135648700)

[5.2 Inserting into Database 52](#_Toc135648701)

# Phase 1: Fact-Finding, Information Gathering, and Conceptual Database Design

## Fact-Finding Techniques and Information Gathering

### 1.1.1 Introduction to Enterprise

The company of our database will be created for the HWS register of a university. The HWS register consists of a register with all the current information of the students which is required to identify the student and decide whether they can graduate or not. This requires information about the student, for example: their name, address, phone number, student ID, number of credits accumulated, course name… With the information collected we can check if all goals are satisfied, all goals must be satisfied in order to graduate.

### 1.1.2 Description of Fact-Finding Techniques

In order to acquire and use material for our research, I used the internet and well-known search engines for fact-finding. I looked at a few older and more recent registers to discover more detailed information on HWS registers.

From all the research, I discovered that the registers are very similar. Every university database may be different, but they all have common things.

Many reports will be generated in order to get the HWS register to work as there will be many strong entities and there will be many connections between them.

### 1.1.3 Database Design Focus

HWS registers are very complicated as there is many information and most if it is linked or related. So for this project I decided to focus on what are the important parts and only what the register needs to do the job that they are intended to do.

Keeping the main focus on the students and their information such as credits accumulated, start and end term of a course, studentID, date of birth… Allows to keep the project simple. This way in order to check if a student graduates it would be faster and more efficient.

### 1.1.4 Entity and Relationship Set Description

We were able to solidify our ideas on how to establish an entity thanks to the tutorials and research we have done so far. We listened to a couple groups discuss their concepts for their suggested databases throughout the lab session. We explored how each concept may be enhanced or whether there was a problem with the current paradigm during each presentation. I attempted to refine my own concepts and initial draughts of our entities using this expertise.

The Transcript entity represents the actual condition of a student by which is then decided whether the student graduates or not.

The goal and course entities represent the conditions that a student must meet in order to graduate.

Entities like address, status, email address, phone number, major, minor… Will be linked to Student entity so that all Information can be accessed through the Student ID.

## Conceptual Database Design

### 1.2.1 Entity Set Description

A picture containing text, screenshot, font, number

Description automatically generated-Student: Strong entity.

Primary key: Student ID.

Primary key: DateOfBirth.

Foreign Key: Phone Number

Foreign Key: Email Address

Foreign Key: Address

Attributes:

1. Name: String; no nulls; single-valued and not unique.
2. Date of birth: Date; no nulls; single-valued and primary key.
3. Admit Term: Date; no nulls; single-valued and not unique.
4. Expected Graduation Term: Date; no nulls; single-valued and not unique.
5. Status: String; no nulls; single-valued and not unique.
6. Academic Load: String; no nulls; single-valued.
7. Academic Level: String; no nulls; single-valued.

-Address: Weak entity.

A screen shot of a computer

Description automatically generated with low confidencePrimary key: Address ID.

Foreign key: Student ID.

Attributes:

1. Home: String; no nulls; specifies the street address.
2. Campus: String; no nulls; specifies the street address.
3. Dom: String; no nulls; specifies the street address.
4. Address ID: String; no nulls; specifies the type of address (campus, dorm, home, etc.).
5. Student ID: Integer; no nulls; foreign key that references the Student entity.

A screenshot of a phone number

Description automatically generated with medium confidence-Phone Number: Weak entity.

Primary key: Cell Number.

Foreign key: Student ID.

Attributes:

1. Campus Number: Integer; no nulls; not unique.
2. Cell Number: Integer; no nulls; primary key.
3. Home Number: Integer; no nulls; not unique.
4. Student ID: Integer; no nulls; foreign key that references the Student entity.

A picture containing text, screenshot, font, line

Description automatically generated-Email: Weak entity.

Primary key: Email ID.

Foreign key: Student ID.

Attributes:

1. HWS: String; no nulls; single-valued and simple.
2. Other: String; no nulls; single-valued and simple.
3. Preferred: String; no nulls; single-valued and simple.
4. Email ID: String; no nulls; single-valued and primary key.
5. Student ID: Integer; no nulls; foreign key that references the Student entity.

A screen shot of a computer

Description automatically generated with low confidence-Status: Weak entity.

Primary key: Active/Inactive.

Foreign key: Student ID.

Attributes:

1. Active/Inactive: String; no nulls; single-valued and primary key.
2. Student ID: Student ID: Integer; no nulls; foreign key that references the student entity.

A picture containing text, screenshot, font, line

Description automatically generated

-Academic Load: Weak entity.

Primary key: Full/Part Time

Foreign key: Student ID.

Attributes:

1. Full/Part Time: String; no nulls; single-valued and primary key.
2. Student ID: Student ID: Integer; no nulls; foreign key that references the Student entity.

A picture containing text, screenshot, font, number

Description automatically generated-Academic Level: Weak entity.

Primary key: Credits.

Foreign key: Student ID.

Attributes:

1. FY: String; nulls; single-valued and not unique.
2. SO: String; nulls; single-valued and not unique.
3. JR: String; nulls; single-valued and not unique.
4. SR: String; nulls; single-valued and not unique.
5. Credits: integer; no nulls; single-valued and primary key.
6. Student ID: Student ID: Integer; no nulls; foreign key that references the Student entity.

A picture containing text, screenshot, font, number

Description automatically generated

-Major: Strong entity.

Primary key: Major ID.

Foreign key: Student ID.

Foreign key: Department ID.

Attributes:

1. Major ID: String; no nulls; single-valued and primary key.
2. Department ID: Integer; no nulls; foreign key that references the Department entity.
3. Disciplinary/Interdisciplinary: String; no nulls; single-valued and not unique.
4. Student ID: Student ID: Integer; no nulls; foreign key that references the Student entity.
5. Degree type: String; no nulls; single-valued and not unique.

A picture containing text, screenshot, font, number

Description automatically generated-Minor: Strong entity.

Primary key: Minor ID.

Foreign key: Student ID.

Foreign key: Department ID.

Attributes:

1. Minor ID: String; no nulls; single-valued and primary key.
2. Department ID: Integer; no nulls; foreign key that references the Department entity.
3. Disciplinary/Interdisciplinary: String; no nulls; single-valued and not unique.
4. Student ID: Student ID: Integer; no nulls; foreign key that references the Student entity.

A close-up of a document

Description automatically generated with low confidence

-Department: Strong entity.

Primary key: Department ID.

Attributes:

1. Department ID: Integer; no nulls; single-valued and primary key.
2. Disciplinary/Interdisciplinary: String; no nulls; single-valued and not unique.
3. Department name: String; no nulls; single-valued and not unique.

-Course: Strong entity.

A screen shot of a course

Description automatically generated with low confidencePrimary key: Course ID.

Foreign key: Department ID.

Attributes:

1. Course ID: Integer; no nulls; single-valued and primary key.
2. Department ID: Integer; no nulls; foreign key that references the Department entity.
3. Disciplinary/Interdisciplinary: String; no nulls; single-valued and not unique.
4. Student ID: Student ID: Integer; no nulls; foreign key that references the Student entity.

A screenshot of a computer

Description automatically generated with low confidence-Goal: Strong entity.

Primary key: Goal ID.

Foreign key: Course ID, referring to the Course Entity.

Attributes:

1. Goal ID: Integer; no nulls; single-valued and primary key.
2. Status: String; no nulls; single-valued and not unique.
3. Course ID: Integer; no nulls; single-valued and foreign key.

A screen shot of a computer

Description automatically generated with low confidence-Transcript: Weak entity.

Primary key: Transcript ID.

Foreign key: Student ID.

Foreign key: Course ID.

Attributes:

1. Credits: Integer; no nulls; single-valued and not unique.
2. GPA: Integer; no nulls; single-valued and not unique.
3. Transcript ID: Integer; no nulls; single-valued and primary key.
4. Course ID: Integer; no nulls; single-valued and foreign key.
5. Student ID: Integer; no nulls; foreign key that references the Student entity.

### 1.2.2Relationship Set Description

Student has addresses, phone numbers and email addresses.

A student has multiple addresses, phone numbers, and email addresses. So, there are one-to-many relationships between the Student entity and the Address, Phone, and Email entities.

Student has majors and minors.

A student can have up to two majors and two minors. So, there are one-to-many relationships between the Student entity and the Major and Minor entities.

Each Major and Minor are associated with a department.

Each major and minor is associated with a department, so there is a many-to-one relationship between the Major/Minor entities and the Department entity.

Student has an academic level, an academic load, and a status.

Each student can have an academic load (Full time/ Part time), and an academic level (FY, SO, JR or SR) the number of credits accumulated are checked to see the academic level. Each student has one academic load, one academic level and a status, so the relationship between student and academic load or level and status is one to one.

Goal of the Course & Course belongs to a department.

Each course has many goals so the relationship between course and goal is one to many. Every department has many courses so the relationship between department and course is one to many.

Transcript of the course

A transcript might have many courses and a course might have many transcripts, so the relationship between both is many to many.

Student has Transcript.

Student can have one or more transcripts, so the relationship between the Student entity and Transcript entity is one to many.

### 1.2.3Related Entity Set

When entities are combined to present a more comprehensive picture, this is known as generalisation. By combining various things into one, we can construct a larger entity based on their shared properties and characteristics.

The contrary of generalisation, however, is specialisation. Entities are separated into sub classes according to their attributes when they specialise. You can take an entity such as department and split it into subclasses like major, minor or course.

Because the entities major, minor and course have similar attributes and yet are used for different purposes, to facilitate the understanding of the ER diagram, we decided to not generalise them.

### 1.2.4E-R Diagram

An entity-relationship model, or E-R model, is a representation of entities and their relationships with one another. The connections between entities and their cardinalities are included in this. This approach enables the meaningful organisation of entities.

For this database, we developed the following E-R Model by making use of all prior knowledge and development. It is a very high-level and fundamental knowledge of what our university would need for this database. This is a rough draught since the design can alter as the project develops. We can now build a relational database using this paradigm, though.

A picture containing text, diagram, plan, font

Description automatically generated

# Phase 2: From E-R Model to Relational Model

## 2.1 Conceptual Database and Logical Database

### 2.1.1 E-R Model and Relational Model

There are a few key elements of the E-R paradigm that help in database construction and visualisation. The developer can construct a graphical representation of their database using E-R modelling. A data model may be developed by creating Entities that take the role of actual mental or physical things with independent existence and giving them attributes to fully describe them. Entities can then be connected through connections that have real-world significance.

### 2.1.2 Comparison of Two Different Models

Differences and Similarities

Both the E-R and Relational Model aid in conceptualising and visualising the final database design. The E-R model is used by database developers to visualise the primary conceptual or physical entities, as well as each entity's properties and relationships to other entities. Without some kind of visual aid, creating the relational model could be challenging. The relational model is used to help the developer finalise their proposed database layout as well as to create relations from proposed entities and link them together through foreign keys or relation tables.

Advantages and Disadvantages

The visualisation feature of the E-R paradigm is a benefit over the relational approach. The E-R model uses connected shapes to create a graphic that makes it simple to see entities, characteristics, and connections. Relationships are represented in the relational model as tables, which are less aesthetically pleasing and more difficult to deal with when constructing a database.

The E-R Model completely supports multi-valued and composite characteristics, whereas the Relational Model only fully supports single-valued and simple attributes. This is another benefit the E-R Model has over the Relational Model. This can be an issue when real-world items, like cars with several colours, function better with multi-valued and composite properties. This can be handled via conversion techniques, but it is unquestionably a benefit over the Relational Model.

The fundamental drawback of the E-R Model is that there is no specific query language for it. The SQL language, which is used to retrieve data, is a benefit of the relational model, making it a far more practical approach.

## 2.2 Conversion from E-R Model to a Relational Database

### 2.2.1 Converting Entity Types to Relations

Strong Entity Types

Strong entity types must be transformed into relations in the Relational Model by building a Relation that includes each of the basic characteristics that make up the entity being transformed. The relational model does not support composite or multivalued properties; only basic attributes are supported. The main key of the newly established relation must also be chosen from one of the entity's key characteristics.

Weak Entity Types

A relation can be built utilising all of the weak entity's simple properties when converting weak entity types to relations in the Relational Model. To properly map the relations for the relational model, a foreign key must be constructed from the parent entity's main key because this is a weak entity without a key of its own.

Simple and Composite Attributes

The Relational Model only supports simple characteristics. Simply add all simple characteristics as simple attributes of their relevant relations for the E-R Model's simple attributes. The Relational Model does not support composite properties. As a result, you will divide the composite attribute into a group of basic characteristics that can be utilised with ease in each relation in order to convert E-R Model composite attributes. Instead of dividing composite attributes into basic attributes, you may also establish a separate relation for them.

Single-valued and Multi-valued Attributes

The Relational Model only supports single-valued attributes. Include simple, single-valued qualities in their respective relation for the E-R Model's single-valued attributes.

The Relational Model does not handle multi-valued properties. In order to store the values for each multi-valued attribute, you must establish a separate relation for each such attribute. A foreign key in this new relation will connect this table to its parent relation.

### 2.2.2 Converting Relationship Types to Relations

One to One Relationship

There are three options available for converting binary One to One relationship types. Given entities Student and Academic level or load or status who are in a One to One Relationship:

1. The first choice is to include a foreign key that refers to the other relation in one of the relations. In this situation, full collaboration between the two parties is quite beneficial.
2. Merging the two entities into a single relation is the second alternative. For this to be successful, both parties must be fully involved.
3. Establishing a cross-reference relation is the final possibility for binary One to One interactions.

Many to Many Relationship

For converting binary Many to Many relationship types, there is only one option available.

Given entities Transcript and Course who are in a Many to Many relationships:

1. The final binary option to establish a cross-reference relationship, use many to many relationships. Make a relationship out of the connection between the transcript and the course. The main keys of Transcript and Course will be combined to form the primary key of this relation, R. Simple relationship attributes will be added as attributes of this new relation R.

### 2.2.3 Database Constraints

Certain restrictions, either built into the relational design itself or derived from business requirements, must be adhered to by databases. The database schema, or manner of organisation, is intimately tied to restrictions. The purpose of constraints is to protect the integrity of the data in the database and to specify how the data will be treated while being deleted, added, or updated. .

Domain Constraints

According to domain constraints, each attribute's values in a tuple—such as its datatype or enumerated datatype—must all fall within the designated domain. For instance, if the domain of an attribute is an integer, then the value of that attribute for all tuples must also be an integer. Alternatively, if the domain is an enumerated datatype, all tuples' values for the property must be contained in that enumerated set. The DBMS won't let you update or insert an attribute with a value that doesn't already exist in the attribute's domain.

Key Constraints

There must be a means to distinguish tuples from one another for data retrieval since the Relational Model is founded on set theory, which by definition requires that every element in a set be unique. We can use these special values to locate distinct records in a database by using primary and candidate keys. A relation, however, may contain many candidate keys, also known as unique keys. The primary key for this operation must be chosen as one. The DBMS will not let you construct a key that is not unique while inserting or editing data.

NULL Constraints

Another restriction is when an attribute cannot be null, such as a student's first or last name. If the attribute is set as non-null while updating or inserting, the DBMS will prevent you from inserting null data.

Entity Integrity Constraints

The main key cannot be null under the entity integrity constraint. The main key must always be unique and must not be NULL because it is used to identify distinct records in databases and each table is only allowed one primary key. Primary keys can also be used to link tables together via joins. The DBMS will not let you have a null primary key for adding or updating.

Referential Integrity Constraints

A foreign key in a relation must always refer to an existent tuple from the relation it is referencing under the referential integrity constraint.

The database operations rely on this limitation. For instance, only under certain circumstances may the delete operation be carried out. In our database, it would be against the referential integrity constraint to remove an artist from the Artist connection without also removing any and all associated albums and songs. You cannot create foreign keys that point to tuples in other relations that do not already exist in order to enter or update data in the database. Additionally, this would go against the requirement on referential integrity.

Check Constraint

When a database table is created using SQL, check constraints are established. By ensuring that a certain condition is satisfied, check constraints are used to guarantee the integrity of data that is being updated or inserted. The check constraint will evaluate to TRUE or FALSE depending on whether the value being entered is null or not. The check constraint will evaluate to UNKNOWN if the value being inserted is null, but it won't break the constraint. A tick limitation can state that an item's price is greater than $0.00 as an illustration. So that no database item can have a price that is less than or equal to $0.00 when data is added or modified.

## 2.3 Sample Queries for Database

### 2.3.1 Samples Data

Student Table:

INSERT INTO Student (student\_id, name, date\_of\_birth, admit\_term, expected\_graduation\_term, status, academic\_load, academic\_level)

SELECT

1, 'Calvito', '2002-07-24', 'September 2022', 'May 2025', 'Active', 'Full-time', 'JR'

UNION ALL SELECT

2, 'Eze', '2003-08-13', 'September 2022', 'May 2026', 'Active', 'Full-time', 'SO'

UNION ALL SELECT

3, 'Nene', '2004-01-02', 'September 2023', 'May 2026', 'Inactive', 'Full-time', 'SO'

UNION ALL SELECT

4, 'Aleesha', '2003-10-22', 'September 2023', 'May 2027', 'Inactive', 'Part-time', 'FY'

UNION ALL SELECT

5, 'Eric', '2002-03-16', 'September 2023', 'May 2026', 'Inactive', 'Full-time', 'SR'

;

Address Table:

INSERT INTO Address (address\_id, student\_id, address\_type, street\_address, city, state, country, zip\_code)

SELECT

1, 1, 'Campus', '51 HWS Avenue', 'Birmingham', 'West Midlands', 'Birmingham', 'B321EU'

UNION ALL SELECT

2, 1, 'Home', '110 Ekis De', 'Birmingham', 'West Midlands', 'Birmingham', 'B190AH'

UNION ALL SELECT

3, 1, 'Preferred', '32 Baca Lexera', 'Birmingham', 'West Midlands', 'Birmingham', 'B230DA'

UNION ALL SELECT

4, 2, 'Campus', '51 HWS Avenue', 'Birmingham', 'West Midlands', 'Birmingham', 'B321EU'

UNION ALL SELECT

5, 2, 'Home', '43 Baca Lexera', 'Birmingham', 'West Midlands', 'Birmingham', 'B321EU'

UNION ALL SELECT

6, 2, 'Preferred', '51 HWS Avenue', 'Birmingham', 'West Midlands', 'Birmingham', 'B321EU'

UNION ALL SELECT

7, 3, 'Campus', '51 HWS Avenue', 'Birmingham', 'West Midlands', 'Birmingham', 'B321EU'

UNION ALL SELECT

8, 3, 'Home', '12 Marra Ana', 'Birmingham', 'West Midlands', 'Birmingham', 'B458HR'

UNION ALL SELECT

9, 3, 'Preferred', '51 HWS Avenue', 'Birmingham', 'West Midlands', 'Birmingham', 'B321EU'

UNION ALL SELECT

10, 4, 'Campus', '51 HWS Avenue', 'Birmingham', 'West Midlands', 'Birmingham', 'B321EU'

UNION ALL SELECT

11, 4, 'Home', '76 Xupa Nepes', 'Birmingham', 'West Midlands', 'Birmingham', 'B102SA'

UNION ALL SELECT

12, 4, 'Preferred', '51 HWS Avenue', 'Birmingham', 'West Midlands', 'Birmingham', 'B321EU'

UNION ALL SELECT

13, 5, 'Campus', '51 HWS Avenue', 'Birmingham', 'West Midlands', 'Birmingham', 'B321EU'

UNION ALL SELECT

14, 5, 'Home', '230 Sucla Nalg', 'Birmingham', 'West Midlands', 'Birmingham', 'B562FA'

UNION ALL SELECT

15, 5, 'Preferred', '51 HWS Avenue', 'Birmingham', 'West Midlands', 'Birmingham', 'B321EU'

;

Phone Table:

INSERT INTO Phone (phone\_id, student\_id, phone\_type, phone\_number)

SELECT

1, 1, 'Campus', '0121-6534-2345'

UNION ALL SELECT

2, 1, 'Cell', '0755543294'

UNION ALL SELECT

3, 1, 'Home', '0121-2324-9867'

UNION ALL SELECT

4, 2, 'Campus', '0121-6534-2345'

UNION ALL SELECT

5, 2, 'Cell', '0776542214'

UNION ALL SELECT

6, 2, 'Home', '0121-9247-5347'

UNION ALL SELECT

7, 3, 'Campus', '0121-6534-2345'

UNION ALL SELECT

8, 3, 'Cell', '0766323815'

UNION ALL SELECT

9, 3, 'Home', '0121-9214-1362'

UNION ALL SELECT

10, 4, 'Campus', '0121-6534-2345'

UNION ALL SELECT

11, 4, 'Cell', '0722541113'

UNION ALL SELECT

12, 4, 'Home', '0121-9002-5397'

UNION ALL SELECT

13, 5, 'Campus', '0121-6534-2345'

UNION ALL SELECT

14, 5, 'Cell', '0711252134'

UNION ALL SELECT

15, 5, 'Home', '0121-6574-0192'

;

Email table:

INSERT INTO Email (email\_id, student\_id, email\_type, email\_address)

SELECT

1, 1, 'HWS', ' calvito@hws.co.uk'

UNION ALL SELECT

2, 1, 'Other', 'calvito@gmail.com'

UNION ALL SELECT

3, 1, 'Preferred', 'calvito@gmail.com'

UNION ALL SELECT

4, 2, 'HWS', ' eze@hws.co.uk'

UNION ALL SELECT

5, 2, 'Other', 'eze@gmail.com'

UNION ALL SELECT

6, 2, 'Preferred', 'eze@gmail.com'

UNION ALL SELECT

7, 3, 'HWS', ' nene@hws.co.uk'

UNION ALL SELECT

8, 3, 'Other', 'nene@gmail.com'

UNION ALL SELECT

9, 3, 'Preferred', 'nene@gmail.com'

UNION ALL SELECT

10, 4, 'HWS', ' aleesha@hws.co.uk'

UNION ALL SELECT

11, 4, 'Other', 'aleesha@gmail.com'

UNION ALL SELECT

12, 4, 'Preferred', 'aleesha@hws.co.uk'

UNION ALL SELECT

13, 5, 'HWS', ' eric@hws.co.uk'

UNION ALL SELECT

14, 5, 'Other', 'eric@gmail.com'

UNION ALL SELECT

15, 5, 'Preferred', 'eric@hws.co.uk'

;

Major Table:

INSERT INTO Major (major\_id, student\_id, major\_type, department\_id)

SELECT

1, 1, 'BS', 1

UNION ALL SELECT

2, 1, 'BS', 1

UNION ALL SELECT

3, 2, 'BS', 2

UNION ALL SELECT

4, 3, 'BS', 2

UNION ALL SELECT

5, 3, 'BS', 1

UNION ALL SELECT

6, 4, 'BS', 2

UNION ALL SELECT

7, 4, 'BS', 3

UNION ALL SELECT

8, 5, 'BA', 4

UNION ALL SELECT

9, 5, 'BA', 5

;

Minor Table:

INSERT INTO Minor (minor\_id, student\_id, department\_id)

SELECT

1, 2, 3

UNION ALL SELECT

2, 2, 5

;

Department table:

INSERT INTO Department (department\_id, department\_name)

SELECT

1, 'Department of Biology'

UNION ALL SELECT

2, 'Department of Computer Science'

UNION ALL SELECT

3, 'Department of Psicology'

UNION ALL SELECT

4, 'Department of History'

UNION ALL SELECT

5, 'Department of Law'

;

Course Table:

INSERT INTO Course (course\_id, department\_id, course\_number, title, credits)

SELECT

1, 1, 'BIO101', 'Introduction to Biology', 20

UNION ALL SELECT

2, 2, 'CS201', 'Introduction to Computer Science', 20

UNION ALL SELECT

3, 3, 'PS301', 'Introduction to Psicology', 20

UNION ALL SELECT

4, 4, 'HS401', 'Introduction to History', 20

UNION ALL SELECT

5, 5, 'LW501', 'Introduction to Law', 20

;

Goal table:

INSERT INTO Goal (goal\_id, goal\_name, course\_id)

SELECT

1, 'Quantitative Reasoning', 1

UNION ALL SELECT

2, 'Critical Thinking', 1

UNION ALL SELECT

3, 'Quantitative Reasoning', 2

UNION ALL SELECT

4, 'Critical Thinking', 2

UNION ALL SELECT

5, 'Quantitative Reasoning', 3

UNION ALL SELECT

6, 'Critical Thinking', 3

UNION ALL SELECT

7, 'Quantitative Reasoning', 4

UNION ALL SELECT

8, 'Critical Thinking', 4

UNION ALL SELECT

9, 'Quantitative Reasoning', 5

UNION ALL SELECT

10, 'Critical Thinking', 5

;

Transcript Table:

INSERT INTO Transcript (transcript\_id, student\_id, course\_id, grade, goal\_status)

SELECT

1, 1, 1, 'A', 'Satisfied'

UNION ALL SELECT

2, 2, 2, 'B', 'Partially Satisfied'

UNION ALL SELECT

3, 3, 2, 'A', 'Satisfied'

UNION ALL SELECT

1, 3, 1, 'A', 'Satisfied'

UNION ALL SELECT

4, 4, 2, 'A', 'Satisfied'

UNION ALL SELECT

5, 4, 3, 'B', 'Partially Satisfied'

UNION ALL SELECT

6, 5, 4, 'B', 'Partially Satisfied'

UNION ALL SELECT

7, 5, 5, 'A', 'Satisfied'

;

### 2.3.2 Design of Queries

-Select Address of a student:

SELECT \* FROM Address WHERE student\_id = 1;

-Select phone of a student:

SELECT \* FROM Phone WHERE student\_id = 4;

-Select email of a student:

SELECT \* FROM Email WHERE student\_id = 5;

-Select major of a student:

SELECT \* FROM Major WHERE student\_id = 3;

-Select department of a Major:

SELECT \* FROM Department WHERE department\_id IN (SELECT department\_id FROM Major WHERE student\_id = 2);

-Select course of a student:

SELECT \* FROM Course WHERE course\_id IN (SELECT course\_id FROM Transcript WHERE student\_id = 4);

-Select transcript of a student:

SELECT \* FROM Transcript WHERE student\_id = 1;

-Select goals of a student:

SELECT Goal.goal\_name, Transcript.goal\_status

FROM Goal

JOIN Course\_Goal ON Goal.goal\_id = Course\_Goal.goal\_id

JOIN Transcript ON Course\_Goal.course\_id = Transcript.course\_id

WHERE Transcript.student\_id = 2;

-Get credits accumulated by a studeny:

SELECT SUM(credits) AS total\_credits

FROM Course

WHERE course\_id IN (SELECT course\_id FROM Transcript WHERE student\_id = 5);

-Select courses that satisfy all goals:

SELECT Course.\*

FROM Course

JOIN Course\_Goal ON Course.course\_id = Course\_Goal.course\_id

JOIN Goal ON Course\_Goal.goal\_id = Goal.goal\_id

WHERE Goal.goal\_name = 'Quantitative Reasoning';

-Select all students that have satisfied all goals:

SELECT Student.\*

FROM Student

WHERE student\_id IN (

SELECT student\_id

FROM Transcript

GROUP BY student\_id

HAVING COUNT(DISTINCT goal\_status) = (

SELECT COUNT(DISTINCT goal\_id) FROM Goal

)

);

-Select all students who did not satisfy any goal:

SELECT Student.\*

FROM Student

WHERE student\_id NOT IN (

SELECT student\_id

FROM Transcript

WHERE goal\_status = 'Satisfied'

);

SELECT \* FROM Student WHERE academic\_level = 'SR';

-Select all senior students:

SELECT \* FROM Student WHERE academic\_level = 'SR';

-Select all courses that satisfy multiple goals:

SELECT Course.\*

FROM Course

JOIN Course\_Goal ON Course.course\_id = Course\_Goal.course\_id

WHERE Course.course\_id IN (

SELECT Course.course\_id

FROM Course

JOIN Course\_Goal ON Course.course\_id = Course\_Goal.course\_id

JOIN Goal ON Course\_Goal.goal\_id = Goal.goal\_id

WHERE Goal.goal\_name IN ('Quantitative Reasoning', 'Critical Thinking')

GROUP BY Course.course\_id

HAVING COUNT(DISTINCT Goal.goal\_name) = 2

);

-Select students that have taken specific courses of a department:

SELECT Student.\*

FROM Student

WHERE Student.student\_id IN (

SELECT Transcript.student\_id

FROM Transcript

JOIN Course ON Transcript.course\_id = Course.course\_id

JOIN Department ON Course.department\_id = Department.department\_id

WHERE Department.department\_name = 'Department of Biology'

);

-Select all students who did not take any courses of a department:

SELECT Student.\*

FROM Student

WHERE Student.student\_id NOT IN (

SELECT Transcript.student\_id

FROM Transcript

JOIN Course ON Transcript.course\_id = Course.course\_id

JOIN Department ON Course.department\_id = Department.department\_id

WHERE Department.department\_name = 'Department of Biology'

);

-Update student with student\_id=3 data:

UPDATE Student

SET name = 'Nene',

date\_of\_birth = '2002-03-20',

admit\_term = 'September 2020',

expected\_graduation\_term = 'May 2024',

status = 'Active',

academic\_load = 'Full-time',

academic\_level = 'SR'

WHERE student\_id = 3;

-Update email of a student from Email Table:

UPDATE Email

SET email\_address = 'calvito@hws.co.uk'

WHERE student\_id = 1 AND email\_type = 'Prefferred';

## Phase 3: Implementation of Relational Database

### 3.1 Relation Normalization

#### 3.1.1 Anomalies

It is possible to experience a number of problems in relationships that have not yet become normalised.

Insertion, deletion, and alteration abnormalities are the key problems.

Insertion Errors, Insertion anomalies are frequent for relation schema that are not well described. For instance, if we were inserting a new Album into the Album table and our Album table had all characteristics for the Studio where each album was produced at, we would need to ensure that the Studio attributes were consistent amongst Album tuples. A new Studio would be tough to add to the database as a result because there aren't any albums from it yet.

Deletion Anomalies

The second insertion anomaly is related to the fundamental problem with deletion anomalies. The data that originally existed for a given Studio would be utterly destroyed if we were to delete an Album from the Album table that might very well be the only Album ever recorded there.

Modification Anomalies

Update anomalies occur when changing a value in one tuple can lead to inconsistencies in many other tuples in a relation in a poorly constructed schema. For instance, all Album tuples that hold data for a Studio would need to be changed to reflect the new information if the value of an attribute of that Studio were to change. Massive data inconsistencies may emerge if this is not done properly.

#### 3.1.2 Normalization

In order to reduce superfluous data and data abnormalities, normalisation is the process of analysing database relations in order to make them comply to a particular degree of normality. A series of tests are performed against relations as part of the normalisation process, and chances to decompose the relations are sought out where necessary to reduce potential data concerns.

As was previously said, normalisation aims to reduce insertion, deletion, and update anomalies as well as the amount of data redundancy that takes place. However, there are different levels of normalisation. The following are the primary categories of normalisation: Boyce-Codd Normal Form, the First Normal Form, the Second Normal Form, and the Third Normal Form.

First Normal Form

According to First Normal Form, or 1NF, each attribute's domain can only include atomic values, and each attribute's value must be a single value from that domain. A set of values, tuple of values, or a combination of the two are not permitted in this situation. Nested relations are likewise not permitted by this. The relational model now considers 1NF to be a component of the specification of a relation, and therefore forbids the use of composite values and characteristics with multiple values.

Second Normal Form

The idea of full functional dependency is used in Second Normal Form, or 2NF. If eliminating any attribute C from A results in the dependency not holding, the functional dependency A B is said to have full functional dependency. Therefore, any non-key property for relations with primary keys that contain multiple attributes shouldn't be functionally dependent on a portion of the primary key. 1NF and 2NF must be met.

For instance, if we had a member tuple with the values "SSN, address," fname would be a functional dependence. It would not follow 2NF if we removed the address from the dependent and it continued to hold true.

Third Normal Form

The Third Normal Form, sometimes known as 3NF, makes use of the idea of transitive dependency. If there is a set of attributes C in T that is neither a candidate key nor a subset of any key of T, and if both A C and C B are present, then the functional relationship between A B in a relational schema T is a transitive dependency. To put it another way, no relation should have any non-key attributes that are functionally dependent on another non-key property, group of non-key attributes, or single non-key attribute. 1NF and 2NF must also be satisfied by 3NF.

Boyce-Codd Normal Form

Boyce-Codd Similar to 3NF, Normal Form, or BCNF, is much stricter and more powerful. It must be the case that whenever a nontrivial functional dependence X A holds in a relational schema R, then X is a super key of R for the relational schema to be in BCNF. Since BCNF does not satisfy the 3NF criterion that would allow A to be prime, the two definitions of BCNF only slightly diverge from one another. Since this characteristic is absent, BCNF is a more robust normal variant than 3NF.

#### 3.1.3 Relation Normalization

I adhered to the 1NF rule when creating our relational database, which states that composite values and multi-valued characteristics are prohibited. Since our original E-R Model design did not include any composite values or multi-valued attributes, adhering to 1NF naturally emerged from our design when we converted the E-R model to a relational model.

I was able to remove unnecessary data by adhering to the normalisation criteria, such as for the Buyer relation. Buyer information was initially saved in each Transaction tuple, which raised the possibility of conflicting data. However, by breaking out the Transaction relation, we were able to build a Buyer relation that would centralise all buyer data and remove any superfluous or inconsistent information. 3.2 SQL \*Plus

### 3.2 SQL Queries

-Select Address of a student:

SELECT \* FROM Address WHERE student\_id = 1;

-Select phone of a student:

SELECT \* FROM Phone WHERE student\_id = 4;

-Select email of a student:

SELECT \* FROM Email WHERE student\_id = 5;

-Select major of a student:

SELECT \* FROM Major WHERE student\_id = 3;

-Select department of a Major:

SELECT \* FROM Department WHERE department\_id IN (SELECT department\_id FROM Major WHERE student\_id = 2);

-Select course of a student:

SELECT \* FROM Course WHERE course\_id IN (SELECT course\_id FROM Transcript WHERE student\_id = 4);

-Select transcript of a student:

SELECT \* FROM Transcript WHERE student\_id = 1;

-Select goals of a student:

SELECT Goal.goal\_name, Transcript.goal\_status

FROM Goal

JOIN Course\_Goal ON Goal.goal\_id = Course\_Goal.goal\_id

JOIN Transcript ON Course\_Goal.course\_id = Transcript.course\_id

WHERE Transcript.student\_id = 2;

-Get credits accumulated by a studeny:

SELECT SUM(credits) AS total\_credits

FROM Course

WHERE course\_id IN (SELECT course\_id FROM Transcript WHERE student\_id = 5);

-Select courses that satisfy all goals:

SELECT Course.\*

FROM Course

JOIN Course\_Goal ON Course.course\_id = Course\_Goal.course\_id

JOIN Goal ON Course\_Goal.goal\_id = Goal.goal\_id

WHERE Goal.goal\_name = 'Quantitative Reasoning';

-Select all students that have satisfied all goals:

SELECT Student.\*

FROM Student

WHERE student\_id IN (

SELECT student\_id

FROM Transcript

GROUP BY student\_id

HAVING COUNT(DISTINCT goal\_status) = (

SELECT COUNT(DISTINCT goal\_id) FROM Goal

)

);

-Select all students who did not satisfy any goal:

SELECT Student.\*

FROM Student

WHERE student\_id NOT IN (

SELECT student\_id

FROM Transcript

WHERE goal\_status = 'Satisfied'

);

SELECT \* FROM Student WHERE academic\_level = 'SR';

-Select all senior students:

SELECT \* FROM Student WHERE academic\_level = 'SR';

-Select all courses that satisfy multiple goals:

SELECT Course.\*

FROM Course

JOIN Course\_Goal ON Course.course\_id = Course\_Goal.course\_id

WHERE Course.course\_id IN (

SELECT Course.course\_id

FROM Course

JOIN Course\_Goal ON Course.course\_id = Course\_Goal.course\_id

JOIN Goal ON Course\_Goal.goal\_id = Goal.goal\_id

WHERE Goal.goal\_name IN ('Quantitative Reasoning', 'Critical Thinking')

GROUP BY Course.course\_id

HAVING COUNT(DISTINCT Goal.goal\_name) = 2

);

-Select students that have taken specific courses of a department:

SELECT Student.\*

FROM Student

WHERE Student.student\_id IN (

SELECT Transcript.student\_id

FROM Transcript

JOIN Course ON Transcript.course\_id = Course.course\_id

JOIN Department ON Course.department\_id = Department.department\_id

WHERE Department.department\_name = 'Department of Biology'

);

-Select all students who did not take any courses of a department:

SELECT Student.\*

FROM Student

WHERE Student.student\_id NOT IN (

SELECT Transcript.student\_id

FROM Transcript

JOIN Course ON Transcript.course\_id = Course.course\_id

JOIN Department ON Course.department\_id = Department.department\_id

WHERE Department.department\_name = 'Department of Biology'

);

-Update student with student\_id=3 data:

UPDATE Student

SET name = 'Nene',

date\_of\_birth = '2002-03-20',

admit\_term = 'September 2020',

expected\_graduation\_term = 'May 2024',

status = 'Active',

academic\_load = 'Full-time',

academic\_level = 'SR'

WHERE student\_id = 3;

-Update email of a student from Email Table:

UPDATE Email

SET email\_address = 'calvito@hws.co.uk'

WHERE student\_id = 1 AND email\_type = 'Prefferred';

## Phase 4: Stored Procedures, Packages, and Triggers

### 4.1 Oracle PL/SQL

#### 4.1.1 What is PL/SQL?

Procedural Language Extensions to SQL, or PL/SQL, is a language that Oracle uses to give SQL greater programming power so that it can create more sophisticated objects and actions. The procedural language features of PL/SQL include conditions, loops, the ability to declare constants and variables, functions, and error handling.

Advantages of PL/SQL Create stored procedures, functions, and triggers using PL/SQL. These items are particularly practical since They allow for database automation. Instead of having to worry about complicated updates and deletes, procedures can be created that will handle all of this for you automatically, instead of having to execute the individual statements in the client, which can be less secure.

Control Statements

There are several control statements in PL/SQL that are particularly helpful for writing procedures.

PL/SQL control statements fall into three kinds. As follows:

1. Conditional Selection Statement: Executes many statements depending on the value of the input data. These are the IF and CASE statements.

2. Use loop statements to repeat the same instructions using a variety of data values. The LOOP, FOR, and WHILE loop statements are examples of loop statements.

3. Sequential control statements can be used in PL/SQL but are not required. GOTO and NULL are the two consecutive control statements.

#### 4.1.2 PL/SQL Syntax

Stored Procedure

Like functions, stored procedures are a collection of PL/SQL statements that can be performed countless times. These processes can incorporate factors as well. Stored procedures come in handy since it is more efficient to call a stored procedure once rather than repeatedly manually performing the same set of PL/SQL statements.

Syntax:

CREATE [OR REPLACE] PROCEDURE <procedure name> [list of parameters]

IS

<Declaration Section>

BEGIN

<procedure body>

END;

Stored Functions

A stored function is a collection of PL/SQL commands that can be used to invoke a function. With the exception that a function delivers a value to the place it is called, they are virtually identical to stored procedures. A method might or might not return a value.

Syntax:

CREATE [OR REPLACE] FUNCTION <function name> [parameters]

RETURN <return datatype>;

IS

<Declaration section>

BEGIN

<Function Body>

Return <return variable>; EXCEPTION

<Exception section>

Return <return variable>;

END;

Triggers

Programmes or procedures known as triggers are implicitly executed before, after, or in place of an update, insert, or delete and are saved in the database. With the aid of this technique, rules may be specified and will be enforced anytime data in a table is changed. Trigger syntax is demonstrated in the code that follows.

Syntax:

CREATE TRIGGER <trigger name>

<BEFORE | AFTER | INSTEAD OF>

<INSERT | DELETE | UPDATE>

<OF column name>

ON <table name>

FOR EACH ROW

WHEN <conditions>

BEGIN

<Desired statement go here>

END;

Cursor

When SQL statements are executed, a temporary work area called a cursor is established in the system memory. In order to access the data retrieved in a select statement in a process, you can assign a select statement a name using cursors. Cursors come in two varieties: explicit and implicit. Both have the same functionality, but they are accessed in distinct ways.

Syntax:

CURSOR <cursor name> IS

<Desired FUNCTIONALITY>

BEGIN

OPEN <cursor name>

<BODY>

CLOSE <cursor name>

END;

### 4.2 MS SQL Server and MySQL Stored Procedures

#### 4.2.1 Microsoft SQL Server and T-SQL

T-SQL, sometimes known as Transaction-SQL, is an extension of SQL that is used by Microsoft SQL Server. It is similar to PL/SQL in that it offers a number of capabilities that are not included in SQL, such as procedural programming and variables that enable the construction of stored procedures.The inclusion of the FROM clause in both the DELETE and UPDATE commands enables the use of JOINS, which makes filtering records and deleting data much simpler than in PL/SQL. This is a key feature of T-SQL that distinguishes it from Oracle.

T-SQL Procedure

Passing parameters is the primary distinction between T-SQL and PL/SQL. To distinguish between various parameter types, PL/SQL use the keywords IN, OUT, and INOUT. The same concept as what Oracle implements, but with a different syntax, is accomplished in T-SQL by the usage of OUT, OUTPUT, and READONLY. The @ signifying the use of a variable is another significant distinction. When assigning SELECT results into a variable in PL/SQL, the keyword INTO must be used. T-SQL, however, does not require this. The general structure resembles that of PL/SQL notwithstanding these differences.

Syntax:

CREATE { PROCEDURE | PROC } [schema\_name.]procedure\_name

[ @parameter [type\_schema\_name.] datatype

[ VARYING ] [ = default ] [ OUT | OUTPUT | READONLY ]

, @parameter [type\_schema\_name.] datatype

[ VARYING ] [ = default ] [ OUT | OUTPUT | READONLY ] ]

[ WITH { ENCRYPTION | RECOMPILE | EXECUTE AS Clause } ]

[ FOR REPLICATION ]

AS

BEGIN

[declaration\_section] executable\_section END;

END;

#### 4.2.2 MySQL Server Routines

MySQL Stored Procedure

MySQL uses what are called Routines, which are equivalent to Procedures in PL/SQL, and are very similar to Oracle Procedures in terms of syntax and semantics, as they have the ability to use cursors, as well as all control statements Oracle has, such as IF, ELSE, case statements, and loops. Similar to Oracle, MySQL also allows the use of IN, OUT, and INOUT parameter passing. And MySQL also requires that you use the keyword INTO when using SELECT in a procedure, which Oracle also requires.

Syntax:

MySQL Function

Although MySQL functions and Oracle functions are quite similar, MySQL does not support OUT or INOUT parameters. All parameters are IN by default, and this cannot be modified. In contrast to Oracle, which permits the use of all three types of parameters in functions and procedures, this is a significant difference. Additionally, MySQL functions are only permitted to return one value.

Syntax:

CREATE

[DEFINER = { user | CURRENT\_USER }]

FUNCTION sp\_name ([func\_parameter[,...]]) RETURNS type

[characteristic ...]

BEGIN routine\_body

End

## Phase 5: Graphical User Interface

### 5.1 SQL Tables

#### 5.1 Tables

-Student table:

CREATE TABLE Student (

student\_id INT PRIMARY KEY,

name VARCHAR(255) NOT NULL,

date\_of\_birth DATE,

admit\_term VARCHAR(20),

expected\_graduation\_term VARCHAR(20),

status VARCHAR(10),

academic\_load VARCHAR(15),

academic\_level VARCHAR(2)

);

-Address Table:

CREATE TABLE Address (

address\_id INT PRIMARY KEY,

student\_id INT,

address\_type VARCHAR(20) NOT NULL,

street\_address VARCHAR(255) NOT NULL,

city VARCHAR(100) NOT NULL,

state VARCHAR(100) NOT NULL,

country VARCHAR(100) NOT NULL,

zip\_code VARCHAR(20) NOT NULL,

FOREIGN KEY (student\_id) REFERENCES Student(student\_id)

);

-Phone table:

CREATE TABLE Phone (

phone\_id INT PRIMARY KEY,

student\_id INT,

phone\_type VARCHAR(20) NOT NULL,

phone\_number VARCHAR(20) NOT NULL,

FOREIGN KEY (student\_id) REFERENCES Student(student\_id)

);

-Email Table:

CREATE TABLE Email (

email\_id INT PRIMARY KEY,

student\_id INT,

email\_type VARCHAR(20) NOT NULL,

email\_address VARCHAR(255) NOT NULL,

FOREIGN KEY (student\_id) REFERENCES Student(student\_id)

);

-Major Table

CREATE TABLE Major (

major\_id INT PRIMARY KEY,

student\_id INT,

major\_type VARCHAR(2) NOT NULL,

department\_id INT,

FOREIGN KEY (student\_id) REFERENCES Student(student\_id),

FOREIGN KEY (department\_id) REFERENCES Department(department\_id)

);

-Minor Table:

CREATE TABLE Minor (

minor\_id INT PRIMARY KEY,

student\_id INT,

department\_id INT,

FOREIGN KEY (student\_id) REFERENCES Student(student\_id),

FOREIGN KEY (department\_id) REFERENCES Department(department\_id)

);

-Department Table:

CREATE TABLE Department (

department\_id INT PRIMARY KEY,

department\_name VARCHAR(255) NOT NULL

);

-Course Table:

CREATE TABLE Course (

course\_id INT PRIMARY KEY,

department\_id INT,

course\_number VARCHAR(10),

title VARCHAR(255),

credits INT,

FOREIGN KEY (department\_id) REFERENCES Department(department\_id)

);

-Goal Table:

CREATE TABLE Goal (

course\_id INT,

goal\_id INT,

FOREIGN KEY (course\_id) REFERENCES Course(course\_id),

FOREIGN KEY (goal\_id) REFERENCES Goal(goal\_id),

PRIMARY KEY (course\_id, goal\_id)

);

-Transcript Table:

CREATE TABLE Transcript (

transcript\_id INT PRIMARY KEY,

student\_id INT,

course\_id INT,

grade VARCHAR(2),

goal\_status VARCHAR(20),

FOREIGN KEY (student\_id) REFERENCES Student(student\_id),

FOREIGN KEY (course\_id) REFERENCES Course(course\_id)

);

### 5.2 Inserting into Database

#### 5.2.1 Database Insert into Tables

Student Table:

INSERT INTO Student (student\_id, name, date\_of\_birth, admit\_term, expected\_graduation\_term, status, academic\_load, academic\_level)

SELECT

1, 'Calvito', '2002-07-24', 'September 2022', 'May 2025', 'Active', 'Full-time', 'JR'

UNION ALL SELECT

2, 'Eze', '2003-08-13', 'September 2022', 'May 2026', 'Active', 'Full-time', 'SO'

UNION ALL SELECT

3, 'Nene', '2004-01-02', 'September 2023', 'May 2026', 'Inactive', 'Full-time', 'SO'

UNION ALL SELECT

4, 'Aleesha', '2003-10-22', 'September 2023', 'May 2027', 'Inactive', 'Part-time', 'FY'

UNION ALL SELECT

5, 'Eric', '2002-03-16', 'September 2023', 'May 2026', 'Inactive', 'Full-time', 'SR'

;

Address Table:

INSERT INTO Address (address\_id, student\_id, address\_type, street\_address, city, state, country, zip\_code)

SELECT

1, 1, 'Campus', '51 HWS Avenue', 'Birmingham', 'West Midlands', 'Birmingham', 'B321EU'

UNION ALL SELECT

2, 1, 'Home', '110 Ekis De', 'Birmingham', 'West Midlands', 'Birmingham', 'B190AH'

UNION ALL SELECT

3, 1, 'Preferred', '32 Baca Lexera', 'Birmingham', 'West Midlands', 'Birmingham', 'B230DA'

UNION ALL SELECT

4, 2, 'Campus', '51 HWS Avenue', 'Birmingham', 'West Midlands', 'Birmingham', 'B321EU'

UNION ALL SELECT

5, 2, 'Home', '43 Baca Lexera', 'Birmingham', 'West Midlands', 'Birmingham', 'B321EU'

UNION ALL SELECT

6, 2, 'Preferred', '51 HWS Avenue', 'Birmingham', 'West Midlands', 'Birmingham', 'B321EU'

UNION ALL SELECT

7, 3, 'Campus', '51 HWS Avenue', 'Birmingham', 'West Midlands', 'Birmingham', 'B321EU'

UNION ALL SELECT

8, 3, 'Home', '12 Marra Ana', 'Birmingham', 'West Midlands', 'Birmingham', 'B458HR'

UNION ALL SELECT

9, 3, 'Preferred', '51 HWS Avenue', 'Birmingham', 'West Midlands', 'Birmingham', 'B321EU'

UNION ALL SELECT

10, 4, 'Campus', '51 HWS Avenue', 'Birmingham', 'West Midlands', 'Birmingham', 'B321EU'

UNION ALL SELECT

11, 4, 'Home', '76 Xupa Nepes', 'Birmingham', 'West Midlands', 'Birmingham', 'B102SA'

UNION ALL SELECT

12, 4, 'Preferred', '51 HWS Avenue', 'Birmingham', 'West Midlands', 'Birmingham', 'B321EU'

UNION ALL SELECT

13, 5, 'Campus', '51 HWS Avenue', 'Birmingham', 'West Midlands', 'Birmingham', 'B321EU'

UNION ALL SELECT

14, 5, 'Home', '230 Sucla Nalg', 'Birmingham', 'West Midlands', 'Birmingham', 'B562FA'

UNION ALL SELECT

15, 5, 'Preferred', '51 HWS Avenue', 'Birmingham', 'West Midlands', 'Birmingham', 'B321EU'

;

Phone Table:

INSERT INTO Phone (phone\_id, student\_id, phone\_type, phone\_number)

SELECT

1, 1, 'Campus', '0121-6534-2345'

UNION ALL SELECT

2, 1, 'Cell', '0755543294'

UNION ALL SELECT

3, 1, 'Home', '0121-2324-9867'

UNION ALL SELECT

4, 2, 'Campus', '0121-6534-2345'

UNION ALL SELECT

5, 2, 'Cell', '0776542214'

UNION ALL SELECT

6, 2, 'Home', '0121-9247-5347'

UNION ALL SELECT

7, 3, 'Campus', '0121-6534-2345'

UNION ALL SELECT

8, 3, 'Cell', '0766323815'

UNION ALL SELECT

9, 3, 'Home', '0121-9214-1362'

UNION ALL SELECT

10, 4, 'Campus', '0121-6534-2345'

UNION ALL SELECT

11, 4, 'Cell', '0722541113'

UNION ALL SELECT

12, 4, 'Home', '0121-9002-5397'

UNION ALL SELECT

13, 5, 'Campus', '0121-6534-2345'

UNION ALL SELECT

14, 5, 'Cell', '0711252134'

UNION ALL SELECT

15, 5, 'Home', '0121-6574-0192'

;

Email table:

INSERT INTO Email (email\_id, student\_id, email\_type, email\_address)

SELECT

1, 1, 'HWS', ' calvito@hws.co.uk'

UNION ALL SELECT

2, 1, 'Other', 'calvito@gmail.com'

UNION ALL SELECT

3, 1, 'Preferred', 'calvito@gmail.com'

UNION ALL SELECT

4, 2, 'HWS', ' eze@hws.co.uk'

UNION ALL SELECT

5, 2, 'Other', 'eze@gmail.com'

UNION ALL SELECT

6, 2, 'Preferred', 'eze@gmail.com'

UNION ALL SELECT

7, 3, 'HWS', ' nene@hws.co.uk'

UNION ALL SELECT

8, 3, 'Other', 'nene@gmail.com'

UNION ALL SELECT

9, 3, 'Preferred', 'nene@gmail.com'

UNION ALL SELECT

10, 4, 'HWS', ' aleesha@hws.co.uk'

UNION ALL SELECT

11, 4, 'Other', 'aleesha@gmail.com'

UNION ALL SELECT

12, 4, 'Preferred', 'aleesha@hws.co.uk'

UNION ALL SELECT

13, 5, 'HWS', ' eric@hws.co.uk'

UNION ALL SELECT

14, 5, 'Other', 'eric@gmail.com'

UNION ALL SELECT

15, 5, 'Preferred', 'eric@hws.co.uk'

;

Major Table:

INSERT INTO Major (major\_id, student\_id, major\_type, department\_id)

SELECT

1, 1, 'BS', 1

UNION ALL SELECT

2, 1, 'BS', 1

UNION ALL SELECT

3, 2, 'BS', 2

UNION ALL SELECT

4, 3, 'BS', 2

UNION ALL SELECT

5, 3, 'BS', 1

UNION ALL SELECT

6, 4, 'BS', 2

UNION ALL SELECT

7, 4, 'BS', 3

UNION ALL SELECT

8, 5, 'BA', 4

UNION ALL SELECT

9, 5, 'BA', 5

;

Minor Table:

INSERT INTO Minor (minor\_id, student\_id, department\_id)

SELECT

1, 2, 3

UNION ALL SELECT

2, 2, 5

;

Department table:

INSERT INTO Department (department\_id, department\_name)

SELECT

1, 'Department of Biology'

UNION ALL SELECT

2, 'Department of Computer Science'

UNION ALL SELECT

3, 'Department of Psicology'

UNION ALL SELECT

4, 'Department of History'

UNION ALL SELECT

5, 'Department of Law'

;

Course Table:

INSERT INTO Course (course\_id, department\_id, course\_number, title, credits)

SELECT

1, 1, 'BIO101', 'Introduction to Biology', 20

UNION ALL SELECT

2, 2, 'CS201', 'Introduction to Computer Science', 20

UNION ALL SELECT

3, 3, 'PS301', 'Introduction to Psicology', 20

UNION ALL SELECT

4, 4, 'HS401', 'Introduction to History', 20

UNION ALL SELECT

5, 5, 'LW501', 'Introduction to Law', 20

;

Goal table:

INSERT INTO Goal (goal\_id, goal\_name, course\_id)

SELECT

1, 'Quantitative Reasoning', 1

UNION ALL SELECT

2, 'Critical Thinking', 1

UNION ALL SELECT

3, 'Quantitative Reasoning', 2

UNION ALL SELECT

4, 'Critical Thinking', 2

UNION ALL SELECT

5, 'Quantitative Reasoning', 3

UNION ALL SELECT

6, 'Critical Thinking', 3

UNION ALL SELECT

7, 'Quantitative Reasoning', 4

UNION ALL SELECT

8, 'Critical Thinking', 4

UNION ALL SELECT

9, 'Quantitative Reasoning', 5

UNION ALL SELECT

10, 'Critical Thinking', 5

;

Transcript Table:

INSERT INTO Transcript (transcript\_id, student\_id, course\_id, grade, goal\_status)

SELECT

1, 1, 1, 'A', 'Satisfied'

UNION ALL SELECT

2, 2, 2, 'B', 'Partially Satisfied'

UNION ALL SELECT

3, 3, 2, 'A', 'Satisfied'

UNION ALL SELECT

1, 3, 1, 'A', 'Satisfied'

UNION ALL SELECT

4, 4, 2, 'A', 'Satisfied'

UNION ALL SELECT

5, 4, 3, 'B', 'Partially Satisfied'

UNION ALL SELECT

6, 5, 4, 'B', 'Partially Satisfied'

UNION ALL SELECT

7, 5, 5, 'A', 'Satisfied'

;