**Project Background and Objectives**

The Operations Department of an aviation maintenance college faces persistent data fragmentation. Key records, including student attendance, instructor compliance, and exam results, are dispersed across spreadsheets and legacy systems. This lack of centralisation hinders performance tracking, complicates audit readiness, and risks non-compliance with regulatory standards. As Hogan (2018) highlights, such fragmentation impairs data quality, consistency, and usability.

To address these challenges, this project proposes a centralised relational database aligned with the college’s academic workflows. The system will provide secure data storage, structured user access, and real-time reporting, with built-in scalability and integration with tools such as Power BI and Streamlit.

**Logical Design: Entities, Relationships, and Data Types**

PostgreSQL was selected as the database engine due to the structured nature of the data. Most values fall into four predictable types: integers, strings, dates, and UUIDs. Examples include student IDs, task names, exam scores, attendance dates, and instructor statuses. A relational model supports this structure efficiently through strict schemas, constraints, and indexing (Sciore, 2007). PostgreSQL’s support for foreign keys, UUID generation, and multi-user access made it preferable to NoSQL databases, which prioritise flexibility over structure. Since this project manages tightly coupled entities such as students, instructors, and assessments, relational integrity is essential (RelationalDBDesign, n.d.).

The schema was normalised to reduce redundancy and ensure consistency. Each table represents a distinct academic or administrative process. Core entities include students, instructors, attendance records, assessment results, and exam or task definitions. Primary and foreign keys maintain relationships between these tables and support validation.

Each table serves a specific purpose:

* The **Student List** table stores identifiers, government IDs, curriculum, and class designations. It serves as the foundation for linking all other student-related records.
* **Instructor Compliance** captures status information such as licensing and hiring, ensuring that only qualified personnel are associated with student-facing tasks.
* **Student Attendance** logs attendance on specific dates and includes both student and instructor IDs.
* **Assessment Result Tables** record performance on theory exams (computer-based), practical tasks, and capstone projects, covering all assessment types the students undertake.
* **Exam Lists** define the exams the student will take during the semester, including theory assessments, practical tasks, and final capstone tasks. Information such as subtasks and the number of questions is included.

A visual representation of the database schema, including data types and relationships, is shown in Figure 1.

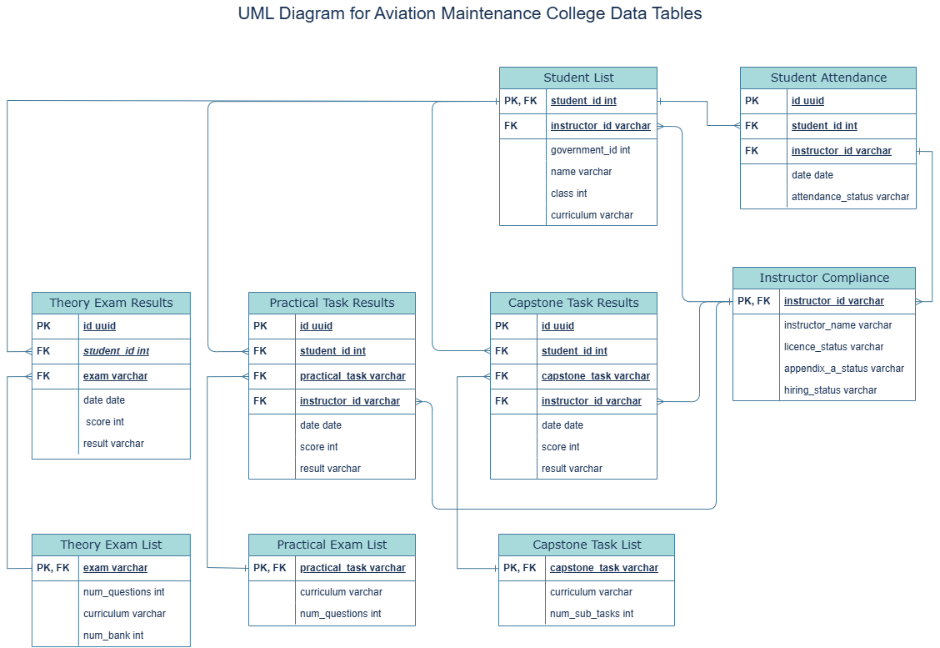


Figure 1: UML diagram showcasing the varies tables, attributes, data-types and their relationships

**Figure 1. UML Diagram for Data Tables**

This UML diagram outlines the database schema, showing key tables, their attributes, data types, and relationships. Entities such as students, instructors, attendance, and assessments are connected via primary and foreign keys to ensure consistency and traceability. Data types include VARCHAR for names and statuses, INT for numeric fields, DATE for time-based entries, and UUID for unique identifiers where required. The diagram also shows relationship arrows: single perpendicular strokes indicate one-to-one links, while forks indicate one-to-many relationships. The diagram reflects a structured, relational design supporting reliable reporting and future scalability.

**Database Platform Proposal: Supabase**

The proposed platform for deploying this database is Supabase, an open-source cloud platform built on PostgreSQL. Supabase extends PostgreSQL’s core strengths by adding modern features well suited to educational environments (Supabase, n.d.). Its built-in support for Row-Level Security (RLS) allows fine-grained access control. For example, instructors are restricted to viewing only their assigned students, while administrators can access data across their department. This supports compliance with GDPR and UK data protection legislation (GDPR.eu, n.d.; GOV.UK, n.d.).

Supabase also enables seamless integration with frontend tools. Its RESTful API endpoints allow data to be retrieved securely from tools such as Power BI, Retool, and Streamlit, empowering both technical and non-technical staff to interact with the system (Microsoft, n.d.). Its browser-based CSV upload functionality supports efficient workflows for users unfamiliar with SQL.

Supabase was the most practical choice compared to alternatives. Firebase lacks relational querying and security constraints (MIT OpenCourseWare, 2003). A self-hosted PostgreSQL instance offers more control but would require technical staffing the college does not currently have. Supabase offers a cost-effective, secure, and easy-to-deploy platform without sacrificing performance.

**Data Pipeline and Cleaning Process**

Figure 2 outlines each data table, its source system, update frequency, format, and reliability and Figure 3 shows the method of insertion of each table into the database.

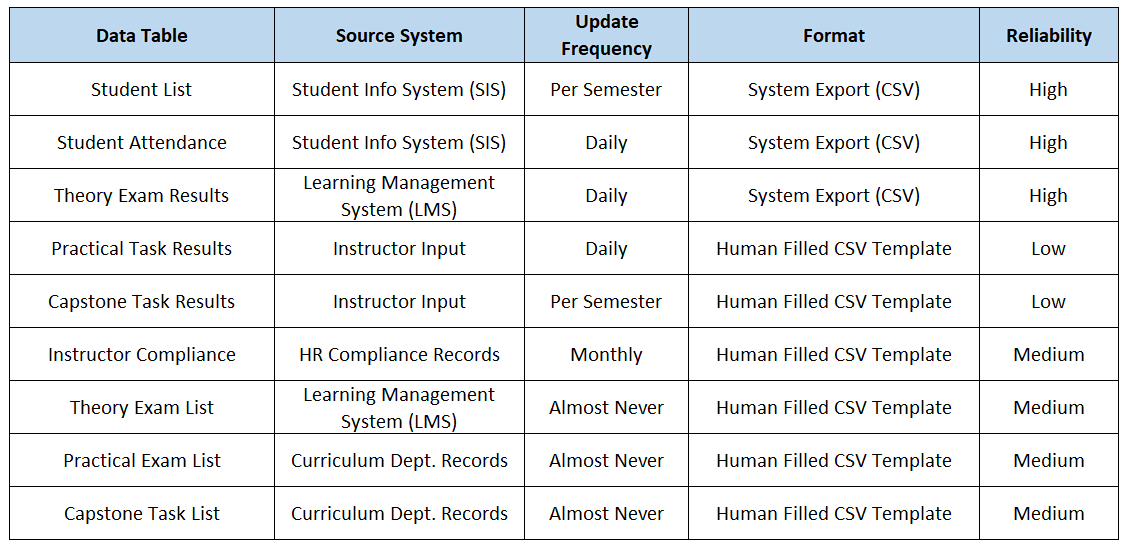


Figure 2: **Data Tables, Sources, Formats, and Reliability**.

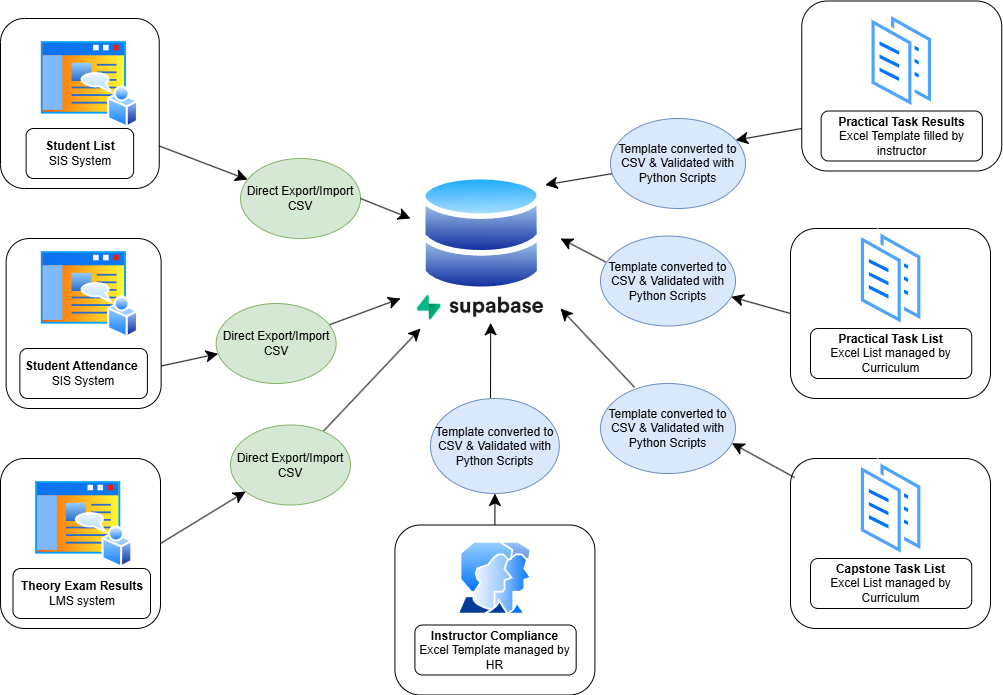


Figure 3: **Data table sources and method of Insertion into the database**

All data that will populate the database begins in CSV format. These files are sourced either from system exports or manually filled templates. This setup requires a flexible, semi-automated pipeline to ensure clean and consistent data before upload.

System exports originate from the Student Information System (SIS) and Learning Management System (LMS). These generate standardised CSV files containing the student list, attendance data, and theory exam results. Because these systems enforce strict formats, their files are reliable and are imported directly into the database.

Other tables, such as practical task results and instructor compliance, are populated using Excel templates issued to each department. These templates include:

* Dropdown menus for categorical fields
* Score and date format restrictions
* Conditional formatting to highlight missing or invalid data

Once filled, templates are saved as CSV files and uploaded to a web application built in Streamlit. A Python script using Pandas runs a set of validation checks:

* Ensures headers match the expected schema
* Verifies student and instructor IDs exist and are properly formatted
* Confirms score ranges are valid
* Checks for duplicate rows, or previous upload of the same results.

If any issues are found, the application returns the erroneous rows, allowing users to fix them. If validation passes, the user can upload the data using the psycopg library. The data is inserted into Supabase, and the upload is logged for auditing (European Data Protection Board, n.d.).

This semi-automated approach balances oversight with efficiency. Full automation was avoided for human-entered data due to the higher risk of error. As Gonçalves et al. (2023) note, integrating validation into user-facing systems helps improve data reliability and prevents downstream issues.

As the system matures, further automation can be implemented using Retool Workflows or Power Automate to handle recurring data imports and trigger alerts when anomalies are detected (Retool, n.d.; Microsoft, n.d.).

**Conclusion**

This relational database system addresses the college’s operational needs while reducing data silos. Its structure supports academic workflows, protects student data, and enables accurate performance reporting. PostgreSQL provides reliability and integrity, while Supabase simplifies access, integration, and deployment.

Initial deployment will focus on a pilot phase, refining validation scripts and onboarding users. Future iterations will improve automation and introduce stored procedures to streamline reporting. This system offers a sustainable solution for managing institutional data and preparing for future growth.

**References**

European Data Protection Board (n.d.) *Guidelines, recommendations, and best practices*. Available at: <https://edpb.europa.eu/edpb_en>

GDPR.eu (n.d.) *A guide to GDPR data privacy requirements*. Available at: <https://gdpr.eu/>

Gonçalves, C.T., Gonçalves, M.J.A. and Campante, M.I. (2023) ‘Developing integrated performance dashboards visualisations using Power BI as a platform’, *Information*, 14(11), p. 614. Available at: <https://www.mdpi.com/2078-2489/14/11/614>

GOV.UK (n.d.) *Data protection: The UK's data protection legislation*. Available at: <https://www.gov.uk/data-protection>

Hogan, R. (2018) *A Practical Guide to Database Design*. Boca Raton, FL: CRC Press.

MIT OpenCourseWare (2003) *Spatial database management and advanced geographic information systems*. Available at: <https://ocw.mit.edu/courses/11-521-spring-2003/a69668cb3f76aeb74b005dbbaa446d09_lect7.pdf>

Microsoft (n.d.) *Power BI implementation planning: Embed for your customers*. Available at: <https://learn.microsoft.com/en-us/power-bi/developer/embedded/embed-scenarios>

RelationalDBDesign (n.d.) *Relationships and referential integrity*. Available at: <https://www.relationaldbdesign.com/database-analysis/module7/relationships-referential-integrity.php>

Retool (n.d.) *Retool Workflows Documentation*. Available at: <https://docs.retool.com/docs/workflows>

Sciore, E. (2007) *Database Management: A Systems approach using JAVA*. Available at: <https://www.researchgate.net/publication/225084162_Database_Design_And_Implementation>

Streamlit (n.d.) *Streamlit Documentation*. Available at: <https://docs.streamlit.io>

Supabase (n.d.) *Supabase Documentation*. Available at: <https://supabase.com/docs>