SQL BASICS

Objective

- To design and implement a relational database for managing students, instructors, courses, and enrollments.
- To practice SQL operations including filtering, sorting, aliases, joins, grouping, subqueries, and schema modifications.
- To apply Data Definition Language (DDL) and Data Manipulation Language (DML) commands in database management.

Pre-requisite

- Basic knowledge of relational database concepts such as tables, keys, and relationships.
- Familiarity with SQL commands like CREATE, INSERT, SELECT, UPDATE, DELETE, and ALTER.
- Access to MySQL Workbench (or a similar SQL environment) for creating and executing queries.
- Understanding of schema design and normalization principles.

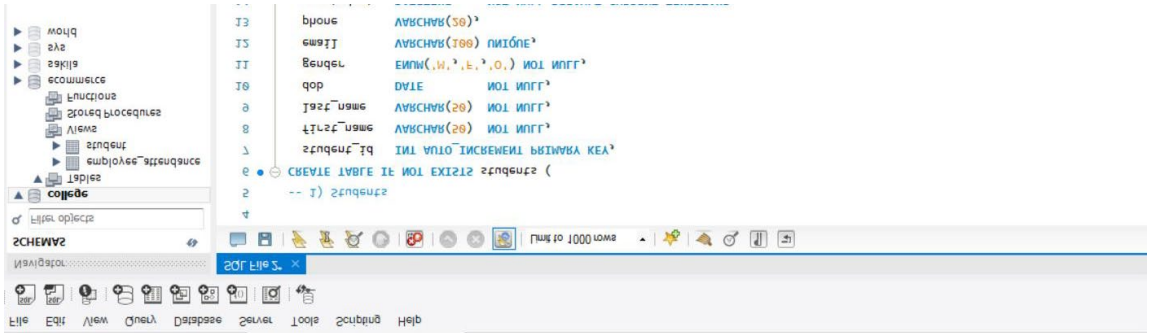
Procedure:

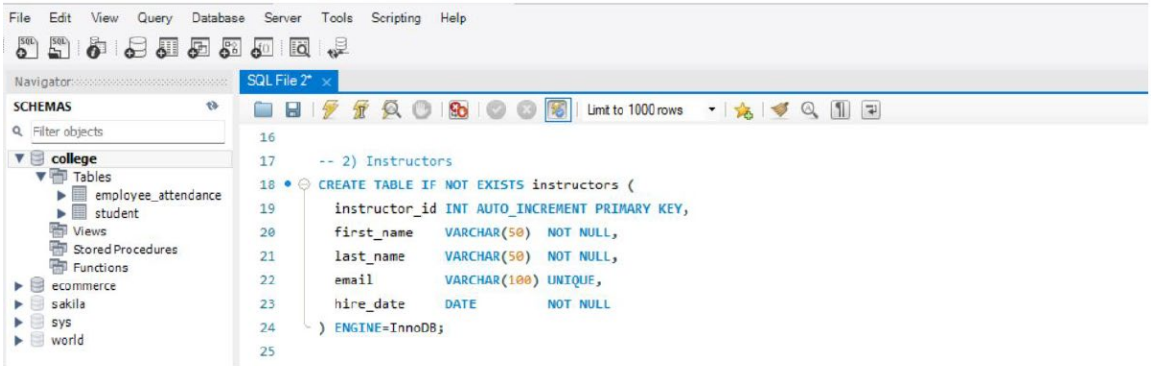
STEP 1: Create the 4 core tables in schema college

In Workbench, open a new SQL tab (File ➜ New Query Tab or Ctrl+T).

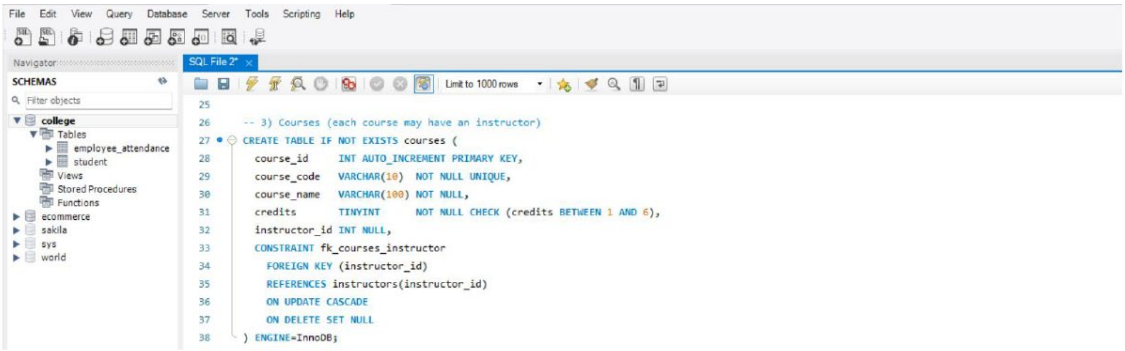
Make sure **college** is the default schema (it' s bold in the left panel). If not, right-click **college**

➜ **Set as Default Schema**.

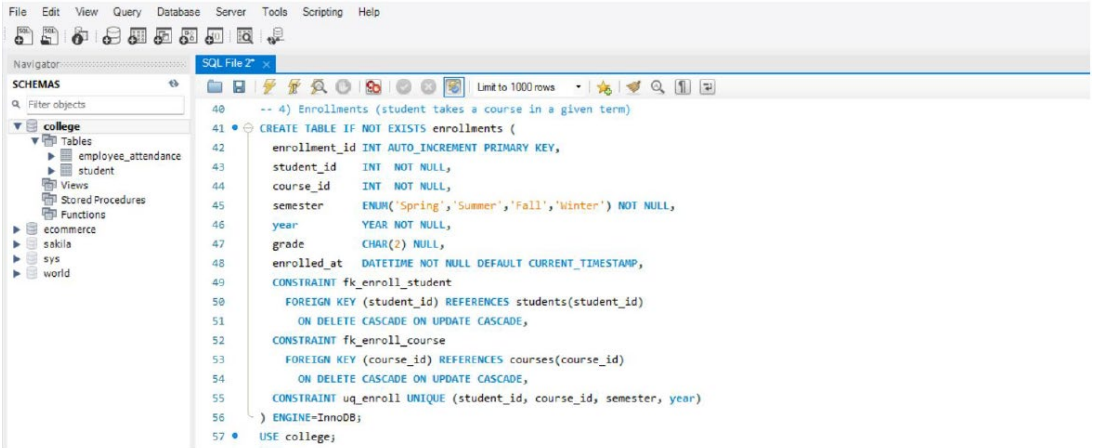




```
-- 2) Instructors
CREATE TABLE IF NOT EXISTS instructors (
  instructor_id INT AUTO_INCREMENT PRIMARY KEY,
  first_name    VARCHAR(50)  NOT NULL,
  last_name     VARCHAR(50)  NOT NULL,
  email         VARCHAR(100) UNIQUE,
  hire_date     DATE         NOT NULL
) ENGINE=InnoDB;
```



```
-- 3) Courses (each course may have an instructor)
CREATE TABLE IF NOT EXISTS courses (
  course_id     INT AUTO_INCREMENT PRIMARY KEY,
  course_code   VARCHAR(10)  NOT NULL UNIQUE,
  course_name   VARCHAR(100) NOT NULL,
  credits       TINYINT      NOT NULL CHECK (credits BETWEEN 1 AND 6),
  instructor_id INT NULL,
  CONSTRAINT fk_courses_instructor
    FOREIGN KEY (instructor_id)
    REFERENCES instructors(instructor_id)
    ON UPDATE CASCADE
    ON DELETE SET NULL
) ENGINE=InnoDB;
```



```
-- 4) Enrollments (student takes a course in a given term)
CREATE TABLE IF NOT EXISTS enrollments (
  enrollment_id INT AUTO_INCREMENT PRIMARY KEY,
  student_id    INT  NOT NULL,
  course_id     INT  NOT NULL,
  semester      ENUM('Spring','Summer','Fall','Winter') NOT NULL,
  year          YEAR NOT NULL,
  grade         CHAR(2) NULL,
  enrolled_at   DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP,
  CONSTRAINT fk_enroll_student
    FOREIGN KEY (student_id) REFERENCES students(student_id)
      ON DELETE CASCADE ON UPDATE CASCADE,
  CONSTRAINT fk_enroll_course
    FOREIGN KEY (course_id) REFERENCES courses(course_id)
      ON DELETE CASCADE ON UPDATE CASCADE,
  CONSTRAINT uq_enroll UNIQUE (student_id, course_id, semester, year)
) ENGINE=InnoDB;
USE college;
```

STEP 2 : Insert Sample Data

```
INSERT INTO instructors (first_name, last_name, email, hire_date)
VALUES
('John', 'Smith', 'jsmith@college.edu', '2015-08-20'),
('Emily', 'Johnson', 'ejohnson@college.edu', '2017-02-10'),
('Michael', 'Brown', 'mbrown@college.edu', '2018-06-01');
SELECT * FROM instructors;
-- Insert Courses again
INSERT INTO courses (course_code, course_name, credits, instructor_id)
VALUES
('CS101', 'Intro to Computer Science', 4, 1),
('MATH201', 'Calculus I', 3, 2),
('ENG150', 'English Literature', 2, 3);
SELECT * FROM courses;
INSERT INTO enrollments (student_id, course_id, semester, year, grade)
VALUES
(1, 1, 'Fall', 2023, 'A'),   -- Alice → CS101
(1, 2, 'Fall', 2023, 'B'),   -- Alice → MATH201
(2, 1, 'Fall', 2023, 'B'),   -- Bob → CS101
(3, 3, 'Fall', 2023, 'A');   -- Charlie → ENG150
SELECT * FROM enrollments;
```

SELECT

SELECT * FROM students;

```
select * from student;
```

Result Grid

student_id	first_name	last_name	dob	gender	email	phone	created_at
1	Alice	Williams	2001-05-14	F	alice.w@college.edu	1234567890	2025-08-25 21:04:21
2	Bob	Davis	2000-09-30	M	bob.d@college.edu	9876543210	2025-08-25 21:04:21
3	Charlie	Miller	2002-01-22	M	charlie.m@college.edu	5554443333	2025-08-25 21:04:21
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

SELECT * FROM courses;

```
select * from student;
```

Result Grid

course_id	course_code	course_name	credits	instructor_id
1	CS101	Intro to Computer Science	4	1
2	MATH201	Calculus I	3	2
3	ENG150	English Literature	2	3
NULL	NULL	NULL	NULL	NULL

FILTERING (WHERE):

```
SELECT first_name, last_name, gender
FROM students
WHERE gender = 'F';
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

first_name	last_name	gender
Alice	Williams	F

```
SELECT course_name, credits
FROM courses
WHERE credits > 3;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

course_name	credits
Intro to Computer Science	4

SORTING (ORDER BY):

Students by DOB

```
SELECT first_name, last_name, dob
FROM students
ORDER BY dob ASC;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

first_name	last_name	dob
Bob	Davis	2000-09-30
Alice	Williams	2001-05-14
Charlie	Miller	2002-01-22

Instructors by Hire Date

```
SELECT first_name, last_name, hire_date
FROM instructors
ORDER BY hire_date DESC;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

first_name	last_name	hire_date
Michael	Brown	2018-06-01
Emily	Johnson	2017-02-10
John	Smith	2015-08-20

ALIASES:

Instructors with column aliases :

```
SELECT first_name AS "First", last_name AS "Last", email AS "Email Address"
FROM instructors;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

First	Last	Email Address
John	Smith	jsmith@college.edu
Emily	Johnson	ejohnson@college.edu
Michael	Brown	mbrown@college.edu

Student full names with alias :

```
SELECT CONCAT(first_name, ' ', last_name) AS "Student Name", email
FROM students;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Student Name	email
Alice Williams	alice.w@college.edu
Bob Davis	bob.d@college.edu
Charlie Miller	charlie.m@college.edu

COUNT (How many students are enrolled in each course):

```
SELECT c.course_name, COUNT(e.student_id) AS total_students
FROM courses c
JOIN enrollments e ON c.course_id = e.course_id
GROUP BY c.course_name;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

course_name	total_students
Intro to Computer Science	2
Calculus I	1
English Literature	1

AVG (Average grade per course):

```sql
SELECT c.course_name,
       AVG(
         CASE e.grade
           WHEN 'A' THEN 4
           WHEN 'B' THEN 3
           ELSE NULL
         END
       ) AS avg_grade_points
FROM courses c
JOIN enrollments e ON c.course_id = e.course_id
GROUP BY c.course_name;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

course_name	avg_grade_points
Intro to Computer Science	3.5000
Calculus I	3.0000
English Literature	4.0000

JOIN Courses + Instructors (Who teaches which course):

```sql
SELECT c.course_code, c.course_name,
       i.first_name AS instructor_first,
       i.last_name AS instructor_last
FROM courses c
JOIN instructors i ON c.instructor_id = i.instructor_id;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

course_code	course_name	instructor_first	instructor_last
CS101	Intro to Computer Science	John	Smith
MATH201	Calculus I	Emily	Johnson
ENG150	English Literature	Michael	Brown

GROUP BY with HAVING (Only courses with more than 1 student):

```sql
SELECT c.course_name, COUNT(e.student_id) AS total_students
FROM courses c
JOIN enrollments e ON c.course_id = e.course_id
GROUP BY c.course_name
HAVING COUNT(e.student_id) > 1;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

course_name	total_students
Intro to Computer Science	2

Sub query (Find students enrolled in “Intro to Computer Science”):

```
SELECT first_name, last_name
FROM students
WHERE student_id IN (
  SELECT student_id
  FROM enrollments
  WHERE course_id = (
    SELECT course_id
    FROM courses
    WHERE course_name = 'Intro to Computer Science'
  )
);
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

first_name	last_name
Alice	Williams
Bob	Davis

Sub query with Aggregate (Find student(s) with highest grade points):

```
SELECT s.first_name, s.last_name
FROM students s
WHERE s.student_id IN (
  SELECT e.student_id
  FROM enrollments e
  WHERE CASE e.grade
          WHEN 'A' THEN 4
          WHEN 'B' THEN 3
          ELSE 0
        END = (
          SELECT MAX(
            CASE grade
              WHEN 'A' THEN 4
              WHEN 'B' THEN 3
              ELSE 0
            END
          )
          FROM enrollments
        )
);
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

first_name	last_name
Alice	Williams
Charlie	Miller

UPDATE (Change a grade):

```
UPDATE enrollments
SET grade = 'A'
WHERE student_id = 2 AND course_id = 1;

SELECT * FROM enrollments WHERE student_id = 2;
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

enrollment_id	student_id	course_id	semester	year	grade	enrolled_at
7	2	1	Fall	2023	A	2025-08-25 21:09:10
NULL	NULL	NULL	NULL	NULL	NULL	NULL

DELETE (Remove a student enrollment):

```
DELETE FROM enrollments
WHERE student_id = 3 AND course_id = 3;

SELECT * FROM enrollments;
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

enrollment_id	student_id	course_id	semester	year	grade	enrolled_at
5	1	1	Fall	2023	A	2025-08-25 21:09:10
6	1	2	Fall	2023	B	2025-08-25 21:09:10
7	2	1	Fall	2023	A	2025-08-25 21:09:10
NULL	NULL	NULL	NULL	NULL	NULL	NULL

enrollments 26 × Apply

ALTER TABLE (Add a new column):

```
ALTER TABLE students
ADD gpa DECIMAL(3,2) NULL;

-- Update sample GPA values
UPDATE students
SET gpa = CASE student_id
            WHEN 1 THEN 3.6
            WHEN 2 THEN 3.2
            WHEN 3 THEN 3.8
          END;

SELECT * FROM students;
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

student_id	first_name	last_name	dob	gender	email	phone	created_at	gpa
1	Alice	Williams	2001-05-14	F	alice.w@college.edu	1234567890	2025-08-25 21:04:21	NULL
2	Bob	Davis	2000-09-30	M	bob.d@college.edu	9876543210	2025-08-25 21:04:21	NULL
3	Charlie	Miller	2002-01-22	M	charlie.m@college.edu	5554443333	2025-08-25 21:04:21	NULL
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Conclusion:

In this lab, a College Management System database was successfully designed and implemented using SQL. Core tables were created for students, instructors, courses, and enrollments with appropriate relationships. Sample data was inserted, and multiple queries were executed to perform filtering, sorting, aggregation, joins, and subqueries. Additionally, schema modifications were carried out using ALTER, UPDATE, and DELETE operations. This exercise provided practical experience in database design and query execution, building a strong foundation for real-world applications in academic systems.