

42.1 Programming languages

Programming languages

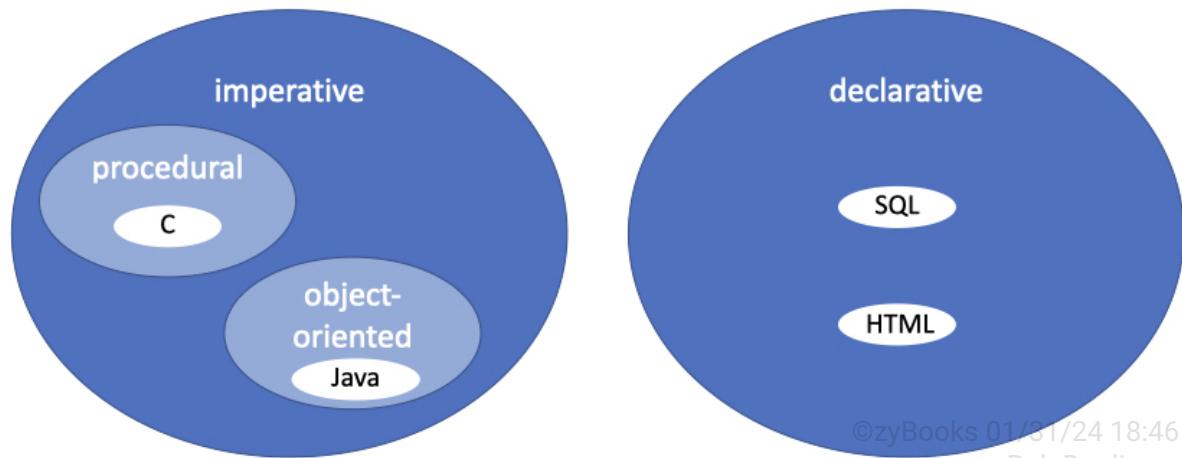
Programming languages fall into two broad categories, or paradigms: imperative and declarative.

Imperative languages contain control flow statements that determine the execution order of program steps. Control flow statements include loops for repeatedly executing code and conditionals for conditionally executing code. Two popular types of imperative languages are:

1. **Procedural languages** are composed of procedures, also called functions or subroutines. Most languages developed prior to 1990 are procedural. Ex: C and COBOL.
2. **Object-oriented languages** organize code into classes. A class combines variables and procedures into a single construct. Most languages developed since 1990 are object-oriented. Ex: Java, Python, and C++.

Declarative languages do not contain control flow statements. Each statement declares *what* result is desired, using logical expressions, rather than *how* the result is processed. SQL and HTML are leading examples of declarative languages.

Figure 42.1.1: Programming languages.



©zyBooks 01/31/24 18:46 1939727
Rob Daglio
MDCCOP2335Spring2024

Programming language categories

The purpose of this section is to provide context for database programming, not a complete classification of programming languages. A complete classification might include additional categories, such as functional programming and logic programming.

PARTICIPATION ACTIVITY**42.1.1: Programming languages.**

©zyBooks 01/31/24 18:46 1939727

Rob Daglio
MDCCOP2335Spring2024

- 1) _____ languages combine data and procedures into classes.

Check**Show answer**

- 2) _____ languages do not explicitly specify how results are processed.

Check**Show answer**

- 3) _____ languages contain control flow statements but no classes.

Check**Show answer**

- 4) Programming languages are either _____ or declarative.

Check**Show answer**

©zyBooks 01/31/24 18:46 1939727

Rob Daglio
MDCCOP2335Spring2024

Database programming

Declarative language is commonly used for database queries for two reasons:

- *Easier programming.* With procedural language, complex queries are coded with loops and conditional statements. With declarative language, complex queries are coded in just one statement.
- *Faster execution.* With procedural language, a programmer specifies how to process a query. With declarative language, the optimizer determines how to process a query. Since complex queries can be processed in many ways, optimized queries usually execute faster than programmed queries.

©zyBooks 01/31/24 18:46 1939727

Rob Daglio

MDCCOP2335Spring2024

For the reasons above, as well as early backing by IBM, SQL has become the dominant query language.

Complete applications need control flow statements as well as database queries. Since SQL has no control flow statements, complete applications combine SQL with a procedural or object-oriented language. Building applications with both SQL and a general-purpose language is called **database programming**.

Database programming presents two challenges:

- *Syntax gap.* The syntax of SQL and most other languages are quite different. Ex: Groups of statements, called blocks, are fundamental to most procedural and object-oriented languages. SQL has no blocks.
- *Paradigm gap.* Query processing is fundamentally different in SQL and procedural or object-oriented languages. Ex: A result set with many rows is processed in a single SQL statement, but usually requires loops and conditional statements in other languages.

Database programming must bridge the syntax and paradigm gaps.

PARTICIPATION ACTIVITY

42.1.2: Syntax and paradigm gaps.



Declarative language

```
SELECT AirlineName, FlightNumber
FROM Flight
WHERE AirportCode = 'JFK' AND DepartureTime > '12:00';
```

©zyBooks 01/31/24 18:46 1939727

Rob Daglio

MDCCOP2335Spring2024

Procedural language

```
printf("%20s %12s \n", "AirlineName ", "FlightNumber");
for (int i = 1; i <= rowCount(flight); i++) {
    if (strcmp(getAirportCode(i), "JFK") == 0 && getDepartureTime(i) >
12)
        printf("%20s %12d \n", getAirlineName(i), getFlightNumber(i));
}
```

AirlineName	FlightNumber
American Airlines	1154

Air India
Lufthansa852
920

Animation content:

Static figure:

©zyBooks 01/31/24 18:46 1939727

Rob Daglio

MDCCOP2335Spring2024

A SELECT statement appears with caption declarative language:

Begin SQL code:

```
SELECT AirlineName, FlightNumber
```

```
FROM Flight
```

```
WHERE AirportCode = 'JFK' AND DepartureTime > '12:00';
```

```
End SQL code.
```

An equivalent C code fragment appears with caption procedural language:

Begin C code:

```
printf("%20s %12s \n", "AirlineName ", "FlightNumber");
for (int i = 1; i <= rowCount(flight); i++) {
    if (strcmp(getAirportCode(i), "JFK") == 0 || getDepartureTime(i) > 12)
        printf("%20s %12d \n", getAirlineName(i), getFlightNumber(i));
}
```

End C code.

A console displays the output of the C code:

```
AirlineName FlightNumber
```

```
American Airlines 1154
```

```
Air India 852
```

```
Lufthansa 920
```

Animation captions:

1. A declarative language specifies what result is desired. The SQL selects all airline names and flight numbers departing from JFK airport after noon.
2. A procedural language specifies how the result is processed. The code fragment is written in C.
3. printf() displays column headings. The string containing % characters specifies the heading format.
4. The for loop checks every row of the Flight table. Variable i is the row number.
5. The if statement selects flights departing from JFK airport after noon.
6. printf() displays the airline name and flight number of selected flights.

©zyBooks 01/31/24 18:46 1939727

Rob Daglio

MDCCOP2335Spring2024

PARTICIPATION ACTIVITY

42.1.3: Database programming.

1) _____ is the leading declarative language.

Check**Show answer**

©zyBooks 01/31/24 18:46 1939727

Rob Daglio

MDCCOP2335Spring2024

2) A(n) _____ determines how to process statements in a declarative language.

Check**Show answer**

3) C contains control flow statements and SQL does not. This is an example of the _____ gap.

Check**Show answer**

4) SQL uses many keywords, like SELECT, FROM, and WHERE. C relies heavily on punctuation, like {} and #. This is an example of the _____ gap.

Check**Show answer**

©zyBooks 01/31/24 18:46 1939727

Rob Daglio

MDCCOP2335Spring2024

Database programming techniques

To bridge the syntax and paradigm gaps, three database programming techniques have emerged:

- **Embedded SQL** codes SQL statements directly in a program written in another language. The keywords `EXEC SQL` precede all SQL statements so the compiler can distinguish SQL from other statements. Ex: Embedded SQL in C for Oracle Database is called Pro*C.

- **Procedural SQL** extends the SQL language with control flow statements, creating a new programming language. Procedural SQL is limited compared to general-purpose languages, however, and is used primarily for database applications. Ex: In Oracle Database, procedural SQL is called PL/SQL.
- An **application programming interface**, or **API**, is a library of procedures or classes. The library links an application programming language to a computer service, such as a database, email, or web service. The procedure or class declarations are written in the application programming language. Ex: JDBC is a library of Java classes that access relational databases.

MDCCOP2335Spring2024

Embedded SQL was the earliest database programming technique, developed along with SQL in the 1980s. Procedural SQL emerged soon after embedded SQL in the late 1980s. The first widely used database API, ODBC, was released in 1992.

Procedural SQL and APIs are often used together. A common database task is coded in an SQL procedure. The compiled procedure is called via an API and can be reused in many applications.

PARTICIPATION ACTIVITY

42.1.4: API example using C# and ODBC.



```

string reservationConnString =
    "DRIVER={MySQL ODBC 3.51 Driver};" +
    "SERVER=localhost;" +
    "DATABASE=Reservation;" +
    "UID=samsnead;" +
    "PASSWORD=e%8nH!";
OdbcConnection reservationConnection = new OdbcConnection(reservationConnString);
reservationConnection.Open();

OdbcCommand listFlights = new OdbcCommand("SELECT * FROM Flight",
                                             reservationConnection);

OdbcDataReader flightReader;
flightReader = listFlights.ExecuteReader();

while (flightReader.Read()) {
    Console.WriteLine("Flight Number: " + flightReader.GetInt32(0));
    Console.WriteLine("Airline Name: " + flightReader.GetString(1));
}

flightReader.Close();

```

©zyBooks 01/31/24 18:46 1939727
Rob Daglio
MDCCOP2335Spring2024

Flight Number: 208
Airline Name: United Airlines
Flight Number: 920
Airline Name: Lufthansa

Animation content:

Static figure:

A C# code fragment that calls the ODBC API appears:

Begin C# code:

```
string reservationConnString =  
    "DRIVER={MySQL ODBC 3.51 Driver};" +  
    "SERVER=localhost;" +  
    "DATABASE=Reservation;" +  
    "UID=samsnead;" +  
    "PASSWORD=e%8nH!";
```

©zyBooks 01/31/24 18:46 1939727

Rob Daglio

MDCCOP2335Spring2024

```
OdbcConnection reservationConnection = new OdbcConnection(reservationConnString);  
reservationConnection.Open();  
OdbcCommand listFlights = new OdbcCommand("SELECT * FROM Flight", reservationConnection);  
OdbcDataReader flightReader;  
flightReader = listFlights.ExecuteReader();  
while (flightReader.Read()) {  
    Console.WriteLine("Flight Number: " + flightReader.GetInt32(0));  
    Console.WriteLine("Airline Name: " + flightReader.GetString(1));  
}  
flightReader.Close();  
End C# code.
```

A console displays the output of the C# code fragment:

Flight Number: 208

Airline Name: United Airlines

Flight Number: 920

Airline Name: Lufthansa

Animation captions:

1. This example illustrates ODBC embedded in C#. The reservationConnection object connects the program to the database.
2. The listFlights object contains the SELECT statement and connects to the Reservation database.
3. The flightReader object points to individual rows of the result table.
4. ExecuteReader() executes the SELECT statement stored in the listFlights object.
5. Read() moves flightReader to the next result table row. The while loop repeats until all rows have been read.
6. GetInt32(0) and GetString(1) return the first and second columns of the current FlightReader

Rob Daglio

MDCCOP2335Spring2024





1) Different database programming techniques cannot be combined in one application.

- True
- False

2) The EXEC SQL keyword is used in procedural SQL.

©zyBooks 01/31/24 18:46 1939727

Rob Daglio

MDCCOP2335Spring2024

- True
- False

3) Procedural SQL is specified in the SQL standard.



- True
- False

4) The ODBC API for the Java language consists of Java classes.



- True
- False

Advantages and disadvantages

Embedded SQL allows the programmer to write SQL statements directly within another programming language. This approach is conceptually simple but suffers three problems:

- *Gaps*. Most programming today uses object-oriented languages. The syntax and paradigm gap between object-oriented language and SQL is wide, so programs are difficult to write and maintain.
- *Compile steps*. The SQL statements and host language must be compiled with different compilers, in two steps.
- *Network traffic*. Application programs usually run on a client computer, connected to the database server via a network. With embedded SQL, each query generates a 'round-trip' from client to server, increasing network traffic and execution time.

©zyBooks 01/31/24 18:46 1939727

Rob Daglio

MDCCOP2335Spring2024

For the above reasons, embedded SQL is no longer widely used.

Procedural SQL has several advantages over embedded SQL:

- *Gaps*. Procedural SQL gracefully extends SQL. The syntax and paradigm gaps are minimal so programs are relatively easy to write and maintain.
- *Compile steps*. Procedural SQL is compiled in one step rather than two.

- *Network traffic.* Client applications typically call SQL procedures for execution on the database server. Each procedure executes multiple database operations, reducing network traffic and execution time.
- *Optimization level.* The database optimizes entire procedures rather than individual queries, resulting in better optimization and faster query execution.

Procedural SQL languages vary significantly by database and do not offer the full capabilities of general-purpose languages. Ex: Most procedural SQL languages are not object-oriented. For these reasons, procedural SQL is commonly used for limited database tasks.

APIs address the limitations of embedded SQL and procedural SQL:

- *Gaps.* Since API procedures and class declarations are written in the application language, applications are compiled in one step, the syntax gap disappears, and the paradigm gap is reduced.
- *Applications.* Database APIs are available for general-purpose, object-oriented languages. Applications are not limited to database processing tasks and procedural languages.
- *Database independence.* Unlike procedural SQL, language and API syntax is independent of data source.

For the reasons above, APIs are the most commonly used database programming technique.

Table 42.1.1: Advantages and disadvantages.

	Paradigm gap	Syntax gap	Object-oriented language	Applications	Compile steps	Network traffic	Database-independent
Embedded SQL	High	High	Limited	General	2	High	No
Procedural SQL	Moderate	Low	Limited	Database processing	1	Low	No
API	Moderate	Low	Yes	General	1	Variable	Yes

PARTICIPATION ACTIVITY

42.1.6: Advantages and disadvantages.



Select the database programming technique that best matches the description.



1) Usually generates a network round trip for each SQL query.

- Embedded SQL
- Procedural SQL
- API

2) Compiles and stores complex data processing tasks on the database server.

- Embedded SQL
- Procedural SQL
- API

3) Has become less popular with the rise of object-oriented programming.

- Embedded SQL
- Procedural SQL
- API

4) Affords the greatest database security.

- Embedded SQL
- Procedural SQL
- API



©zyBooks 01/31/24 18:46 193977

Rob Daglio

MDCCOP2335Spring2024



Exploring further:

- [Declarative programming](#)
- [Procedural and object-oriented programming](#)

©zyBooks 01/31/24 18:46 193977

Rob Daglio

MDCCOP2335Spring2024

Overview

Embedded SQL statements appear in programs written in another language, called the **host language**. To distinguish SQL from host language statements, embedded SQL begins with the keyword

EXEC SQL, followed by an SQL statement and a semicolon.

Programs containing embedded SQL are compiled in two steps:

1. A **precompiler** translates SQL statements into host language statements and function calls.
2. The host language compiler translates the host language program to an executable program.

Embedded SQL is defined in the SQL standard and commonly supported in older host languages, such as C, COBOL, or FORTRAN. Ex: The implementation in C for Oracle Database is called Pro*C.

MDCCOP2335Spring2024
Embedded SQL is an early database programming technique, developed soon after the SQL language, and is not supported in newer languages and databases such as Python and MySQL.

SQLJ is a variant of embedded SQL designed for Java. SQLJ is similar to standard embedded SQL, but the syntax is adapted to Java. SQLJ is not widely supported, since object-oriented languages like Java commonly use an API rather than embedded SQL.

This section describes embedded SQL in C and Oracle Database.

PARTICIPATION
ACTIVITY

42.2.1: Embedded SQL in a C program.



```
int main() {
    /* Other C statements */
    printf("Updating flight number 308");

    EXEC SQL
        UPDATE Flight
        SET AirlineName = 'British Airways', AirportCode = 'JFK'
        WHERE FlightNumber = 308;

    /* Other C statements */
    return 0;
}
```



C
code

Animation content:

Static figure:

©zyBooks 01/31/24 18:46 1939727

Begin C code:

Rob Daglio

int main()

MDCCOP2335Spring2024

/* Other C statements */

printf("Updating flight number 308");

EXEC SQL

UPDATE Flight

SET AirlineName = 'British Airways', AirportCode = 'JFK'

```
    WHERE FlightNumber = 308;  
/* Other C statements */  
return 0;  
}  
End C code.
```

The EXEC SQL statement is highlighted. An arrow from the EXEC SQL statement points to a rectangle with caption C code.

©zyBooks 01/31/24 18:46 1939727
Rob Daglio
MDCCOP2335Spring2024

Animation captions:

1. The C program begins with int main() and ends with return 0.
2. printf() is a C statement that outputs text.
3. An embedded SQL statement begins with EXEC SQL and ends with a semicolon.
4. The precompiler translates embedded SQL to C statements and function calls.

PARTICIPATION
ACTIVITY

42.2.2: Embedded SQL.



- 1) Which statement is correct embedded SQL syntax?



- ```
INSERT INTO Employee (ID,
Name, Salary)
VALUES (2538, 'Lisa
Ellison', 45000);
```
- ```
EXEC SQL  
INSERT INTO Employee ( ID,  
Name, Salary)  
VALUES (2538, 'Lisa  
Ellison', 45000)
```
- ```
EXEC SQL
INSERT INTO Employee (ID,
Name, Salary)
VALUES (2538, 'Lisa
Ellison', 45000);
```
- ```
EXEC SQL INSERT INTO  
Employee ( ID, Name,  
Salary);  
EXEC SQL VALUES (2538,  
'Lisa Ellison', 45000);
```

©zyBooks 01/31/24 18:46 1939727
Rob Daglio
MDCCOP2335Spring2024



2) Embedded SQL is commonly supported in which language?

- C
- Python
- HTML

3) A precompiler translates embedded SQL to:

- Machine language
- Host language
- SQL

©zyBooks 01/31/24 18:46 19397
Rob Daglio
MDCCOP2335Spring2024

Connections

Programs containing embedded SQL must connect to a database prior to executing any queries. A connection is a communication link between a host program and a database server. A connection identifies the database location and provides login credentials. After a connection is established, all SQL queries execute on the specified database.

Connections are managed with three embedded SQL statements:

- **CONNECT TO Database AS ConnectionName USER AccountName USING Password;** creates a connection called ConnectionName to the database specified as Database. The syntax of Database depends on the computing system and might, for example, be an internet address.
- **SET CONNECTION ConnectionName;** sets the database associated with ConnectionName as the 'active' database. Subsequent embedded SQL statements are directed to the associated database.
- **DISCONNECT ConnectionName;** terminates the connection. All resources associated with the connection, such as memory for query results, are released.

A host program can create multiple connections to communicate with multiple databases, but only one connection can be active at any time.

PARTICIPATION ACTIVITY

42.2.3: Connections in a C program fragment.

©zyBooks 01/31/24 18:46 19397
Rob Daglio
MDCCOP2335Spring2024

```
EXEC SQL
  CONNECT TO 100.320.420.999 AS FlightConnection
  USER "SamSnead" USING "s#-8aye9(";

EXEC SQL
```

```

SET CONNECTION FlightConnection;

printf("Updating flight number 308");
EXEC SQL
    UPDATE Flight
    SET AirlineName = 'British Airways', AirportCode = 'JFK'
    WHERE FlightNumber = 308;

printf("Releasing Reservation database connection");

EXEC SQL
    DISCONNECT FlightConnection;

```

©zyBooks 01/31/24 18:46 1939727
Rob Daglio
MDCCOP2335Spring2024

Animation content:

Static figure:

Begin Embedded SQL code:

```

EXEC SQL CONNECT TO 100.320.420.999 AS FlightConnection USER "SamSnead" USING
"s#-8aye9";
EXEC SQL SET CONNECTION FlightConnection;
printf("Updating flight number 308");
EXEC SQL
    UPDATE Flight
    SET AirlineName = 'British Airways', AirportCode = 'JFK'
    WHERE FlightNumber = 308;
printf("Releasing Reservation database connection");
EXEC SQL DISCONNECT FlightConnection;
End Embedded SQL code.

```

Step 1: The CONNECT TO statement creates a connection to the database at internet address 100.320.420.999. The EXEC SQL CONNECT statement is highlighted.

Step 2: FlightConnection is active. The EXEC SQL SET CONNECTION statement is highlighted.

Step 3: Subsequent queries refer to the FlightConnection database. The EXEC SQL UPDATE statement is highlighted.

Step 4: When the connection is no longer needed, release the associated resources. The EXEC SQL DISCONNECT statement is highlighted.

Rob Daglio
MDCCOP2335Spring2024

Animation captions:

1. The CONNECT TO statement creates a connection to the database at internet address 100.320.420.999.
2. FlightConnection is active.
3. Subsequent queries refer to the FlightConnection database.

4. When the connection is no longer needed, release the associated resources.

PARTICIPATION ACTIVITY**42.2.4: Connections.**

1) Several connections can exist at the same time.



- True
- False

©zyBooks 01/31/24 18:46 1939727

Rob Daglio

MDCCOP2335Spring2024

2) A connection consists of a database address only.



- True
- False

3) All embedded SQL statements require an active connection.



- True
- False

4) Each database connection consumes computer resources.



- True
- False

Shared variables

A **shared variable** is a host language variable that appears in SQL statements. The results of SELECT statements are assigned to shared variables for further processing in the host language. Shared variables must be declared between `BEGIN DECLARE SECTION` and `END DECLARE SECTION` statements.

A shared variable declaration specifies a host language data type for the shared variable. Host language data types do not always match database data types, so the precompiler must generate code to convert data types. To minimize conversion problems, the programmer matches the data types of shared variables and columns as closely as possible.

©zyBooks 01/31/24 18:46 1939727
Rob Daglio
MDCCOP2335Spring2024

Shared variables appear in an `INTO` clause of a `SELECT` statement. To distinguish shared variables from database columns, a colon precedes shared variables. When the `SELECT` statement executes, values returned by columns in the `SELECT` clause are assigned to shared variables in the `INTO` clause.

The `INTO` clause works only for statements that return a single row. If the statement returns multiple rows, an error occurs. Statements that return multiple rows are processed with cursors, described below.

PARTICIPATION ACTIVITY
42.2.5: Shared variables in a C program fragment.


©zyBooks 01/31/24 18:46 1939727

Rob Daglio

MDCCOP2335Spring2024

```
EXEC SQL BEGIN DECLARE SECTION;
    int flight;
    char airline[21];
    char airport[4];
EXEC SQL END DECLARE SECTION;

printf("Enter flight number: ");
scanf("%d", &flight);

EXEC SQL
    SELECT AirlineName, AirportCode
    INTO :airline, :airport
    FROM Flight
    WHERE FlightNumber = :flight;

printf("Flight number: %d\nAirline name: %s\nAirport code: %s",
    flight, airline, airport);
```

Enter flight number: 308
 Flight number: 308
 Airline name: India Air
 Airport code: MSN

Animation content:

Static figure:

Begin Embedded SQL code:

```
EXEC SQL BEGIN DECLARE SECTION;
```

```
    int flight;
```

```
    char airline[21];
```

```
    char airport[4];
```

```
EXEC SQL END DECLARE SECTION;
```

```
printf("Enter flight number: ");
```

```
scanf("%d", &flight);
```

```
EXEC SQL
```

```
    SELECT AirlineName, AirportCode
```

```
    INTO :airline, :airport
```

```
    FROM Flight
```

```
    WHERE FlightNumber = :flight;
```

```
printf("Flight number: %d\nAirline name: %s\nAirport code: %s", flight, airline, airport);
```

End Embedded SQL code.

A console displays code input and output:

©zyBooks 01/31/24 18:46 1939727

Rob Daglio

MDCCOP2335Spring2024

Enter flight number: 308

Filght number: 308

Airline name: India Air

Airport code: MSN

Step 1: Three shared variables are declared between BEGIN DECLARE SECTION and END DECLARE SECTION. The code from BEGIN DECLARE SECTION through END DECLARE SECTION is highlighted.

©zyBooks 01/31/24 18:46 1939727

Rob Daglio

MDCCOP2335Spring2024

Step 2: The C code allows the user to input a flight number. The shared variable flight is assigned the number. The first printf is highlighted. The scanf statement is highlighted.

Step 3: The query selects rows with column FlightNumber equal to the shared variable flight. The WHERE clause is highlighted.

Step 4: The shared variable airline is assigned AirlineName. The shared variable airport is assigned AirportCode. The SELECT and INTO clauses are highlighted.

Step 5: The C code outputs the result of the SELECT statement. The second printf statement is highlighted. Output appears on the console.

Animation captions:

1. Three shared variables are declared between BEGIN DECLARE SECTION and END DECLARE SECTION.
2. The C code allows the user to input a flight number. The shared variable flight is assigned the number.
3. The query selects rows with column FlightNumber equal to the shared variable flight.
4. The shared variable airline is assigned AirlineName. The shared variable airport is assigned AirportCode.
5. The C code outputs the result of the SELECT statement.

PARTICIPATION
ACTIVITY

42.2.6: Shared variables.



- 1) Shared variables can appear in which clause of an embedded SELECT statement?

- The INTO clause only
- The WHERE clause only
- Both INTO and WHERE clauses

©zyBooks 01/31/24 18:46 1939727

Rob Daglio

MDCCOP2335Spring2024



2) The Employee table has columns ID and Name, with 1000 rows. The host program declares shared variables employeeNumber and employeeName. What is wrong with the statement below?

```
EXEC SQL
SELECT ID, Name
INTO :employeeNumber,
:employeeName
FROM Employee;
```

©zyBooks 01/31/24 18:46 1939727
Rob Daglio
MDCCOP2335Spring2024

- The statement should be on one line, not four.
 - The statement should return one row, not many.
 - A colon should appear before ID and Name.
- 3) In the declaration section, shared variables are assigned a data type in:

- The host language
- The database
- The SQL standard



Cursors

A **cursor** is an embedded SQL variable that identifies an individual row of a result table. Cursors are necessary to process queries that return multiple rows. Cursors are managed with four embedded SQL statements:

- **DECLARE CursorName CURSOR FOR Statement;** creates a cursor named CursorName that is associated with the query Statement.
- **OPEN CursorName;** executes the query associated with CursorName and positions the cursor before the first row of the result table.
- **FETCH FROM CursorName INTO :SharedVariable1, :SharedVariable2...**; advances CursorName to the next row of the result table and copies selected values into the shared variables.
- **CLOSE CursorName;** releases the result table associated with the cursor.

©zyBooks 01/31/24 18:46 1939727
Rob Daglio

As result table rows are fetched, the cursor eventually moves past the last table row and cannot fetch any more data. The program can detect this condition by checking the value of SQLSTATE, an

embedded SQL variable. A value of "00000" indicates the cursor is pointing at a valid row. Other values indicate the cursor has moved past the last row, or other error conditions.

The statement `INCLUDE SQLCA` is required to make `SQLSTATE` and other embedded SQL variables available to the host program.

PARTICIPATION ACTIVITY
42.2.7: Cursors in a C program fragment.

©zyBooks 01/31/24 18:46 1939727

Rob Daglio

MDCCOP2335Spring2024



```

EXEC SQL INCLUDE SQLCA;

EXEC SQL BEGIN DECLARE SECTION;
    int flight;
    char airline[21];
    char airport[4];
EXEC SQL END DECLARE SECTION;

EXEC SQL DECLARE FlightCursor CURSOR FOR
    SELECT FlightNumber, AirlineName, AirportCode
    FROM Flight;

EXEC SQL OPEN FlightCursor;

EXEC SQL FETCH FROM FlightCursor INTO :flight, :airline, :airport;

while (strcmp(SQLSTATE, "00000") == 0) {
    printf("%s %d at %s", airline, flight, airport);
    EXEC SQL FETCH FROM FlightCursor INTO :flight, :airline, :airport;
}

EXEC SQL CLOSE FlightCursor;

```

United Airlines 504 at JF
India Air 308 at MSN
Lufthansa 552 at SEA

Animation content:

Static figure:

Begin Embedded SQL code:

`EXEC SQL INCLUDE SQLCA;`

`EXEC SQL BEGIN DECLARE SECTION;`

`int flight;`

`char airline[21];`

`char airport[4];`

`EXEC SQL END DECLARE SECTION;`

`EXEC SQL DECLARE FlightCursor CURSOR FOR`

`SELECT FlightNumber, AirlineName, AirportCode`

`FROM Flight;`

`EXEC SQL OPEN FlightCursor;`

`EXEC SQL FETCH FROM FlightCursor INTO :flight, :airline, :airport;`

©zyBooks 01/31/24 18:46 1939727

Rob Daglio

MDCCOP2335Spring2024

```
while (strcmp(SQLSTATE, "00000") == 0) {  
    printf("%s %d at %s", airline, flight, airport);  
    EXEC SQL FETCH FROM FlightCursor INTO :flight, :airline, :airport;  
}  
EXEC SQL CLOSE FlightCursor;  
End Embedded SQL code.
```

A console displays code output:

United Airlines 504 at JFK

India Air 308 at MSN

Lufthansa 552 at SEA

©zyBooks 01/31/24 18:46 1939727

Rob Daglio

MDCCOP2335Spring2024

Step 1: DECLARE CURSOR creates a cursor named FlightCursor that is associated with the SELECT query. The EXEC SQL DECLARE CURSOR statement is highlighted.

Step 2: OPEN executes the query and positions FlightCursor before the result table's first row. The EXEC SQL OPEN statement is highlighted.

Step 3: FETCH moves the cursor to first row and assigns column values to corresponding shared variables. The first EXEC SQL FETCH statement is highlighted.

Step 4: INCLUDE declares SQLSTATE to a C program. The value "00000" indicates the cursor is pointing to a valid row. The EXEC SQL INCLUDE statement is highlighted. The while clause is highlighted.

Step 5: While the cursor points to a valid row, output the query result and fetch the next row. The printf and second EXEC SQL FETCH statements are highlighted. Output appears on the console.

Step 6: When the cursor moves past the last row, the while loop terminates and the cursor is released. The EXEC SQL CLOSE statement is highlighted.

Animation captions:

1. DECLARE CURSOR creates a cursor named FlightCursor that is associated with the SELECT query.
2. OPEN executes the query and positions FlightCursor before the result table's first row.
3. FETCH moves the cursor to first row and assigns column values to corresponding shared variables.
4. INCLUDE declares SQLSTATE to a C program. The value "00000" indicates the cursor is pointing to a valid row.
5. While the cursor points to a valid row, output the query result and fetch the next row.

**PARTICIPATION
ACTIVITY**

42.2.8: Cursors.



- 1) The _____ statement moves a cursor to the next row of a result table.

Check**Show answer**

©zyBooks 01/31/24 18:46 1939727

Rob Daglio

MDCCOP2335Spring2024



- 2) The _____ statement executes a query.

Check**Show answer**

- 3) The _____ statement associates a cursor name with a query.

Check**Show answer**

- 4) The _____ statement releases resources associated with a cursor.

Check**Show answer**

- 5) The _____ statement assigns query results to shared variables.

Check**Show answer**

©zyBooks 01/31/24 18:46 1939727

Rob Daglio

MDCCOP2335Spring2024





- 6) The _____ statement positions a cursor before the first row of a result table.

[Show answer](#)

©zyBooks 01/31/24 18:46 1939727

Rob Daglio

MDCCOP2335Spring2024

Dynamic SQL

In some cases, embedded SQL statements are generated while the program is running. Ex: In a web shopping program, the host program may generate a string variable containing an SQL statement based on user interaction.

Embedded SQL statements that are generated at run-time are called **dynamic SQL**. Embedded SQL statements that are fixed in program code are called **static SQL**. Static SQL statements are compiled by the precompiler, while dynamic SQL is compiled as the program executes. For this reason, static SQL executes faster than dynamic SQL.

Dynamic SQL is managed with two embedded SQL statements:

- **PREPARE StatementName FROM :StatementString;** compiles the query in the shared variable StatementString and associates the variable with StatementName.
- **EXECUTE StatementName;** executes the query associated with StatementName.

The **EXECUTE** statement has several variations, including:

- **EXECUTE StatementName USING :SharedVariable1, :SharedVariable2, ...;** substitutes the values of shared variables for placeholders in the StatementString, as illustrated in the animation below.
- **EXECUTE IMMEDIATE :StatementString;** both compiles and executes the query in StatementString. **EXECUTE IMMEDIATE** does the work of **PREPARE** and **EXECUTE**, but the compiled query is not named and saved.

PREPARE names and saves a compiled query for use by multiple **EXECUTE** statements. Executing a compiled query with **EXECUTE** is more efficient than **EXECUTE IMMEDIATE** when a query is executed many times.

©zyBooks 01/31/24 18:46 1939727

Rob Daglio

Programming with dynamic SQL is challenging because the host language code depends on the specific dynamic query. Ex: A query that deletes a single row is coded differently than a query that selects multiple rows using a cursor. Therefore, dynamic SQL is most effective when the program generates a specific query type with limited variations.



```

EXEC SQL BEGIN DECLARE SECTION;
  char queryString[200];
  int flight;
EXEC SQL END DECLARE SECTION;

printf("Enter query: ");
scanf("%s", queryString);
EXEC SQL PREPARE QueryName FROM :queryString;

printf("Enter flight number: ");
scanf("%d", &flight);
EXEC SQL EXECUTE QueryName USING :flight;

```

Memory

DELETE FROM Flight :flight WHERE FlightNumber = :placeHolder;	queryString
692	flight
©zyBooks 01/31/24 18:46 1939727	Rob Daglio
	MDCCOP2335Spring2024

Animation content:

Static figure:

Begin Embedded SQL code:

```
EXEC SQL BEGIN DECLARE SECTION;
```

```
  char queryString[200];
```

```
  int flight;
```

```
EXEC SQL END DECLARE SECTION;
```

```
printf("Enter query: ");
```

```
scanf("%s", queryString);
```

```
EXEC SQL PREPARE QueryName FROM :queryString;
```

```
printf("Enter flight number: ");
```

```
scanf("%d", &flight);
```

```
EXEC SQL EXECUTE QueryName USING :flight;
```

End Embedded SQL code.

A memory diagram appears with three cells. The first cell has caption queryString and contains the statement:

DELETE FROM Flight WHERE FlightNumber = placeholder;

The second cell has caption flight and contains 692.

Step 1: Two shared variables are declared. queryString will later contain an SQL statement. The code from EXEC SQL BEGIN to EXEC SQL END is highlighted. Captions queryString and flight appear next to memory cells.

Step 2: The user inputs an SQL query, which is saved in queryString. The first printf and scanf statements are highlighted. The DELETE statement appears in the queryString memory cell.

Step 3: The query is compiled and saved as QueryName. The EXEC SQL PREPARE statement is highlighted.

Step 4: The user inputs flight number 692. The query executes and deletes rows containing flight number 692. The second printf and scan statements are highlighted. The EXEC SQL EXECUTE statement is highlighted. 692 appears in the flight memory cell.

Animation captions:

1. Two shared variables are declared. queryString will later contain an SQL statement.
2. The user inputs an SQL query, which is saved in queryString.
3. The query is compiled and saved as QueryName.
4. The user inputs flight number 692. The query executes and deletes rows containing flight number 692.

©zyBooks 01/31/24 18:46 1939727

Rob Daglio

MDCCOP2335Spring2024

PARTICIPATION ACTIVITY

42.2.10: Dynamic SQL.



- 1) A(n) _____ statement must appear before an EXECUTE statement without the IMMEDIATE keyword.

Check

Show answer



- 2) Shared variables appear in an EXECUTE statement with the _____ clause.

Check

Show answer



- 3) Statements that are written explicitly in program code are _____ SQL.

Check

Show answer



©zyBooks 01/31/24 18:46 1939727

Rob Daglio

MDCCOP2335Spring2024



- 4) The precompiler cannot compile
_____ SQL statements.

Check**Show answer**

©zyBooks 01/31/24 18:46 1939727

Rob Daglio

MDCCOP2335Spring2024

**CHALLENGE ACTIVITY****42.2.1: Embedded SQL.**

539740.3879454.qx3zqy7

Start

Given the following tables:

```
CREATE TABLE Room (
    Name CHAR(40),
    Capacity INT,
    PRIMARY KEY (Name)
);
```

```
CREATE TABLE Seat (
    RoomName CHAR(40),
    SeatNumber INT,
    PRIMARY KEY (RoomName, SeatNumber)
);
```

Fill in the keywords to open, use, and close a connection.

```
int main() {      // how a C program begins
    EXEC SQL
        CONNECT ____ (A)____ 127.0.0.1 AS ____ (B)____
        USER "student42" ____ (C)____ "99x442";
    EXEC SQL
        ____ (D)____ CONNECTION RoomConnection;
    EXEC SQL
        INSERT INTO Room VALUES ('Lecture', 100);
    EXEC SQL
        DISCONNECT ____ (E)____;
    return 0;      // how a C program ends
}
```

(A) Ex: KEYWORD(B) (C) (D) (E)

©zyBooks 01/31/24 18:46 1939727

Rob Daglio

MDCCOP2335Spring2024

1**2****3****Check****Next**

Exploring further:

- [Database and language support for embedded SQL](#)
- [Oracle embedded SQL](#)
- [Oracle dynamic SQL](#)

©zyBooks 01/31/24 18:46 1939727
Rob Daglio
MDCCOP2335Spring2024

42.3 Procedural SQL

Overview

Procedural SQL is an extension of the SQL language that includes procedures, functions, conditionals, and loops. Procedural SQL is intended for database applications and does not offer the full capabilities of general-purpose languages such as Java, Python, and C.

Procedural SQL can, in principle, be used for complete programs. More often, however, the language is used to create procedures that interact with the database and accomplish limited tasks. These procedures are compiled by and stored in the database, and therefore are called **stored procedures**.

Stored procedures can be called from a command line or a host program written in another language. Host program calls to stored procedures can be coded with embedded SQL or built into an API.

SQL/Persistent Stored Modules (SQL/PSM) is a standard for procedural SQL that extends the core SQL standard. SQL/PSM is implemented in many relational databases with significant variations and different names. Many names incorporate the acronym 'PL', which stands for 'Procedural Language'.

MySQL conforms closely to the SQL/PSM standard. MySQL procedural SQL is considered part of the SQL language and does not have a special name. This section describes the MySQL implementation.

Table 42.3.1: Procedural SQL.

©zyBooks 01/31/24 18:46 1939727
Rob Daglio
MDCCOP2335Spring2024

Database	Programming language
Oracle Database	PL/SQL
SQL Server	Transact-SQL

DB2	SQL PL
PostgreSQL	PL/pgSQL
MySQL	SQL

©zyBooks 01/31/24 18:46 1939727

Rob Daglio

MDCCOP2335Spring2024

PARTICIPATION ACTIVITY**42.3.1: Procedural SQL.**

- 1) Procedural SQL is compiled in two steps, with a precompiler followed by a compiler.



- True
- False

- 2) A stored procedure is compiled every time the procedure is called.



- True
- False

- 3) Procedural SQL languages differ significantly, depending on the database.



- True
- False

- 4) Most Procedural SQL languages implement the entire SQL/PSM standard.



- True
- False

©zyBooks 01/31/24 18:46 1939727

Rob Daglio

MDCCOP2335Spring2024

Stored procedures

A stored procedure is declared with a **CREATE PROCEDURE** statement, which includes the procedure name, optional parameter declarations, and the procedure body.

A **parameter declaration** has an optional **IN**, **OUT**, or **INOUT** keyword, a **ParameterName**, and a **Type**. The **Type** is any data type supported by the database. The **IN**, **OUT**, and **INOUT** keywords limit the parameter to input, output, or both. If no keyword appears, the parameter is limited to input.

The procedure **body** consists of either a simple statement, such as **SELECT** and **DELETE**, or a compound statement. Compound statements, described below, are commonly used in stored procedures to code complex database tasks.

Figure 42.3.1: CREATE PROCEDURE statement.

```
CREATE PROCEDURE ProcedureName ( <parameter-declaration>, <parameter-declaration>, ... )
Body;

<parameter-declaration>:
[ IN | OUT | INOUT ] ParameterName Type
```

©zyBooks 01/31/24 18:46 1939727

Bob DeMillo
MDCCOP2335Spring2024

A stored procedure is executed with a **CALL** statement. The **CALL** statement includes the procedure name and a parameter list. The types of the CALL parameters must be compatible with the types of the corresponding **CREATE PROCEDURE** parameters.

Figure 42.3.2: CALL Statement.

```
CALL ProcedureName( ParameterName, ParameterName,
... )
```

Stored procedures can be called from:

- Other stored procedures
- The MySQL Command-Line Client
- C, C++, Java, Python, and other programming languages

When called from another stored procedure, call arguments are literals, like 10 or 'alpha', or stored procedure variables as described below. When called from the command line, call arguments are literals or user-defined variables. A **user-defined variable** is an SQL variable that must begin with an @ character. When called from a different programming language, call parameter syntax varies greatly, depending on the language and API.

PARTICIPATION
ACTIVITY

42.3.2: Stored procedure call from the command line.

```
CREATE PROCEDURE FlightCount(IN airline VARCHAR(20), OUT quantity INT)
  SELECT COUNT(*)
    INTO quantity
   FROM Flight
 WHERE AirlineName = airline;
```

```
mysql> CALL FlightCount('British Airways', @result);
1 row in set (0.00 sec)

mysql> SELECT @result;
+-----+
| @result |
+-----+
| 41      |
+-----+
1 row in set (0.00 sec)
```

21/24 18:46 1939727
41 Rob Daglio
MDCCOP2335Spring2024

Animation content:

Static figure:

Begin SQL code:

```
CREATE PROCEDURE FlightCount(IN airline VARCHAR(20), OUT quantity INT)
  SELECT COUNT(*) INTO quantity FROM Flight WHERE AirlineName = airline;
End SQL code.
```

A console displays operator input and database output:

```
mysql> CALL FlightCount('British Airways', @result);
1 row in set (0.00 sec)
mysql> SELECT @result;
+-----+
| @result |
+-----+
| 41      |
+-----+
1 row in set (0.00 sec)
```

An arrow points from British Airways on the console to IN airline in the CREATE PROCEDURE statement. Another arrow points from OUT quantity in the CREATE PROCEDURE statement to @result on the console. 41 appears below @result on the console and is highlighted.

Step 1: The FlightCount stored procedure has an input parameter airline and an output parameter quantity. The CREATE PROCEDURE clause is highlighted.

Step 2: The stored procedure body is a single SELECT statement. The SELECT statement is highlighted.

Step 3: The INTO clause assigns the COUNT(*) value to the quantity parameter. The INTO clause is highlighted.

Step 4: The stored procedure is called from the command line. airline is assigned 'British Airways'. @result is assigned the resulting quantity value. The console appears. The first console command appears. The arrows appear. 41 appears below @result on the console.

©zyBooks 01/31/24 18:46 1939727
Rob Daglio

Step 5: The value of the user-defined variable @result is displayed with a SELECT statement. The Flight table has 41 British Airways flights. The second console command and output appear.

Animation captions:

1. The FlightCount stored procedure has an input parameter airline and an output parameter quantity.
2. The stored procedure body is a single SELECT statement.
3. The INTO clause assigns the COUNT(*) value to the quantity parameter.
4. The stored procedure is called from the command line. airline is assigned 'British Airways'. @result is assigned the resulting quantity value.
5. The value of the user-defined variable @result is displayed with a SELECT statement. The Flight table has 41 British Airways flights.

PARTICIPATION ACTIVITY

42.3.3: Stored procedures.



Match the language element to the description.

If unable to drag and drop, refresh the page.

CALL

body

<parameter-declaration>

INOUT

@

First character in a user-defined variable.

Consists of a simple or compound statement.

©zyBooks 01/31/24 18:46 1939727
Rob Daglio
MDCCOP2335Spring2024

May appear in the body of a stored procedure.

Precedes a parameter name in a <parameter-declaration>.

Optional in CREATE PROCEDURE statements.

Reset

Compound statements

©zyBooks 01/31/24 18:46 1939727

Rob Daglio

A **compound statement** is a series of statements between a **BEGIN** and an **END** keyword. The statements may be variable declarations, assignment statements, if statements, while loops, and other common programming constructs.

The **DECLARE** statement creates variables for use inside stored procedures. A variable has a name, data type, and an optional default value. Variable scope extends from the **DECLARE** statement to the next **END** keyword. Variables should not have the same name as a table column.

Figure 42.3.3: DECLARE statement.

```
DECLARE VariableName Type [ DEFAULT Value  
];
```

The **SET** statement assigns a variable with an expression. The variable must be declared prior to the **SET** statement, and the expression may be any valid SQL expression.

Figure 42.3.4: SET statement.

```
SET VariableName =  
<expression>;
```

The **IF** statement is similar to statements in other programming languages. The statement includes a logical expression and a statement list. If the expression is true, the statements in the list are executed. If the expression is false, the statements are ignored.

IF statements have optional **ELSEIF** and **ELSE** clauses. **ELSEIF** and **ELSE** clauses are executed when the expression in the **IF** clause is false, as in other programming languages.

Figure 42.3.5: IF statement.

```

IF expression THEN statement list
[ ELSEIF expression THEN statement
list ]
[ ELSE StatementList ]
END IF;

```

©zyBooks 01/31/24 18:46 1939727

Rob Daglio

The **WHILE** statement also includes an expression and a statement list. The statement list repeatedly executes as long as the expression is true. When the expression is false, the **WHILE** statement terminates, and the statement following **END WHILE** executes.

Figure 42.3.6: WHILE statement.

```

WHILE expression
DO
    statement list
END WHILE;

```

Other common programming statements are supported within the compound statement, including **CASE**, **REPEAT**, and **RETURN**. Compound statements also support cursors, described below.

PARTICIPATION ACTIVITY

42.3.4: Compound statements in a stored procedure.



```

CREATE PROCEDURE AddFlights(IN startTime TIME, IN endTime TIME)
BEGIN
    DECLARE departTime TIME DEFAULT startTime;
    DECLARE flightNum INT DEFAULT 540;

    WHILE departTime <= endTime DO
        INSERT INTO Flight
        VALUES (flightNum, 'Lufthansa Airlines', departTime, 'FRA');
        SET flightNum = flightNum + 100;
        IF departTime < '12:00' THEN
            SET departTime = ADDTIME(departTime, '00:30');
        ELSE
            SET departTime = ADDTIME(departTime, '1:00');
        END IF;

    END WHILE;
END;

```

©zyBooks 01/31/24 18:46 1939727

Rob Daglio

MDCCOP2335Spring2024

14:00	departTime
640	flightNum

540	Lufthansa Airlines	13:00	FRA

©zyBooks 01/31/24 18:46 1939727

Rob Daglio

MDCCOP2335Spring2024

Animation content:

Static figure:

Begin SQL code:

```
CREATE PROCEDURE AddFlights(IN startTime TIME, IN endTime TIME)
```

```
BEGIN
```

```
    DECLARE departTime TIME DEFAULT startTime;
```

```
    DECLARE flightNum INT DEFAULT 540;
```

```
    WHILE departTime <= endTime DO
```

```
        INSERT INTO Flight VALUES (flightNum, 'Lufthansa Airlines', departTime, 'FRA');
```

```
        SET flightNum = flightNum + 100;
```

```
        IF departTime < '12:00' THEN SET departTime = ADDTIME(departTime, '00:30');
```

```
        ELSE SET departTime = ADDTIME(departTime, '1:00');
```

```
        END IF;
```

```
    END WHILE;
```

```
END;
```

End SQL code.

A memory diagram appears with two cells. The first cell has caption departTime and contains 14:00. The second cell has caption flightNum and contains 680.

A table contains one row:

540, Lufthansa Airlines, 13:00, FRA

Step 1: The AddFlights() stored procedure inserts new flights into the Flight table. The body is a compound statement. CREATE PROCEDURE, BEGIN, and END are highlighted.

Step 2: The declare statements create and initialize two variables. departTime is assigned the startTime parameter. flightNum is assigned 540. The DECLARE statements are highlighted. 13:00 appears in memory cell departTime. 540 appears in memory cell flightNum.

Step 3: The WHILE loop repeats as long as departTime is \leq endTime. The WHILE clause is highlighted.

Step 4: Each pass through the loop inserts a new flight. The INSERT statement is highlighted. The

values 13:00 and 540 move from memory to the INSERT statement. A row is inserted into the table:
540, Lufthansa Airlines, 13:00, FRA

Step 5: flightNum is incremented by 100. If departTime is before noon, add 30 minutes. If departTime is noon or later, add an hour. The SET statement is highlighted. The IF ELSE END IF statement is highlighted. The memory cell values change to 14:00 and 640.

Animation captions:

©zyBooks 01/31/24 18:46 1939727

Rob Daglio

MDCCOP2335Spring2024

1. The AddFlights() stored procedure inserts new flights into the Flight table. The body is a compound statement.
2. The declare statements create and initialize two variables. departTime is assigned the startTime parameter. flightNum is assigned 540.
3. The WHILE loop repeats as long as departTime is \leq endTime.
4. Each pass through the loop inserts a new flight.
5. flightNum is incremented by 100. If departTime is before noon, add 30 minutes. If departTime is noon or later, add an hour.

PARTICIPATION ACTIVITY

42.3.5: Compound statements.



The following stored procedure awards bonuses based on employee performance ratings:

©zyBooks 01/31/24 18:46 1939727

Rob Daglio

MDCCOP2335Spring2024

```
CREATE PROCEDURE AwardBonus()
BEGIN

    DECLARE bonusAmount INT;
    DECLARE ratingCount INT;
    __A__ rating INT DEFAULT 10;

    __B__ rating > 0 DO

        SELECT COUNT(*)
        __C__ ratingCount
        FROM Employee
        WHERE PerformanceRating = rating;

        __D__ ratingCount > 80
        SET bonusAmount = 1000 * rating;
        ELSE
        SET bonusAmount = 100 * rating;
        END IF;

        UPDATE Employee
        SET EmployeeBonus = bonusAmount
        WHERE PerformanceRating = rating;

        __E__ rating = rating - 1;

    END WHILE;

END;
```

©zyBooks 01/31/24 18:46 1939727

Rob Daglio

MDCCOP2335Spring2024

1) What is keyword A?

**Check****Show answer**

2) What is keyword B?

**Check****Show answer**

3) What is keyword C?

**Check****Show answer**

©zyBooks 01/31/24 18:46 1939727

Rob Daglio

MDCCOP2335Spring2024



4) What is keyword D?

Check

Show answer

5) What is keyword E?

Check

Show answer

©zyBooks 01/31/24 18:46 1939727

Rob Daglio

MDCCOP2335Spring2024

Cursors

A cursor is a special variable that identifies an individual row of a result table. Cursors are necessary to process queries that return multiple rows. Cursor syntax in procedural SQL is similar to embedded SQL:

- **DECLARE CursorName CURSOR FOR Statement** creates a cursor named CursorName that is associated with the query Statement. The Statement may not have an INTO clause.
- **DECLARE CONTINUE HANDLER FOR NOT FOUND Statement** specifies a Statement that executes when the cursor eventually moves past the last row and cannot fetch any more data. Reading data from a cursor is performed within a loop, so the Statement usually sets a variable indicating the loop should terminate.
- **OPEN CursorName** executes the query associated with CursorName and positions the cursor before the first row of the result table. Multiple cursors with different names can be open at the same time.
- **FETCH FROM CursorName INTO variable [, variable, ...]** advances CursorName to the next row of the result table and copies selected values into the variables. The number of INTO clause variables must match the number of SELECT clause columns.
- **CLOSE CursorName** releases the result table associated with the cursor.

The animation below shows a stored procedure that uses a cursor to change departure times for all flights.

©zyBooks 01/31/24 18:46 1939727

Rob Daglio

MDCCOP2335Spring2024



PARTICIPATION ACTIVITY

42.3.6: Cursor modifies flight departure times.

```
CREATE PROCEDURE ChangeDepartureTime(IN airportIn CHAR(3))
BEGIN
    DECLARE flightNum INT;
```

```

DECLARE airport CHAR(3);
DECLARE departTime TIME;
DECLARE finishedReading BOOL DEFAULT FALSE;
DECLARE flightCursor CURSOR FOR
    SELECT FlightNumber, AirportCode, DepartureTime
    FROM Flight;
DECLARE CONTINUE HANDLER FOR NOT FOUND SET finishedReading = TRUE;

OPEN flightCursor;
FETCH FROM flightCursor INTO flightNum, airport, departTime;

WHILE NOT finishedReading DO
    IF airport = airportIn THEN
        UPDATE Flight
        SET DepartureTime = ADDTIME(departTime, '1:00')
        WHERE FlightNumber = flightNum;
    ELSE
        UPDATE Flight
        SET DepartureTime = SUBTIME(departTime, '0:30')
        WHERE FlightNumber = flightNum;
    END IF;

    FETCH FROM flightCursor INTO flightNum, airport, departTime;
END WHILE;

CLOSE flightCursor;
END;

```

@zyBooks 01/31/24 18:46 1939727

Rob Daglio

MDCCOP2335Spring2024

Animation content:

Static figure:

Begin SQL code:

```
CREATE PROCEDURE ChangeDepartureTime(IN airportIn CHAR(3))
```

```
BEGIN
```

```

DECLARE flightNum INT;
DECLARE airport CHAR(3);
DECLARE departTime TIME;
DECLARE finishedReading BOOL DEFAULT FALSE;
DECLARE flightCursor CURSOR FOR SELECT FlightNumber, AirportCode, DepartureTime FROM
Flight;
DECLARE CONTINUE HANDLER FOR NOT FOUND SET finishedReading = TRUE;
OPEN flightCursor;
FETCH FROM flightCursor INTO flightNum, airport, departTime; ©zyBooks 01/31/24 18:46 1939727
WHILE NOT finishedReading DO
    IF airport = airportIn THEN UPDATE Flight SET DepartureTime = ADDTIME(departTime, '1:00')
    WHERE FlightNumber = flightNum;
    ELSE UPDATE Flight SET DepartureTime = SUBTIME(departTime, '0:30') WHERE FlightNumber =
flightNum;
    END IF;
    FETCH FROM flightCursor INTO flightNum, airport, departTime;

```

Rob Daglio

MDCCOP2335Spring2024

```
END WHILE;  
CLOSE flightCursor;  
END;  
End SQL code.
```

Step 1: The cursor named flightCursor retrieves data for all flights. The DECLARE CURSOR statement is highlighted.

©zyBooks 01/31/24 18:46 1939727
Rob Daglio

Step 2: The finishedReading variable is initially FALSE. The handler sets finishedReading to TRUE when the cursor moves past the last row. The DECLARE departTime and DECLARE CONTINUE HANDLER statements are highlighted.

Step 3: OPEN executes flightCursor's associated query and sets the cursor prior to the first result table row. The OPEN statement is highlighted.

Step 4: FETCH moves the cursor to the first row and assigns column values to procedure variables. The first FETCH statement is highlighted.

Step 5: The WHILE loop repeats until the cursor moves past the last row. The WHILE clause is highlighted.

Step 6: The IF statement changes the departure time for the selected flight. The IF ELSE END IF statement is highlighted.

Step 7: FETCH moves the cursor to the next row. The second FETCH statement is highlighted.

Step 8: CLOSE releases cursor resources when no longer needed. The CLOSE statement is highlighted.

Animation captions:

1. The cursor named flightCursor retrieves data for all flights.
2. The finishedReading variable is initially FALSE. The handler sets finishedReading to TRUE when the cursor moves past the last row.
3. OPEN executes flightCursor's associated query and sets the cursor prior to the first result table row.
4. FETCH moves the cursor to the first row and assigns column values to procedure variables.
5. The WHILE loop repeats until the cursor moves past the last row.
6. The IF statement changes the departure time for the selected flight.
7. FETCH moves the cursor to the next row.
8. CLOSE releases cursor resources when no longer needed.

©zyBooks 01/31/24 18:46 1939727
MDCCOP2335Spring2024

Refer to code in the above animation.

- 1) Which name refers to a single value selected by flightCursor?

- airportIn
- airport
- AirportCode

©zyBooks 01/31/24 18:46 1939727

Rob Daglio

MDCCOP2335Spring2024

- 2) What happens to flights that depart from the input airport?

- Departure time is one hour later.
- Departure time is a half-hour earlier.
- Departure time is not changed.



- 3) Where does flightCursor point after the WHILE loop executes three times?

- The third row of the result table
- The fourth row of the result table
- Between the third and fourth row of the result table



Stored functions

A **stored function** is like a stored procedure that returns a single value. A stored function is declared with a **CREATE FUNCTION** statement. **CREATE FUNCTION** syntax is similar to **CREATE PROCEDURE** syntax, with several differences:

- Function parameters are for input only and therefore have no **IN**, **OUT**, or **INOUT** keyword.
- Functions return a single value. The value follows the **RETURN** keyword in the stored function body. The value data type follows the **RETURNS** keyword in the stored function heading.
- Functions are invoked within an SQL expression rather than with a **CALL** statement.

Unlike **CREATE PROCEDURE** statements, **CREATE FUNCTION** requires at least one of the following keywords:

- **NO SQL** - Function contains no SQL statements.
- **READS SQL DATA** - Function contains SQL statements that read, but not write, table data.

- **DETERMINISTIC** - Function contains SQL statements that write table data.

Figure 42.3.7: CREATE FUNCTION statement.

```
CREATE FUNCTION FunctionName( <parameter-declaration>, <parameter-
declaration>, ... )
    RETURNS Type
    [ NO SQL | READS SQL DATA | DETERMINISTIC ]
Body;
```

<parameter-declaration>:
ParameterName Type

©zyBooks 01/31/24 18:46 1939727
Rob Daglio
MDCCOP2335Spring2024

Stored functions behave like built-in SQL functions, such as COUNT(), SUM(), and AVG(). The only difference is that stored functions are defined by the programmer, while built-in functions are defined by the database.

Stored functions are limited compared to stored procedures. Stored functions return only one value, while stored procedures can return multiple values through different OUT parameters. However, stored functions, like built-in functions, are useful when a computed value is needed within an SQL expression.

PARTICIPATION ACTIVITY

42.3.8: Stored function that computes tax.



```
CREATE FUNCTION ComputeTax(employeeName VARCHAR(20))
    RETURNS INT
    READS SQL DATA
BEGIN

    DECLARE income INT;

    SELECT Salary
    INTO income
    FROM Employee
    WHERE Name = employeeName;

    IF income <= 25000 THEN
        RETURN 0;
    ELSE
        RETURN (income - 25000) * 0.1;
    END IF;

END;
```

©zyBooks 01/31/24 18:46 1939727
Rob Daglio
MDCCOP2335Spring2024

mysql> SELECT ComputeTax('Lisa Ellison');

```
+-----+
| ComputeTax('Lisa Ellison') |
+-----+
|          1500 |
+-----+
1 row in set (0.00 sec)
```

Animation content:

@zyBooks 01/31/24 18:46 1939727

Rob Daglio

MDCCOP2335Spring2024

Static figure:

Begin SQL code:

```
CREATE FUNCTION ComputeTax(employeeName VARCHAR(20)) RETURNS INT READS SQL DATA
BEGIN
    DECLARE income INT;
    SELECT Salary INTO income FROM Employee WHERE Name = employeeName;
    IF income <= 25000 THEN RETURN 0;
    ELSE RETURN (income - 25000) * 0.1;
    END IF;
END;
```

End SQL code.

A console displays input and output:

```
mysql> SELECT ComputeTax('Lisa Ellison');
+-----+
| ComputeTax('Lisa Ellison') |
+-----+
|          1500 |
+-----+
1 row in set (0.00 sec)
```

Step 1: The stored function computes an employee's tax and returns an integer. The function reads SQL data with a SELECT statement. The CREATE FUNCTION statement is highlighted.

Step 2: The employee's salary is retrieved and assigned to the income variable. The DECLARE and SELECT statements are highlighted.

Step 3: Tax is computed as 10% of income exceeding \$25,000 and returned. The IF ELSE END IF statement is highlighted.

@zyBooks 01/31/24 18:46 1939727

Rob Daglio

MDCCOP2335Spring2024

Step 4: Stored functions are called within any SQL expression, like built-in functions. The console appears. The first line appears on the console:

```
mysql> SELECT ComputeTax('Lisa Ellison');
```

Step 5: Lisa Ellison's salary is \$40,000. Tax is $(\$40,000 - \$25,000) * 10\% = \$1,500$, displayed as a result table. The output appears on the console.

Animation captions:

1. The stored function computes an employee's tax and returns an integer. The function reads SQL data with a SELECT statement.
2. The employee's salary is retrieved and assigned to the income variable.
3. Tax is computed as 10% of income exceeding \$25,000 and returned.
4. Stored functions are called within any SQL expression, like built-in functions.
5. Lisa Ellison's salary is \$40,000. Tax is $(\$40,000 - \$25,000) * 10\% = \$1,500$, displayed as a result table.

PARTICIPATION ACTIVITY

42.3.9: Stored functions.



- 1) Stored functions have no parameters.



- True
 False

- 2) RETURN and RETURNS are equivalent keywords.



- True
 False

- 3) The function below requires no additional keywords to execute.



```
CREATE FUNCTION Smallest(x INT,  
y INT)  
RETURNS INT  
BEGIN  
IF x < y THEN  
RETURN x;  
ELSE  
RETURN y;  
END IF;  
END;
```

©zyBooks 01/31/24 18:46 1939727
Rob Daglio
MDCCOP2335Spring2024

- True
 False



- 4) Stored functions may be executed from the MySQL command line.

- True
- False

- 5) The <parameter-declaration> has the same syntax in stored procedures and stored functions.

- True
- False

©zyBooks 01/31/24 18:46 1939727
Rob Daglio
MDCCOP2335Spring2024

Triggers

A **trigger** is like a stored procedure or a stored function, with two differences:

- Triggers have neither parameters nor a return value. Triggers read and write tables but do not communicate directly with a calling program.
- Triggers are not explicitly invoked by a **CALL** statement or within an expression. Instead triggers are associated with a specific table and execute whenever the table is changed.

Triggers are used to enforce business rules automatically as data is updated, inserted, and deleted.

The **CREATE TRIGGER** statement specifies a TriggerName followed by four required keywords:

- **ON TableName** identifies the table associated with the trigger.
- **INSERT, UPDATE, or DELETE** indicates that the trigger executes when the corresponding SQL operation is applied to the table.
- **BEFORE or AFTER** determines whether the trigger executes before or after the insert, update, or delete operation.
- **FOR EACH ROW** indicates the trigger executes repeatedly, once for each row affected by the insert, update, or delete operations.

MySQL triggers do not support all features of the SQL standard. Ex: The standard includes keywords **FOR EACH STATEMENT** as an alternative to **FOR EACH ROW**. **FOR EACH STATEMENT** indicates the trigger executes once only, rather than repeatedly for each affected row.

©zyBooks 01/31/24 18:46 1939727

Rob Daglio

MDCCOP2335Spring2024

Like stored procedures, the trigger body may be a simple or compound statement. Within the body, keywords **OLD** and **NEW** represent the name of the table associated with the trigger. **OLD** represents the table values prior to an update or delete operation. **NEW** represents the table values after an insert or update operation.

Figure 42.3.8: CREATE TRIGGER statement.

```
CREATE TRIGGER TriggerName
[ BEFORE | AFTER ] [ INSERT | UPDATE |
DELETE ]
ON TableName
FOR EACH ROW
Body;
```

©zyBooks 01/31/24 18:46 1939727
Rob Daglio
MDCCOP2335Spring2024

PARTICIPATION ACTIVITY

42.3.10: Trigger example.

```
CREATE TRIGGER ExamGrade
BEFORE INSERT
ON Exam
FOR EACH ROW
BEGIN

    IF NEW.Score >= 90 THEN
        SET NEW.Grade = 'A';
    ELSEIF NEW.Score >= 80 THEN
        SET NEW.Grade = 'B';
    ELSEIF NEW.Score >= 70 THEN
        SET NEW.Grade = 'C';
    ELSE
        SET NEW.Grade = 'F';
    END IF;

END;
```

```
mysql> INSERT INTO Exam (ID, Score)
-> VALUES (1, 79), (2, 92), (3, 85);
Query OK, 3 rows affected (0.00 sec)
Records: 3  Duplicates: 0  Warnings: 0

mysql> SELECT * FROM Exam;
+----+-----+-----+
| ID | Score | Grade |
+----+-----+-----+
| 1  |    79 | C    |
| 2  |    92 | A    |
| 3  |    85 | B    |
+----+-----+-----+
```

Animation content:

Static figure:

Begin SQL code:

```
CREATE TRIGGER ExamGrade BEFORE INSERT ON Exam
FOR EACH ROW
BEGIN
```

```
    IF NEW.Score >= 90 THEN SET NEW.Grade = 'A';
    ELSEIF NEW.Score >= 80 THEN SET NEW.Grade = 'B';
    ELSEIF NEW.Score >= 70 THEN SET NEW.Grade = 'C';
    ELSE SET NEW.Grade = 'F';
END IF;
```

©zyBooks 01/31/24 18:46 1939727
Rob Daglio
MDCCOP2335Spring2024

END;
End SQL code.

A console displays input and output:

```
mysql> INSERT INTO Exam (ID, Score) VALUES (1, 79), (2, 92), (3, 85);
```

Query OK, 3 rows affected (0.00 sec)

Records: 3 Duplicates: 0 Warnings: 0

```
mysql> SELECT * FROM Exam;
```

ID	Score	Grade
1	79	C
2	92	A
3	85	B

©zyBooks 01/31/24 18:46 1939727

Rob Daglio

MDCCOP2335Spring2024

Step 1: The ExamGrade trigger executes before inserts to the Exam table. The CREATE TRIGGER statement is highlighted.

Step 2: One insert statement can create multiple rows. The trigger executes once for each new row. The FOR statement is highlighted.

Step 3: The trigger automatically determines exam grade and updates the Grade column in new table rows. The code from BEGIN through END is highlighted.

Step 4: Inserting new rows causes the ExamGrade trigger to execute before the rows are inserted. The first prompt and INSERT statement appear on the console.

Step 5: The Grade column contains values set by the ExamGrade trigger. The second prompt, SELECT statement, and output appear on the console.

Animation captions:

1. The ExamGrade trigger executes before inserts to the Exam table.
2. One insert statement can create multiple rows. The trigger executes once for each new row.
3. The trigger automatically determines exam grade and updates the Grade column in new table rows.
4. Inserting new rows causes the ExamGrade trigger to execute before the rows are inserted.
5. The Grade column contains values set by the ExamGrade trigger.



Refer to the trigger in the animation above.

- 1) What happens when an exam score is updated from 65 to 83?

- Grade changes from C to B.
- Grade does not change.
- Grade is set to NULL.



- 2) NULLs are allowed in Score. What happens when a new row is inserted with a NULL Score?

- The insert fails and no row is created.
- The insert succeeds and Grade is set to the column default value.
- The insert succeeds and ExamGrade is set to 'F'

- 3) An INSERT is attempted, but the inserted primary key is not unique. Does the trigger execute?

- Yes, the trigger executes, and the INSERT succeeds.
- Yes, the trigger executes, but the INSERT fails.
- No, the trigger never executes.



CHALLENGE ACTIVITY

42.3.1: Procedural SQL.



539740.3879454.qx3zqy7

Start

©zyBooks 01/31/24 18:46 1939727
Rob Daglio
MDCCOP2335Spring2024

```
CREATE TABLE Seat (
    RoomName VARCHAR(40),
    SeatNumber INT,
```

PRIMARY KEY (RoomName, SeatNumber)

Fill in the keywords to create a stored procedure that adds entries to the Seat table.

```
____(A)____ PROCEDURE AddSeats(IN roomName VARCHAR(40), ____(B)____ howMa  
____(C)____  
    DECLARE n INT ____(D)____ 1;  
  
    WHILE n <= howMany DO  
        INSERT INTO Seat VALUES (roomName, n);  
        ____(E)____ n = n + 1;  
    END ____(F)____;  
END;
```

©zyBooks 01/31/24 18:46 1939727

Rob Daglio

MDCCOP2335Spring2024

(A) Ex: KEYWORD

(D)

(B)

(E)

(C)

(F)

1

2

Check

Next

Exploring further:

- [MySQL CREATE PROCEDURE and CREATE FUNCTION statements](#)
- [MySQL CALL statement](#)
- [MySQL compound statements and cursors](#)
- [MySQL CREATE TRIGGER statement](#)

©zyBooks 01/31/24 18:46 1939727

Rob Daglio

MDCCOP2335Spring2024

42.4 Application programming interfaces

Overview

An application programming interface (API) specifies the interaction between an application and a computer service, such as a database, email, or web service. If the application is written in a procedural language like C, the API specifies procedure calls. If the application is written in an object-oriented language like Java, the API specifies classes.

Hundreds of APIs are available, connecting major programming languages to commonly used services. Dozens of APIs are available for relational databases and other data sources. Each programming language supports different APIs for different services. Ex: Java supports JMS for message services, JSAPI for speech synthesis and recognition services, JavaMail for email services, and many others. Conversely, each service may have different APIs for different programming languages.

Leading database APIs include:

- **SQL/Call Language Interface (SQL/CLI)**, an extension of the SQL standard
- **ODBC** supports many programming languages and data sources. ODBC was developed in parallel with SQL/CLI and conforms closely to the standard.
- **JDBC** for Java applications
- **DB-API** for Python applications
- **ADO.NET** for .NET applications. .NET is a Microsoft environment, and C# is the most popular .NET language.
- **PDO** for PHP applications. PHP is a widely used programming language for building dynamic websites.

MySQL connectors implement leading APIs between the MySQL database and major programming languages.

Table 42.4.1: Leading database APIs.

Acronym	Derivation	Original Sponsor	Initial release	Application language	MySQL Connector
ODBC	Open Database Connectivity	Microsoft	1992	Numerous	Connector/ODBC
JDBC	Java Database Connectivity	Sun Microsystems (now Oracle)	1997	Java	Connector/J
DB-API	Database API	Python Software Foundation	1996	Python	Connector/Python

ADO.NET	ActiveX Data Objects	Microsoft	2002	.NET languages	Connector/.NET
PDO	PHP Data Objects	The PHP Group	2005	PHP	none

©zyBooks 01/31/24 18:46 1939727

Rob Daglio

MDCCOP2335Spring2024

Terminology

Use of the term **application programming interface**, or **API**, varies. The term always refers to the syntax and expected behavior of procedure or class declarations, as specified by a standards organization. In some cases, the term also refers to the implementation of the procedures or classes in complete code libraries.

PARTICIPATION ACTIVITY

42.4.1: Database APIs.



1) Which API supports only one language ?



- ODBC
- DB-API
- ADO.NET

2) Which API is often used with the C# language ?



- JDBC
- DB-API
- ADO.NET

3) Database APIs support:

©zyBooks 01/31/24 18:46 1939727
Rob Daglio
MDCCOP2335Spring2024



- Relational databases only
- Relational and non-relational databases only
- Databases and other data sources

Capabilities

At a high level, database APIs are similar and support common capabilities, including:

- *Manage connections.* A database connection identifies the database address and provides login credentials. A connection must be created and opened before queries are executed.
- *Prepare queries.* When a query is executed repeatedly, compiling once prior to execution is faster than compiling each time the query is executed. In database APIs, compiling prior to execution is often called 'preparing' a query.
- *Execute queries with single-row results.* Single-row queries are relatively easy to embed in API procedure calls. Procedure parameters transfer query results to program variables.
- *Execute queries with multiple-row results.* Queries that return multiple rows require special processing. A special API object, called a 'cursor' or 'reader', identifies a single row of the result table. The cursor advances through the result table as the application processes one row at a time.
- *Call stored procedures.* Stored procedures are executed with the API 'execute query' capability, like other SQL statements. Special API syntax matches stored procedure parameters with host language variables.

Specific syntax of the above capabilities varies greatly, depending on the API and host language. A JDBC code example appears below. A detailed description of DB-API for Python and MySQL appears elsewhere in this material.

PARTICIPATION ACTIVITY

42.4.2: JDBC example.



Procedural SQL

```
CREATE PROCEDURE FlightCount(IN airline VARCHAR(20), OUT quantity INT)
    SELECT COUNT(*)
    INTO quantity
    FROM Flight
    WHERE AirlineName = airline;
```

Java

```
import java.sql.*;
import java.util.Scanner;
public class JdbcDemo {
    public static void main(String[] args) throws SQLException {
        Connection reservation = DriverManager.getConnection(
            "jdbc:mysql://localhost/Reservation", "samsnead", "b98$xm");
        CallableStatement flightCall = reservation.prepareCall(
            "{ CALL FlightCount(?, ?) }");
        System.out.print("Enter airline name: ");
        Scanner scnr = new Scanner(System.in);
        String airlineName = scnr.nextLine();
```

©zyBooks 01/31/24 18:46 1939727
Rob Daglio
MDCCOP2335Spring2024

```

    flightCall.setString("Airline", airlineName);
    flightCall.registerOutParameter("Quantity", Types.INTEGER);
    flightCall.executeQuery();

    int total = flightCall.getInt("Quantity");
    System.out.println("Airline: " + airlineName + "\nTotal flights: " + total);

    reservation.close();
}
}

```

©zyBooks 01/31/24 18:46 1939727

Rob Daglio

MDCCOP2335Spring2024

Animation content:

Static figure:

SQL code has caption procedural SQL:

Begin SQL code:

```
CREATE PROCEDURE FlightCount(IN airline VARCHAR(20), OUT quantity INT)
```

```
    SELECT COUNT(*) INTO quantity FROM Flight WHERE AirlineName = airline;
```

End SQL code.

Java code has caption java:

Begin Java code:

```
import java.sql.*;
```

```
import java.util.Scanner;
```

```
public class JdbcDemo {
```

```
    public static void main(String[] args) throws SQLException {
```

```
        Connection reservation = DriverManager.getConnection("jdbc:mysql://localhost/Reservation",
"samsnead", "b98$xm");
```

```
        CallableStatement flightCall = reservation.prepareCall("{ CALL FlightCount(?, ?) }");
```

```
        System.out.print("Enter airline name: ");
```

```
        Scanner scnr = new Scanner(System.in);
```

```
        String airlineName = scnr.nextLine();
```

```
        flightCall.setString("Airline", airlineName);
```

```
        flightCall.registerOutParameter("Quantity", Types.INTEGER);
```

```
        flightCall.executeQuery();
```

```
        int total = flightCall.getInt("Quantity");
```

```
        System.out.println("Airline: " + airlineName + "\nTotal flights: " + total);
```

```
        reservation.close();
```

```
}
```

```
}
```

End Java code.

©zyBooks 01/31/24 18:46 1939727

Rob Daglio

MDCCOP2335Spring2024

Step 1: The FlightCount procedure is written in procedural SQL and stored in the Reservation database. Procedural SQL code appears.

Step 2: The java.sql namespace implements the JDBC API. Java code appears. The import

statement is highlighted.

Step 3: The reservation object opens a database connection. close() releases the connection. The Connection and reservation.close statements are highlighted.

Step 4: prepareCall() compiles CALL FlightCount and saves the statement in the flightCall object. ? characters are placeholders for parameters. The CallableStatement statement is highlighted.

©zyBooks 01/31/24 18:46 1939727

Rob Daglio

Step 5: Java code inputs an airline name to the Java variable airlineName. The System.out.print, Scanner, and String statements are highlighted. United Airlines appears next to airlineName in the String statement.

Step 6: The airline parameter is set to airlineName, the quantity parameter is registered as an integer, and the query is executed. The three flightCall statements are highlighted. United Airlines appears under IN airline In the CREATE PROCEDURE statement. 102 appears under OUT quantity In the CREATE PROCEDURE statement.

Step 7: The Java variable total is assigned the Quantity value and displayed. The int total and System.out.println statements are highlighted. 102 appears next to total in the System.out.println statement.

Animation captions:

1. The FlightCount procedure is written in procedural SQL and stored in the Reservation database.
2. The java.sql namespace implements the JDBC API.
3. The reservation object opens a database connection. close() releases the connection.
4. prepareCall() compiles CALL FlightCount and saves the statement in the flightCall object. ? characters are placeholders for parameters.
5. Java code inputs an airline name to the Java variable airlineName.
6. The airline parameter is set to airlineName, the quantity parameter is registered as an integer, and the query is executed.
7. The Java variable total is assigned the Quantity value and displayed.

PARTICIPATION ACTIVITY

42.4.3: JDBC API capabilities.

©zyBooks 01/31/24 18:46 1939727

Rob Daglio

MDCCOP2335Spring2024

Refer to the animation above. Match the Java language element with the description.

If unable to drag and drop, refresh the page.

getInt()

import java.sql.*;

prepareCall()

reservation

flightCall

	Makes JDBC classes available to the Java program.
	Identifies the database location and provides database access credentials.
	Contains an SQL stored procedure call.
	Compiles an SQL statement and associates the statement with a Java object.
	Returns the value of a stored procedure output parameter.

Reset

Architecture

A database API is implemented in source code by API developers. The compiled code is distributed to application developers and linked to API calls when applications are compiled.

Database API implementations typically contain two software layers:

- A **driver** connects directly to the database. Although all relational databases support standard SQL, implementations vary somewhat, and most databases offer non-standard extensions. Consequently, a different driver is necessary for each database.
- The **driver manager** connects the application to the drivers. The application communicates with the driver manager using API procedure calls. The driver manager forwards each call to the correct driver. When one application connects to several databases, the driver manager selects a driver based on the active connection.

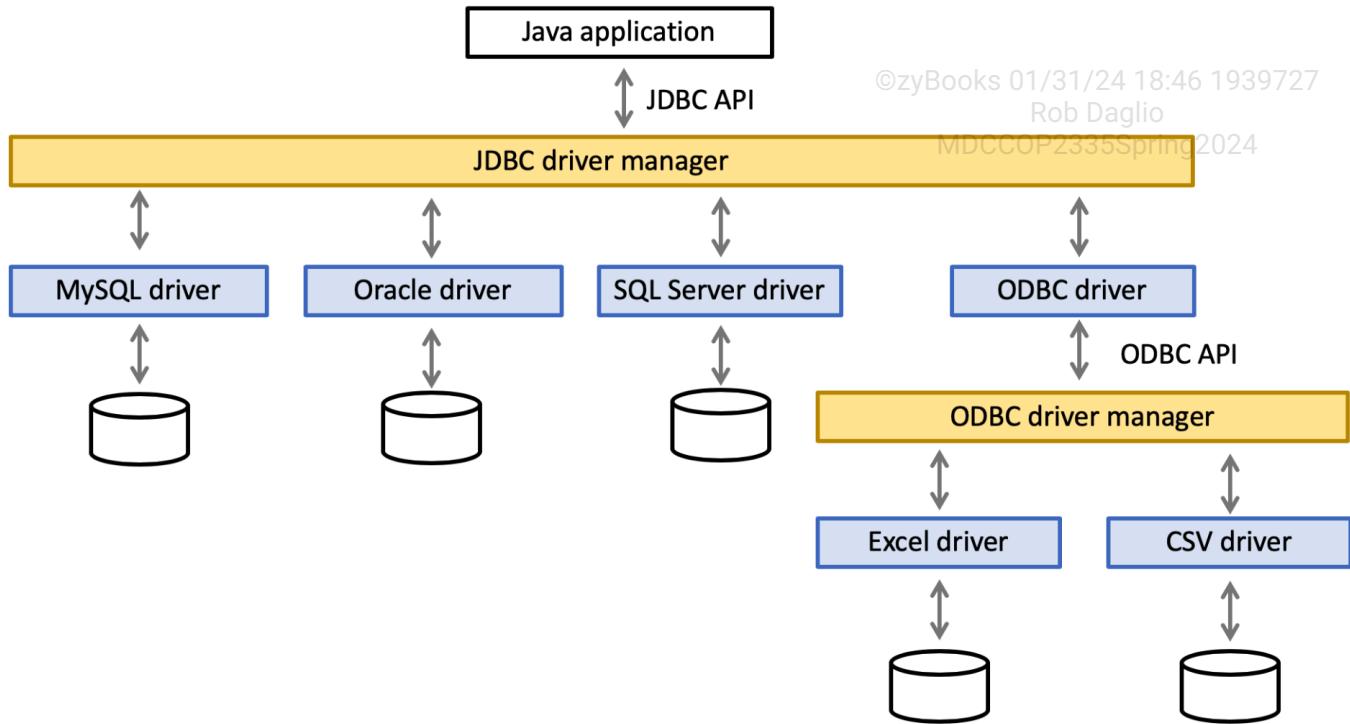
Many database APIs support non-relational data sources such as Excel spreadsheets (.xlsx files) and comma-separated data (.csv files). Each supported data source requires a different driver. Since many data sources do not support SQL capabilities such as transaction management, drivers to non-relational sources may not support the full API.

A primary API may support a secondary API. Ex: A JDBC implementation may include an ODBC driver, enabling access to any ODBC data source from JDBC. Supporting a secondary API is useful when a data source is not supported by the primary API. The secondary API introduces additional layers between application and data source, however, and therefore is inefficient.

PARTICIPATION ACTIVITY



42.4.4: API architecture.



Animation content:

Static figure:

The diagram consists of six horizontal layers. The top layer contains a white rectangle named Java Application. The next layer down contains a wide yellow rectangle named JDBC driver manager. The next layer contains four blue rectangles named MySQL driver, Oracle driver, SQL server driver, and ODBC driver. The MySQL driver, Oracle driver, and SQL server driver are above disk drive icons. The ODBC driver is above a yellow rectangle named ODBC driver manager. The ODBC driver manager is above blue rectangles named Excel driver and CSV driver, which are above disk drive icons. Vertical double-headed arrows connect each rectangle with the rectangles or disk drives directly below.

Animation captions:

©zyBooks 01/31/24 18:46 1939727
Rob Daglio
MDCCOP2335Spring2024

1. Java applications communicate via the JDBC API with the JDBC driver manager.
2. The driver manager communicates with each data source via a different driver.
3. If an API has no driver for a data source, the API may communicate indirectly via another API.
4. The ODBC driver for JDBC communicates with Excel and CSV data sources via the ODBC driver manager.

**PARTICIPATION
ACTIVITY**

42.4.5: Architecture.



- 1) Each database requires a different driver manager.

True
 False

©zyBooks 01/31/24 18:46 1939727

Rob Daglio

MDCCOP2335Spring2024



- 2) The driver manager communicates with all drivers using the same commands.

True
 False

- 3) Every driver implements the full capabilities of the API.

True
 False



- 4) In the animation above, the secondary API is ODBC.

True
 False



- 5) A secondary API can be layered on top of a third API, for a total of three API layers.

True
 False



Exploring further:

- [MySQL connectors](#)

©zyBooks 01/31/24 18:46 1939727

Rob Daglio

MDCCOP2335Spring2024

42.5 Database programming with Python

Connections

This section describes database programming in Python and MySQL using Connector/Python.

Connector/Python is a MySQL component based on the DB-API standard from the Python Software Foundation. This section assumes familiarity with the Python language.

Installing Connector/Python

©zyBooks 01/31/24 18:46 1939727

Rob Daglio

MDCCOP2335Spring2024

The Connector/Python package must be installed on the computer running the Python program. Connector/Python can be downloaded from mysql.com and installed by running an installation program. Connector/Python can also be installed using pip from the Windows or Mac command line:

```
pip install mysql-connector-python
```

A Python program uses Connector/Python by importing the `mysql.connector` module. A **module** is a file containing Python classes, functions, or variables.

Python programs must connect to a database prior to executing queries. A connection is an object of the **MySQLConnection** class that is created with the `mysql.connector.connect()` function, specifying the database server address, database name, login username, and password.

Creating a connection fails if the database is not found or login credentials are invalid. Therefore, connections are usually created within the `try` block of a `try-except` statement. If the connection fails, the `except` block executes and typically prints an error message.

The Python program releases the connection when no longer needed with the `connection.close()` method.

PARTICIPATION ACTIVITY

42.5.1: Creating a connection.



```
import mysql.connector
from mysql.connector import errorcode

try:
    reservationConnection = mysql.connector.connect(
        user='samsnead',
        password='*jksi72$',
        host='127.0.0.1',
        database='Reservation')

except mysql.connector.Error as err:
    if err.errno == errorcode.ER_ACCESS_DENIED_ERROR:
```

©zyBooks 01/31/24 18:46 1939727

Rob Daglio

MDCCOP2335Spring2024

```

        print('Invalid credentials')
    elif err(errno == errorcode.ER_BAD_DB_ERROR):
        print('Database not found')
    else:
        print('Cannot connect to database:', err)

else:
    # Execute database operations...
    reservationConnection.close()

```

©zyBooks 01/31/24 18:46 1939727

Rob Daglio

MDCCOP2335Spring2024

Animation content:

Static figure:

Begin Python code:

```
import mysql.connector
from mysql.connector import errorcode
try:
```

```
    reservationConnection = mysql.connector.connect(
        user='samsnead',
        password='*jksi72$',
        host='127.0.0.1',
        database='Reservation')
```

except mysql.connector.Error as err:

```
    if err(errno == errorcode.ER_ACCESS_DENIED_ERROR):
        print('Invalid credentials')
```

```
    elif err(errno == errorcode.ER_BAD_DB_ERROR):
        print('Database not found')
```

```
    else:
```

```
        print('Cannot connect to database:', err)
```

```
else:
```

```
    # Execute database operations . . .
```

```
    reservationConnection.close()
```

End Python code.

Animation captions:

1. The import statements make Connector/Python code and errorcode module available to the program.
2. The mysql.connector.connect() function creates a MySQLConnection object named reservationConnection.
3. If the connection fails, the except block executes and prints an error message.
4. If the connection succeeds, the else block executes. Subsequent queries use the reservationConnection object.
5. When database processing is finished, close() releases the connection.

©zyBooks 01/31/24 18:46 1939727

Rob Daglio

MDCCOP2335Spring2024

PARTICIPATION ACTIVITY

42.5.2: Python connections.



- 1) A connection must always be created within the try clause of a try-except-else statement.

- True
- False

©zyBooks 01/31/24 18:46 1939727

Rob Daglio

MDCCOP2335Spring2024

- 2) In the animation above, MySQL server is running on a machine with address 127.0.0.1.

- True
- False



- 3) In the animation above, the close() method is called if the password is incorrect.

- True
- False



- 4) A connection must be created prior to executing any database operation.

- True
- False



Cursors

A cursor is an object of the **MySQLCursor** class that executes SQL statements and stores the results. Cursors are created with the **connection.cursor()** method and therefore are always associated with a specific database.

SQL statements are compiled (translated to low-level database operations) and executed with the **cursor.execute()** method. The `cursor.execute()` method has one required parameter, a string containing the SQL statement.

©zyBooks 01/31/24 18:46 1939727
Rob Daglio
MDCCOP2335Spring2024

A cursor is always associated with one connection but may be reused for multiple SQL statements. Cursors must be released with the **cursor.close()** method prior to closing the associated connection.

When using Connector/Python with MySQL and the InnoDB storage engine, the results of INSERT, UPDATE, and DELETE statements are not automatically saved in the database. The **connection.commit()** method saves all changes. The **connection.rollback()** method discards all

changes. The `commit()` and `rollback()` methods must appear prior to releasing the cursor used to execute the changes.

`MySQLCursor` contains additional methods and properties, such as:

- The `cursor.rowcount` property is the number of rows returned or altered by a query.
- The `cursor.column_names` property is a list of column names in a query result.
- The `cursor.fetchwarnings()` method returns a list of warnings generated by a query.

©zyBooks 01/31/24 18:46 1939727

MDCCOP2335Spring2024

PARTICIPATION ACTIVITY

42.5.3: Using cursors.



```

flightCursor = reservationConnection.cursor()

flightCreate = ('CREATE TABLE Flight (
    FlightNumber INT PRIMARY KEY NOT NULL,
    AirlineName VARCHAR(20),
    AirportCode CHAR(3) )')

flightCursor.execute(flightCreate)

flightQuery = ("INSERT INTO Flight "
    "(FlightNumber, AirlineName, AirportCode) "
    "VALUES (280, 'China Airlines', 'PEK')")

flightCursor.execute(flightQuery)

reservationConnection.commit()
flightCursor.close()

```

Animation content:

Static figure:

Begin Python code:

```

flightCursor = reservationConnection.cursor()
flightCreate = ('CREATE TABLE Flight (
    FlightNumber INT PRIMARY KEY NOT NULL,
    AirlineName VARCHAR(20),
    AirportCode CHAR(3) )')
flightCursor.execute(flightCreate)
flightQuery = ("INSERT INTO Flight "
    "(FlightNumber, AirlineName, AirportCode) "
    "VALUES (280, 'China Airlines', 'PEK')")
flightCursor.execute(flightQuery)
reservationConnection.commit()

```

©zyBooks 01/31/24 18:46 1939727

Rob Daglio

MDCCOP2335Spring2024

```
flightCursor.close()  
End Python code.
```

Animation captions:

1. cursor() creates a cursor called flightCursor, associated with the Reservation database.
2. execute() creates the Flight table.
3. The next execute() inserts a row into the Flight table.
4. commit() saves the insert in the database before the cursor is released.

©zyBooks 01/31/24 18:46 1939727

Rob Daglio

MDCCOP2335Spring2024

**PARTICIPATION
ACTIVITY**

42.5.4: Cursors.



1) Which class contains the cursor() method?

- MySQLConnection
- MySQLCursor
- reservationConnection



2) The execute() method:

- Executes an SQL statement that must be previously compiled.
- Compiles but does not execute an SQL statement
- Both compiles and executes an SQL statement.



3) Which MySQLCursor method should always be called prior to closing the cursor's connection?

- close()
- commit()
- rollback()
- Either commit() or rollback()



©zyBooks 01/31/24 18:46 1939727

Rob Daglio

MDCCOP2335Spring2024



4) If the INSERT in the animation above successfully inserts a new flight, what is `flightCursor.rowcount` after the INSERT executes?

- 0
- 1
- 1

©zyBooks 01/31/24 18:46 1939727
Rob Daglio
MDCCOP2335Spring2024

Query parameters

The `cursor.execute()` method can substitute Python values into the given SQL statement using query parameters:

- The SQL statement uses placeholder characters `%s` to represent the query values.
- A tuple containing values to be assigned to the placeholders is passed as the second argument to `cursor.execute()`.

The tuple values are mapped to the `%s` placeholders, in sequence, when the SQL statement executes. Python data types are automatically converted to MySQL data types.

A database programmer must be cautious when inserting input data into an SQL statement. An **SQL injection attack** is when a user intentionally enters values that alter the intent of an SQL statement. The `cursor.execute()` method prevents SQL injection when assigning values to placeholders.

PARTICIPATION ACTIVITY

42.5.5: Using query parameters to insert a new flight.



```
print('Flight number? ', end = ' ')
flightNum = input()
print('Airline? ', end = ' ')
airline = input()
print('Airport code? ', end = ' ')
airportCode = input()

flightCursor = reservationConnection.cursor()
flightQuery = ('INSERT INTO Flight '
              '(FlightNumber, AirlineName, AirportCode)
              VALUES (%s, %s, %s)')
flightData = (flightNum, airline, airportCode)
flightCursor.execute(flightQuery, flightData)

reservationConnection.commit()
flightCursor.close()
```

©zyBooks 01/31/24 18:46 1939727
Rob Daglio
MDCCOP2335Spring2024

Flight number? 105

Airline? Air India
Airport code? DEL

Animation content:

Static figure:

Begin Python code:

```
print('Flight number? ', end = "")  
flightNum = input()  
print('Airline? ', end = "")  
airline = input()  
print('Airport code? ', end = "")  
airportCode = input()  
flightCursor = reservationConnection.cursor()  
flightQuery = ('INSERT INTO Flight '  
    '(FlightNumber, AirlineName, AirportCode) '  
    'VALUES (%s, %s, %s)')  
flightData = (flightNum, airline, airportCode)  
flightCursor.execute(flightQuery, flightData)  
reservationConnection.commit()  
flightCursor.close()  
End Python code.
```

An arrow points from each parameter in:

```
flightData = (flightNum, airline, airportCode)  
to the corresponding %s in:  
'VALUES (%s, %s, %s)'
```

A console displays code input and output:

Flight number? 105

Airline? Air India

Airport code? DEL

Animation captions:

©zyBooks 01/31/24 18:46 1939727

Rob Daglio

MDCCOP2335Spring2024

1. The user enters flight data.
2. flightQuery contains three %s placeholders. Values are assigned when the cursor executes.
3. flightData tuple contains three values.
4. When the cursor executes, flightData values are assigned to placeholders. Python data types are automatically converted to MySQL data types.
5. commit() saves the insert in the database, and close() releases the cursor resources.

PARTICIPATION ACTIVITY**42.5.6: Query parameters.**

- 1) Which line assigns the correct tuple for the UPDATE statement?

```
flightQuery = ('UPDATE Flight '
               'SET AirlineName =
%s '
               'WHERE
FlightNumber = %s')
flightCursor.execute(flightQuery,
flightData)
```

©zyBooks 01/31/24 18:46 1939727
Rob Daglio
MDCCOP2335Spring2024

- flightData = ('United Airlines', 'PEK')
- flightData = ('United Airlines', 'PEK', 105)
- flightData = ('United Airlines', 105)

- 2) What does the code output?

```
try:
    flightQuery = ('UPDATE Flight
'
               'SET
AirlineName = %s '
               'WHERE
FlightNumber = %s')
    flightData = (301, 'United
Airlines')

flightCursor.execute(flightQuery,
flightData)
    print('Flight updated.')
except
mysql.connector.errors.DataError:
    print('Unable to update
flight.')
```

- Flight updated.
- Unable to update flight.
- Nothing is output.

©zyBooks 01/31/24 18:46 1939727
Rob Daglio
MDCCOP2335Spring2024



- 3) Assume the airline and flightNum values are obtained from user input. What is wrong with the following code?

```
flightQuery = ('UPDATE Flight SET
AirlineName = "' + airline +
    '" WHERE
FlightNumber = ' + flightNum)
flightCursor.execute(flightQuery)
```

- execute() is missing a second argument.
- The code is susceptible to an SQL injection attack.
- Nothing is wrong.

©zyBooks 01/31/24 18:46 1939727

Rob Daglio

MDCCOP2335Spring2024

Fetching values

When cursor.execute() executes a SELECT statement, the query results are accessed with fetch methods:

- **cursor.fetchone()** returns a tuple containing a single result row or the value `None` if no rows are selected. If a query returns multiple rows, cursor.fetchone() may be executed repeatedly until it returns `None`.
- **cursor.fetchall()** returns a list of tuples containing all result rows. The tuple list can be processed in a loop. Ex: `for rowTuple in cursor.fetchall()` assigns each row to rowTuple and terminates when all rows are processed.

PARTICIPATION ACTIVITY

42.5.7: Fetching flight results.



```
flightCursor = reservationConnection.cursor()
flightQuery = ('SELECT FlightNumber, DepartureTime FROM Flight '
    'WHERE AirportCode = %s AND AirlineName = %s')
flightData = ('PEK', 'China Airlines')
flightCursor.execute(flightQuery, flightData)
```

©zyBooks 01/31/24 18:46 1939727

Rob Daglio

MDCCOP2335Spring2024

```
for row in flightCursor.fetchall():
    print('Flight', row[0], 'departs at', row[1])
flightCursor.close()
```

Flight 107 departs at 13:30:00
 Flight 350 departs at 8:15:00
 Flight 482 departs at 10:07:00

Animation content:

Static figure:

Begin Python code:

```
flightCursor = reservationConnection.cursor()  
flightQuery = ('SELECT FlightNumber, DepartureTime FROM Flight '  
    'WHERE AirportCode = %s AND AirlineName = %s')  
flightData = ('PEK', 'China Airlines')  
flightCursor.execute(flightQuery, flightData)  
for row in flightCursor.fetchall():  
    print('Flight', row[0], 'departs at', row[1])  
flightCursor.close()  
End Python code.
```

©zyBooks 01/31/24 18:46 1939727

Rob Daglio

MDCCOP2335Spring2024

A console displays program output:

Flight 107 departs at 13:30:00

Flight 350 departs at 8:15:00

Flight 482 departs at 10:07:00

Animation captions:

1. `flightCursor` executes a `SELECT` query.
2. `fetchall()` returns a set of tuples. Each tuple is a row of the result table.
3. The `for` statement processes each tuple in the set. `row[0]` contains data from the first column, and `row[1]` from the second column.

PARTICIPATION
ACTIVITY

42.5.8: Fetching values.



Refer to the code below.

©zyBooks 01/31/24 18:46 1939727

Rob Daglio

MDCCOP2335Spring2024

```
flightQuery = ('SELECT AirlineName, DepartureTime, AirportCode FROM Flight  
    'WHERE FlightNumber = %s')  
flightData = (140,) # Comma required for single value tuple  
flightCursor.execute(flightQuery, flightData)  
  
row = flightCursor.__A__( )  
if row is __B__:  
    print("Flight not found")  
else:  
    print('Flight departs from', __C__ )  
  
flightCursor.close()
```

©zyBooks 01/31/24 18:46 1939727
Rob Daglio
MDCCOP2335Spring2024

1) What is identifier A?

Check**Show answer**

2) What is identifier B?

Check**Show answer**

3) What is identifier C?

Check**Show answer**

Stored procedure calls

Stored procedures are part of MySQL procedural SQL. Stored procedures are saved in the database and often called from general-purpose languages like Python.

Stored procedures are called with the **cursor.callproc()** method, which has two parameters:

- The name of the stored procedure
- An input tuple containing a value for each stored procedure parameter

©zyBooks 01/31/24 18:46 1939727
Rob Daglio
MDCCOP2335Spring2024

The method returns an output tuple, also containing one value for each stored procedure parameter.

Stored procedure parameters are designated IN, OUT, or INOUT, indicating the parameter is for input, output, or both. For an OUT parameter, the corresponding value of the input tuple is ignored. For an IN parameter, the corresponding value of the output tuple is the input value.



Procedural SQL

```
CREATE PROCEDURE FlightCount(IN airline VARCHAR(20), OUT quantity INT)
    SELECT COUNT(*)
    INTO quantity
    FROM Flight
    WHERE AirlineName = airline;
```

©zyBooks 01/31/24 18:46 1939727



Python

```
flightCursor = reservationConnection.cursor()
flightData = ('China Airlines', 0)
result = flightCursor.callproc('FlightCount', flightData)
print(result[1])
flightCursor.close()
```

23

Animation content:

Static figure:

An SQL statement has caption procedural SQL:

Begin SQL code:

```
CREATE PROCEDURE FlightCount(IN airline VARCHAR(20), OUT quantity INT)
    SELECT COUNT(*) INTO quantity
    FROM Flight
    WHERE AirlineName = airline;
```

End SQL code.

Begin Python code:

```
flightCursor = reservationConnection.cursor()
flightData = ('China Airlines', 0)
result = flightCursor.callproc('FlightCount', flightData)
print(result[1])
flightCursor.close()
```

End Python code.

A memory diagram has four cells. One value appears in each cell:

©zyBooks 01/31/24 18:46 1939727

China Airlines

Rob Daglio

MDCCOP2335Spring2024

0

China Airlines

23

The first two cells have caption flightData. The last two cells have caption result.

A console displays program output:

23

Step 1: FlightCount counts the number of flights for an airline. The SQL code appears.

Step 2: flightData contains an airline name and a 0 placeholder for the quantity parameter. The Python code appears. The second Python statement is highlighted:

```
flightData = ('China Airlines', 0)
```

The memory diagram appears. Values China Airlines and 0 appear in the first two cells, with caption
©zyBooks 01/31/24 18:46 193972
Rob Daglio
MDCCOP2335Spring2024

Step 3: callproc() calls FlightCount with the input tuple flightData and returns the output tuple to result. The third Python statement is highlighted:

```
result = flightCursor.callproc('FlightCount', flightData)
```

The stored procedure executes and counts 23 rows in the Flight table for China Airlines. Values China Airlines and 23 appear in the last two cells, with caption result.

Step 4: result[1] is the second entry of the result tuple. The fourth Python statement is highlighted:

```
print(result[1])
```

23 appears on the console.

Animation captions:

1. FlightCount counts the number of flights for an airline.
2. flightData contains an airline name and a 0 placeholder for the quantity parameter.
3. callproc() calls FlightCount with the input tuple flightData and returns the output tuple to result.

PARTICIPATION ACTIVITY

42.5.10: Stored procedural calls.

Refer to the animation above. Match the language element with the description.

If unable to drag and drop, refresh the page.

cursor()

result

callproc()

flightData

quantity

©zyBooks 01/31/24 18:46 193972

Rob Daglio

MDCCOP2335Spring2024

A Python output tuple

A stored procedure parameter

A Python input tuple

A method of the MySQLCursor class

A method of the MySQLConnection class

Reset**CHALLENGE ACTIVITY**

42.5.1: Database programming with Python.

©zyBooks 01/31/24 18:46 1939727
Rob Daglio
MDCCOP2335Spring2024

539740.3879454.qx3zqy7

Start

The Python code below prints out the names and capacities of all the rooms in the database. Fill in the missing keywords.

```
from mysql import connector # load connector functions
connection = connector._____ (A)_____
    user='Blaise',
    password='Pascal',
    host='localhost',
    database='college'
)
rowcursor = connection._____ (B)_____
query = 'SELECT * FROM Room'
_____ (C)_____.execute(_____ (D)_____ )
for row in rowcursor._____ (E)_____: print(row)
rowcursor._____ (F)_____
connection._____ (G)_____ ()
```

- (A) Ex: keyword (D) (G)
(B) (E)
(C) (F)

1

©zyBooks 01/31/24 18:46 1939727
Rob Daglio
MDCCOP2335Spring2024**Check****Try again**

Exploring further:

- [Download MySQL Connector/Python](#)
- [MySQL Connector/Python Developer Guide](#)
- [Python DB-API standard](#)

©zyBooks 01/31/24 18:46 1939727

Rob Daglio
MDCCOP2335Spring2024

42.6 Database programming with Java

Connections

This section describes database programming in Java and MySQL using Connector/J. Connector/J is the official JDBC (Java Database Connectivity) driver for MySQL. This section assumes familiarity with the Java language.

Installing Connector/J

The Connector/J package must be installed on the computer running the Java program. Instructions for downloading and installing Connector/J are available at [Connector/J Installation](#).

A Java program connects to the MySQL database by importing and using the classes `java.sql.Connection` and `java.sql.DriverManager`.

Java programs must connect to a database prior to executing queries. A connection is an object of the **Connection** interface that is created with the **DriverManager.getConnection()** method, specifying the database server address, database name, login username, and password.

Creating a connection fails if the database is not found or login credentials are invalid. Therefore, connections are usually created within the **try** block of a **try-catch** statement. If the connection fails, the **catch** block executes and typically prints an error message.

The Java program releases the connection when no longer needed with the **connection.close()** method.

PARTICIPATION ACTIVITY

42.6.1: Creating a connection.



```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;

public class Reservation {

    public static void main(String[] args) {
        Connection reservationConn = null;
        try {
            reservationConn = DriverManager.getConnection(
                "jdbc:mysql://127.0.0.1/reservation?"
                + "user=samsnead&password=*jksi72$");
        } catch (SQLException e) {
            System.out.println("Cannot connect to database:"
                + e.getMessage());
        }
    }
}
```

MDCCOP2335Spring2024

Rob Daglio

01/31/24 18:46 1939727

Animation content:

Static figure:

Begin Java code:

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
public class Reservation {
    public static void main(String[] args) {
        Connection reservationConn = null;
        try {
            reservationConn = DriverManager.getConnection("jdbc:mysql://127.0.0.1/reservation?" +
"user=samsnead&password=*jksi72$");
            // Execute database operations...
            reservationConn.close();
        }
        catch (SQLException e) {
            System.out.println("Cannot connect to database:" + e.getMessage());
        }
    }
}
```

MDCCOP2335Spring2024

Rob Daglio

01/31/24 18:46 1939727

End Java code.

Animation captions:

1. The import statements make Connector/J code available to the program.
2. The `DriverManager.getConnection()` method creates a `Connection` object named `reservationConn`.
3. If the connection succeeds, subsequent queries use the `reservationConn` object. When the database processing is finished, `close()` releases the connection.
4. If the connection fails, the catch block executes and prints an error message from the `SQLException` object.

©zyBooks 01/31/24 18:46 1939727

Rob Daglio

MDCCOP2335Spring2024

PARTICIPATION ACTIVITY

42.6.2: Java connections.



- 1) A connection must always be created within the try clause of a try-catch statement.

True
 False

- 2) In the animation above, MySQL server is running on a machine with address 127.0.0.1.

True
 False

- 3) In the animation above, the `close()` method is called if the password is incorrect.

True
 False

- 4) A connection must be created prior to executing any database operation.

True
 False



©zyBooks 01/31/24 18:46 1939727

Rob Daglio

MDCCOP2335Spring2024

Executing queries

The **Statement** interface defines and executes SQL queries. Statement objects are created with the Connection interface **`createStatement()`** method. A statement object is associated with the same connection, and therefore the same database, at all times. However, a statement object may execute multiple times with different queries.

SQL queries are defined and executed with Statement methods:

- **executeQuery()** associates a statement object with a SELECT query, compiles the query to low-level database operations and executes the query. executeQuery() has one required parameter, a string containing the SQL query, and returns a ResultSet object, described below.
- **executeUpdate()** executes INSERT, UPDATE, and DELETE queries. executeUpdate() returns the number of rows matched by the query, not the number of rows that were modified.
- **execute()** executes any SQL statement but is commonly used for data definition statements such as CREATE TABLE. execute() returns a boolean value. If the value is true, the statement has generated a ResultSet object which can be accessed with the Statement getResultSet() method.
- **close()** releases statement objects and must be executed prior to closing the associated connection.

The Statement interface must be imported to a Java program with

```
import java.sql.Statement;
```

PARTICIPATION ACTIVITY

42.6.3: Using cursors.



```
Statement flightStatement = reservationConn.createStatement();

String flightCreate = "CREATE TABLE Flight (
    + "FlightNumber INT PRIMARY KEY NOT NULL,
    + "AirlineName VARCHAR(20),
    + "AirportCode CHAR(3))";

flightStatement.execute(flightCreate);

String flightInsert = "INSERT INTO Flight "
    + "(FlightNumber, AirlineName, AirportCode) "
    + "VALUES (280, 'China Airlines', 'PEK')";

flightStatement.executeUpdate(flightInsert);

flightStatement.close();
```

Animation content:

©zyBooks 01/31/24 18:46 1939727

Rob Daglio

MDCCOP2335Spring2024

Static figure:

Begin Java code:

```
Statement flightStatement = reservationConn.createStatement();
```

```
String flightCreate = "CREATE TABLE Flight (FlightNumber INT PRIMARY KEY NOT NULL,
AirlineName VARCHAR(20), AirportCode CHAR(3));
```

```
flightStatement.execute(flightCreate);
```

```
String flightInsert = "INSERT INTO Flight(FlightNumber, AirlineName, AirportCode) VALUES (280,
```

```
'China Airlines', 'PEK');");
flightStatement.executeUpdate(flightInsert);
flightStatement.close();
End Java code.
```

Animation captions:

1. createStatement() creates a statement called flightStatement, associated with the Reservation database.
2. execute() executes the CREATE TABLE statement, creating the Flight table.
3. executeUpdate() executes the INSERT statement, inserting a row into Flight.
4. close() releases the statement resources.

©zyBooks 01/31/24 18:46 1939727

Rob Daglio

MDCCOP2335Spring2024

PARTICIPATION ACTIVITY

42.6.4: Statement and ResultSet.



1) Which interface contains the createStatement() method?

- Connection
- MySQLConnection
- reservationConn



2) The executeQuery() method:

- Executes an SQL statement that must be previously compiled.
- Compiles but does not execute an SQL statement
- Both compiles and executes an SQL statement.



Fetching values

Connector/J does not support cursors. Instead, query results are retrieved with the **ResultSet** interface.

©zyBooks 01/31/24 18:46 1939727

Rob Daglio

MDCCOP2335Spring2024

The executeQuery() method returns a ResultSet object containing query results. Each ResultSet object maintains a pointer to the **current row** of the query result. Initially, the current row is the first row of the query result. The **next()** method advances the current row to the next row. next() returns true if the current row advances, or false if the current row is at the last row of the query result.

Values in the current row are accessed with ResultSet methods:

- **getInt()** takes a column name parameter and returns the integer value of the column.

- **getDouble()** takes a column name parameter and returns the Double value of the column.
- **getString()** takes a column name parameter and returns the String value of the column.

Other ResultSet methods return additional information:

- **getMetaData()** returns a ResultSetMetaData object, containing information about the columns of a ResultSet object.
- **getWarnings()** returns the first warning generated by a query.

©zyBooks 01/31/24 18:46 1939727

Rob Daglio

MDCCOP2335Spring2024

The Result interface must be imported to a Java program with `import java.sql.ResultSet;`.

PARTICIPATION
ACTIVITY

42.6.5: Querying flight results.



```
Statement flightStatement = reservationConn.createStatement();

String flightQuery = "SELECT FlightNumber, DepartureTime FROM "
    + "Flight WHERE AirportCode='PEK' AND AirlineName='China Airlines'";
ResultSet resultSet = flightStatement.executeQuery(flightQuery);

while (resultSet.next()) {
    System.out.println("Flight " + resultSet.getString("FlightNumber")
        + " departs at " + resultSet.getString("DepartureTime"));
}

resultSet.close();
```

Flight 107 departs at 13:30:00
 Flight 350 departs at 8:15:00
 Flight 482 departs at 10:07:00

Animation content:

Static figure:

©zyBooks 01/31/24 18:46 1939727

Rob Daglio

MDCCOP2335Spring2024

Begin Java code:

```
Statement flightStatement = reservationConn.createStatement();
```

```
String flightQuery = "SELECT FlightNumber, DepartureTime FROM Flight WHERE AirportCode='PEK' AND AirlineName='China Airlines'";
```

```
ResultSet resultSet = flightStatement.executeQuery(flightQuery);
```

```
while (resultSet.next()) {
```

```
    System.out.println("Flight " + resultSet.getString("FlightNumber") + " departs at " +
```

```
resultSet.getString("DepartureTime"));
}
resultSet.close();
End Java code.
```

A console displays program output:

```
Flight 107 departs at 13:30:00
Flight 350 departs at 8:15:00
Flight 482 departs at 10:07:00
```

©zyBooks 01/31/24 18:46 1939727
Rob Daglio
MDCCOP2335Spring2024

Animation captions:

1. The flightStatement object executes a SELECT query and returns a ResultSet object.
2. The while statement advances through each row of the query results by calling the next() method.
3. For each resultSet row, the FlightNumber and DepartureTime columns are retrieved and printed.
4. close() releases the resultSet resources.

PARTICIPATION ACTIVITY

42.6.6: Fetching values.



Refer to the code below.

```
Statement flightStatement = reservationConn.createStatement();
String flightQuery = "SELECT AirlineName, DepartureTime, AirportCode FROM
Flight "
+ "WHERE FlightNumber = '140'";

ResultSet resultSet = flightStatement.__A__(flightQuery);
int rows = 0;
while(resultSet.__B__){
    System.out.println(resultSet.getString("AirlineName") +
        " departs from " + resultSet.getString("AirportCode"));
    rows++;
}
if (__C__){
    System.out.println("Flight not found");
}
resultSet.close();
```

©zyBooks 01/31/24 18:46 1939727
Rob Daglio
MDCCOP2335Spring2024

- 1) What is identifier A?

Check

Show answer



2) What is identifier B?

Check**Show answer**

3) What is identifier C?

Check**Show answer**

©zyBooks 01/31/24 18:46 1939727

Rob Daglio

MDCCOP2335Spring2024

Query parameters

The **PreparedStatement** interface extends the Statement interface and supports SQL query parameters.

PreparedStatement objects are created with the Connection interface **prepareStatement()** method. **prepareStatement()** takes an SQL query String parameter, unlike **createStatement()**, which takes no parameter.

PreparedStatement inherits the **executeQuery()** and **executeUpdate()** methods from Statement. Unlike the corresponding Statement methods, these PreparedStatement methods do not take a query parameter, since prepared statements are assigned a query when created.

PreparedStatement supports SQL query parameters:

- The SQL query contains one or more ? characters to represent query parameters.
- PreparedStatement methods such as **setString()** and **setInt()** assign values to placeholders. Ex: **preparedStatement.setString(1, flightNumber)** sets the first ? in the preparedStatement object to the String value of flightNumber.

A database programmer must be cautious when inserting input data into an SQL statement. An **SQL injection attack** is when a user intentionally enters values that alter the intent of an SQL statement.

The **preparedStatement.executeQuery()** method prevents SQL injection when assigning values to placeholders.

PARTICIPATION ACTIVITY

42.6.7: Using query parameters to insert a new flight.

©zyBooks 01/31/24 18:46 1939727

Rob Daglio

MDCCOP2335Spring2024



```
Scanner scnr = new Scanner(System.in);

System.out.println("Flight number? ");
int flightNum = scnr.nextInt();
System.out.println("Airline? ");
```

```

String airline = scnr.nextLine();
System.out.println("Airport code? ");
String airportCode = scnr.nextLine();

String flightInsert = "INSERT INTO Flight (FlightNumber"
    + ", AirlineName, AirportCode) VALUES (?, ?, ?)";

PreparedStatement flightPreparedStatement =
reservationConn.prepareStatement(flightInsert);

flightPreparedStatement.setInt(1, flightNum);
flightPreparedStatement.setString(2, airline);
flightPreparedStatement.setString(3, airportCode);

flightPreparedStatement.executeUpdate();

flightPreparedStatement.close();

```

Flight number? 105
Airline? Air India
Airport code? DEL

©zyBooks 01/31/24 18:46 1939727
Rob Daglio
MDCCOP2335Spring2024

Animation content:

Static figure:

Begin Java code:

```

Scanner scnr = new Scanner(System.in);
System.out.println("Flight number? ");
int flightNum = scnr.nextInt();
System.out.println("Airline? ");
String airline = scnr.nextLine();
System.out.println("Airport code? ");
String airportCode = scnr.nextLine();
String flightInsert = "INSERT INTO Flight (FlightNumber, AirlineName, AirportCode) VALUES (?, ?, ?)";
PreparedStatement flightPreparedStatement = reservationConn.prepareStatement(flightInsert);
flightPreparedStatement.setInt(1, flightNum);
flightPreparedStatement.setString(2, airline);
flightPreparedStatement.setString(3, airportCode);
flightPreparedStatement.executeUpdate();
flightPreparedStatement.close();
End Java code.

```

A console displays program input and output:

Flight number? 105

Airline? Air India

Airport code? DEL

©zyBooks 01/31/24 18:46 1939727
Rob Daglio
MDCCOP2335Spring2024

Animation captions:

1. The user enters flight data.

2. flightInsert contains three ? placeholders. Values are assigned when the prepared statement executes.
3. The PreparedStatement object is created with the SQL statement in the flightInsert string.
4. When executeUpdate() executes, Java values are assigned to placeholders. Java data types are automatically converted to MySQL data types.

PARTICIPATION
ACTIVITY

42.6.8: Query parameters.

©zyBooks 01/31/24 18:46 1939727

Rob Daglio

MDCCOP2335Spring2024



- 1) Which lines assign the correct values for the UPDATE query?

```
flightUpdate = "UPDATE Flight SET AirlineName =  
? "  
+ "WHERE flightNumber = ?";  
PreparedStatement flightPS =  
  
reservationConn.prepareStatement(flightUpdate);  
// Update values here  
flightPS.executeUpdate();  
  
 flightPS.setString(0, "United Airlines");  
flightPS.setInt(1, 105);  
  
 flightPS.setString("United Airlines");  
flightPS.setInt(105);  
  
 flightPS.setString(1, "United Airlines");  
flightPS.setInt(2, 105);
```



©zyBooks 01/31/24 18:46 1939727

Rob Daglio

MDCCOP2335Spring2024



2) What does the code output?

```

try {
    flightUpdate = "UPDATE Flight SET
AirlineName = ? "
    + "WHERE FlightNumber = ?";
    PreparedStatement flightPS =
reservationConn.prepareStatement(flightUpdate);
    flightPS.setInt(1, 301);
    flightPS.setString(2, "United Airlines");
    flightPS.executeUpdate();
    System.out.println("Flight updated.");
}
catch (SQLException e) {
    System.out.println("Unable to update
flight.");
}

```

©zyBooks 01/31/24 18:46 1939727
Rob Daglio
MDCCOP2335Spring2024

- Flight updated.
- Unable to update flight.
- Nothing is output.

3) Assume the airline and flightNum values are obtained from user input. What is wrong with the following code?

```

flightUpdate = "UPDATE Flight SET AirlineName =
" + airline
    + "WHERE FlightNumber = " + flightNum;
PreparedStatement flightPS =
reservationConn.prepareStatement(flightUpdate);
flightPS.executeUpdate();

```

- executeUpdate() is missing an argument.
- The code is susceptible to an SQL injection attack.
- Nothing is wrong.

Stored procedure calls

Stored procedures are part of MySQL procedural SQL. Stored procedures are saved in the database and often called from general-purpose languages like Java. Stored procedure parameters are labeled IN or OUT. IN parameters input data to the stored procedure. OUT parameters output data from the stored procedure.

MDCCOP2335Spring2024

The **CallableStatement** interface extends PreparedStatement with methods that call stored procedures. CallableStatement objects are created with the Connection **prepareCall()** method and executed with the CallableStatement **executeQuery()** method. Additional CallableStatement methods associate Java values with stored procedure parameters:

- **setInt()** sets the IN parameter to the given Java int value.
- **setString()** sets the IN parameter to the given Java String value.
- **registerOutParameter()** sets the OUT parameter to the specified JDBC type.

PARTICIPATION
ACTIVITY

42.6.9: Calling a stored procedure to get flight count.



©zyBooks 01/31/24 18:46 1939727

Rob Daglio

MDCCOP2335Spring2024

Procedural SQL

```
CREATE PROCEDURE FlightCount(IN airline VARCHAR(20), OUT quantity INT)
    SELECT COUNT(*)
    INTO quantity
    FROM Flight
    WHERE AirlineName = airline;
```

Java

```
CallableStatement flightCS = reservationConn.
    prepareCall("{call FlightCount(?, ?)}");
flightCS.setString(1, "China Airlines");
flightCS.registerOutParameter(2, java.sql.Types.INTEGER);
flightCS.executeQuery();

System.out.println(flightCS.getString(2));
```

Memory

China Airlines
23

flightC

23

Animation content:

Static figure:

An SQL statement has caption procedural SQL:

Begin SQL code:

```
CREATE PROCEDURE FlightCount(IN airline VARCHAR(20), OUT quantity INT)
    SELECT COUNT(*) INTO quantity
    FROM Flight
    WHERE AirlineName = airline;
```

End SQL code.

Begin Java code:

```
CallableStatement flightCS = reservationConn.prepareCall("{call FlightCount(?, ?)}");
flightCS.setString(1, "China Airlines");
flightCS.registerOutParameter(2, java.sql.Types.INTEGER);
flightCS.executeQuery();
System.out.println(flightCS.getString(2));
End Java code.
```

©zyBooks 01/31/24 18:46 1939727

Rob Daglio

MDCCOP2335Spring2024

A memory diagram appears with two cells, with caption flightCS. The cells contain values China Airlines and 23.

A console displays program output:

23

Step 1: The FlightCount stored procedure counts the number of flights for an airline. The procedural SQL appears.

Step 2: The prepareCall() method creates a CallableStatement object. The Java code appears. The CallableStatement statement is highlighted. The memory diagram appears with ? in both cells.

Step 3: setString() binds the airline parameter. registerOutParameter() binds the quantity parameter. executeQuery() executes the stored procedure. The three flightCS statements are highlighted. The stored procedure executes and counts 23 rows in the Flight table for China Airlines. Values China Airlines and 23 appear in memory, with caption flightCS.

Step 4: getString(2) returns the quantity parameter. 23 appears on the console.

Animation captions:

1. The FlightCount stored procedure counts the number of flights for an airline.
2. The prepareCall() method creates a CallableStatement object.
3. setString() binds the airline parameter. registerOutParameter() binds the quantity parameter. executeQuery() executes the stored procedure.

PARTICIPATION ACTIVITY

42.6.10: Stored procedural calls.



Refer to the animation above. Match the language element with the description.

If unable to drag and drop, refresh the page.

CallableStatement

registerOutParameter

java.sql.Types.VARCHAR

prepareCall

setInt

©zyBooks 01/31/24 18:46 1939727
Rob Daglio

MDCCOP2335Spring2024

Interface that extends
PreparedStatement

Method that creates a
CallableStatement object

Method that binds the OUT parameter
of a stored procedure

Method that binds the IN parameter of
a stored procedure

Java String data type

©zyBooks 01/31/24 18:46 1939727

Rob Daglio
MDCCOP2335Spring2024

Reset

**CHALLENGE
ACTIVITY**

42.6.1: Database programming with Java.



539740.3879454.qx3zqy7

Start

The Java code below prints out the room numbers of all the rooms in the database located on the missing keywords.

```
Connection roomConn = null;  
roomConn = DriverManager._____ (A)_____ (  
    "jdbc:mysql://127.0.0.1/reservation?"  
    + "user=samsnead&password=*jksi72$");  
String roomQuery = "SELECT RoomNumber FROM Room WHERE FloorNumber  
PreparedStatement roomStatement = roomConn._____ (B)_____ (roomQuery  
_____ (C)_____.setInt(1, 3);  
ResultSet resultSet = roomStatement._____ (D)_____ ();  
while (resultSet.next()) {  
    System.out.println("Found room " + resultSet.getInt("_____ (E)_  
        + " located on the 3rd floor.");  
}  
_____ (F)_____.close();  
roomStatement.close();  
roomConn._____ (G)_____ ();
```

(A) Ex: keyword

(E)

(B)

(F)

(C)

(G)

(D) ©zyBooks 01/31/24 18:46 1939727

Rob Daglio
MDCCOP2335Spring2024

1

Check

Try again

Exploring further:

- [MySQL Connector/J Installation](#)
- [MySQL Connector/J Developer Guide](#)
- [Java API for the java.sql library](#)

©zyBooks 01/31/24 18:46 1939727

Rob Daglio

MDCCOP2335Spring2024

42.7 Database programming for the web

Connections

This section describes database programming for the web using PHP and MySQL. **PHP** is the most widely used programming language used to create dynamic websites. Other alternatives to PHP include Python, Java, Node.js, and ASP.NET. This section assumes familiarity with HTML, the language used to build web pages, and PHP.

A PHP application uses **PHP Data Objects (PDO)**, an API to interact with a wide range of databases, including MySQL. To use PDO for a specific database, a database-specific PDO driver is required. Most PHP installations include a PDO driver for MySQL.

PHP scripts must connect to a database prior to executing queries. A PHP script establishes a connection by creating a **PDO object**. The PDO constructor specifies a DSN and a MySQL username and password. A **Data Source Name (DSN)** is a string containing database connection information:

- DSN prefix - "mysql:" for MySQL
- host - Database server's hostname or address
- port - Database server's port number (if the default port number is not used)
- dbname - Database name

If the PDO constructor fails to connect to MySQL successfully, the PDO constructor throws a **PDOException**, which results in a fatal error. PHP can be configured to handle fatal errors in various ways, including outputting the error message to the web page.

©zyBooks 01/31/24 18:46 1939727

Rob Daglio

MDCCOP2335Spring2024

PARTICIPATION ACTIVITY

42.7.1: Connecting to MySQL.

web browser

<http://example.org/db.php>

web server

response

MySQL server

Reservation

Fatal error: Uncaught
PDOException: Access
denied...

db.php



database

db.php

```
<?php

$dsn = "mysql:host=localhost;dbname=Reservation";
$username = "samsnead";
$password = "*jksi72$";
$pdo = new PDO($dsn, $username, $password); @zyBooks 01/31/24 18:46 1939727
// Use $pdo to perform database operations MDCCOP2335Spring2024

?>
```

Animation content:

Static figure:

A diagram has three rectangles with captions web browser, web server, and MySQL server. The web browser contains a URL and error message. The web server contains another rectangle with caption db.php. The MySQL server contains another rectangle with caption reservation database. The web browser and web server are connected by a double-headed arrow with caption response. The web server and MySQL server are connected by a double-headed arrow under a large X, with caption connection.

Begin PHP code:

```
<?php
$dsn = "mysql:host=localhost;dbname=Reservation";
$username = "samsnead";
$password = "*jksi72$";
$pdo = new PDO($dsn, $username, $password);
// Use $pdo to perform database operations
?>
```

End PHP code.

Step 1: The web browser sends a request to the web server for a PHP script called db.php. The web server executes the PHP script. The web browser, web server, MySQL server, and PHP script appear. The web browser contains the URL only, not the error message. Only one arrow appears, pointing from web browser to web server, with caption request.

Step 2: The DSN indicates that PDO should connect to a MySQL server running on localhost (127.0.0.1) and use the database named "Reservation". The first script line is highlighted:

```
$dsn = "mysql:host=localhost;dbname=Reservation";
```

Step 3: The PDO constructor uses the DSN and MySQL username and password to create a database connection. The script line is highlighted:

```
$pdo = new PDO($dsn, $username, $password);
```

The connection arrow appears between web server and MySQL server, without the large X.

Step 4: If the connection succeeds, the PHP script uses the PDO object to execute queries. The script comment is highlighted:

```
// Use $pdo to perform database operations
```

©zyBooks 01/31/24 18:46 1939727

Rob Daglio

MDCCOP2335Spring2024

Step 5: If the connection fails, the PDOException produces a fatal error. The caption PDOException: Fatal error appears next to the script comment. A large X appears above the connection arrow.

Step 6: If PHP is configured to show fatal errors, the web server responds with a web page showing the error details, which is then displayed by the web browser. The arrow from web browser to web server becomes double-headed with caption response. The fatal error message moves from web server to web browser.

Animation captions:

1. The web browser sends a request to the web server for a PHP script called db.php. The web server executes the PHP script.
2. The DSN indicates that PDO should connect to a MySQL server running on localhost (127.0.0.1) and use the database named "Reservation".
3. The PDO constructor uses the DSN and MySQL username and password to create a database connection.
4. If the connection succeeds, the PHP script uses the PDO object to execute queries.
5. If the connection fails, the PDOException produces a fatal error.
6. If PHP is configured to show fatal errors, the web server responds with a web page showing the error details, which is then displayed by the web browser.

PARTICIPATION ACTIVITY

42.7.2: Connecting to a database.



- 1) What DSN value is omitted from the animation above?

- DSN prefix
- host
- port
- dbname

©zyBooks 01/31/24 18:46 1939727

Rob Daglio

MDCCOP2335Spring2024



2) What values are required to create a PDO object?

- Username and password only
- Hostname only
- Hostname, database name, username, and password

3) Referring to the animation above, why might the PDO constructor throw a PDOException?

- The username and password are correct.
- The MySQL server is running on localhost.
- The Reservation database does not exist.

©zyBooks 01/31/24 18:46 1939727

Rob Daglio
MDCCOP2335Spring2024



Executing statements

The PDO object's **query()** method executes an SQL statement. The **query()** method returns a **PDOStatement** object or FALSE if an error occurs executing the statement.

Instead of checking the **query()** return value for FALSE, good practice is to call **setAttribute()** to make the PDO object throw a PDOException when an SQL error occurs. The PDOException provides details about the SQL error. The PDO object's **setAttribute()** method sets a PDO attribute to a value. Ex:

`$pdo->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION)` sets the error mode to throw a PDOException.

The PDOStatement method **rowCount()** returns the number of rows that are inserted, updated, or deleted.

PARTICIPATION ACTIVITY

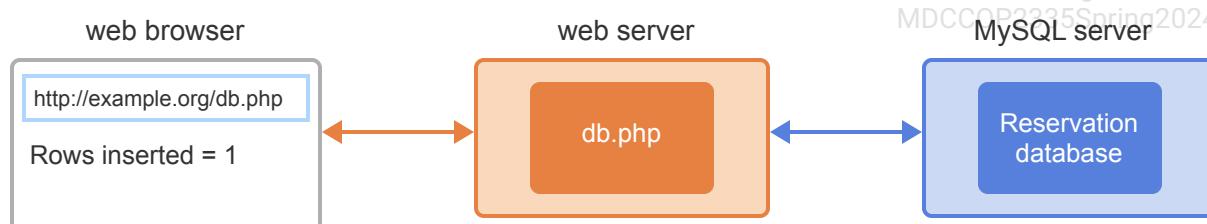
42.7.3: Insert a new flight.



©zyBooks 01/31/24 18:46 1939727

Rob Daglio
MDCCOP2335Spring2024

MySQL server



```
$dsn = "mysql:host=localhost;dbname=Reservation";
$pdo = new PDO($dsn, "samsnead", "*jksi72$");

$pdo->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

$sql = "INSERT INTO Flight ".
    "(FlightNumber, AirlineName, AirportCode) " .
    "VALUES (280, 'China Airlines', 'PEK')";

$statement = $pdo->query($sql);

echo "Rows inserted = " . $statement->rowCount();
```

©zyBooks 01/31/24 18:46 1939727
Rob Daglio
MDCCOP2335Spring2024

Animation content:

Static figure:

A diagram has three rectangles with captions web browser, web server, and MySQL server. The web browser contains a URL and the message rows inserted = 1. The web server contains another rectangle with caption db.php. The MySQL server contains another rectangle with caption reservation database. The web browser and web server are connected by a double-headed arrow. The web server and MySQL server are connected by a double-headed arrow.

A PHP script appears with caption db.php:

Begin PHP code:

```
$dsn = "mysql:host=localhost;dbname=Reservation";
$pdo = new PDO($dsn, "samsnead", "*jksi72$");
$pdo->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
$sql = "INSERT INTO Flight ".
    "(FlightNumber, AirlineName, AirportCode) " .
    "VALUES (280, 'China Airlines', 'PEK')";

$statement = $pdo->query($sql);
echo "Rows inserted = " . $statement->rowCount();
End PHP code.
```

Step 1: The PDO object creates a connection to MySQL. The diagram and script appear. The web browser displays the URL but not the message. The first two script lines are highlighted:

```
$dsn = "mysql:host=localhost;dbname=Reservation";
$pdo = new PDO($dsn, "samsnead", "*jksi72$");
```

©zyBooks 01/31/24 18:46 1939727
Rob Daglio
MDCCOP2335Spring2024

Step 2: setAttribute() makes PDO throw a PDOException if an error occurs executing an SQL query. The next script line is highlighted:

```
$pdo->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
```

Step 3: The query() method executes the INSERT statement in the \$sql string. The flight is inserted

into the Flight table. The following script lines are highlighted:

```
$sql = "INSERT INTO Flight ".  
    "(FlightNumber, AirlineName, AirportCode) ".  
    "VALUES (280, 'China Airlines', 'PEK');"
```

```
$statement = $pdo->query($sql);
```

The caption INSERT moves from query(\$sql) to the web server, and then to MySQL server.

©zyBooks 01/31/24 18:46 1939727

Step 4: MySQL server returns information to the web server indicating the result of the INSERT statement. The information is returned from query() as a PDOStatement. The message flight inserted moves from MySQL server to the web server.

Rob Daglio

MDCCOP2335Spring2024

Step 5: The rowCount() method returns 1 because one row was inserted into the Flight table. The last script line is highlighted:

```
echo "Rows inserted = " . $statement->rowCount();
```

The message rows inserted = 1 moves from the echo statement to the web server, and then appears in the web browser.

Animation captions:

1. The PDO object creates a connection to MySQL.
2. setAttribute() makes PDO throw a PDOException if an error occurs executing an SQL query.
3. The query() method executes the INSERT statement in the \$sql string. The flight is inserted into the Flight table.
4. MySQL server returns information to the web server indicating the result of the INSERT statement. The information is returned from query() as a PDOStatement.

PARTICIPATION ACTIVITY

42.7.4: Executing statements.



Refer to the animation above.

- 1) Removing the call to setAttribute()
stops the query() method from
executing the SQL statement.



- True
- False

©zyBooks 01/31/24 18:46 1939727
Rob Daglio
MDCCOP2335Spring2024



2) If FlightNumber is the primary key, executing the INSERT statement twice results in a PDOException being thrown.

- True
- False

3) If \$sql is changed to insert two flights, the call to rowCount() returns 2.

- True
- False

©zyBooks 01/31/24 18:46 1939727

Rob Daglio

MDCCOP2335Spring2024



Prepared statements

An SQL statement may be created dynamically from user input. A **prepared statement** is an SQL statement that can be customized with parameter values retrieved from user input. Prepared statements are immune to most SQL injection attacks.

Prepared statements are created and executed in four steps:

1. *Define* - A prepared statement uses an SQL statement with parameter identifiers as placeholders for values. A **parameter identifier** can be a question mark (?) or a name following a colon (:name).
2. *Prepare* - The PDO method **prepare()** prepares a parameterized SQL statement for execution and returns a PDOStatement.
3. *Bind* - The PDOStatement method **bindValue()** binds a value to a parameter identifier in the SQL statement. Ex: `$statement->bindValue(1, "Bob")` binds "Bob" to the first question mark, and `$statement->bindValue("name", "Bob")` binds "Bob" to the :name parameter identifier.
4. *Execute* - The PDOStatement method **execute()** executes the SQL statement with the bound parameters and throws a PDOException if an error occurs.

PARTICIPATION ACTIVITY

42.7.5: Prepared statement inserts a new flight.

©zyBooks 01/31/24 18:46 1939727

Rob Daglio

MDCCOP2335Spring2024



addflight.php

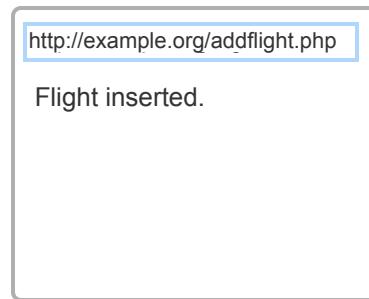
```
$sql = "INSERT INTO Flight " .
       "(FlightNumber, AirlineName, AirportCode) " .
       "VALUES (?, ?, ?);"

$statement = $pdo->prepare($sql);
$statement->bindValue(1, $_POST["flightNum"]);
```

	\$_POST
flightNum	105
airlineName	Air India
airlineCode	DEL

```
$statement->bindValue(2, $_POST["airlineName"]);
$statement->bindValue(3, $_POST["airlineCode"]);
$statement->execute();

echo "Flight inserted.;"
```



©zyBooks 01/31/24 18:46 1939727
Rob Daglio
MDCCOP2335Spring2024

Animation content:

Static figure:

A PHP script has caption addflight.php:

Begin PHP code:

```
$sql = "INSERT INTO Flight ".
```

```
    "(FlightNumber, AirlineName, AirportCode) " .
```

```
    "VALUES (?, ?, ?);
```

```
$statement = $pdo->prepare($sql);
```

```
$statement->bindValue(1, $_POST["flightNum"]);
```

```
$statement->bindValue(2, $_POST["airlineName"]);
```

```
$statement->bindValue(3, $_POST["airlineCode"]);
```

```
$statement->execute();
```

```
echo "Flight inserted.";
```

End PHP code.

A memory diagram has caption \$_POST and three cells. The first cell has caption flightNum and contains 105. The second cell has caption airlineName and contains Air India. The third cell has caption airlineCode and contains DEL.

A web browser displays a URL and the message flight inserted.

©zyBooks 01/31/24 18:46 1939727

Rob Daglio

MDCCOP2335Spring2024

Step 1: A user enters flight information into a form. The form submits to addflight.php, and submitted values are placed in the \$_POST array. The PHP script appears. The web browser appears with the URL, prompts, user responses, and a submit button. The prompts and responses are:

Num? 105

Airline? Air India

Code? DEL

The submit button is clicked and the responses move to the memory cells.

Step 2: The INSERT statement uses ? as placeholders for three values. The first PHP statement is highlighted:

```
$sql = "INSERT INTO Flight ".  
    "(FlightNumber, AirlineName, AirportCode) ".  
    "VALUES (?, ?, ?);"
```

Step 3: prepare() prepares the parameterized statement and returns a PDOStatement. The next PHP statement is highlighted:

©zyBooks 01/31/24 18:46 1939727
Rob Daglio
MDCCOP2335Spring2024

```
$statement = $pdo->prepare($sql);
```

Step 4: bindValue() binds the values from \$_POST to the parameter identifiers. The next three PHP statements are highlighted:

```
$statement->bindValue(1, $_POST["flightNum"]);  
$statement->bindValue(2, $_POST["airlineName"]);  
$statement->bindValue(3, $_POST["airlineCode"]);
```

105, Air India, and DEL move from memory to the three statements and then to VALUES(?, ?, ?).

Step 5: execute() executes the INSERT statement. addflight.php displays a confirmation message. The last two PHP statements are highlighted:

```
$statement->execute();  
echo "Flight inserted.";
```

The message flight inserted appears in the web browser.

Animation captions:

1. A user enters flight information into a form. The form submits to addflight.php, and submitted values are placed in the \$_POST array.
2. The INSERT statement uses ? as placeholders for three values.
3. prepare() prepares the parameterized statement and returns a PDOStatement.
4. bindValue() binds the values from \$_POST to the parameter identifiers.

PARTICIPATION ACTIVITY

42.7.6: Prepared statements.



©zyBooks 01/31/24 18:46 1939727
Rob Daglio
MDCCOP2335Spring2024



1) Which values are missing?

```
$sql = "UPDATE Flight " .  
       "SET AirlineName = ? " .  
       "WHERE FlightNumber =  
?";  
  
$statement = $pdo->prepare($sql);  
$statement->bindValue(1, ____);  
$statement->bindValue(2, ____);  
$statement->execute();
```

©zyBooks 01/31/24 18:46 1939727
Rob Daglio
MDCCOP2335Spring2024

- "United Airlines" and "PEK"
- 305 and "United Airlines"
- "United Airlines" and 305

2) Which values are missing?

```
$sql = "UPDATE Flight " .  
       "SET AirlineName =  
:airlineName " .  
       "WHERE FlightNumber =  
:flightNum";  
  
$statement = $pdo->prepare($sql);  
$statement->bindValue(____,  
"Southwest Airlines");  
$statement->bindValue(____,  
445);  
$statement->execute();
```

- AirlineName and FlightNumber
- "airlineName" and "flightNum"
- :airlineName and :flightNum

©zyBooks 01/31/24 18:46 1939727
Rob Daglio
MDCCOP2335Spring2024



- 3) Assume the airline and flightNum values are obtained from a submitted form. What is wrong with the following code?

```
$sql = "UPDATE Flight " .
    "SET AirlineName =
'$_POST[airline]' .
    "WHERE FlightNumber =
$_POST[flightNum]";

$statement = $pdo->query($sql);
```

©zyBooks 01/31/24 18:46 1939727
Rob Daglio
MDCCOP2335Spring2024

- The SQL statement is missing parameter identifiers.
- The code is susceptible to an SQL injection attack.
- Nothing is wrong.

Fetching values

When executing a PDOStatement with a SELECT statement, the execute() method creates a cursor object. Query results are obtained from the cursor with the fetch() method. The PDOStatement method **fetch()** returns an array containing data from one row or FALSE if no row is selected. The row values are indexed in two ways:

- By column name. Ex: ["FlightNumber"] => 350, ["DepartureTime"] => "08:15:00", ["AirportCode"] => "PEK".
- By column number. Ex: [0] => 350, [1] => "08:15:00", [2] => "PEK".

If a SELECT statement returns more than one row, fetch() may be called in a loop. Each call to fetch() returns the next row from the result table. When no more rows exist, fetch() returns FALSE.

The fetch() method has an optional style parameter that returns the fetched row in other formats. Common style values include:

- **PDO::FETCH_NUM** - Returns only an array indexed by column number. Ex: `$row[0]` is the data from the first column.
- **PDO::FETCH_ASSOC** - Returns only an associative array indexed by column name. Ex: `$row["colName"]` is the data from the colName column.
- **PDO::FETCH_OBJ** - Returns an object with property names that match column names. Ex: `$row->colName` is the data from the colName column.

©zyBooks 01/31/24 18:46 1939727
Rob Daglio
MDCCOP2335Spring2024



```

$sql = "SELECT FlightNumber, DepartureTime FROM Flight " .
    "WHERE AirportCode = :code AND AirlineName = :name";
$statement = $pdo->prepare($sql);
$statement->bindValue("code", "PEK");
$statement->bindValue("name", "China Airlines");
$statement->execute();

while ($row = $statement->fetch()) {
    echo "<p>Flight $row[FlightNumber] departs at $row[DepartureTime]</p>";
}

```

Rob Daglio
MDCCOP2335Spring2024

<http://example.org/flights.php>

Flight 107 departs at 13:30:00
 Flight 350 departs at 08:15:00
 Flight 482 departs at 10:07:00

Animation content:

Static figure:

Begin PHP code:

```

$sql = "SELECT FlightNumber, DepartureTime FROM Flight " .
    "WHERE AirportCode = :code AND AirlineName = :name";
$statement = $pdo->prepare($sql);
$statement->bindValue("code", "PEK");
$statement->bindValue("name", "China Airlines");
$statement->execute();
while ($row = $statement->fetch()) {
    echo "<p>Flight $row[FlightNumber] departs at $row[DepartureTime]</p>";
}

```

End PHP code.

A web browser displays a URL and three messages:

Flight 107 departs at 13:30:00

Flight 350 departs at 08:15:00

Flight 482 departs at 10:07:00

©zyBooks 01/31/24 18:46 1939727
 Rob Daglio
MDCCOP2335Spring2024

Step 1: The prepared statement is executed(), selecting China Airlines flights out of PEK. The PHP script appears. All lines prior to the while statement are highlighted. The web browser appears with URL but no messages.

Step 2: fetch() returns an array containing a single row from the results table. The first line of the while statement is highlighted.

Step 3: The while loop processes each row until all rows are processed. Result values are accessed from \$row using the column names. The complete while statement is highlighted. The three messages appear in the web browser.

Animation captions:

1. The prepared statement is executed(), selecting China Airlines flights out of PEK.
2. fetch() returns an array containing a single row from the results table.
3. The while loop processes each row until all rows are processed. Result values are accessed from \$row using the column names.

PARTICIPATION ACTIVITY

42.7.8: Fetching values.



Refer to the code below.

```
$flightNum = 140;  
$sql = "SELECT AirlineName, DepartureTime, AirportCode FROM Flight ".  
       "WHERE FlightNumber = ?";  
$statement = $pdo->prepare($sql);  
$statement->bindValue(__A__, $flightNum);  
$statement->execute();  
  
$row = $statement->__B__();  
if ($row) {  
    echo "$row[AirlineName] $flightNum departs from $row[__C__]";  
}  
else {  
    echo "Flight not found";  
}
```

1) What is identifier A?

**Check****Show answer**

2) What is identifier B?

**Check****Show answer**

©zyBooks 01/31/24 18:46 1939727
Rob Daglio
MDCCOP2335Spring2024

3) What is identifier C?

**Check****Show answer**

Stored procedure calls

Stored procedures are part of MySQL procedural SQL. Stored procedures are saved in the database and may be called from a PHP script.

Stored procedures are called with a prepared statement. Values are bound to the stored procedure's IN parameters. Stored procedures with an OUT or INOUT parameter require a call with an SQL user-defined variable. A second query is required to fetch the value of the user-defined variable that is holding the output parameter value.

PARTICIPATION ACTIVITY

42.7.9: Calling a stored procedure to get flight count.



Procedural SQL

```
CREATE PROCEDURE FlightCount(IN airline VARCHAR(20), OUT quantity INT)
    SELECT COUNT(*) INTO quantity
    FROM Flight
    WHERE AirlineName = airline;
```

16

flights.php

```
$statement = $pdo->prepare("CALL FlightCount(?, @count)");
$statement->bindValue(1, "China Airlines");
$statement->execute();

$statement = $pdo->query("SELECT @count AS count");
$row = $statement->fetch();
echo "Count = $row[count];"
```

http://example.org/flights.php
Count = 16

Animation content:

Static figure:

SQL code appears with caption procedural SQL.

Begin SQL code:

```
CREATE PROCEDURE FlightCount(IN airline VARCHAR(20), OUT quantity INT)
```

```
    SELECT COUNT(*) INTO quantity
    FROM Flight
    WHERE AirlineName = airline;
```

End SQL code.

©zyBooks 01/31/24 18:46 1939727
Rob Daglio
MDCCOP2335Spring2024

A PHP script appears with caption flights.php.

Begin PHP code:

```
$statement = $pdo->prepare("CALL FlightCount(?, @count)");
$statement->bindValue(1, "China Airlines");
$statement->execute();
$statement = $pdo->query("SELECT @count AS count");
```

```
$row = $statement->fetch();
echo "Count = $row[count]";
End PHP code.
```

In the PHP code, an arrow points from China Airlines to ?. In the SQL code, China Airlines appears below IN airline and 16 appears below OUT quantity. Another arrow points from 16 to @count in the PHP code.

©zyBooks 01/31/24 18:46 1939727

Rob Daglio

A web browser appears with a URL and the message count = 16. MDCCOP2335Spring2024

Step 1: FlightCount counts the number of flights for an airline. The SQL code appears.

Step 2: The prepared statement uses a CALL statement to call FlightCount with the airline "China Airlines". The PHP code appears. The web browser appears with the URL but no message. The first two PHP statements are highlighted. The arrow from China Airlines to ? appears.

Step 3: execute() calls FlightCount. The OUT parameter quantity is assigned to the @count variable. The next PHP statement is highlighted:

```
$statement->execute();
```

China Airlines moves from the PHP code to IN airline. 16 moves from SELECT COUNT(*) to OUT quantity. The arrow from 16 to @count appears.

Step 4: A second query fetches the value of @count. The last three PHP statements are highlighted. The message count = 16 appears in the web browser.

Animation captions:

1. FlightCount counts the number of flights for an airline.
2. The prepared statement uses a CALL statement to call FlightCount with the airline "China Airlines".
3. execute() calls FlightCount. The OUT parameter quantity is assigned to the @count variable.

PARTICIPATION
ACTIVITY

42.7.10: Stored procedure call with PDO.



©zyBooks 01/31/24 18:46 1939727

Rob Daglio

MDCCOP2335Spring2024

Refer to the stored procedure that, given an airport and airline, finds the flight number of the last flight. Place the PHP code in order.

```

CREATE PROCEDURE LastFlight(IN airport CHAR(3), IN airline VARCHAR(45), OUT
flightNum INT)
BEGIN
    SELECT FlightNumber
    INTO flightNum
    FROM Flight
    WHERE AirportCode = airport AND AirlineName = airline
    ORDER BY DepartureTime DESC
    LIMIT 1;
END;

```

©zyBooks 01/31/24 18:46 1939727

Rob Daglio

MDCCOP2335Spring2024

If unable to drag and drop, refresh the page.

```

$statement->bindValue(1, "DEN");
$statement->bindValue(2, "United Airlines");

```

```

$statement = $pdo->query("SELECT @flightNum AS flightNum");
$row = $statement->fetch();

```

```

$statement = $pdo->prepare("CALL LastFlight(?, ?, @flightNum)");

```

```

echo "Flight number = $row[flightNum]";

```

```

$statement->execute();

```

Step 1

Step 2

Step 3

Step 4

Step 5

Reset

©zyBooks 01/31/24 18:46 1939727

Rob Daglio

MDCCOP2335Spring2024

Exploring further:

- [PHP Data Objects documentation](#) from PHP.net

42.8 LAB - Database programming with Python (SQLite)

Complete the Python program to create a Horse table, insert one row, and display the row. The main program calls four functions:

©zyBooks 01/31/24 18:46 1939727

Rob Daglio

MDCCOP2335Spring2024

1. `create_connection()` creates a connection to the database.
2. `create_table()` creates the Horse table.
3. `insert_horse()` inserts one row into Horse.
4. `select_all_horses()` outputs all Horse rows.

Complete all four functions. Function parameters are described in the template. Do not modify the main program.

The Horse table should have five columns, with the following names, data types, constraints, and values:

Name	Data type	Constraints	Value
Id	integer	primary key, not null	1
Name	text		'Babe'
Breed	text		'Quarter horse'
Height	double		15.3
BirthDate	text		'2015-02-10'

The program output should be:

```
All horses:  
(1, 'Babe', 'Quarter Horse', 15.3, '2015-02-10')
```

This lab uses the SQLite database rather than MySQL. The Python API for SQLite is similar to MySQL Connector/Python. Consequently, the API is as described in the text with a few exceptions:

Rob Daglio

MDCCOP2335Spring2024

- Use the import library provided in the program template.
- Create a connection object with the function `sqlite3.connect(":memory:")`.
- Use the character `?` instead of `%s` as a placeholder for query parameters.
- Use data type `text` instead of `char` and `varchar`.

SQLite reference information can be found at [SQLite Python Tutorial](#), but is not necessary to complete this lab.

**LAB
ACTIVITY**

42.8.1: LAB - Database programming with Python (SQLite)

0 / 10

**main.py**[Load default template...](#)

```
1 import sqlite3
2 from sqlite3 import Error
3
4 # Creates connection to sqlite in-memory database
5 def create_connection():
6     """
7         Create a connection to in-memory database
8         :return: Connection object
9     """
10
11     # YOUR CODE HERE
12     # Use sqlite3.connect(":memory:") to create connection object
13
14     return conn
15
16
```

©zyBooks 01/31/24 18:46 1939727

Rob Daglio

MDCCOP2335Spring2024

Develop mode**Submit mode**

Run your program as often as you'd like, before submitting for grading. Below, type any needed input values in the first box, then click **Run program** and observe the program's output in the second box.

Enter program input (optional)

If your code requires input values, provide them here.

Run program

Input (from above)

**main.py**
(Your program)

Output

Program output displayed here

©zyBooks 01/31/24 18:46 1939727

Rob Daglio

MDCCOP2335Spring2024

Coding trail of your work

[What is this?](#)

History of your effort will appear here once you begin working on this zyLab.

42.9 LAB - Database programming with Java (SQLite)

©zyBooks 01/31/24 18:46 1939727

Pub Date: MDCCOP2335Spring2024

Complete the Java program to create a Horse table, insert one row, and display the row. The main program calls four methods:

1. `createConnection()` creates a connection to the database.
2. `createTable()` creates the Horse table.
3. `insertHorse()` inserts one row into Horse.
4. `selectAllHorses()` outputs all Horse rows.

Complete all four methods. Method parameters are described in the template. Do not modify the main program.

The Horse table should have five columns, with the following names, data types, constraints, and values:

Name	Data type	Constraints	Value
Id	integer	primary key, not null	1
Name	text		'Babe'
Breed	text		'Quarter horse'
Height	double		15.3
BirthDate	text		'2015-02-10'

The program output should be:

```
All horses:  
(1, 'Babe', 'Quarter Horse', 15.3, '2015-02-10')
```

©zyBooks 01/31/24 18:46 1939727

This lab uses the SQLite database rather than MySQL. Both SQLite and MySQL Connector/J implement the JDBC API. Consequently, the API is as described in the text, with a few exceptions:

- Use the connection string "jdbc:sqlite::in-memory" to connect to an in-memory database.
- Use the `text` data type instead of `char` and `varchar`.

SQLite reference information can be found at [SQLite Java Tutorial](#), but is not necessary to complete this lab.

**LAB
ACTIVITY**

42.9.1: LAB - Database programming with Java (SQLite)

0 / 10

**LabProgram.java**[Load default template...](#)

```
1 import java.sql.*;                                     ©zyBooks 01/31/24 18:46 1939727
2                                                 Rob Daglio
3 public class LabProgram {                           MDCCOP2335Spring2024
4
5     // Create a connection to a sqlite in-memory database
6     // Returns Connection object
7     public static Connection createConnection() {
8
9         // YOUR CODE HERE
10        // Use connection string "jdbc:sqlite::memory:"
11
12    }
13
14    // Create Horse table
15    // Parameter conn is database connection created in createConnection()
16
```

Develop mode**Submit mode**

Run your program as often as you'd like, before submitting for grading. Below, type any needed input values in the first box, then click **Run program** and observe the program's output in the second box.

Enter program input (optional)

If your code requires input values, provide them here.

Run program

Input (from above) →

LabProgram.java
(Your program) →**Program output displayed here**

©zyBooks 01/31/24 18:46 1939727
Rob Daglio
MDCCOP2335Spring2024

Coding trail of your work

[What is this?](#)

History of your effort will appear here once you begin working on this zyLab.