COMS404 UNIVERSITY COURSE MANAGEMENT DATABASE

Student ID: 2003128

University of Chichester 1250-1375 word limit

DESIGN

SPECIFICATIONS AND DESIGN

GOAL

To create an online normalised database with a simple administrative web-based interface for a university course management system. The interface will provide functionality for the three main elements of the university: departments, staff and students. Information about these elements should be intuitively displayed and accessible, and modification to the database should be possible through the interface in a user-friendly way.

DATABASE TABLES

Having separate tables for each section of the system is highly pertinent in creating a normalized and cohesive relational database. There will be a table for staff, students, courses, modules, departments and individual bridge tables allowing for the creation of many-to-many mappings between tables.

Initially, staff and students were going to be a single “users” table with a user type column distinguishing between staff and students. This is a suitable solution; yet it raises issues regarding different criteria between teaching modules and being assigned modules as a student in that multiple bridge tables would have to be made between modules and users to distinguish the functions (for example for student grades which tutors would not have). Either approach would work, however for clarity the decision was made to separate students and staff into two separate tables.

All tables’ attributes have been atomised and bridge tables have been used to maintain a normalized form, so data is relevant to the table it is in (2NF) and has either a designated primary key in the form of an ID, or a combined composite key to prevent duplicate entries of data and make sure each record can be uniquely defined (3NF). For example, this allows us to know which departments a tutor belong to simply by knowing which modules they teach, without needing to directly store this information.

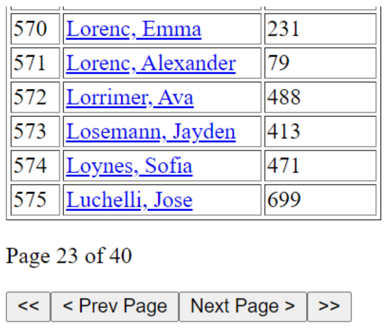
The primary keys for the most part makes use of AUTOINCREMENT (AI), as this automatically keeps the column unique by incrementing a counter every time a new record is entered - or more accurately when an attempt is made. If the query fails, the counter may still be incremented which was a notable consideration as this means some numbers would not be present as an ID. The primary key for the modules table, however, does not make use of AI and instead relies on backend PHP validation to cancel a new entry query if the generated key already exists in the database. Whilst this does have the effect that not *every* entry will be accepted on these grounds, it does allow for more complex varchar primary keys which is beneficial as having an AI primary key as well as a varchar module code stored in the database makes having both columns redundant due to them both being unique, breaking normalization.

As courses can have multiple modules, students can be assigned multiple modules and staff can teach multiple modules (and modules can have multiple students assigned, be taught by multiple teachers and belong to multiple courses), a many-to-many relationship between these tables can be setup using bridge tables. These tables leverage composite keys to establish unique records which ultimately saves having a redundant primary key column that takes up extra storage resources in the database. This makes adding and removing many-to-many relationships very easy, although it does mean you have to update multiple tables when removing students, staff, courses and especially modules. If a module were to be removed from the database, it would have to be removed from the modules table, and the three bridge tables it also occupies.

One-to-many relationships have also been set up where appropriate, for example as students can only be on one course at a time, the course ID of the course they are on can be stored in the students table as a foreign key to allow for the details of the course to be displayed on the student's profile page making use of an INNER JOIN query between the student and course tables.

WEBSITE

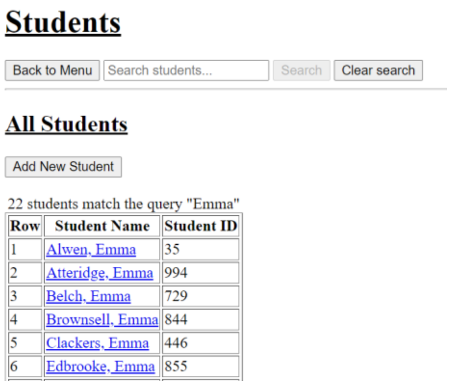
The website has been designed with usability in mind. Pagination has been used on relevant webpages to prevent too much information being on screen at once by limiting the SELECT query to only look at the relevant 25 results based on a page number and an offset. Validated search bars have also been used to make use of URL query strings to easily look through the data (Fig 1 & 2).



570	Lorenc, Emma	231
571	Lorenc, Alexander	79
572	Lorrimer, Ava	488
573	Losemann, Jayden	413
574	Loynes, Sofia	471
575	Luchelli, Jose	699

Page 23 of 40

Figure 1



Students

All Students

22 students match the query "Emma"

Row	Student Name	Student ID
1	Alwen, Emma	35
2	Atteridge, Emma	994
3	Belch, Emma	729
4	Brownsell, Emma	844
5	Clackers, Emma	446
6	Edbrooke, Emma	855

Figure 2

Furthermore, a search bar is available from the main menu that allows quick search for any of the database tables in one go. This is a very user-friendly way of searching as having a single search box for everything reduces the number of places the user would need to go to be able to search the relevant table they are trying to look through.

Restrictions on all input boxes are present to help mitigate risk of SQL injection attacks, as well as more benign misinput or invalid data being input into the text boxes when adding new students, staff, modules or courses. Only certain whitelisted characters will allow for submission of the input form to prevent invalid characters from being submitted (using JavaScript to make use of real-time client-side validation). For further future improvements, backend PHP input validation would add an extra layer of protection for malformed inputs which would significantly reduce the chances of a malicious attack causing damage to the database. Appropriate use of input devices has been made, for example number inputs for determining course length in years or date input to describe date of birth for staff and students. This vastly improves the user experience by making inputting data much easier and far less prone to errors. Whilst it does mean some characters are unusable for things like module or course names, the benefits outweigh the cons especially when considering scaling the project to a larger audience in the future where malicious attacks or multiple users will increase the chances of these concerns.

When submitting a form that requires multiple queries to be executed, transactions and prepared statements are used to greatly simply the process and make it safer. The use of transactions creates an ACID compliant database by prevents the task from being interrupted part of the way through and causing only some of the data to be written to the database. Either the whole task completes, or it is rolled back to a safe state, so no data becomes corrupted or unusable. This is shown in Fig 1:

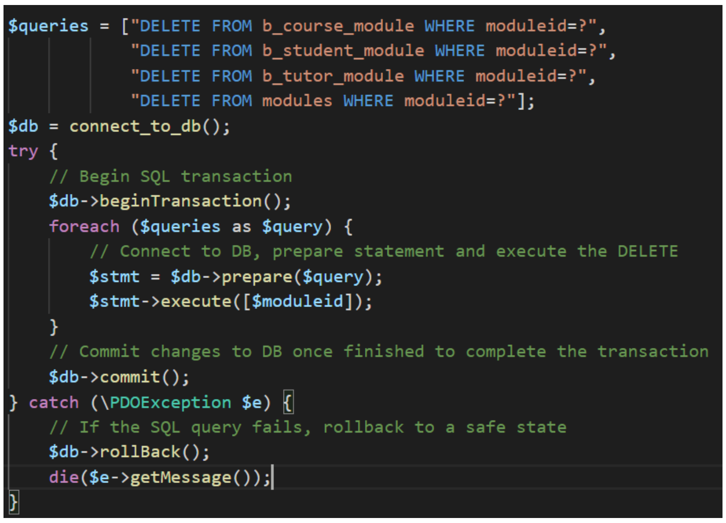

```php
$queries = ["DELETE FROM b_course_module WHERE moduleid=?",
            "DELETE FROM b_student_module WHERE moduleid=?",
            "DELETE FROM b_tutor_module WHERE moduleid=?",
            "DELETE FROM modules WHERE moduleid=?"];
$db = connect_to_db();
try {
    // Begin SQL transaction
    $db->beginTransaction();
    foreach ($queries as $query) {
        // Connect to DB, prepare statement and execute the DELETE
        $stmt = $db->prepare($query);
        $stmt->execute([$moduleid]);
    }
    // Commit changes to DB once finished to complete the transaction
    $db->commit();
} catch (\PDOException $e) {
    // If the SQL query fails, rollback to a safe state
    $db->rollBack();
    die($e->getMessage());
}
```

Figure 3: Use of transactions with SQL queries

One of the major limitations of the website currently is the lack of ability to modify existing data, such as changing the name of a student or staff member. This would not be difficult to implement in the future, either by switching to the coms404-template package for the provided functionality or by writing it using the current website design. This functionality would be very beneficial when it comes to human error as currently if you make a mistake with a new entry (such as misspelling a new staff member's name), you would need to delete the whole entry and redo it as a new entry.

EVALUATION

Overall, the database and web-based interface successfully fulfil the given specification, whilst using good database design practices and providing useful functionality for the end user. Further functionality can be added easily in the future if necessary to provide more useful features and interaction with the database and the scalability potential is entirely available.