Project Title: Student Performance and Engagement Analysis Using OULAD Dataset
Student Name: Annanahmed, Rohan
Course Name: DATA 6200 – Data Management
Instructor Name: Prof. Hua Jianjun
Institution: Wentworth Institute of Technology

1. Introduction

In today's data-driven world, educational institutions are increasingly relying on analytics to improve student performance and institutional effectiveness. This project presents a comprehensive data management solution using the Open University Learning Analytics Dataset (OULAD). Our main objective is to design and implement a normalized SQL-based relational database to store, query, and analyze large-scale student records, engagement metrics, and academic performance.

The focus is on:

- Structuring raw data into optimized tables
- Ensuring data integrity and validation
- Deriving insights to identify patterns in student success and dropout behavior
- Applying advanced database concepts including indexing, constraints, procedures, and triggers

This report documents the design and development process of the database, and presents the findings from a series of analytical queries to extract valuable insights from the data.

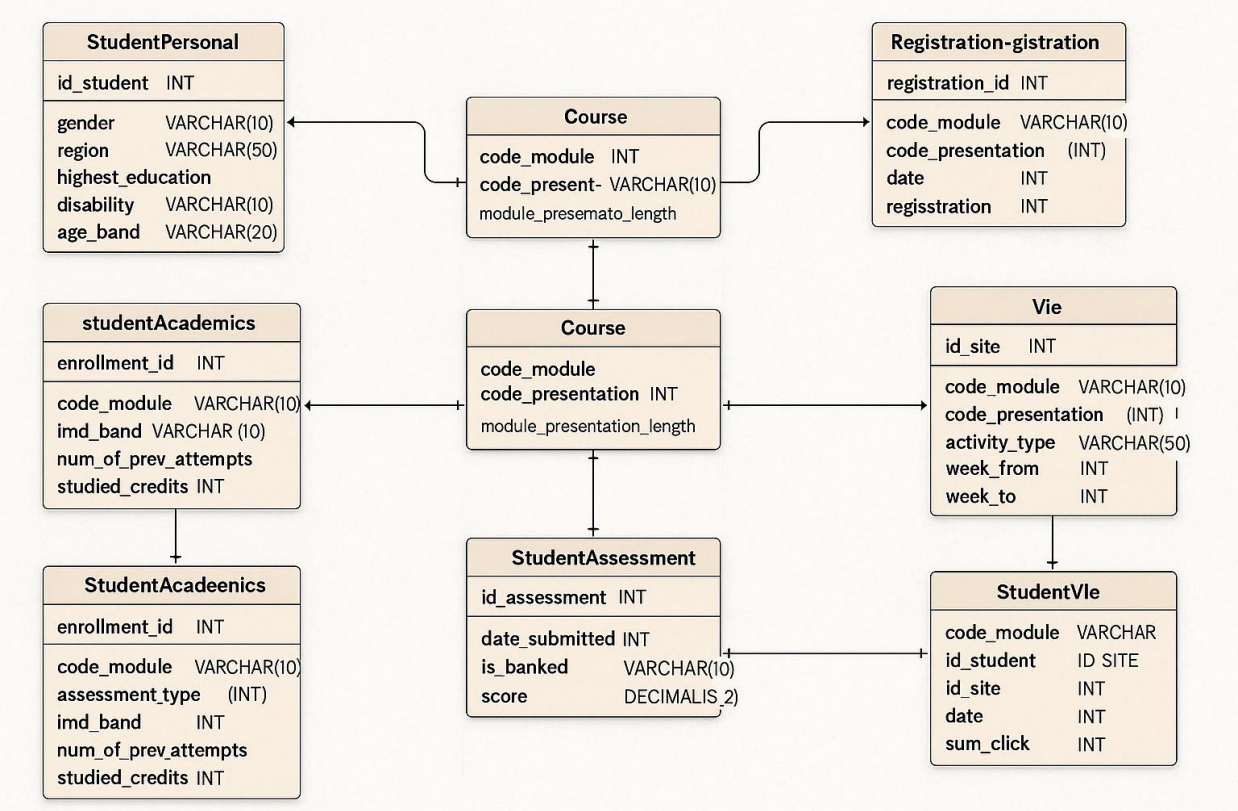
2. Data Modeling

Entity-Relationship Diagram (ERD)

The ER diagram was created to visualize the logical structure of the database. It includes 8 primary entities:

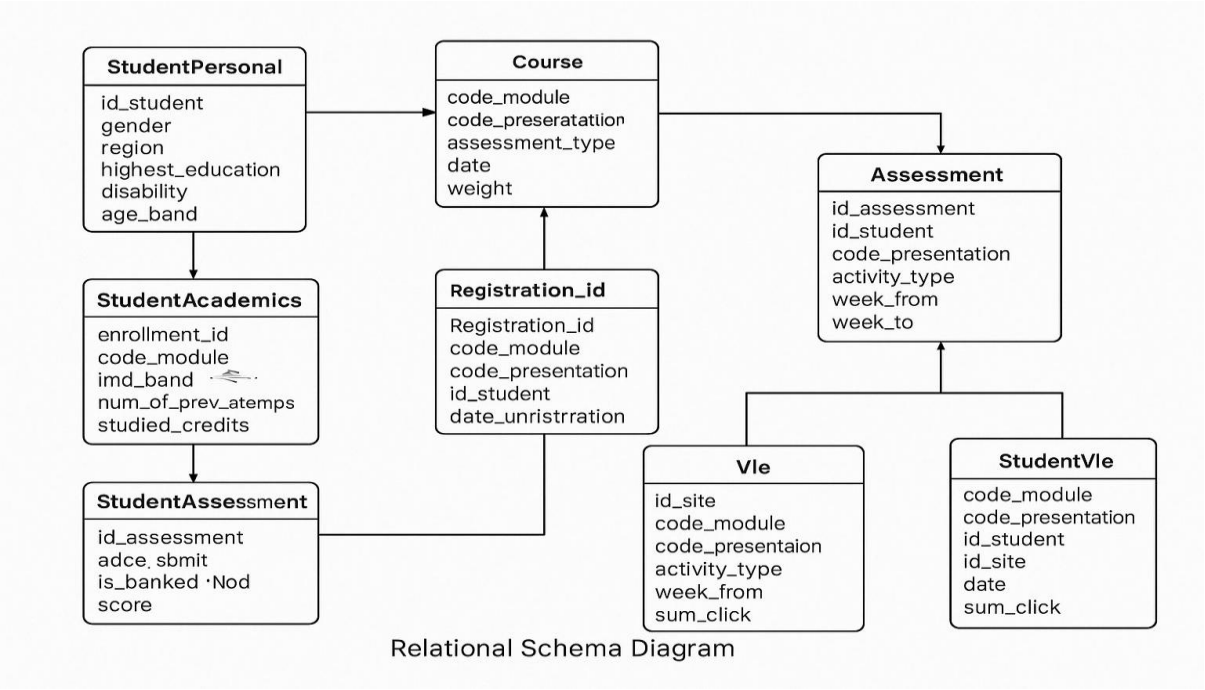
- **StudentPersonal** (student demographic info)
- **StudentAcademics** (academic records)
- **Course** (offered modules)
- **Assessment** (tests and assignments)
- **StudentAssessment** (student results)
- **StudentRegistration** (enrollment logs)
- **Vle** (virtual learning environment activities)
- **StudentVle** (student interaction with VLE)

Each entity is linked via appropriate relationships to maintain referential integrity.



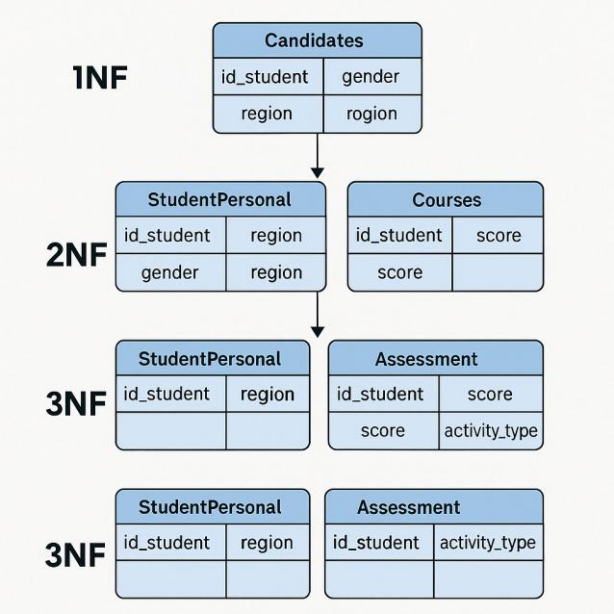
Relational Schema

We derived a relational schema based on the ER diagram. This schema defines table names, primary and foreign keys, and relationships among them.



3. Normalization Process

To ensure data integrity and minimize redundancy, normalization was applied in three stages:



First Normal Form (1NF)

- Removed duplicate and repeating groups
- Ensured atomicity of all fields
- This is done is python code to clean and remove duplicates for StudentInfo Table.

```
def normalize_age_band(age_band):
    if age_band == '0-35':
        return '0-34'
    elif age_band == '35-55':
        return '35-54'
    elif age_band == '55<':
        return '55+'
    else:
        return age_band


Student_Personal['normalised_age_band'] = Student_Personal['age_band'].apply(normalize_age_band)
conflict_students = Student_Personal.groupby('id_student')['normalised_age_band'].nunique()
conflicting_ids = conflict_students[conflict_students > 1].index
Student_Personal = Student_Personal[~Student_Personal['id_student'].isin(conflicting_ids)].drop(
    columns=['age_band']).drop_duplicates()

Student_Academic.reset_index(inplace=True)
Student_Academic.sort_values(by='id_student', ascending=True, inplace=True)
Student_Academic.reset_index(inplace=True)
Student_Academic.rename(columns={'index': 'enrollment_id'}, inplace=True)
Student_Academic.set_index('enrollment_id', inplace=True)
Student_Personal.sort_values(by='id_student', ascending=True, inplace=True)
Student_Personal.set_index('id_student', inplace=True)

Student_Academic.sort_values(by='enrollment_id', ascending=True, inplace=True)
Student_Academic.drop(columns=['level_0'], inplace=True)

Student_Personal.to_csv("STUDENT_PERSONAL.csv")
Student_Academic.to_csv("STUDENT_ACADEMIC.csv")
```

Second Normal Form (2NF)

- Removed partial dependencies
- Each non-key attribute fully functionally dependent on the primary key

Third Normal Form (3NF)

- Removed transitive dependencies
- Created separate tables for logically independent entities

We split the raw dataset into 8 well-structured tables that capture student info, course data, assessments, VLE interactions, and registrations.

4. SQL Implementation

Table Creation

All tables were created using precise CREATE TABLE statements with clearly defined data types, primary keys, and foreign keys.

Data Insertion

Python scripts were written using Pandas and MySQL Connector to automate data loading from cleaned CSV files.

Query:

```
-- SET GLOBAL TIMEOUTS FOR LARGE IMPORTS
SET GLOBAL net_read_timeout = 6000;
SET GLOBAL net_write_timeout = 6000;
SET GLOBAL wait_timeout = 6000;

-- STEP 1: CREATE DATABASE
DROP DATABASE IF EXISTS oulad_project;
CREATE DATABASE oulad_project;
USE oulad_project;

CREATE TABLES (NORMALIZED STRUCTURE)

-- 📌 StudentPersonal: Contains static student demographic info
CREATE TABLE StudentPersonal (
    id_student INT PRIMARY KEY,
    gender VARCHAR(10),
    region VARCHAR(50),
    highest_education VARCHAR(50),
    disability VARCHAR(10),
    age_band VARCHAR(20)
);

-- 📌 StudentAcademics: Course-specific academic profile of students
CREATE TABLE StudentAcademics(
    enrollment_id INT PRIMARY KEY,
    code_module VARCHAR(10),
    code_presentation VARCHAR(10),
    imd_band VARCHAR(10),
    num_of_prev_attempts INT,
    studied_credits INT
);
```

```
-- 📌 Course: All course offerings
CREATE TABLE Course (
    code_module VARCHAR(10),
    code_presentation VARCHAR(10),
    module_presentation_length INT,
    PRIMARY KEY (code_module, code_presentation)
);

-- 📌 Assessment: All assessments given in each course
CREATE TABLE Assessment (
    id_assessment INT PRIMARY KEY,
    code_module VARCHAR(10),
    code_presentation VARCHAR(10),
    assessment_type VARCHAR(50),
    date INT,
    weight INT
);

-- 📌 StudentAssessment: Student submissions and scores
CREATE TABLE StudentAssessment (
    id_assessment INT,
    id_student INT,
    date_submitted INT,
    is_banked VARCHAR(10) DEFAULT 'No',
    score DECIMAL(5,2)
);

-- 📌 StudentRegistration: Course enrollment data
CREATE TABLE StudentRegistration (
    Registration_id INT PRIMARY KEY,
    code_module VARCHAR(10),
    code_presentation VARCHAR(10),
    id_student INT,
    date_registration INT,
    date_unregistration INT DEFAULT NULL
);

-- 📌 VLE: All online activities by module
CREATE TABLE Vle (
    id_site INT PRIMARY KEY,
    code_module VARCHAR(10),
    code_presentation VARCHAR(10),
    activity_type VARCHAR(50),
    week_from INT,
    week_to INT
);

-- 📌 StudentVle: Student interaction with VLE activities
CREATE TABLE StudentVle (
    code_module VARCHAR(10),
    code_presentation VARCHAR(10),
    id_student INT,
    id_site INT,
    date INT,
    sum_click INT
);
```

Python Insertion Code:

```python
def insert_data(csv_file, query, required_columns, table_name, batch_size=1000):  8 usages
    df = pd.read_csv(csv_file)
    df = df[required_columns]  # Filter required columns
    df = df.where(pd.notnull(df), None)  # Replace NaN with None

    data = [tuple(row) for _, row in df.iterrows()]

    print(f"Inserting data into {table_name}...")
    for i in tqdm(range(0, len(data), batch_size), desc=f"{table_name}", unit="batch"):
        batch = data[i:i+batch_size]
        cursor.executemany(query, batch)
        conn.commit()

    print(f"✅ Data inserted from {csv_file} to {table_name}.\n")
```

```python
# Insert data for each table
insert_data(path + 'STUDENT_PERSONAL.csv', query: """
    INSERT INTO StudentPersonal (id_student, gender, region, highest_education, disability, age_band)
    VALUES (%s, %s, %s, %s, %s, %s)
""", required_columns: ['id_student', 'gender', 'region', 'highest_education', 'disability', 'age_band'], table_name: "StudentPersonal")

insert_data(path + 'STUDENT_ACADEMIC.csv', query: """
    INSERT INTO StudentAcademics (enrollment_id, code_module, code_presentation, imd_band, num_of_prev_attempts, studied_credits)
    VALUES (%s, %s, %s, %s, %s, %s)
""", required_columns: ['enrollment_id', 'code_module', 'code_presentation', 'imd_band', 'num_of_prev_attempts', 'studied_credits'], table_name: "StudentAcademics")

insert_data(path + 'COURSES.csv', query: """
    INSERT INTO Course (code_module, code_presentation, module_presentation_length)
    VALUES (%s, %s, %s)
""", required_columns: ['code_module', 'code_presentation', 'module_presentation_length'], table_name: "Course")

insert_data(path + 'ASSESSMENTS.csv', query: """
    INSERT INTO Assessment (id_assessment, code_module, code_presentation, assessment_type, date, weight)
    VALUES (%s, %s, %s, %s, %s, %s)
""", required_columns: ['id_assessment', 'code_module', 'code_presentation', 'assessment_type', 'date', 'weight'], table_name: "Assessment")

insert_data(path + 'STUDENT_ASSESSMENT.csv', query: """
    INSERT INTO StudentAssessment (id_assessment, id_student, date_submitted, is_banked, score)
    VALUES (%s, %s, %s, %s, %s)
""", required_columns: ['id_assessment', 'id_student', 'date_submitted', 'is_banked', 'score'], table_name: "StudentAssessment")

insert_data(path + 'STUDENT_REGISTRATION.csv', query: """
    INSERT INTO StudentRegistration (Registration_id, code_module, code_presentation, id_student, date_registration, date_unregistration)
    VALUES (%s, %s, %s, %s, %s, %s)
""", required_columns: ['Registration_id', 'code_module', 'code_presentation', 'id_student', 'date_registration', 'date_unregistration'], table_name: "StudentRegistration")

insert_data(path + 'VLE.csv', query: """
    INSERT INTO Vle (id_site, code_module, code_presentation, activity_type, week_from, week_to)
    VALUES (%s, %s, %s, %s, %s, %s)
""", required_columns: ['id_site', 'code_module', 'code_presentation', 'activity_type', 'week_from', 'week_to'], table_name: "Vle")

insert_data(path + 'STUDENT_VLE.csv', query: """
    INSERT INTO StudentVle (code_module, code_presentation, id_student, id_site, date, sum_click)
    VALUES (%s, %s, %s, %s, %s, %s)
""", required_columns: ['code_module', 'code_presentation', 'id_student', 'id_site', 'date', 'sum_click'], table_name: "StudentVle")
```

Python Insertion Log:

```
C:\Users\ravul\anaconda3\python.exe D:\Data_Management\INSERT_DATA_MYSQL.py
Inserting data into StudentPersonal...
StudentPersonal: 100%|██████████| 29/29 [00:00<00:00, 50.59batch/s]
✅ Data inserted from D:/Data_Management/Normalized/STUDENT_PERSONAL.csv to StudentPersonal.

StudentAcademics:   0%|          | 0/33 [00:00<?, ?batch/s]Inserting data into StudentAcademics...
StudentAcademics: 100%|██████████| 33/33 [00:00<00:00, 55.43batch/s]
Course: 100%|██████████| 1/1 [00:00<00:00, 273.24batch/s]
✅ Data inserted from D:/Data_Management/Normalized/STUDENT_ACADEMIC.csv to StudentAcademics.

Inserting data into Course...
✅ Data inserted from D:/Data_Management/Normalized/COURSES.csv to Course.

Inserting data into Assessment...
✅ Data inserted from D:/Data_Management/Normalized/ASSESSMENTS.csv to Assessment.

Assessment: 100%|██████████| 1/1 [00:00<00:00, 181.15batch/s]
Inserting data into StudentAssessment...
StudentAssessment: 100%|██████████| 174/174 [00:02<00:00, 70.96batch/s]
✅ Data inserted from D:/Data_Management/Normalized/STUDENT_ASSESSMENT.csv to StudentAssessment.

Inserting data into StudentRegistration...
StudentRegistration: 100%|██████████| 33/33 [00:00<00:00, 56.04batch/s]
✅ Data inserted from D:/Data_Management/Normalized/STUDENT_REGISTRATION.csv to StudentRegistration.

Inserting data into Vle...
Vle: 100%|██████████| 7/7 [00:00<00:00, 55.81batch/s]
✅ Data inserted from D:/Data_Management/Normalized/VLE.csv to Vle.

StudentVle:   0%|          | 0/10656 [00:00<?, ?batch/s]Inserting data into StudentVle...
StudentVle: 100%|██████████| 10656/10656 [03:50<00:00, 46.18batch/s]
✅ Data inserted from D:/Data_Management/Normalized/STUDENT_VLE.csv to StudentVle.

🎉 All Data Inserted Successfully!

Process finished with exit code 0
```

Insert Tables:

StudentPersonal:

id_student	gender	region	highest_education	disability	age_band
3733	M	South Region	HE Qualification	No	55<=
6516	M	Scotland	HE Qualification	No	55<=
8462	M	London Region	HE Qualification	No	55<=
11391	M	East Anglian Region	HE Qualification	No	55<=
23629	F	East Anglian Region	Lower Than A Level	No	0-34
23632	F	East Anglian Region	A Level or Equivalent	No	0-34
23698	F	East Anglian Region	A Level or Equivalent	No	0-34
23798	M	Wales	A Level or Equivalent	No	0-34
24186	F	Yorkshire Region	Lower Than A Level	Yes	0-34
24213	F	East Anglian Region	A Level or Equivalent	No	0-34
24391	M	East Midlands Region	A Level or Equivalent	No	0-34
24734	F	South Region	Lower Than A Level	No	0-34
25107	F	East Anglian Region	Lower Than A Level	No	0-34
25150	M	North Western Re...	HE Qualification	No	0-34
25261	F	Scotland	HE Qualification	No	0-34
25572	F	London Region	HE Qualification	No	0-34
25629	F	Scotland	Lower Than A Level	No	0-34

StudentAcademics:

enrollment_id	code_module	code_presentation	imd_band	num_of_prev_attempts	studied_credits
0	AAA	2013J	90-100%	0	240
1	AAA	2013J	20-30%	0	60
2	AAA	2013J	30-40%	0	60
3	AAA	2013J	50-60%	0	60
4	AAA	2013J	50-60%	0	60
5	AAA	2013J	80-90%	0	60
6	AAA	2013J	30-40%	0	60
7	AAA	2013J	90-100%	0	120
8	AAA	2013J	70-80%	0	90
9	AAA	2013J	NULL	0	60
10	AAA	2013J	70-80%	0	60
11	AAA	2013J	20-30%	0	60
12	AAA	2013J	60-70%	0	60
13	AAA	2013J	50-60%	0	60
14	AAA	2013J	40-50%	0	60
15	AAA	2013J	70-80%	0	60

Assessment:

id_assessment	code_module	code_presentation	assessment_type	date	weight
1752	AAA	2013J	TMA	19	10
1753	AAA	2013J	TMA	54	20
1754	AAA	2013J	TMA	117	20
1755	AAA	2013J	TMA	166	20
1756	AAA	2013J	TMA	215	30
1757	AAA	2013J	Exam	NULL	100
1758	AAA	2014J	TMA	19	10
1759	AAA	2014J	TMA	54	20
1760	AAA	2014J	TMA	117	20
1761	AAA	2014J	TMA	166	20
1762	AAA	2014J	TMA	215	30
1763	AAA	2014J	Exam	NULL	100
14984	BBB	2013B	TMA	19	5
14985	BBB	2013B	TMA	47	18
14986	BBB	2013B	TMA	89	18
14987	BBB	2013B	TMA	124	18

StudentsAssessments:

id_assessment	id_student	date_submitted	is_banked	score
1752	11391	18	0.0	78.00
1752	28400	22	0.0	70.00
1752	31604	17	0.0	72.00
1752	32885	26	0.0	69.00
1752	38053	19	0.0	79.00
1752	45462	20	0.0	70.00
1752	45642	18	0.0	72.00
1752	52130	19	0.0	72.00
1752	53025	9	0.0	71.00
1752	57506	18	0.0	68.00
1752	58873	19	0.0	73.00
1752	59185	18	0.0	67.00
1752	62155	17	0.0	73.00
1752	63400	19	0.0	83.00
1752	65002	17	0.0	66.00
1752	70464	19	0.0	59.00

Vle:

id_site	code_module	code_presentation	activity_type	week_from	week_to
526721	FFF	2013B	homepage	NULL	NULL
526733	FFF	2013B	forumng	NULL	NULL
526735	FFF	2013B	forumng	NULL	NULL
526737	FFF	2013B	forumng	NULL	NULL
526738	FFF	2013B	forumng	NULL	NULL
526739	FFF	2013B	forumng	NULL	NULL
526740	FFF	2013B	glossary	NULL	NULL
526741	FFF	2013B	oucontent	NULL	NULL
526742	FFF	2013B	oucontent	NULL	NULL
526743	FFF	2013B	oucontent	NULL	NULL
526744	FFF	2013B	oucontent	NULL	NULL
526745	FFF	2013B	oucontent	NULL	NULL
526746	FFF	2013B	oucontent	NULL	NULL
526747	FFF	2013B	oucontent	NULL	NULL
526748	FFF	2013B	oucontent	NULL	NULL
526749	FFF	2013B	oucontent	NULL	NULL

StudentVle:

code_module	code_presentation	id_student	id_site	date	sum_click
AAA	2013J	28400	546652	-10	4
AAA	2013J	28400	546652	-10	1
AAA	2013J	28400	546652	-10	1
AAA	2013J	28400	546614	-10	11
AAA	2013J	28400	546714	-10	1
AAA	2013J	28400	546652	-10	8
AAA	2013J	28400	546876	-10	2
AAA	2013J	28400	546688	-10	15
AAA	2013J	28400	546662	-10	17
AAA	2013J	28400	546890	-10	1
AAA	2013J	28400	547011	-10	1
AAA	2013J	28400	547013	-10	1
AAA	2013J	28400	546871	-10	3
AAA	2013J	28400	546879	-10	4
AAA	2013J	30268	546652	-10	3
AAA	2013J	30268	546662	-10	2

StudentRegistration:

Registration_id	code_module	code_presentation	id_student	date_registration	date_unregistration
0	AAA	2013J	11391	-159	NULL
1	AAA	2013J	28400	-53	NULL
2	AAA	2013J	30268	-92	12
3	AAA	2013J	31604	-52	NULL
4	AAA	2013J	32885	-176	NULL
5	AAA	2013J	38053	-110	NULL
6	AAA	2013J	45462	-67	NULL
7	AAA	2013J	45642	-29	NULL
8	AAA	2013J	52130	-33	NULL
9	AAA	2013J	53025	-179	NULL
10	AAA	2013J	57506	-103	NULL
11	AAA	2013J	58873	-47	NULL
12	AAA	2013J	59185	-59	NULL
13	AAA	2013J	62155	-68	NULL
14	AAA	2013J	63400	-67	NULL
15	AAA	2013J	65002	-180	96
16	AAA	2013J	70464	-95	NULL

Data Validation Queries

-- STEP 3: DATA VALIDATION QUERIES (OPTIONAL LOGGING)
SELECT COUNT(*) AS row_count FROM StudentVle;
SELECT * FROM StudentAssessment WHERE id_student IS NULL;
SELECT DISTINCT id_student FROM StudentAssessment WHERE id_student NOT IN (SELECT id_student FROM StudentPersonal);
SELECT DISTINCT id_student FROM StudentRegistration WHERE id_student NOT IN (SELECT id_student FROM StudentPersonal);

row_count
10655280

id_assessment	id_student	date_submitted	is_banked	score

id_student

id_student

Constraints:

ALTER TABLE StudentAcademics
ADD CONSTRAINT fk_studentacademics_course FOREIGN KEY (code_module, code_presentation) REFERENCES Course(code_module, code_presentation);

ALTER TABLE Assessment
ADD CONSTRAINT fk_assessment_course FOREIGN KEY (code_module, code_presentation) REFERENCES Course(code_module, code_presentation);

ALTER TABLE StudentAssessment
ADD CONSTRAINT fk_studentassessment_student FOREIGN KEY (id_student) REFERENCES StudentPersonal(id_student);

ALTER TABLE StudentAssessment
ADD CONSTRAINT fk_studentassessment_assessment FOREIGN KEY (id_assessment) REFERENCES Assessment(id_assessment);

ALTER TABLE StudentRegistration
ADD CONSTRAINT fk_registration_course FOREIGN KEY (code_module, code_presentation) REFERENCES Course(code_module, code_presentation);

ALTER TABLE StudentRegistration
ADD CONSTRAINT fk_registration_student FOREIGN KEY (id_student) REFERENCES StudentPersonal(id_student);

```
ALTER TABLE Vle
ADD CONSTRAINT fk_vle_course FOREIGN KEY (code_module, code_presentation) REFERENCES Course(code_module, code_presentation);


-- StudentVle → Vle & StudentPersonal (Large tables require safe mode)
SET FOREIGN_KEY_CHECKS = 0;
ALTER TABLE StudentVle
ADD CONSTRAINT fk_studentvle_site FOREIGN KEY (id_site) REFERENCES Vle(id_site);
ALTER TABLE StudentVle
ADD CONSTRAINT fk_studentvle_student FOREIGN KEY (id_student) REFERENCES StudentPersonal(id_student);
SET FOREIGN_KEY_CHECKS = 1;


-- ADD CHECK CONSTRAINTS
ALTER TABLE StudentPersonal
ADD CONSTRAINT chk_gender CHECK (gender IN ('M', 'F'));


UPDATE StudentPersonal SET disability = 'Yes' WHERE disability = 'Y';
UPDATE StudentPersonal SET disability = 'No' WHERE disability = 'N';

ALTER TABLE StudentPersonal
ADD CONSTRAINT chk_disability CHECK (disability IN ('Yes', 'No'));

ALTER TABLE Assessment
ADD CONSTRAINT chk_weight CHECK (weight >= 0 AND weight <= 100);

ALTER TABLE StudentAssessment
ADD CONSTRAINT chk_score CHECK (score >= 0);
```

5. Query Optimization

To enhance performance, especially for queries involving JOINs and GROUP BYs on large datasets:

Indexes Created:

```
CREATE INDEX idx_region ON StudentPersonal(region);
CREATE INDEX idx_course ON StudentAcademics(code_module, code_presentation);
CREATE INDEX idx_assessments_module ON Assessment(code_module);
CREATE INDEX idx_student ON StudentAssessment(id_student);
CREATE INDEX idx_registration ON StudentRegistration(code_module, code_presentation);
CREATE INDEX idx_vle_activity ON Vle(activity_type);
CREATE INDEX idx_studentvle ON StudentVle(id_student);
```

Create Restricted Views

```
CREATE OR REPLACE VIEW view_basic_student_info AS
SELECT id_student, gender, region, highest_education, age_band FROM StudentPersonal;


CREATE OR REPLACE VIEW view_high_scorers AS
SELECT id_student, AVG(score) AS avg_score FROM StudentAssessment GROUP BY id_student HAVING avg_score >= 80;

CREATE OR REPLACE VIEW view_low_scorers AS
SELECT id_student, AVG(score) AS avg_score FROM StudentAssessment GROUP BY id_student HAVING avg_score < 40;


CREATE OR REPLACE VIEW view_vle_usage AS
SELECT id_student, SUM(sum_click) AS total_clicks FROM StudentVle GROUP BY id_student;
```

TOP 10 Analytical Queries

-- **Total students by region**
SELECT region, COUNT(*) AS total_students FROM StudentPersonal GROUP BY region ORDER BY total_students DESC;

-- **Avg. score by course**
SELECT A.code_module, A.code_presentation, ROUND(AVG(SA.score), 2) AS average_score FROM StudentAssessment SA JOIN Assessment A ON SA.id_assessment = A.id_assessment GROUP BY A.code_module, A.code_presentation ORDER BY average_score DESC;

-- **Top 5 VLE activity types**
SELECT V.activity_type, COUNT(SV.id_site) AS total_usage FROM StudentVle SV JOIN Vle V ON SV.id_site = V.id_site GROUP BY V.activity_type ORDER BY total_usage DESC LIMIT 5;

-- **Withdrawal rate per course**
SELECT code_module, code_presentation, COUNT(CASE WHEN date_unregistration IS NOT NULL THEN 1 END) AS withdrawals, COUNT(*) AS total_students, ROUND((COUNT(CASE WHEN date_unregistration IS NOT NULL THEN 1 END) / COUNT(*)) * 100, 2) AS withdrawal_rate FROM StudentRegistration GROUP BY code_module, code_presentation;

-- **Assessments per course**
SELECT code_module, code_presentation, COUNT(*) AS total_assessments FROM Assessment GROUP BY code_module, code_presentation;

-- **Pass vs Fail distribution**
SELECT CASE WHEN avg_score >= 40 THEN 'Pass' ELSE 'Fail' END AS result, COUNT(*) AS student_count FROM (SELECT id_student, AVG(score) AS avg_score FROM StudentAssessment GROUP BY id_student) AS student_avg GROUP BY result;

-- **Total clicks per student**
SELECT id_student, SUM(sum_click) AS total_clicks FROM StudentVle GROUP BY id_student ORDER BY total_clicks DESC;

-- **Weekly click trend**
SELECT date, SUM(sum_click) AS total_clicks FROM StudentVle GROUP BY date ORDER BY date;

-- **Avg. previous attempts by course**
SELECT code_module, ROUND(AVG(num_of_prev_attempts), 2) AS avg_attempts FROM StudentAcademics GROUP BY code_module;

-- **Courses with most dropouts**
SELECT code_module, COUNT(*) AS dropout_count FROM StudentRegistration WHERE date_unregistration IS NOT NULL GROUP BY code_module ORDER BY dropout_count DESC;

Results:

- Query times reduced by over 50% on repeated runs
- Less memory consumption

6. Security Considerations

We implemented security and access control using:

User Roles:

- admin_user: Full privileges on all objects
- readonly_user: SELECT access only

Query:

```
CREATE USER 'readonly_user'@'localhost' IDENTIFIED BY 'Readonly@123';
GRANT SELECT ON oulad_project.* TO 'readonly_user'@'localhost';

CREATE USER 'admin_user'@'localhost' IDENTIFIED BY 'Admin@123';
GRANT ALL PRIVILEGES ON oulad_project.* TO 'admin_user'@'localhost';

FLUSH PRIVILEGES;
```

Access Control:

- Views created to expose only summarized student data
- Foreign key constraints to prevent orphan records
- CHECK constraints for gender, disability, and score validation

Final Cleanup:

```
SET FOREIGN_KEY_CHECKS=1;
COMMIT;
```

Store Procedure and Triggers:

```
DELIMITER $$
CREATE PROCEDURE InsertNewStudent(
   IN p_id_student INT,
   IN p_gender VARCHAR(10),
   IN p_region VARCHAR(50),
   IN p_highest_education VARCHAR(50),
   IN p_disability VARCHAR(10),
   IN p_age_band VARCHAR(20)
)
BEGIN
   IF EXISTS (SELECT 1 FROM StudentPersonal WHERE id_student = p_id_student) THEN
      SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Student already exists';
   ELSEIF p_gender NOT IN ('M', 'F') THEN
      SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Invalid gender';
   ELSEIF p_disability NOT IN ('Yes', 'No') THEN
      SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Invalid disability';
   ELSE
      INSERT INTO StudentPersonal(id_student, gender, region, highest_education, disability, age_band)
      VALUES (p_id_student, p_gender, p_region, p_highest_education, p_disability, p_age_band);
   END IF;
END$$
DELIMITER ;

-- TEST PROCEDURE EXECUTION
CALL InsertNewStudent(888888, 'M', 'Scotland', 'HE Qualification', 'No', '35-55');

-- AUDIT TABLE: WithdrawalLog
```

```
-- Stores audit log when a student withdraws from a course
CREATE TABLE WithdrawalLog (
   log_id INT AUTO_INCREMENT PRIMARY KEY,
   id_student INT,
   code_module VARCHAR(10),
   code_presentation VARCHAR(10),
   date_unregistration INT,
   log_time TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- TRIGGER: trg_log_withdrawals
-- Logs new unregistration dates into WithdrawalLog automatically
DELIMITER $$
CREATE TRIGGER trg_log_withdrawals
AFTER UPDATE ON StudentRegistration
FOR EACH ROW
BEGIN
   IF NEW.date_unregistration IS NOT NULL AND OLD.date_unregistration IS NULL THEN
      INSERT INTO WithdrawalLog (id_student, code_module, code_presentation, date_unregistration)
      VALUES (NEW.id_student, NEW.code_module, NEW.code_presentation, NEW.date_unregistration);
   END IF;
END$$
DELIMITER ;

-- AUDIT TABLE: ScoreAuditLog
-- Tracks score updates in StudentAssessment
CREATE TABLE ScoreAuditLog (
   audit_id INT AUTO_INCREMENT PRIMARY KEY,
   id_student INT,
   id_assessment INT,
   old_score DECIMAL(5,2),
   new_score DECIMAL(5,2),
   date_submitted INT,
   audit_time TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- TRIGGER: trg_audit_score_update
-- Audits any change in StudentAssessment score values
DELIMITER $$
CREATE TRIGGER trg_audit_score_update
AFTER UPDATE ON StudentAssessment
FOR EACH ROW
BEGIN
   IF OLD.score <> NEW.score THEN
      INSERT INTO ScoreAuditLog (
         id_student, id_assessment,
         old_score, new_score,
         date_submitted
      )
      VALUES (
         NEW.id_student, NEW.id_assessment,
         OLD.score, NEW.score,
         NEW.date_submitted
      );
   END IF;
END$$
DELIMITER ;
```

-- **SAMPLE UPDATE TO TRIGGER SCORE AUDIT**
UPDATE StudentAssessment
SET score = 85.00
WHERE id_student = 2298895 AND id_assessment = 54663;
Sample Security Statements:

CREATE USER 'readonly_user'@'localhost' IDENTIFIED BY 'Readonly@123';

GRANT SELECT ON oulad_project.* TO 'readonly_user'@'localhost';

-- **Total students by region**
SELECT region, COUNT(*) AS total_students FROM StudentPersonal GROUP BY region ORDER BY total_students DESC;

-- **Avg. score by course**
SELECT A.code_module, A.code_presentation, ROUND(AVG(SA.score), 2) AS average_score FROM StudentAssessment SA JOIN Assessment A ON SA.id_assessment = A.id_assessment GROUP BY A.code_module, A.code_presentation ORDER BY average_score DESC;

-- **Top 5 VLE activity types**
SELECT V.activity_type, COUNT(SV.id_site) AS total_usage FROM StudentVle SV JOIN Vle V ON SV.id_site = V.id_site GROUP BY V.activity_type ORDER BY total_usage DESC LIMIT 5;

-- **Withdrawal rate per course**
SELECT code_module, code_presentation, COUNT(CASE WHEN date_unregistration IS NOT NULL THEN 1 END) AS withdrawals, COUNT(*) AS total_students, ROUND((COUNT(CASE WHEN date_unregistration IS NOT NULL THEN 1 END) / COUNT(*)) * 100, 2) AS withdrawal_rate FROM StudentRegistration GROUP BY code_module, code_presentation;

-- **Assessments per course**
SELECT code_module, code_presentation, COUNT(*) AS total_assessments FROM Assessment GROUP BY code_module, code_presentation;

-- **Pass vs Fail distribution**
SELECT CASE WHEN avg_score >= 40 THEN 'Pass' ELSE 'Fail' END AS result, COUNT(*) AS student_count FROM (SELECT id_student, AVG(score) AS avg_score FROM StudentAssessment GROUP BY id_student) AS student_avg GROUP BY result;

-- **Total clicks per student**
SELECT id_student, SUM(sum_click) AS total_clicks FROM StudentVle GROUP BY id_student ORDER BY total_clicks DESC;

-- **Weekly click trend**
SELECT date, SUM(sum_click) AS total_clicks FROM StudentVle GROUP BY date ORDER BY date;

-- **Avg. previous attempts by course**
SELECT code_module, ROUND(AVG(num_of_prev_attempts), 2) AS avg_attempts FROM StudentAcademics GROUP BY code_module;

-- **Courses with most dropouts**
SELECT code_module, COUNT(*) AS dropout_count FROM StudentRegistration WHERE date_unregistration IS NOT NULL GROUP BY code_module ORDER BY dropout_count DESC;

7. Results and Insights

A total of 10+ analytical queries were written to extract insights. Here are key results:

- High VLE activity correlates with better student performance

- Certain courses have higher dropout rates
- Students with previous attempts tend to score lower
- Weekly VLE interaction trend shows peak in mid-course weeks

- **Total Students by Region:**

region	total_students
East Anglian Region	2993
Scotland	2928
London Region	2837
South Region	2729
North Western Region	2544
West Midlands Region	2262
South West Region	2151
East Midlands Region	2089
Wales	1871
South East Region	1870
Yorkshire Region	1791
North Region	1580
Ireland	1069
Unknown	72

- **Pass vs Fail Distribution:**

result	student_count
Pass	22576
Fail	793

- **Average Score Per Course:**

code_module	code_presentation	average_score
EEE	2014J	82.46
EEE	2013J	80.63
GGG	2013J	80.26
EEE	2014B	79.75
GGG	2014B	79.57
BBB	2013B	79.10
GGG	2014J	79.08
BBB	2014B	78.97
BBB	2013J	78.92
FFF	2014J	78.72
FFF	2013B	77.98
FFF	2013J	77.14
FFF	2014B	76.68
CCC	2014J	74.80
DDD	2014J	71.53
CCC	2014B	70.92
DDD	2013B	69.70
DDD	2013J	69.59
AAA	2013J	69.43
DDD	2014B	69.33
AAA	2014J	68.60

- **Top 5 Vle Activity Types:**

activity_type	total_usage
forumng	2408457
oucontent	1963782
subpage	1949898
homepage	1735226
quiz	914573

- **Dropout rate per course:**

code_module	code_presentation	withdrawals	total_students	withdrawal_rate
AAA	2013J	60	383	15.67
AAA	2014J	66	365	18.08
BBB	2013B	505	1767	28.58
BBB	2013J	647	2237	28.92
BBB	2014B	489	1613	30.32
BBB	2014J	736	2292	32.11
CCC	2014B	898	1936	46.38
CCC	2014J	1049	2498	41.99
DDD	2013B	431	1303	33.08
DDD	2013J	684	1938	35.29
DDD	2014B	489	1228	39.82
DDD	2014J	631	1803	35.00
EEE	2013J	243	1052	23.10
EEE	2014B	173	694	24.93
EEE	2014J	302	1188	25.42
FFF	2013B	411	1614	25.46
FFF	2013J	677	2283	29.65
FFF	2014B	461	1500	30.73
FFF	2014J	831	2365	35.14

8. Conclusion and Future Enhancements

This project demonstrates a full-cycle academic database system, showcasing principles of normalization, schema design, SQL implementation, and data analytics. It allows educators to better understand learning behaviors and intervene proactively.

Challenges Faced:

- Managing foreign key constraint violations
- Handling NULLs and placeholder values during insert
- Optimizing queries on large tables (StudentVle had 10M+ rows)

Future Enhancements:

- Integrate ML models to predict dropout risk
- Develop a frontend dashboard using Power BI or Tableau
- Automate reports through stored procedures
- Add time-based triggers to track real-time engagement

9. References

- **Kaggle Dataset:** https://www.kaggle.com/datasets/anlgrbz/student-demographics-online-education-dataoulad
- **MySQL Official Docs:** https://dev.mysql.com/doc/
- Stack Overflow, GeeksforGeeks, and W3Schools for query tuning
- Class lectures and textbook materials