Azure SQL Lab Manual

A Step-by-Step Guide to Learning SQL with Azure SQL Database

Table of Contents

Introduction

This lab manual provides a step-by-step guide to learning SQL using Azure SQL Database. It covers basic to advanced SQL concepts, including:

- SQL Basics: SELECT, INSERT, UPDATE, DELETE
- Joins: INNER JOIN, LEFT JOIN, RIGHT JOIN, FULL JOIN
- Aggregate Functions: COUNT, SUM, AVG, MIN, MAX
- Advanced Topics: Subqueries, Window Functions

By following this guide, you will gain hands-on experience with SQL and Azure SQL Database, enabling you to build and query relational databases effectively.

Lab Setup

1. Create an Azure SQL Database:
   - Log in to the Azure Portal.
   - Navigate to "SQL Databases" and click "Create."
   - Fill in the required details (e.g., resource group, server name, database name).
   - Click "Review + Create" and then "Create."

2. Connect to the Database:
   - Use Azure Cloud Shell or a local SQL client (e.g., sqlcmd, Azure Data Studio).
   - Run the following command to connect:

```bash
sqlcmd -S my-sql-server.database.windows.net -U myadmin -P MyStrongP@ssword
```

```
PS /home/awwal> $serverName = "my-sql-server.database.windows.net"
PS /home/awwal> $databaseName = "PracticeDB"
PS /home/awwal> $username = "myadmin"
PS /home/awwal> $password = "MyStrongP@ssword"
PS /home/awwal> $connectionString = "Server=$serverName; Database=$databaseName; User ID=$username; Pass
word=$password;"
PS /home/awwal>
```

3. Create Tables:

- Run the SQL scripts provided in the Appendix to create the Students, Courses, and Enrollments tables.

```
PS /home/awwal> Invoke-SqlCmd -ConnectionString $connectionString -Query "SELECT name FROM sys.schemas


name
----
dbo
guest
INFORMATION_SCHEMA
sys
SchoolSchema
db_owner
db_accessadmin
db_securityadmin
db_ddladmin
db_backupoperator
db_datareader
db_datawriter
db_denydatareader
db_denydatawriter

PS /home/awwal>
```

The `Students` table will store information about students.

Command to Create the `Students` Table:

powershell Copy

```powershell
Invoke-SqlCmd -ConnectionString $connectionString -Query @"
CREATE TABLE SchoolSchema.Students (
    StudentID INT PRIMARY KEY IDENTITY(1,1), -- Auto-incrementing primar
y key
    FirstName NVARCHAR(50) NOT NULL,
    LastName NVARCHAR(50) NOT NULL,
    Age INT NOT NULL,
    Grade CHAR(1) NOT NULL,
    CONSTRAINT CHK_Students_Age CHECK (Age >= 16 AND Age <= 30), -- Stud
ents must be between 16 and 30
    CONSTRAINT CHK_Students_Grade CHECK (Grade IN ('A', 'B', 'C', 'D',
'F')) -- Only valid grades
);
"@
```

The `Courses` table will store information about courses.

Command to Create the `Courses` Table:

```powershell
Invoke-SqlCmd -ConnectionString $connectionString -Query @"
CREATE TABLE SchoolSchema.Courses (
    CourseID INT PRIMARY KEY IDENTITY(1,1), -- Auto-incrementing primary
key
    CourseName NVARCHAR(100) NOT NULL,
    Credits INT NOT NULL,
    CONSTRAINT CHK_Courses_Credits CHECK (Credits > 0) -- Credits must b
e positive
);
"@
```

The `Enrollments` table will link students to courses.

Command to Create the `Enrollments` Table:

```powershell
Invoke-SqlCmd -ConnectionString $connectionString -Query @"
CREATE TABLE SchoolSchema.Enrollments (
    EnrollmentID INT PRIMARY KEY IDENTITY(1,1), -- Auto-incrementing pri
mary key
    StudentID INT NOT NULL,
    CourseID INT NOT NULL,
    EnrollmentDate DATE NOT NULL DEFAULT GETDATE(), -- Defaults to the c
urrent date
    FOREIGN KEY (StudentID) REFERENCES SchoolSchema.Students(StudentID),
    FOREIGN KEY (CourseID) REFERENCES SchoolSchema.Courses(CourseID)
);
"@
```

4. Insert Sample Data:

- Run the SQL scripts provided in the Appendix to insert sample data into the tables.

1. Insert Data into `Students`:

```powershell
Invoke-SqlCmd -ConnectionString $connectionString -Query @"
INSERT INTO SchoolSchema.Students (FirstName, LastName, Age, Grade)
VALUES ('John', 'Doe', 20, 'A'),
       ('Jane', 'Smith', 22, 'B'),
       ('Alice', 'Johnson', 21, 'C'),
       ('Bob', 'Brown', 19, 'A'),
       ('Emily', 'Davis', 23, 'B'),
       ('Michael', 'Wilson', 24, 'A'),
       ('Sarah', 'Taylor', 20, 'C'),
       ('David', 'Anderson', 22, 'B');
"@
```

2. Insert Data into `Courses`:

```powershell
Invoke-SqlCmd -ConnectionString $connectionString -Query @"
INSERT INTO SchoolSchema.Courses (CourseName, Credits)
VALUES ('Mathematics', 3),
       ('Physics', 4),
       ('Chemistry', 3),
       ('Biology', 4),
       ('Computer Science', 3),
       ('History', 2),
       ('Literature', 2),
       ('Economics', 3);
"@
```

3. Insert Data into `Enrollments`:

```powershell
Invoke-SqlCmd -ConnectionString $connectionString -Query @"
INSERT INTO SchoolSchema.Enrollments (StudentID, CourseID, EnrollmentDat
e)
VALUES (1, 1, '2024-01-15'), -- John enrolled in Mathematics
       (1, 2, '2024-01-16'), -- John enrolled in Physics
       (2, 3, '2024-01-17'), -- Jane enrolled in Chemistry
       (3, 4, '2024-01-18'), -- Alice enrolled in Biology
       (4, 5, '2024-01-19'), -- Bob enrolled in Computer Science
       (5, 6, '2024-01-20'), -- Emily enrolled in History
       (6, 7, '2024-01-21'), -- Michael enrolled in Literature
       (7, 8, '2024-01-22'), -- Sarah enrolled in Economics
       (8, 1, '2024-01-23'); -- David enrolled in Mathematics
"@
```

Verify the Tables and Data

1. Check the `Students` table:

```powershell
Invoke-SqlCmd -ConnectionString $connectionString -Query "SELECT * F
ROM SchoolSchema.Students;"
```

2. Check the `Courses` table:

```powershell
Invoke-SqlCmd -ConnectionString $connectionString -Query "SELECT * F
ROM SchoolSchema.Courses;"
```

3. Check the `Enrollments` table:

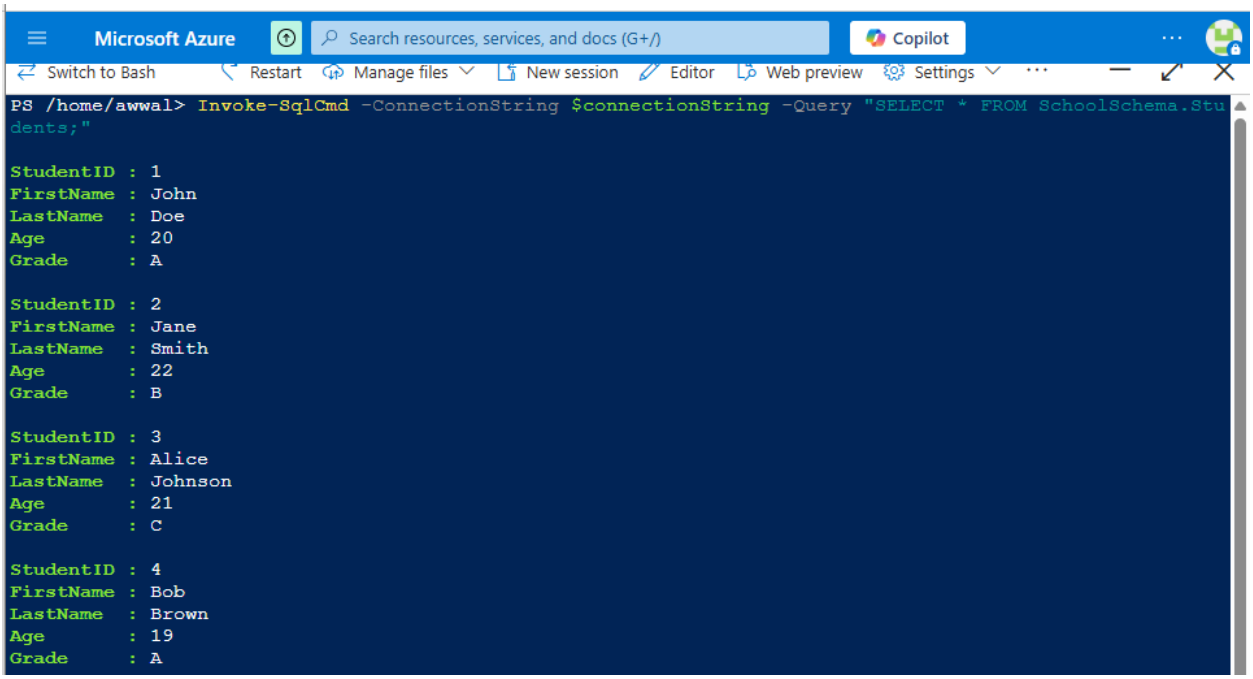
```powershell
Invoke-SqlCmd -ConnectionString $connectionString -Query "SELECT * F
ROM SchoolSchema.Enrollments;"
```

SQL Basics

1. SELECT:

- Retrieve all students:

```sql
SELECT * FROM SchoolSchema.Students;
```



2. INSERT:

- Add a new student:

```sql
INSERT INTO SchoolSchema.Students (FirstName, LastName, Age, Grade)
VALUES ('Emma', 'Watson', 22, 'A');
```

3. UPDATE:

- Update a student's grade:

```sql
UPDATE SchoolSchema.Students
SET Grade = 'B'
WHERE FirstName = 'Emma' AND LastName = 'Watson';
```

4. DELETE:

- Delete a student:

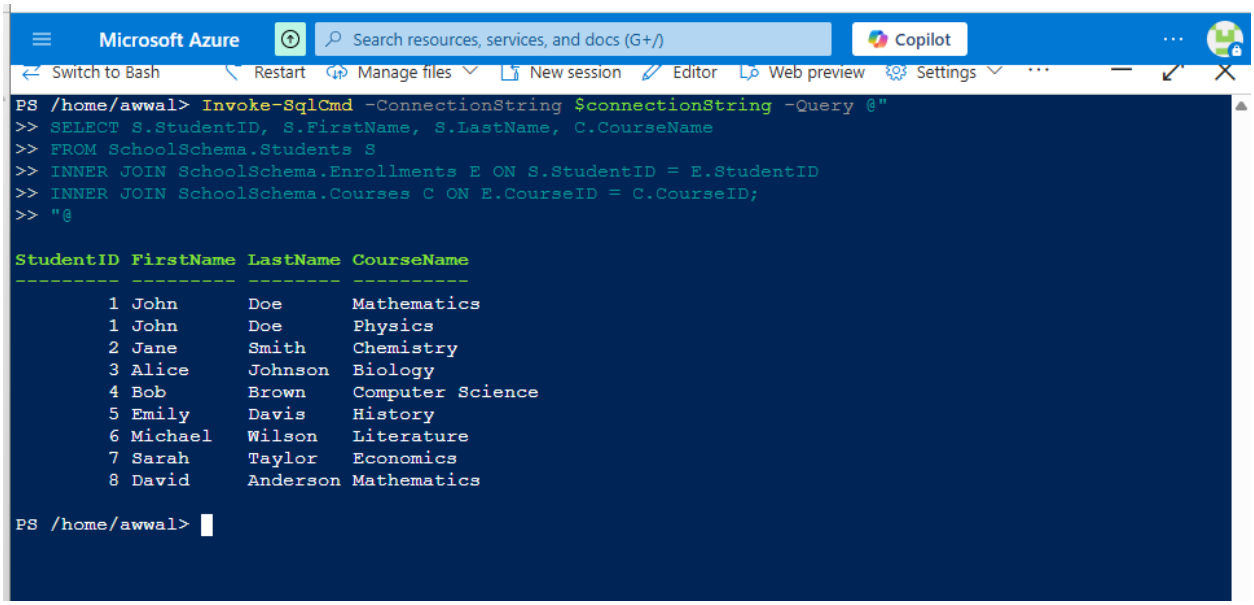
```sql
DELETE FROM SchoolSchema.Students
WHERE FirstName = 'Emma' AND LastName = 'Watson';
```

Joins

1. INNER JOIN:

- Retrieve students and their enrolled courses:

```sql
SELECT S.StudentID, S.FirstName, S.LastName, C.CourseName
FROM SchoolSchema.Students S
INNER JOIN SchoolSchema.Enrollments E ON S.StudentID = E.StudentID
INNER JOIN SchoolSchema.Courses C ON E.CourseID = C.CourseID;
```



2. LEFT JOIN:

- Retrieve all students and their enrolled courses (including students without enrollments):

```sql
SELECT S.StudentID, S.FirstName, S.LastName, C.CourseName
FROM SchoolSchema.Students S
LEFT JOIN SchoolSchema.Enrollments E ON S.StudentID = E.StudentID
LEFT JOIN SchoolSchema.Courses C ON E.CourseID = C.CourseID;
```

```
PS /home/awwal> Invoke-SqlCmd -ConnectionString $connectionString -Query @"
>> SELECT S.StudentID, S.FirstName, S.LastName, C.CourseName
>> FROM SchoolSchema.Students S
>> LEFT JOIN SchoolSchema.Enrollments E ON S.StudentID = E.StudentID
>> LEFT JOIN SchoolSchema.Courses C ON E.CourseID = C.CourseID;
>> "@

StudentID FirstName LastName  CourseName
--------- --------- --------  ----------
        1 John      Doe       Mathematics
        1 John      Doe       Physics
        2 Jane      Smith     Chemistry
        3 Alice     Johnson   Biology
        4 Bob       Brown     Computer Science
        5 Emily     Davis     History
        6 Michael   Wilson    Literature
        7 Sarah     Taylor    Economics
        8 David     Anderson  Mathematics
        9 Emma      Watson
       10 Daniel    Radcliffe
       12 Emma      Watson
```

3. RIGHT JOIN:

- Retrieve all courses and enrolled students (including courses without enrollments):

```sql
SELECT C.CourseName, S.FirstName, S.LastName
FROM SchoolSchema.Students S
RIGHT JOIN SchoolSchema.Enrollments E ON S.StudentID = E.StudentID
RIGHT JOIN SchoolSchema.Courses C ON E.CourseID = C.CourseID;
```

```
PS /home/awwal> Invoke-SqlCmd -ConnectionString $connectionString -Query @"
>> SELECT C.CourseName, S.FirstName, S.LastName
>> FROM SchoolSchema.Students S
>> RIGHT JOIN SchoolSchema.Enrollments E ON S.StudentID = E.StudentID
>> RIGHT JOIN SchoolSchema.Courses C ON E.CourseID = C.CourseID;
>> "@

CourseName       FirstName LastName
----------       --------- --------
Mathematics      John      Doe
Mathematics      David     Anderson
Physics          John      Doe
Chemistry        Jane      Smith
Biology          Alice     Johnson
Computer Science Bob       Brown
History          Emily     Davis
Literature       Michael   Wilson
Economics        Sarah     Taylor
```

4. FULL JOIN:

- Retrieve all students and all courses (including unmatched rows):

```sql
SELECT S.StudentID, S.FirstName, S.LastName, C.CourseName
FROM SchoolSchema.Students S
FULL JOIN SchoolSchema.Enrollments E ON S.StudentID = E.StudentID
FULL JOIN SchoolSchema.Courses C ON E.CourseID = C.CourseID;
```

```
Switch to Bash   Restart   Manage files   New session   Editor   Web preview   Settings
PS /home/awwal> Invoke-SqlCmd -ConnectionString $connectionString -Query @"
>> SELECT S.StudentID, S.FirstName, S.LastName, C.CourseName
>> FROM SchoolSchema.Students S
>> FULL JOIN SchoolSchema.Enrollments E ON S.StudentID = E.StudentID
>> FULL JOIN SchoolSchema.Courses C ON E.CourseID = C.CourseID;
>> "@

StudentID FirstName LastName  CourseName
--------- --------- --------  ----------
        1 John      Doe       Mathematics
        1 John      Doe       Physics
        2 Jane      Smith     Chemistry
        3 Alice     Johnson   Biology
        4 Bob       Brown     Computer Science
        5 Emily     Davis     History
        6 Michael   Wilson    Literature
        7 Sarah     Taylor    Economics
        8 David     Anderson  Mathematics
        9 Emma      Watson
       10 Daniel    Radcliffe
       12 Emma      Watson
```

Aggregate Functions

1. COUNT:

- Count the total number of students:

```sql
SELECT COUNT(*) AS TotalStudents FROM SchoolSchema.Students;
```

```
PS /home/awwal> Invoke-SqlCmd -ConnectionString $connectionString -Query "SELECT COUNT(*) AS TotalStuden
ts FROM SchoolSchema.Students;"

TotalStudents
-------------
           11

PS /home/awwal>
```

2. SUM:

- Calculate the total credits of all courses:

```sql
SELECT SUM(Credits) AS TotalCredits FROM SchoolSchema.Courses;
```

```
PS /home/awwal> Invoke-SqlCmd -ConnectionString $connectionString -Query "SELECT SUM(Credits) AS TotalCr
edits FROM SchoolSchema.Courses;"

TotalCredits
------------
          24
```

3. AVG:

- Calculate the average age of students:

```sql
SELECT AVG(Age) AS AverageAge FROM SchoolSchema.Students;
```

```
PS /home/awwal> Invoke-SqlCmd -ConnectionString $connectionString -Query "SELECT AVG(Age) AS AverageAge
FROM SchoolSchema.Students;"

AverageAge
----------
        21
```

4. MIN:

- Find the minimum age of students:

```sql
SELECT MIN(Age) AS MinAge FROM SchoolSchema.Students;
```

```
Switch to Bash   Restart   Manage files   New session   Editor   Web preview   Settings
PS /home/awwal> Invoke-SqlCmd -ConnectionString $connectionString -Query "SELECT MIN(Age) AS MinAge FROM
 SchoolSchema.Students;"

MinAge
------
    19

PS /home/awwal>
```

5. MAX:

- Find the maximum age of students:

```sql
SELECT MAX(Age) AS MaxAge FROM SchoolSchema.Students;
```

```
PS /home/awwal> Invoke-SqlCmd -ConnectionString $connectionString -Query "SELECT MAX(Age) AS MaxAge FROM
 SchoolSchema.Students;"

MaxAge
------
    24
```

Advanced Topics

1. Subqueries:

- Retrieve students older than the average age:

```sql
SELECT * FROM SchoolSchema.Students
WHERE Age > (SELECT AVG(Age) FROM SchoolSchema.Students);
```

```
PS /home/awwal> Invoke-SqlCmd -ConnectionString $connectionString -Query @"
>> SELECT FirstName, LastName, Age
>> FROM SchoolSchema.Students
>> WHERE Age > (SELECT AVG(Age) FROM SchoolSchema.Students);
>> "@

FirstName LastName  Age
--------- --------  ---
Jane      Smith      22
Emily     Davis      23
Michael   Wilson     24
David     Anderson   22
Emma      Watson     22
Daniel    Radcliffe  24
Emma      Watson     22
```

2. Window Functions:

- Rank students by age:

```sql
SELECT StudentID, FirstName, LastName, Age,
    RANK() OVER (ORDER BY Age DESC) AS AgeRank
```

FROM SchoolSchema.Students;

```
```

```
PS /home/awwal> Invoke-SqlCmd -ConnectionString $connectionString -Query @"
>> SELECT StudentID, FirstName, LastName, Age,
>>        ROW_NUMBER() OVER (ORDER BY Age DESC) AS AgeRank
>> FROM SchoolSchema.Students;
>> "@

StudentID : 6
FirstName : Michael
LastName  : Wilson
Age       : 24
AgeRank   : 1

StudentID : 10
FirstName : Daniel
LastName  : Radcliffe
Age       : 24
AgeRank   : 2

StudentID : 5
FirstName : Emily
LastName  : Davis
Age       : 23
AgeRank   : 3

StudentID : 2
FirstName : Jane
LastName  : Smith
Age       : 22
AgeRank   : 4
```

Conclusion

This lab manual has provided a comprehensive guide to learning SQL using Azure SQL Database. By following the step-by-step instructions and examples, you have gained hands-on experience with:

- SQL Basics: SELECT, INSERT, UPDATE, DELETE

- Joins: INNER JOIN, LEFT JOIN, RIGHT JOIN, FULL JOIN

- Aggregate Functions: COUNT, SUM, AVG, MIN, MAX

- Advanced Topics: Subqueries, Window Functions

Appendix

1. SQL Scripts:

- Create Tables:

```sql
CREATE TABLE SchoolSchema.Students (
    StudentID INT PRIMARY KEY IDENTITY(1,1),
    FirstName NVARCHAR(50) NOT NULL,
    LastName NVARCHAR(50) NOT NULL,
    Age INT NOT NULL,
    Grade CHAR(1) NOT NULL
);
```

- Insert Data:

```sql
INSERT INTO SchoolSchema.Students (FirstName, LastName, Age, Grade)
VALUES ('John', 'Doe', 20, 'A');
```

2. Screenshots and Diagrams:

- Entity-Relationship (ER) Diagram: [Insert ER Diagram]
- Query Results: [Insert Screenshots].