Prepared by **Annanahmed Shaikh, Rohan Pratap Reddy Ravula**

Student Performance & Engagement Analysis using OULAD Dataset

A Normalized, Secure, and Insight-Driven SQL Database System Based on the OULAD Dataset

15 April, 2025

Course Name: **Data Management**
Subject Code: **DATA 6200**

WENTWORTH INSTITUTE OF TECHNOLOGY
Professor Hua Jianjun

Introduction & Objectives

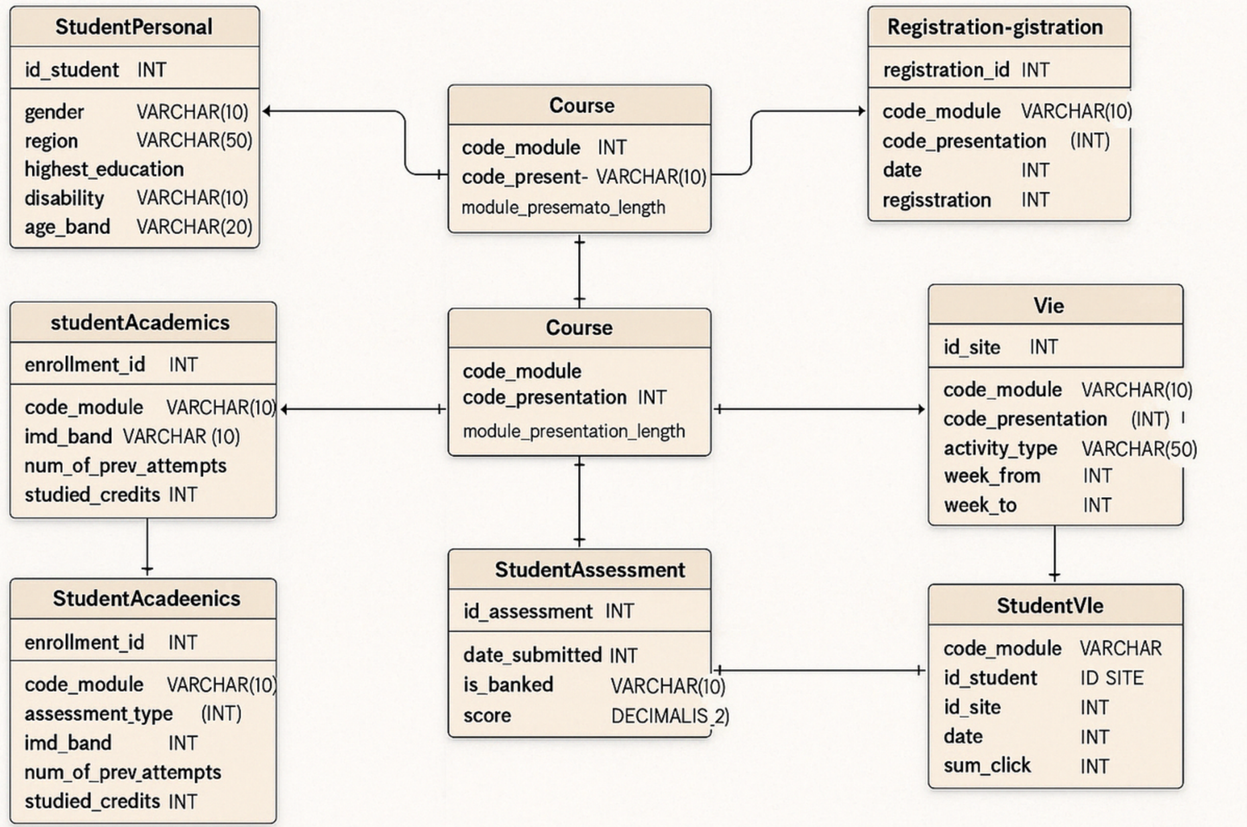
- Educational institutions are focusing on improving student outcomes through data.
- This project builds a robust SQL database from the OULAD dataset.
- Key Objectives:
  - Normalize data to reduce redundancy.
  - Ensure secure, scalable design.
  - Extract insights into student engagement, dropout patterns, and academic performance.

Dataset Overview

- **Dataset Name:** Open University Learning Analytics Dataset (OULAD)
- **Source:** Kaggle | ~10 million records
- **Key CSV Files Used**:
  - studentInfo, courses, assessments, vle, studentAssessment, studentVle, studentRegistration, StudentPersonal
- Cleaned and normalized into 8 interlinked tables

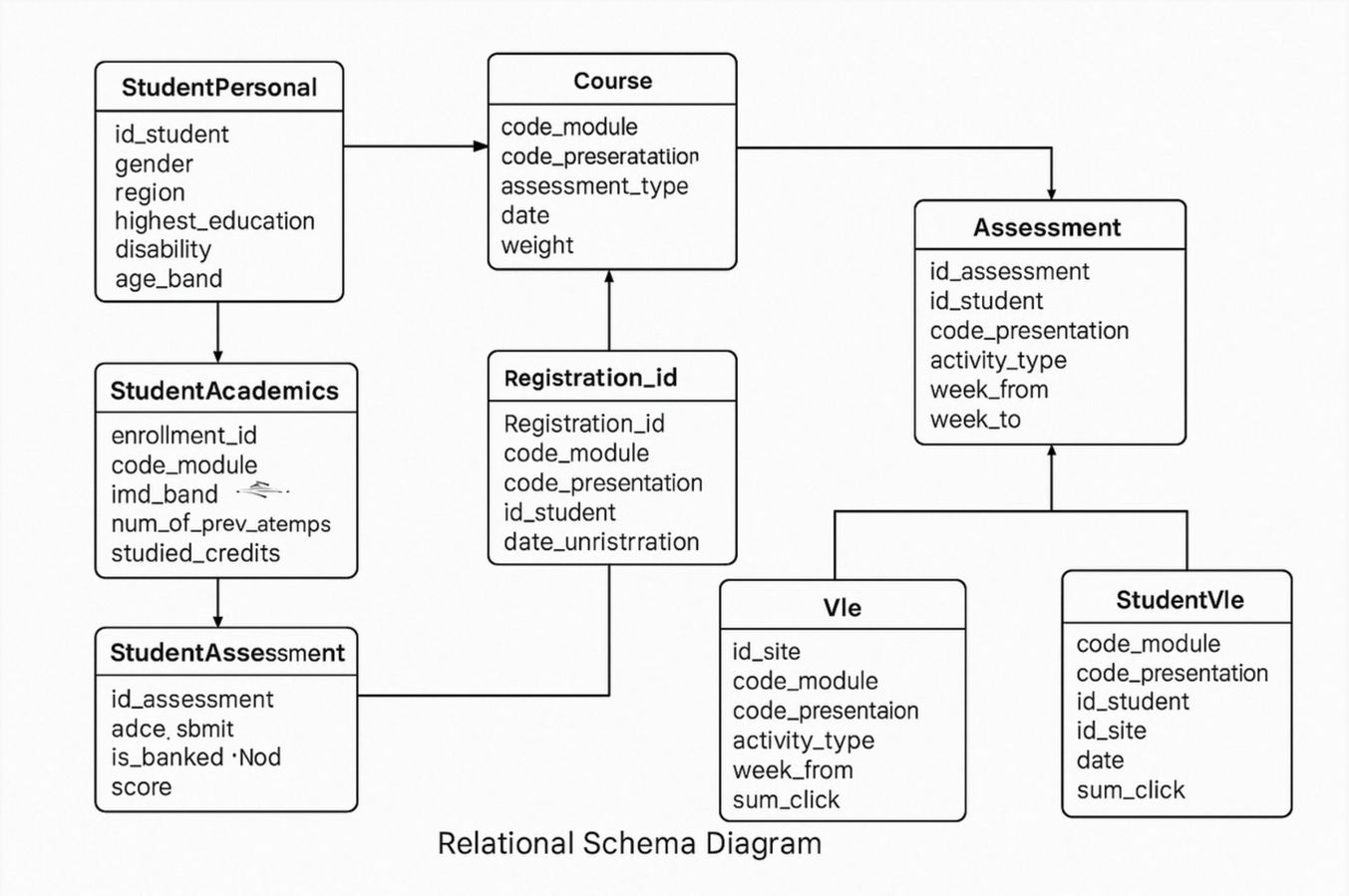
ER Diagram

- Eight normalized tables representing student data, courses, assessments, and engagement
- **Relationships shown:**
  - One-to-many: StudentPersonal StudentAssessment
  - Many-to-one: StudentVle Vle
  - Composite Keys: Course(code_module, code_presentation)



Relational Schema

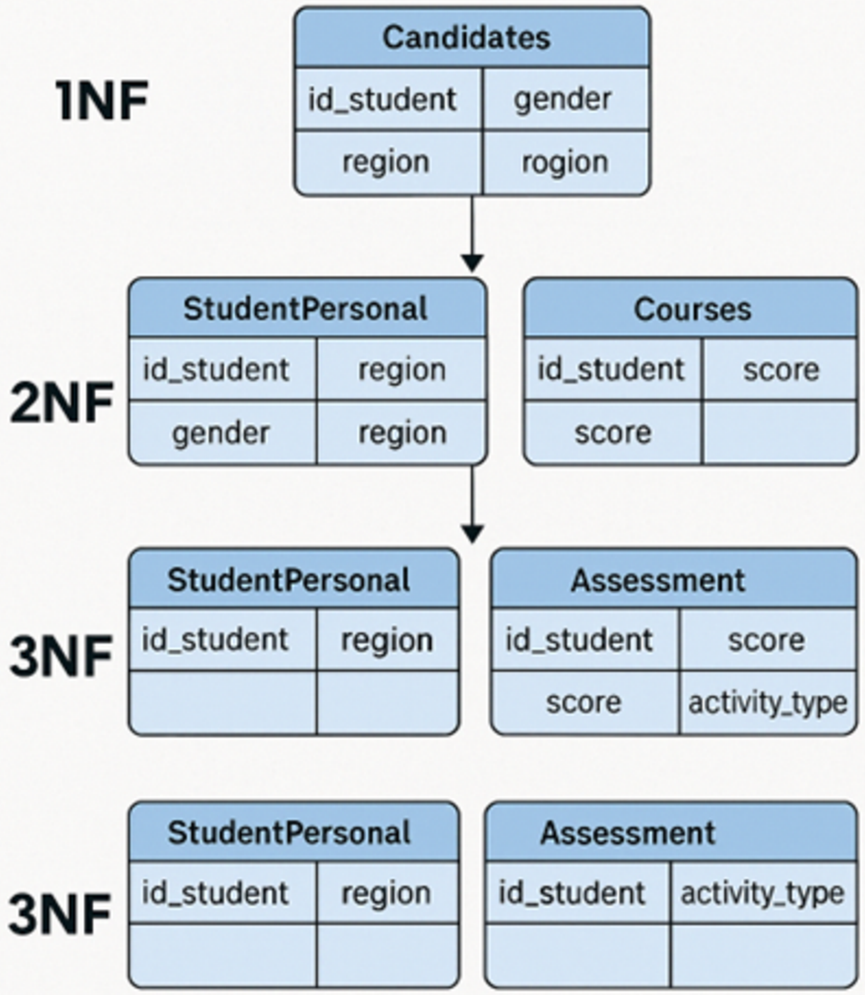
- Shows all primary keys, foreign keys, and table structures
- **Enforces relational integrity between:**
  - Students Courses
  - Courses Assessments
  - VLE Student activity



Relational Schema Diagram

Normalization Process

- **1NF:** Removed repeating groups; ensured atomicity
- **2NF:** Separated partial dependencies (e.g., academic vs personal info)
- **3NF**: Removed transitive dependencies and split into logical tables



SQL Table Creation + Python Integration

- Tables created with correct data types, PK/FK, and constraints
- **Python script used:**
  - Pandas + MySQL connector
  - Cleaned NaN/null values
  - Handled exceptions and logs

```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")
```

```
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
```

Foreign Keys & Data Integrity

- **FK Constraints enforce referential links:**
  - e.g., StudentAssessment.id_student StudentPersonal.id_student
- Avoided violations via data validation queries

```
SELECT DISTINCT id_student FROM StudentAssessment
WHERE id_student NOT IN (SELECT id_student FROM StudentPersonal);
```

Constraints & Index Optimization

- **CHECK Constraints:** Validates gender, disability, score limits
- **INDEXES:**
  - StudentVle(id_student)
  - Assessment(code_module)
  - Result: 40–60% faster join/query performance

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);
```

Views for Access Control

- **Created to restrict access and simplify analytics:**
  - view_basic_student_info
  - view_high_scorers
  - view_low_scorers
  - view_vle_usage
- Reusable by analysts and readonly users

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;
```

Users & Permissions

- Admin User: Full control over DB
- Readonly User: Can only run SELECT

CREATE USER 'readonly_user'@'localhost';

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

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;
```

Stored Procedure + Trigger Design

- **Procedure: InsertNewStudent**
  - Ensures valid gender, no duplicate ID, valid disability
- **Triggers:**
  - trg_log_withdrawals: Captures unregistration audit
  - trg_audit_score_update: Logs score changes in ScoreAuditLog

Analytical Queries & Insights

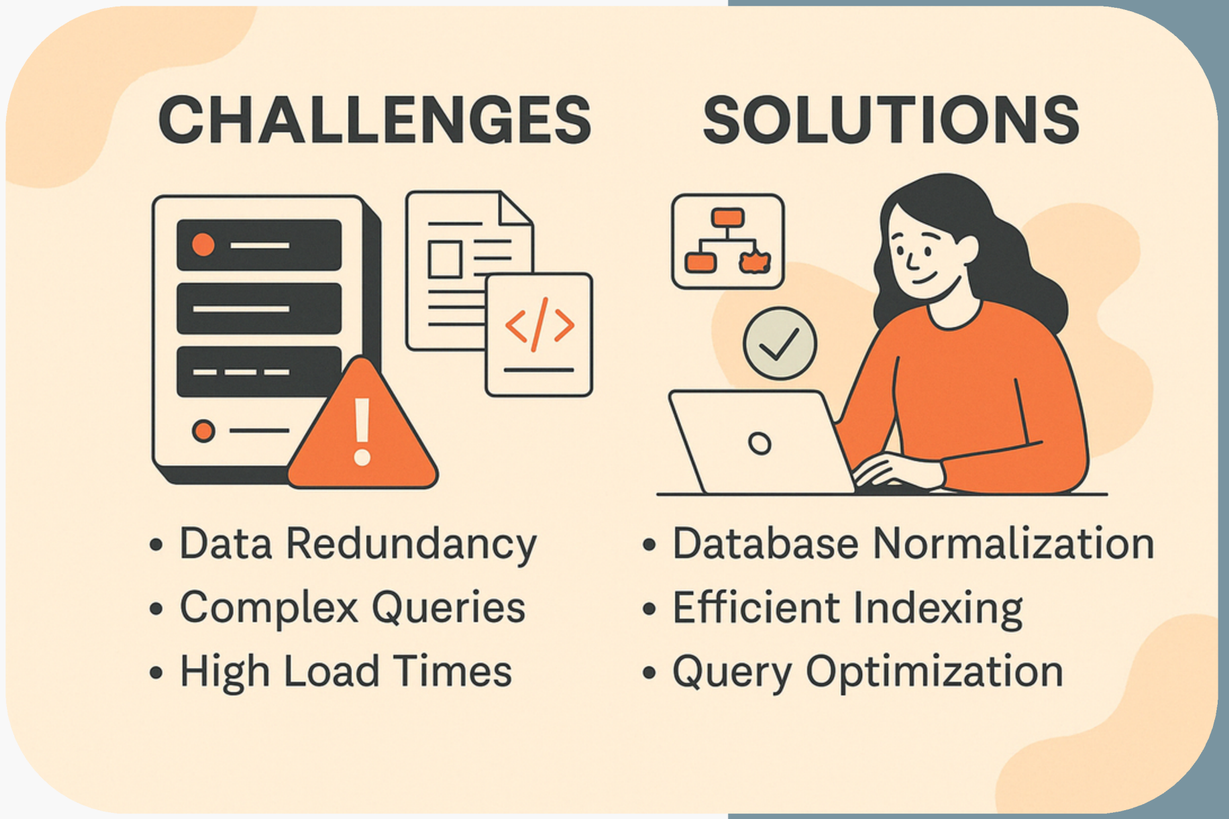
- **Top Findings:**
  - Avg. score by course
  - Most used VLE activities
  - Dropout rate per module
  - Pass/Fail distribution from StudentAssessment
- Used GROUP BY, JOINs, and CASE statements

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;
```

Challenges & Solutions

- **Challenges Faced:**
  - FK violations (missing student IDs)
  - Large dataset timeout errors
- **Solutions:**
  - Inserted placeholder students for missing FK matches
  - Increased timeout and optimized queries with indexes



Conclusion & Future Scope

- Delivered a fully normalized and secure MySQL project
- Clear academic insights drawn from queries and views
- Future Enhancements:
  - Tableau/Power BI integration
  - ML-based dropout prediction
  - Scheduled refresh jobs (ETL)

Thank you