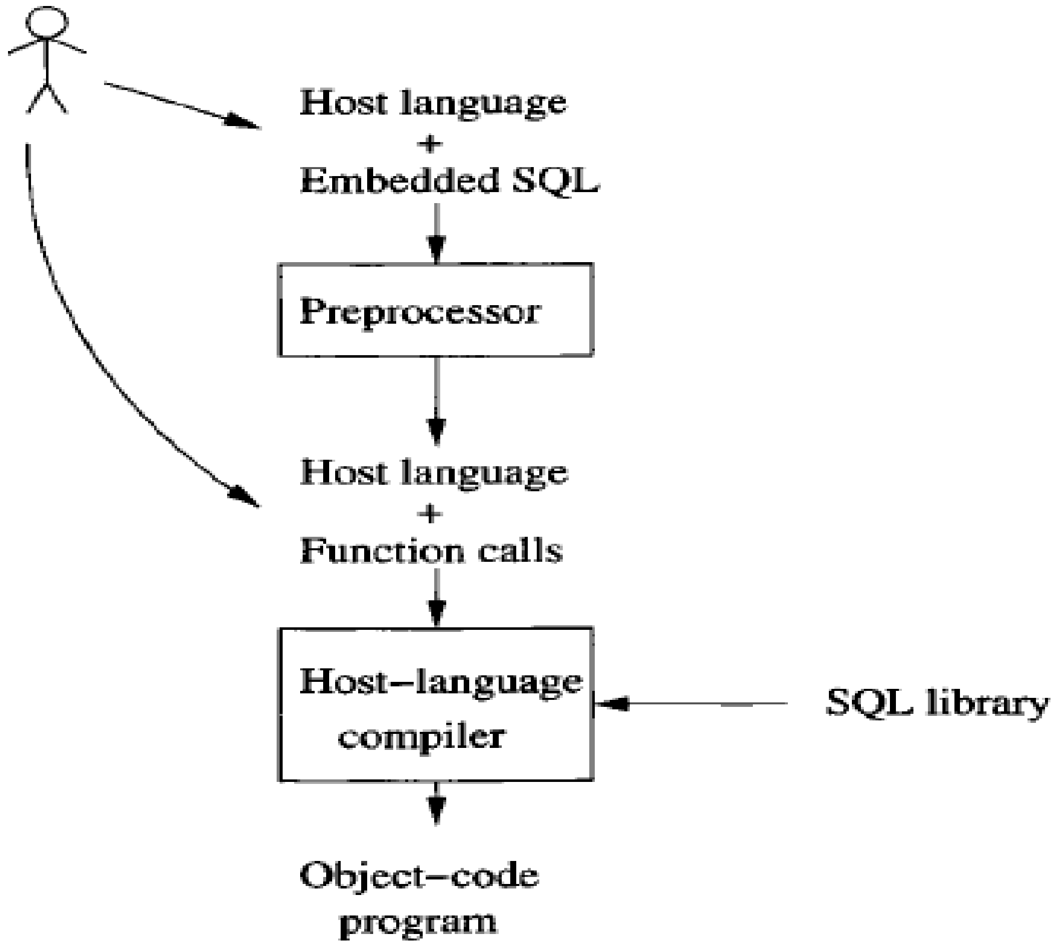
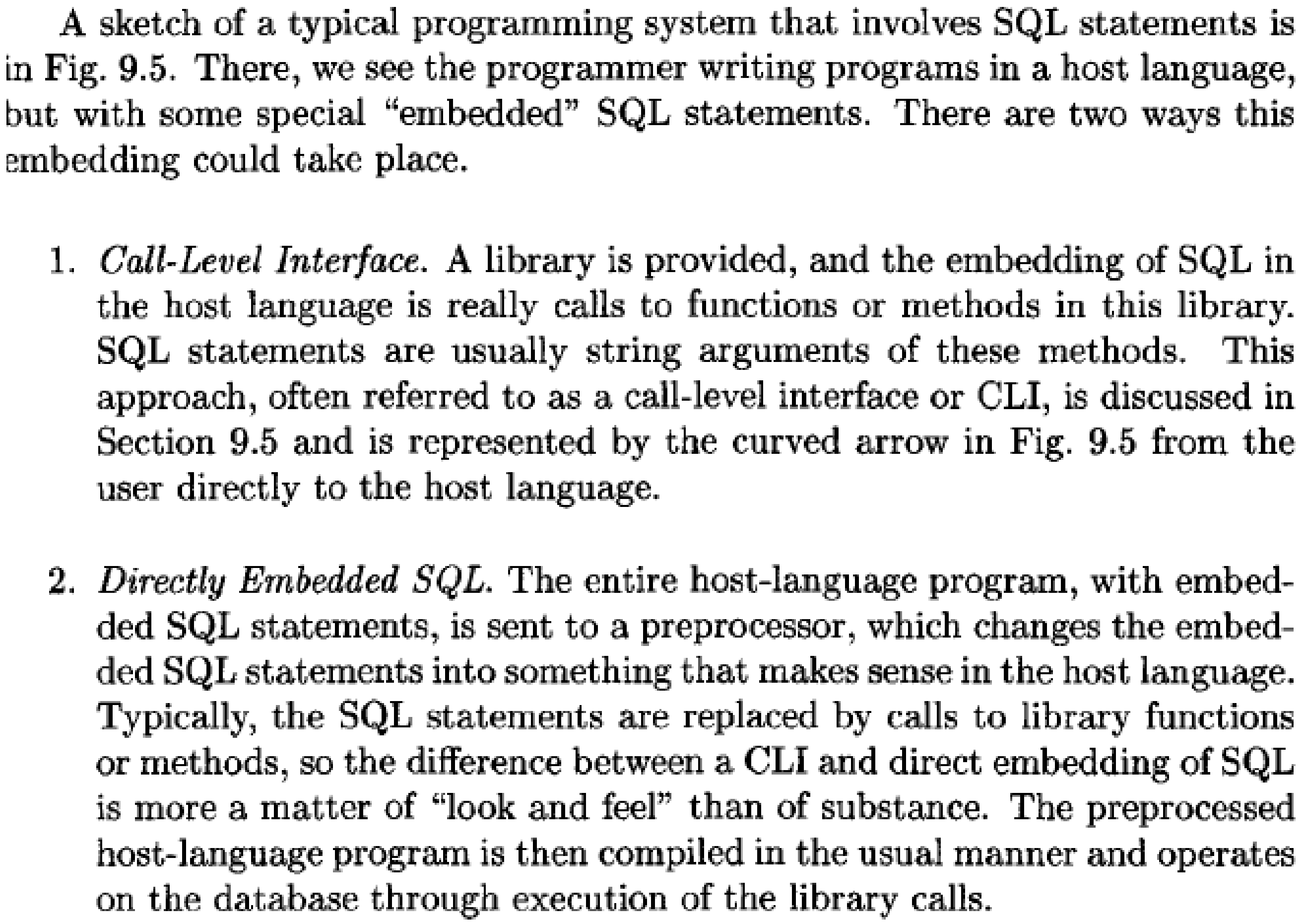
Using SQL with High Level Languages

Java, php and Python

Schematic Diagram of Embedded SQL Processing from the Text



CLI vs Directly Embedded SQL: our use of SQL with php, java and python is of the CLI variety



A sketch of a typical programming system that involves SQL statements is in Fig. 9.5. There, we see the programmer writing programs in a host language, but with some special "embedded" SQL statements. There are two ways this embedding could take place.

1. *Call-Level Interface.* A library is provided, and the embedding of SQL in the host language is really calls to functions or methods in this library. SQL statements are usually string arguments of these methods. This approach, often referred to as a call-level interface or CLI, is discussed in Section 9.5 and is represented by the curved arrow in Fig. 9.5 from the user directly to the host language.

2. *Directly Embedded SQL.* The entire host-language program, with embedded SQL statements, is sent to a preprocessor, which changes the embedded SQL statements into something that makes sense in the host language. Typically, the SQL statements are replaced by calls to library functions or methods, so the difference between a CLI and direct embedding of SQL is more a matter of "look and feel" than of substance. The preprocessed host-language program is then compiled in the usual manner and operates on the database through execution of the library calls.

ODBC (wikipedia)

In computing, **ODBC** (**Open Database Connectivity**) is a standard programming language middleware API for accessing database management systems (DBMS). The designers of ODBC aimed to make it independent of database systems and operating systems. An application written using ODBC can be ported to other platforms, both on the client and server side, with few changes to the data access code.

ODBC accomplishes DBMS independence by using an **ODBC driver** as a translation layer between the application and the DBMS. The application uses ODBC functions through an **ODBC driver manager** with which it is linked, and the driver passes the query to the DBMS. An ODBC driver can be thought of as analogous to a printer driver or other driver, providing a standard set of functions for the application to use, and implementing DBMS-specific functionality.

JDBC (wikipedia)

A **JDBC driver** is a software component enabling a Java application to interact with a database.[1] JDBC drivers are analogous to ODBC drivers, ADO.NET data providers, and OLE DB providers.

To connect with individual databases, JDBC (the Java Database Connectivity API) requires drivers for each database. The JDBC driver gives out the connection to the database and implements the protocol for transferring the query and result between client and database.

JDBC technology drivers fit into one of four categories.[2]

JDBC-ODBC bridge

Native-API driver

Network-Protocol driver (Middleware driver)

Database-Protocol driver (Pure Java driver)

Note: OLE is another database access protocol which is not as portable as ODBC and JDBC but can access spreadsheets and email systems and semi-structured data sources.

Example of making a database connection in Java

```
private void connectToMySQLDB() {
   try {
     try {
        Class.forName("com.mysql.jdbc.Driver");
       System.out.println("Driver loaded");
     }
     catch (Exception ex) {
       System.out.println(" CLASS NOT FOUND EXCEPTION .");
     }
     con = DriverManager.getConnection(
       "jdbc:mysql://134.74.126.107:3306/F15336syou", "F15336syou",
       "password");
   System.out.println(" CONNECTED TO MySQL DB......");
   stt = con.createStatement();
   }
   catch (Exception ex) {
       System.out.println("ERROR OCCURED.");
       ex.printStackTrace();
   }
  }

/* Shenghua You 2015*/
```

Example of using the result set of a query in Java

```
private static void ShowTradeFunction()
{
   JFrame f = new JFrame("STOCK_TRADE");
   JLabel title = new JLabel("ID   TRADE_DATE   TRADE_SEQ_NBR   TRADING_SYMBOL   TRADE_TIME   TRADE_PRICE   TRADE_SIZE");
   JPanel jp = new JPanel();
   jp.setPreferredSize(new Dimension(666, 400));
   jp.add(title);
    try{
      stt.execute("use F15336team2");
      ResultSet rs=stt.executeQuery("select * from STOCK_TRADE");
      while(rs.next())
     {
       String l0=rs.getString("INSTRUMENT_ID");
        String l1=rs.getString("TRADE_DATE");
        int l2=rs.getInt("TRADE_SEQ_NBR");
        String l3=rs.getString("TRADING_SYMBOL");
         String l4=rs.getString("TRADE_TIME");
         int l5=rs.getInt("TRADE_PRICE");
         int l6=rs.getInt("TRADE_SIZE");
         JLabel q = new JLabel(l0+"    "+l1+"    "+l2+"    "+l3+"    "+l4+"    "+l5+"    "+l6);
         jp.add(q);
     }
    }
  catch (Exception ex) {
       System.out.println("can't fetch data.");
     }
/* Shenghua You 2015   Had to leave out Swing display stuff at end*/
```

Example of a prepared statement in Java

```
private void s1Retrieve(Connection conn)
        {
        //method to load the fields of s1 from database
        try {
        Statement stmt = conn.createStatement();
        ResultSet lambres = stmt.executeQuery("select currwork, currcat from lambcurrent");
        while(lambres.next()){
        wn =lambres.getInt(1);
        int cn =lambres.getInt(2);
        System.out.println("Values from lambcurrent:" + wn +" "+cn);
        }
        stmt.close();
        PreparedStatement ps1 = conn.prepareStatement("select workno, catno, artfirst, artlast, gertitle,
engtitle, othertitle, datemonth, dateyear, datetext, medium, dimh, dimw, dimd, inscr, catrais, credit, access,
place, subject,housecat, needtofind, needwork, dated, iden, colorrep, varn from lambase where workno = ?");
        ps1.setInt(1, wn);
        ResultSet results = ps1.executeQuery();
        //now use result set
        while (results.next())
        {
        int thiswn = results.getInt(1);
        System.out.println("Work no passed:"+ wn);
        System.out.println("Work no retrieved:"+thiswn);
```

Php Connect to MySQL and simple select in a loop

```
// Credentials
 $host = "134.74.126.107";
$username = "USERNAME";
 $password = "PASSWORD";
 $stock_database = "stockmarket";
 $team_database = "F15336team4";

 // Connection
 $db = new mysqli($host, $username, $password);

 $stocks = array("BAA", "BAB", "BAC", "BAD", "BAE", "BAF", "BAG", "BAH", "BAI", "BAJ");

 foreach ($stocks as $stock){
      $price = $db->query("SELECT s.TRADE_PRICE as price
                      FROM stockmarket.STOCK_TRADE s
                      WHERE s.TRADING_SYMBOL='$stock'
                      AND s.TRADE_DATE='2005-02-08'
                      LIMIT 1;");
      echo $db->error;
      $price = $price->fetch_assoc()['price'];
      $db->query("INSERT INTO F15336team4.trades VALUES ('$stock', 0, $price, 1);");
      echo $db->error;
      echo "Inserted " . $stock . " with price " . $price;
      echo "<br>";

}

?>
/* MD Islam 2105 */
```

Php code fragment illustrating iteration through output of a query

```
//To maintain a global counter, we start a session that keeps track of a global offset variable
session_start();
                    if (!isset($_SESSION['offset'])) {
                      $x = 0;
                                        }
                    else {
                    $x = $_SESSION['offset'];
                                        }

try{
                    //SQL commands to select 5 rows at a time using the global offset variable
                    $sql_quote = "SELECT * FROM stockmarket.STOCK_QUOTE ORDER BY QUOTE_TIME LIMIT 5 OFFSET $x";
                    $sql_trade = "SELECT * FROM stockmarket.STOCK_TRADE ORDER BY TRADE_TIME LIMIT 5 OFFSET $x";

                    //Stock Quote Section

                    //Print table title in HTML
                    print "<h3>Latest 5 Stock Quote Data</h3><br/>";

                    //Initialize HTML table for the stock quote results
                    $quote_table = "<table border='solid'><thead><tr><td>ID</td><td>Date</td><td>Seq
#</td><td>Symbol</td><td>Date and Time</td><td>Ask Price</td><td>Ask Size</td><td>Bid Price</td><td>Bid Size</td></tr>";

                    //Iterate through each row of the SELECT SQL output
                    foreach ($dbh->query($sql_quote) as $row){
                                        //$row is an associative array that needs to be converted
                                        //Create an array of all the data in a particular row
                                        $quote_raw = array();
                                        for ($i = 0; $i < sizeof($row)/2; $i++){
                                                            array_push($quote_raw,$row[$i]);
                                        }

                                        //Create a string for INSERT SQL
                                        $quote_data =  "'" . implode("','",$quote_raw) . "'";
```

/*Amy Wong and Allen Kim 2015*/

Php prepared statement example from W3schools

```
<?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDB";

// Create connection
$conn = new mysqli($servername, $username, $password, $dbname);

// Check connection
if ($conn->connect_error) {
    die("Connection failed: " . $conn->connect_error);
}

// prepare and bind
$stmt = $conn->prepare("INSERT INTO MyGuests (firstname, lastname, email) VALUES (?, ?, ?)");
$stmt->bind_param("sss", $firstname, $lastname, $email);

// set parameters and execute
$firstname = "John";
$lastname = "Doe";
$email = "john@example.com";
$stmt->execute();

$firstname = "Mary";
$lastname = "Moe";
$email = "mary@example.com";
$stmt->execute();

$firstname = "Julie";
$lastname = "Dooley";
$email = "julie@example.com";
$stmt->execute();

echo "New records created successfully";

$stmt->close();
$conn->close();
?>
```

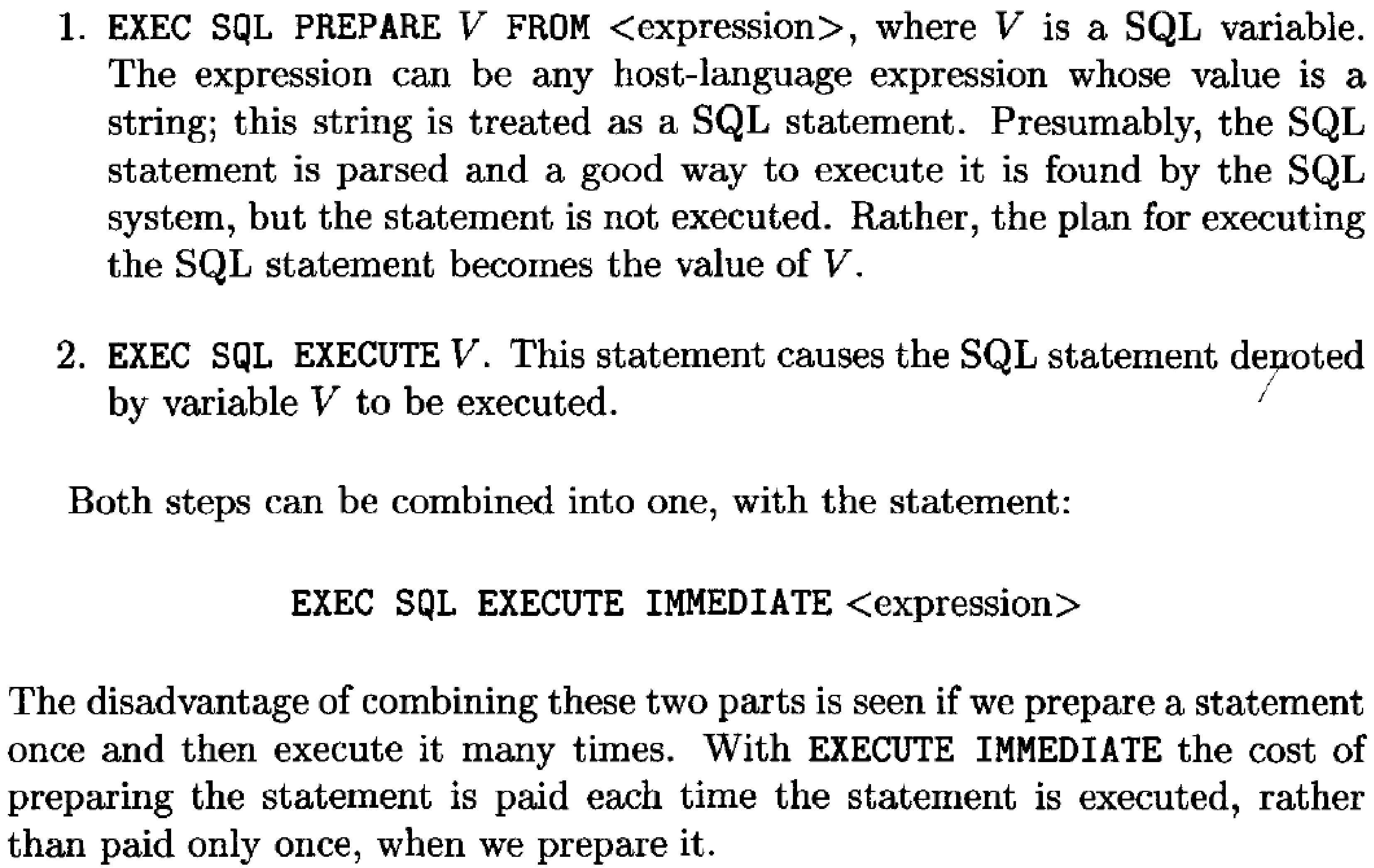
DB connect and iterate through result set in Python

```
#!/usr/bin/python

import pymysql
import pymysql.cursors

# Open database connection
db = pymysql.connect("localhost","pbarnett","42lambie","pubs2" )

# prepare a cursor object using cursor() method
cursor = db.cursor()

sql = "select au_id, au_lname, au_fname from authors where state ='CA'"
try:
  # Execute the SQL command
  cursor.execute(sql)
  # Fetch all the rows in a list of lists.
  results = cursor.fetchall()
  for row in results:
    au_id = row[0]
    au_lname = row[1]
    au_fname = row[2]

    # Now print fetched result
    print "au_id=%s,au_lname=%s,au_fname=%s" % (fname, lname, age, sex, income )

except:
  print "Error: unable to fecth data"

# disconnect from server
db.close()
```

Prepared statements in Python

```
/* python code snippet with cursor but without prepared statement feature*/
def new_title(connection, row_user, author_id):
        row_user['title_id'] = gen_id(connection, 0)

        with connection.cursor() as c:

                c.execute("SELECT au_id from authors where au_id = '%s'" % author_id )
                if c.rowcount == 0:
                        c.execute("INSERT INTO authors (au_id) VALUES ('%s')" % author_id )

                c.execute("SELECT pub_id from publishers where pub_id = '%s'" % row_user['pub_id'] )
                if c.rowcount == 0:
                        c.execute("INSERT INTO publishers (pub_id) VALUES ('%s')" % row_user['pub_id'] )


                #inserts title_id + au_id connection in titleauthor
                statement = ("INSERT INTO titleauthor (au_id, title_id) VALUES ('%s', '%s'" % (author_id, row_user['title_id']))  + ")"
                c.execute(statement)
/*  Fran and Max 2016*/
/* unrelated python code snippet with cursor and prepared statement  from PyMySQL documentation*/

import mysql.connector

cnx = mysql.connector.connect(database='employees')

cursor = cnx.cursor(prepared=True)

stmt = "SELECT fullname FROM employees WHERE id = %s" # (1)
cursor.execute(stmt, (5,))                         # (2)
# ... fetch data ...
cursor.execute(stmt, (10,))                         # (3)
# ... fetch data ...
```

“Dynamic SQL” has several different senses and is often associated with prepared statements: here is our authors’; I will mention another.



1. **EXEC SQL PREPARE** *V* **FROM** <expression>, where *V* is a SQL variable. The expression can be any host-language expression whose value is a string; this string is treated as a SQL statement. Presumably, the SQL statement is parsed and a good way to execute it is found by the SQL system, but the statement is not executed. Rather, the plan for executing the SQL statement becomes the value of *V*.

2. **EXEC SQL EXECUTE** *V*. This statement causes the SQL statement denoted by variable *V* to be executed.

Both steps can be combined into one, with the statement:

EXEC SQL EXECUTE IMMEDIATE <expression>

The disadvantage of combining these two parts is seen if we prepare a statement once and then execute it many times. With **EXECUTE IMMEDIATE** the cost of preparing the statement is paid each time the statement is executed, rather than paid only once, when we prepare it.