# WMS - Use Cases and Main Features

## Student Use Cases (5)

### 1. Login

Students authenticate using their assigned credentials to access the system dashboard and enrollment tools.

---

### 2. Add Course to Cart

Students can search for and select courses before enrolling.

#### 2.1 Filter Courses by Department

- Select a department to narrow down available courses
- System displays only courses belonging to the chosen department

#### 2.2 View Courses as a Dropdown

- View **all courses** or **filtered courses** in a dropdown list
- Dropdown shows course code + course title for quick scanning

#### 2.3 View Course Details

Students can check core information before adding a course to their cart:

- Instructor
- Venue
- Timings
- Credit Hours (CHs)
- Remaining Seats

---

### 3. Validate Courses

The system checks:

- Time clashes
- Prerequisite requirements
- Credit hour limits
- Duplicate course selections
- Seat cap

Students must resolve validation errors before enrolling.

---

## 4. Enroll

Enrollment finalizes the selected cart.

Includes the **course validation** step automatically before enrollment is confirmed.

---

## 5. View Calendar-Based Schedule

After enrolling, students can view their complete weekly schedule in a **calendar format** inside the dashboard.

## 6. Drop a course

A student can drop a course.

---

# Admin Use Cases (5)

## 1. Add Student

Admins can create student profiles and assign academic attributes (refer to Std.Student entity).

### 1.1 Assign School

- SDSB
- SSE

- HSS
- SAHSOL

### 1.2 Assign Department

- e.g., CS, MGMT, HIST, ECON, MATH, etc.
- Department must belong to the selected school

---

## 2. Add Instructor

Admins can register new instructors and associate them with academic units.

### 2.1 Assign School

- SDSB, SSE, HSS, SAHSOL

### 2.2 Assign Department

- Instructors are tied to one or more departments within their school

---

## 3. Add Course

Admins create new course entries with full configuration options (refer to Course.Course entity).

### 3.1 Assign Instructor

- Select an instructor responsible for teaching the course

### 3.2 Assign Department

- Department determines which courses are returned by a filter

### 3.3 Set Course Cap

- Maximum number of students allowed to enroll

### 3.4 Set Prerequisite Courses

- Define which courses must be completed before enrolling

---

## 4. Force Enroll Student

Admins may override constraints to enroll a student manually, such as:

- Full course capacity

---

## 5. View Summary Statistics (Dashboard)

Admins have a dashboard showing key system-level metrics:

- Students by school and department
- Instructors by department
- Total number of courses
- System-wide enrollment counts and trends

---

# Main Features (8)

## Student Features

### 1. Filtering System (Student)

Students can filter courses by **department** to easily find relevant classes.

Filtered results update in real time in dropdowns and course lists.

### 2. Course Validation (Triggered During Enrollment)

When students attempt to enroll, the system validates:

- Prerequisite completion
- Time clashes
- Remaining seats
- Minimum & maximum credit hour limits

Validation is automatic and required before enrollment proceeds.

### 3. Calendar Schedule View

Students can view their complete weekly schedule in a **calendar format**, showing:

- Course timings
- Venues
- Instructor names

### 4. Waitlist Management (System / Background)

A background system feature that manages course waitlists automatically.

When a student drops a course that was previously full, the system checks for an existing waitlist.

- The next student in line is **automatically enrolled**, *or*

**Purpose:**

Enables automated seat redistribution and is well-suited for implementation via:

- **Database triggers**
- **Stored procedures**
- **Scheduled processes** (if needed)

---

## Admin Features

### 1. Filtering System (Admin)

Admins can filter:

- Instructors by department or school
- Courses by department
- Students by school or department (if needed)

This supports faster data management and cleaner dropdowns.

### 2. Course Validation Engine

A central validation system admins rely on when overseeing or modifying enrollments.

Checks include:

1. Prerequisites completed
2. Time clash detection
3. Seat availability

4. Credit hour limits

## 3. Course Configuration Tools

Admins can fully configure any course by setting:

- Instructor assignment
- Course cap
- Prerequisite courses

These settings define how courses appear to students.

## 4. Force Enrollment

Admins may override all rules to enroll a student manually—for cases such as:

- Full capacity courses
- Special academic approvals
- Administrative exceptions

## 5. Summary Statistics Dashboard

Admins have access to a system dashboard showing:

- Students by school and department
- Instructors by department
- Total courses
- Enrollment counts and other metrics

## 6. Audit Sensitive Changes

Tracks and logs all high-impact administrative actions.

**Description:**

Whenever an admin performs a sensitive operation—such as:

- Using **Force Enroll** to override system rules
- Manually changing course **capacity**, **grades**, or other protected fields

...the system must automatically record **who** performed the action and **when** it occurred.

**Purpose:**

Provides traceability and accountability. Ideal for implementation through:

- **AFTER triggers** on relevant tables
- Audit log tables
- Immutable history records

---

# Feature Mapping Matrix

## 1. Stored Procedures (Min: 2)

*Rule: Use these for transactional operations that involve multiple steps (validation + insert).*

- **Feature 1:** `sp_EnrollStudent`

  **Use Case:** Student Enrollment.

  - **Logic:**
    1. Input: `StudentID`, `CourseID`.
    2. Check if `RemainingSeats > 0`.
    3. Check `fn_DetectTimeClash` (see Functions below).
    4. Check `fn_CheckPrerequisites` (see Functions below).
    5. If all pass, `INSERT` into Enrollment table.
    6. `UPDATE` Course table to decrement seats.

- **Feature 2:** `sp_ForceEnroll`

  **Use Case:** Admin Force Enroll.

  - **Logic:** Accepts Admin credentials. Bypasses the checks in `sp_EnrollStudent`, inserts the record directly, and logs the action into an Audit table.

## 2. User-Defined Functions (Min: 2)

*Rule: Use these for calculations or reusable boolean checks.*

- **Feature 1 (Scalar):** `fn_CalculateGPA`
  - **Use Case:** Student Dashboard / Analytics.

- **Logic:** Input `StudentID`. Sums `(GradePoints * Credits) / TotalCredits` and returns the decimal value.

- **Feature 2 (Scalar/Boolean):** `fn_CheckPrerequisites`

  **Use Case:** Validation.

  - **Logic:** Returns `TRUE` if the student has passed all required courses for the target course, `FALSE` otherwise.

## 3. Triggers (Min: 2 types)

*Rule: Automatic reactions to table events.*

- **Feature 1 (AFTER INSERT):** `trg_AuditForceEnrollment`
  - **Use Case:** Audit Sensitive Changes (New).
  - **Logic:** Watch the `Enrollment` table. If a row is inserted where `IsForced = 1` (or similar flag), insert a record into a separate `AuditLog` table with the timestamp and UserID.
- **Feature 2 (INSTEAD OF DELETE):** `trg_SoftDeleteCourse`

  **Use Case:** Course Management.

  - **Logic:** If an Admin tries to `DELETE` a course that has students enrolled, the trigger intercepts it. Instead of deleting (which breaks integrity), it sets an `IsActive` flag to `0` (Soft Delete).

## 4. Common Table Expressions (CTEs) (Min: 2)

*Rule: Complex temporary datasets.*

- **Feature 1 (Recursive CTE):** `cte_PrerequisitePath`
  - **Use Case:** View Full Prerequisite Tree (New).
  - **Logic:** Since prerequisites can be nested (A requires B, B requires C), use a recursive CTE to walk up the tree and list *all* courses a student needs to take before the target course.
- **Feature 2 (Standard CTE):** `cte_StudentScheduleClashes`

  **Use Case:** Validate Courses (Time Clashes).

  - **Logic:** Before enrolling, create a CTE that selects the time slots of all *currently* enrolled courses for that student. Join this CTE with the *target* course's time slot to check for overlaps.

## 5. Views (Min: 2)

*Rule: Security and simplifying complex joins for the frontend.*

- **Feature 1:** `vw_AvailableCourses`

  **Use Case:** Student Course Browsing.

  - **Logic:** Join `Course`, `Instructor`, and `Department`. Filter out courses where `IsActive = 0` or `Semester != Current`. This ensures students never see archived data.

- **Feature 2:** `vw_AdminDashboardStats`

  **Use Case:** Admin Summary Statistics.

  - **Logic:** A complex view that pre-aggregates data: Total Students per Dept, Average Class Size, and Most Popular Courses. This saves the frontend from calculating this every time.

## 6. Indexes (Min: 2 types)

*Rule: Optimization for the "1 Million Rows" requirement.*

- **Feature 1 (Non-Clustered): Index on** `Enrollment(StudentID)`
  - **Use Case:** Viewing Schedule/History.
  - **Logic:** You will likely query `WHERE StudentID = 'X'` thousands of times.
- **Feature 2 (Filtered Index): Index on** `Course(RemainingSeats) WHERE RemainingSeats > 0`
  - **Use Case:** Filtering for open courses.
  - **Logic:** Speeds up queries looking for courses that aren't full yet.

## 7. Table Partitioning (Min: 1)

*Rule: Managing large datasets.*

- **Feature:** Partition `Enrollment` table by `AcademicYear`.

**Use Case:** System Scalability.

- **Logic:** Since you need 1 Million rows, assume 20 years of history. Create partitions for `2020`, `2021`, `2022`, etc. Queries for the current semester will only scan the relevant partition, making the system faster.