暨南大學
JINAN UNIVERSITY

# Course Paper for Undergraduate Students

| | |
|---|---|
| **Course Type:** | Subject Compulsory |
| **Course Name:** | Database Systems<br>数据库系统(全英) |
| **Course Code:** | 60080020 |
| **Instructor:** | 吴汉瑞，Hanrui Wu |

## 《 Course Paper for Database Systems 》

**Name** _____

**Student ID** _____

**College** International School

**Major** CST

**Email** _____

Submit Date:

## 1. Course Content Summary

The primary objective of the database system course is to apply the knowledge and techniques learned to develop and design database applications, addressing the requirements of information processing. Over this semester, I have gained a solid understanding of database-related concepts, extending to the underlying implementation principles, and enhanced this knowledge through practical application in two projects.

The course began with an introduction to various database concepts, including the evolution of data management technology, the background of database technology development, and fundamental data model concepts. We explored the components and main data models, as well as the three-level schema structure and composition of database systems. This was followed by a detailed examination of relational databases, covering relational data structures, operations, integrity constraints, and relational algebra.

A significant focus was placed on practical SQL language skills, encompassing Data Query Language (DQL), Data Definition Language (DDL), and Data Manipulation Language (DML). Since SQL is pivotal in realizing the main functions of relational database systems, its study provided deeper insights into relational database fundamentals.

However, mastering basic table operations such as addition, deletion, modification, and querying was not the end goal. The course progressed to advanced database aspects, including table structure analysis, table constraints, and more complex technologies like virtual table views, transactions, and table joins. A key tenet was that practice is the ultimate test of truth. After learning basic SQL applications, we dived into how to connect to a database using Java, namely JDBC, culminating in the completion of Project 1 (my project is Electricity Billing System), which integrated our theoretical knowledge into a practical context.

Further, we delved into relational database theory, addressing logical design, potential issues, and data dependence basics. This included learning about paradigms, the Armstrong axiom system, and understanding various types of data dependencies like functional, trivial, non-trivial, partial, complete, and transitive dependencies. We also covered concepts such as keys, candidate keys, foreign keys, multi-valued dependencies, and the determination methods of 1NF, 2NF, 3NF, BCNF, and 4NF.

The course also shed light on data storage and database indexing, explaining the B+ tree data structure as a prelude to database optimization. We then studied query and optimization techniques, with a focus on using index optimization to significantly enhance query efficiency. Toward the end of the course, we deepened our understanding of transactions, exploring their characteristics and the challenges and solutions of concurrent transactions.At the same time, we also completed Project 2 (a Teaching Administration Management system focusing on course selection functions), which also deepened my understanding of database knowledge.

In conclusion, this semester's study of databases was comprehensive and multifaceted, covering a broad spectrum of theoretical and practical aspects. The course not only imparted essential database knowledge but also emphasized the application of this knowledge in real-world scenarios, preparing us for advanced database studies and applications in various fields.

## 2. Project Flashing Points Sharing

While building Project 2 (Courses Selection System), some highlights that I thought were worth sharing are as follows:

### 2.1. Database design

The database architecture for this project is comprised of six interconnected tables that collectively support the functionalities of student information management, course cataloging, enrollment records, and authentication processes.

The following ER diagram and the database table relationship diagram visually represent the structure and relationships within the database, offering a clear and concise understanding of how data is interconnected in the system.

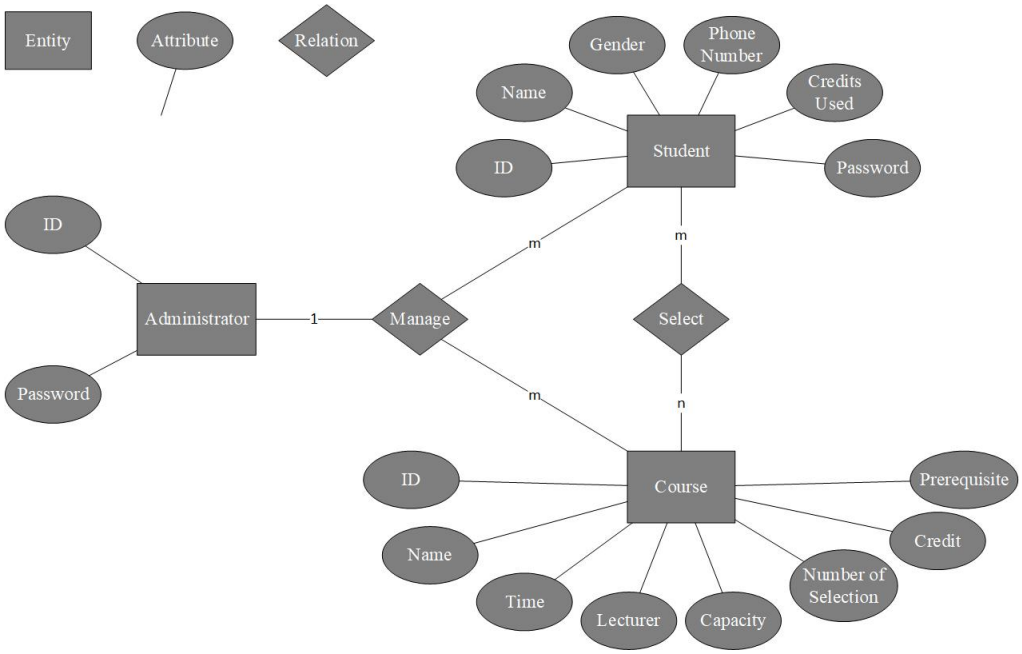


**Figure 1:** System E-R Diagram

These tables are thoughtfully designed with relationships and constraints to ensure data integrity and accuracy. The student and course tables are linked through the course selection and completed courses tables, forming a many-to-many relationship that allows for recording and querying student enrollments and course completions. Additionally, the login tables secure system access.

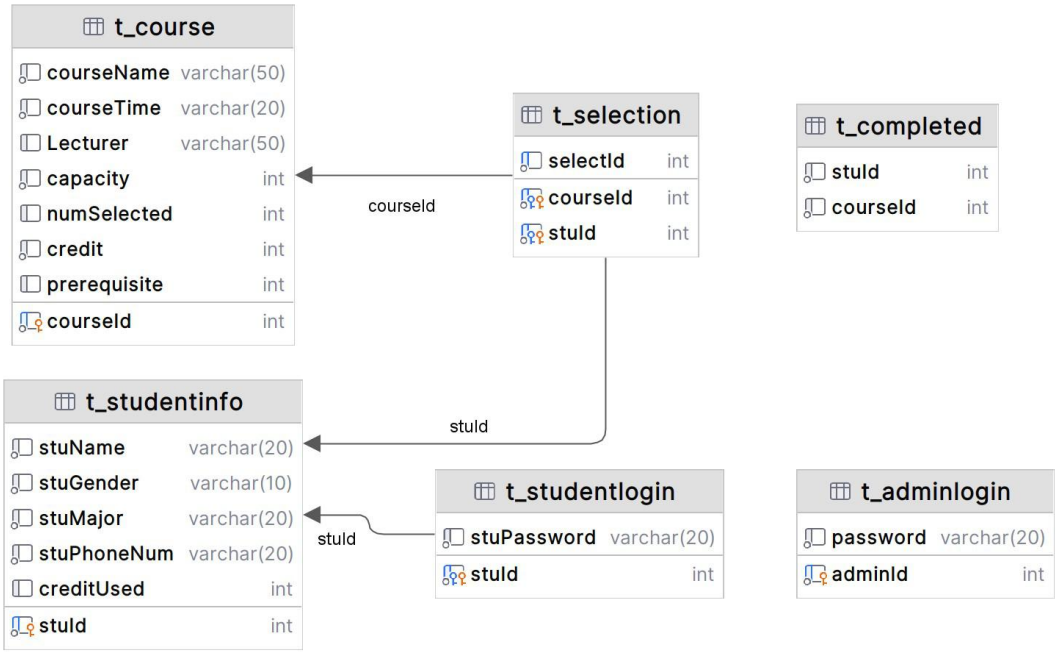


**Figure 2:** Relationship Between Database Tables

The detailed table structures are as follows:

**(1)** "t_studentinfo" (Student Information Table): It records essential details of students such as student ID (stuId), name (stuName), gender (stuGender), major (stuMajor), phone number (stuPhoneNum), and credits used (creditUsed).
**(2)** "t_course" (Course Table): This table includes comprehensive details of courses offered, including course ID (courseId), course name (courseName), course time (courseTime), lecturer (Lecturer), class capacity (capacity), the number of students enrolled (numSelected), credit value (credit), and prerequisite course ID (prerequisite).
**(3)** "t_selection" (Course Selection Table): Representing the many-to-many relationship between students and courses, this table logs the instances of students selecting particular courses, with fields including selection record ID (selectId), course ID (courseId), and student ID (stuId).
**(4)** "t_completed" (Completed Courses Table): This table tracks the courses a student has completed, with fields including student ID (stuId) and course ID (courseId).
**(5)** "t_studentlogin" (Student Login Table): Managing the authentication credentials of students, it contains the student ID (stuId) and corresponding password (stuPassword).
**(6)** "t_adminlogin" (Administrator Login Table): This maintains the login details for administrators, comprising administrator ID (adminId) and password (password).

Incorporated into the database design are triggers that automate the updating and maintenance of related data, such as automatically adjusting the credits used by students. This not only enhances the efficiency of backend code but also ensures the timeliness and accuracy of data updates.

```
DELIMITER $$
CREATE TRIGGER increment_num_selected
    AFTER INSERT ON t_selection FOR EACH ROW
BEGIN
    UPDATE t_course SET numSelected = numSelected + 1
    WHERE courseId = NEW.courseId;
END$$
DELIMITER ;
```

SQL script for trigger construction

Overall, this database design addresses the needs of an educational management system, achieving an efficient organization and management of data structures, providing robust data support, and ensuring an excellent user experience.

### 2.2. Project Design Pattern

In the construction of Project 2, one of the key architectural decisions that stands out is the use of a three-tier architecture. This design pattern significantly streamlines the interaction between the user interface and the database backend, ensuring a clean separation of concerns.

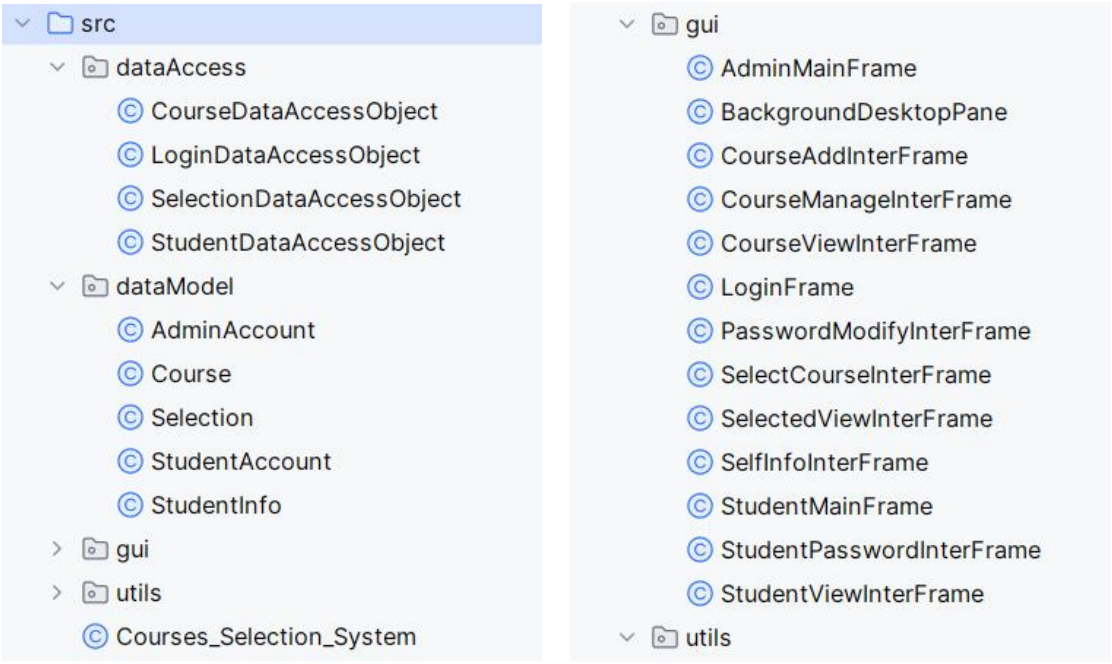


**Figure 3:** Project Code Structure

- **Presentation Layer (GUI):** The graphical user interface is built with user experience in mind, containing frames like "LoginFrame" and "AdminMainFrame" that allow for intuitive navigation and interaction within the application. This layer is strictly for presentation and user interaction, isolated from the business logic and data access code.
- **Business Logic Layer (Data Model):** This layer, encapsulated within the "dataModel" package, defines the system's domain model with classes such as "Course", "StudentInfo", and "Selection". It acts as the intermediary between the

user-facing GUI and the data-centric operations, enforcing business rules and ensuring data integrity.

- **Data Access Layer (DAO):** The "dataAccess" package clearly demonstrates the implementation of the Data Access Object pattern. By abstracting the database access into specific objects like "CourseDataAccessObject", the system promotes loose coupling and high cohesion. This layer handles all interactions with the database, providing a unified interface for CRUD operations while hiding the complexities of direct database manipulation from the other layers.

By employing this architectural pattern, the Courses Selection System ensures that changes in the database schema or the business logic have minimal impact on the user interface, and vice versa. This pattern not only simplifies the maintenance and scalability of the application but also enhances the development process by allowing work on different layers simultaneously without causing conflicts.

### 2.3. Application and Analysis of SQL Statement

When constructing and applying Java methods and their underlying SQL statements in the course selection system, you need to pay special attention to their data operation logic. The following are two representative examples:

The "CreditSum" method calculates the student's total credits by joining the "t_course" and "t_selection" tables and summing the "credit" field. The difficulty lies in accurately aggregating these credits while managing potential concurrent modifications as students add or remove courses. And the relational algebra of this operation is: **Πcredit(σstuId=S (t_course ⋈ t_selection))**

This represents the projection of the "credit" attribute after the selection and join of the "t_course" and "t_selection" tables, where "stuId" is equal to "S", so we can construct an SQL statement in the project method as follows.

```
public int CreditSum(Connection con, int stuId) throws Exception {
    String sql = "select ifnull(sum(c.credit),0) from t_course c,
CoursesSelectionDB.t_selection s where s.stuId=? and s.courseId=c.courseId";
    int sum = 0;
    PreparedStatement pstmt = con.prepareStatement(sql);
    pstmt.setInt(1, stuId);
    ResultSet rs = pstmt.executeQuery();
    while (rs.next()) {
        sum = rs.getInt(1);
    }
    return sum;
}
```

CreditSum Method using PreparedStatement

The "OccTime" method is designed to detect scheduling conflicts by identifying courses with matching "courseTime" entries for a given student. The challenge here is to perform this check efficiently, requiring well-optimized query and indexing

strategies. The relational algebra of this statement is: **σstuId=S ∧ courseTime=T (t_course ⋈ t_selection)**

This represents a selection operation on the natural join between "t_course" and "t_selection", where "stuId" is "S" and "courseTime" is "T", so we can construct an SQL statement as follows:

```
public int OccTime(Connection con, int stuId, String courseTime) throws Exception {
    int occ = 0;
    String sql = "SELECT c.courseId FROM t_course c INNER JOIN t_selection s ON
c.courseId = s.courseId WHERE s.stuId=? AND c.courseTime=?";
    PreparedStatement pstmt = con.prepareStatement(sql);
    pstmt.setInt(1, stuId);
    pstmt.setString(2, courseTime);
    ResultSet rs = pstmt.executeQuery();
    while (rs.next()) {
        occ = 1;
    }
    return occ;
}
```

OccTime Method using PreparedStatement

The system relies on set-based operations succinctly represented by relational algebra. Ensuring that these operations perform optimally under concurrent loads requires strong transaction processing and strict adherence to atomicity, consistency, isolation and durability properties.

### 3. Interesting Selected Topic Discussion

In this section, I delve deeper into a common topic of interest in database-related interviews - the concept of database indexing, with a special emphasis on understanding the underlying mechanics and choices involved.

Database indexing is akin to an efficient filing system within a database table, enabling rapid access to specific data. Consider an index as a streamlined path to find a particular piece of information, much faster than searching every individual row in a table. An apt illustration is how an index can expedite finding an employee's record by searching with their last name.

The core of an index's efficiency lies in its underlying design and data structure. Hash tables, while excellent for quick data retrieval with a time complexity of O(1), fall short in handling range queries effectively. For instance, a query like "*SELECT * FROM user WHERE id > 2;*" becomes problematic with hash indexes due to their inefficiency in managing range searches.

This inadequacy of hash indexes for MySQL leads to exploring other data structures. Binary Search Trees (BST) offer faster search capabilities and can accommodate range searches, which hash indexes cannot. However, BSTs come with a significant

limitation: they can degrade into a linear list in extreme cases, thereby drastically impacting search performance.

The discussion then extends to Red-Black Trees and AVL Trees. Red-Black Trees maintain good average search efficiency but can become unbalanced with sequential insertions, leading to skewed performance. AVL Trees, while avoiding this imbalance, are impractical for MySQL indexing due to their intensive disk IO operations.

The focus then shifts to the choice of B and B+ Trees for MySQL indexing. These tree structures are particularly advantageous because they store a substantial number of indexes in each node. This design reduces the height of the tree, consequently decreasing the required disk IO operations.

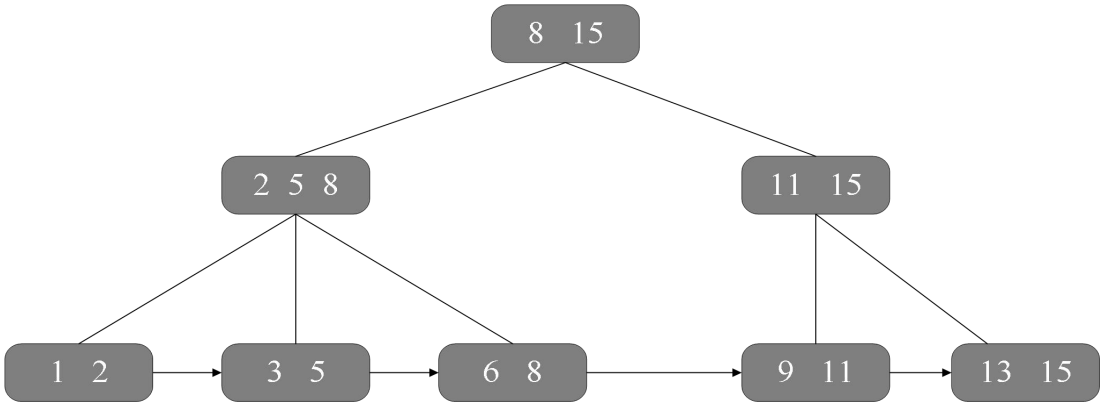


**Figure 4:** Conventional Structure of B+ Tree

Expanding on B+ Trees, they stand out in their architecture and functionality. B+ Trees maintain a balance between the depth of the tree and the number of records each node can hold, optimizing both search efficiency and storage utilization. In a B+ Tree, all records are stored at the leaf nodes, which are interconnected in a linked list format. This arrangement ensures that range searches are highly efficient. For instance, in a range query, the tree navigates to the start of the range at the leaf level and then simply traverses the linked list to cover the entire range. This method significantly outperforms other data structures in scenarios where range queries are frequent.

Moreover, the non-leaf nodes in a B+ Tree act as a guide to these leaf nodes, containing only keys and pointers, which further enhances the search speed for individual records. The balanced nature of B+ Trees ensures that the path from the root to any leaf node is of the same length, providing consistent and predictable performance.

In summary, MySQL's adoption of B+ Trees for indexing is a strategic choice, leveraging their superior capabilities in handling both point queries and range searches efficiently. This in-depth understanding of B+ Trees not only illuminates the rationale behind their use in database indexing but also underscores the importance of aligning data structures with specific database needs. Future lectures could further dissect these concepts, offering students a more nuanced and comprehensive grasp of database indexing strategies.