# B.Sc. In Software Development. Year 3.
# Applications Programming.
# JDBC.

LIMERICK INSTITUTE OF TECHNOLOGY
SCHOOL OF SCIENCE, ENGINEERING & I.T.
*Department of Information Technology*

# Relational Database Model

Relational database

- Composed of tables
  - Rows called records
  - Columns are fields (attributes)
- First field usually primary key
  - Unique for each record
  - Primary key can be more than one field (column)
  - Primary key not required

# Relational Database Structure

Table: **Employee**

| Number | Name | Department | Salary | Location |
|---|---|---|---|---|
| 23603 | JONES, A. | 413 | 1100 | NEW JERSEY |
| 24568 | KERWIN, R. | 413 | 2000 | NEW JERSEY |
| 34589 | LARSON, P. | 642 | 1800 | LOS ANGELES |
| 35761 | MYERS, B. | 611 | 1400 | ORLANDO |
| 47132 | NEUMANN, C. | 413 | 9000 | NEW JERSEY |
| 78321 | STEPHENS, T. | 611 | 8000 | ORLANDO |

A record

Primary Key

A column

# Relational Databases

## Operations

- Projection
  - Taking a subset of a table
- Join
  - Combining tables to form a larger one
- Example
  - Using previous tables, make list of departments and locations

# Relational Databases

Table: **Employee**

| Number | Name | Department | Salary | Location |
|---|---|---|---|---|
| 23603 | JONES, A. | 413 | 1100 | NEW JERSEY |
| 24568 | KERWIN, R. | 413 | 2000 | NEW JERSEY |
| 34589 | LARSON, P. | 642 | 1800 | LOS ANGELES |
| 35761 | MYERS, B. | 611 | 1400 | ORLANDO |
| 47132 | NEUMANN, C. | 413 | 9000 | NEW JERSEY |
| 78321 | STEPHENS, T. | 611 | 8000 | ORLANDO |

Projection (subset)

| Department | Location |
|---|---|
| 413 | NEW JERSEY |
| 611 | ORLANDO |
| 642 | LOS ANGELES |

# Relational Databases

Advantages of relational databases

1. Tables easy to use, understand, and implement
2. Easy to convert other database structures into relational scheme (universal)
3. Projection and join operations easy to implement
4. Performs better than files
5. Easy to modify - very flexible
6. Greater clarity and visibility than other models

*Common Relational Databases include: MySql, MS SQLServer, MS Access and Oracle.*

# The Books Database

Overview of tables in Books database

1. Authors
2. Publisher
3. AuthorISBN
4. Titles

# The Books Database

**Authors** table

- Four fields
  - **AuthorID** - ID number
  - **FirstName**
  - **LastName**
  - **YearBorn**

| AuthorID | FirstName | LastName | YearBorn |
|---|---|---|---|
| 1 | Harvey | Deitel | 1946 |
| 2 | Paul | Deitel | 1968 |
| 3 | Tem | Nieto | 1969 |

# The Books Database

**Publishers** table

- Two fields
  - **PublisherID** - ID number
  - **PublisherName** - abbreviated name of publisher

| PublisherID | PublisherName |
|---|---|
| 1 | Prentice Hall |
| 2 | Prentice Hall PTR |

# The Books Database

**AuthorISBN** table

- Two fields
  - **ISBN** - ISBN number of book
  - **AuthorID** - ID number of author
- Helps link author with title of book

# The Books Database

| ISBN | AuthorID | ISBN | AuthorID | ISBN | AuthorID |
|---|---|---|---|---|---|
| 0-13-010671-2 | 1 | *(continued from bottom of previous row)* | | *(continued from bottom of previous row)* | |
| 0-13-010671-2 | 2 | 0-13-271974-6 | 1 | 0-13-904947-9 | 1 |
| 0-13-020522-2 | 1 | 0-13-271974-6 | 2 | 0-13-904947-9 | 2 |
| 0-13-020522-2 | 2 | 0-13-456955-5 | 1 | 0-13-904947-9 | 3 |
| 0-13-082925-0 | 2 | 0-13-456955-5 | 2 | 0-13-013249-7 | 1 |
| 0-13-082927-7 | 1 | 0-13-456955-5 | 3 | 0-13-013249-7 | 2 |
| 0-13-082927-7 | 2 | 0-13-528910-6 | 1 | 0-13-085609-6 | 1 |
| 0-13-082928-5 | 1 | 0-13-528910-6 | 2 | 0-13-085609-6 | 2 |
| 0-13-082928-5 | 2 | 0-13-565912-4 | 1 | 0-13-085609-6 | 3 |
| 0-13-082928-5 | 3 | 0-13-226119-7 | 2 | 0-13-016143-8 | 1 |
| 0-13-083054-2 | 1 | 0-13-020522-2 | 3 | 0-13-016143-8 | 2 |
| 0-13-083054-2 | 2 | 0-13-082714-2 | 1 | 0-13-016143-8 | 3 |
| 0-13-083055-0 | 1 | 0-13-082714-2 | 2 | 0-13-015870-4 | 1 |
| 0-13-083055-0 | 2 | 0-13-082925-0 | 1 | 0-13-015870-4 | 2 |
| 0-13-118043-6 | 1 | 0-13-565912-4 | 2 | 0-13-015870-4 | 3 |
| 0-13-118043-6 | 2 | 0-13-565912-4 | 3 | 0-13-012507-5 | 1 |
| 0-13-226119-7 | 1 | 0-13-899394-7 | 1 | 0-13-012507-5 | 2 |
| 0-13-226119-7 | 2 | 0-13-899394-7 | 2 | 0-13-085248-1 | 1 |
| *(continued on top of next row)* | | *(continued on top of next row)* | | 0-13-085248-1 | 2 |

# The Books Database

**Titles** table

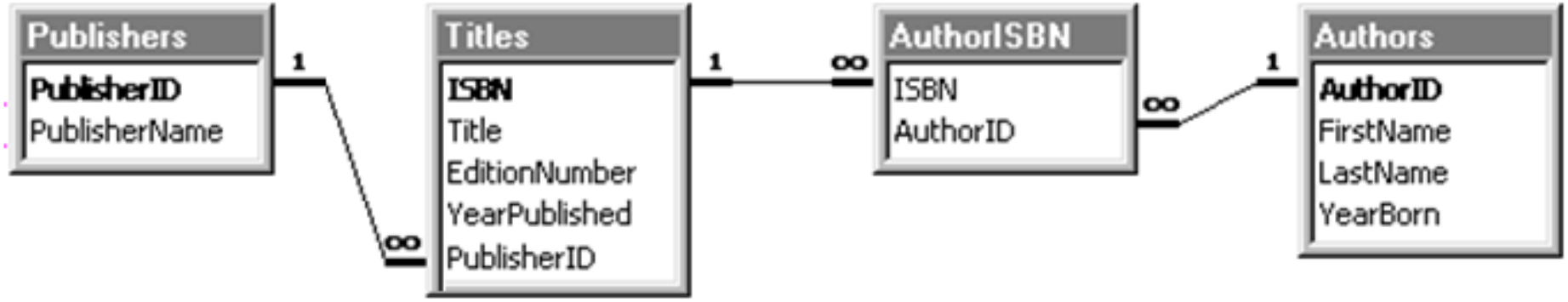
- Five fields
  - **ISBN**
  - **Title** - title of book
  - **EditionNumber**
  - **YearPublished**
  - **PublisherID**
- Table on next slide.

# The Books Database

| ISBN | Title | Edition Number | Year-Published | Publisher ID |
|---|---|---|---|---|
| 0-13-226119-7 | C How to Program | 2 | 1994 | 1 |
| 0-13-528910-6 | C++ How to Program | 2 | 1997 | 1 |
| 0-13-899394-7 | Java How to Program | 2 | 1997 | 1 |
| 0-13-012507-5 | Java How to Program | 3 | 1999 | 1 |
| 0-13-456955-5 | Visual Basic 6 How to Program | 1 | 1998 | 1 |
| 0-13-016143-8 | Internet and World Wide Web How to Program | 1 | 1999 | 1 |
| 0-13-013249-7 | Getting Started with Visual C++ 6 with an Introduction to MFC | 1 | 1999 | 1 |
| 0-13-565912-4 | C++ How to Program Instructor's Manual with Solutions Disk | 2 | 1998 | 1 |
| 0-13-904947-9 | Java How to Program Instructor's Manual with Solution Disk | 2 | 1997 | 1 |
| 0-13-020522-2 | Visual Basic 6 How to Program Instructor's Manual with Solution Disk | 1 | 1999 | 1 |
| 0-13-013870-4 | Internet and World Wide Web How to Program Instructor's Manual with Solutions Disk | 1 | 1999 | 1 |
| 0-13-082925-0 | The Complete C++ Training Course | 2 | 1998 | 2 |
| 0-13-082927-7 | The Complete Java Training Course | 2 | 1997 | 2 |
| 0-13-082928-5 | The Complete Visual Basic 6 Training Course | 1 | 1999 | 2 |
| 0-13-085248-1 | The Complete Java Training Course | 3 | 1999 | 2 |
| 0-13-085609-6 | The Internet and World Wide Web How to Program Complete Training Course | 1 | 1999 | 2 |
| 0-13-082714-2 | C++ How to Program 2/e and Getting Started with Visual C++ 5.0 Tutorial | 2 | 1998 | 1 |
| 0-13-010671-2 | Java How to Program 2/e and Getting Started with Visual J++ 1.1 Tutorial | 2 | 1998 | 1 |
| 0-13-083054-2 | The Complete C++ Training Course 2/e and Getting Started with Visual C++ 5.0 Tutorial | 2 | 1998 | 1 |
| 0-13-083055-0 | The Complete Java Training Course 2/e and Getting Started with Visual J++ 1.1 Tutorial | 2 | 1998 | 1 |
| 0-13-118043-6 | C How to Program | 1 | 1992 | 1 |
| 0-13-271974-6 | Java Multimedia Cyber Classroom | 1 | 1996 | 2 |

# The Books Database

Relationship between tables




- Primary key in **bold**
- Rule of Entity Integrity
  - Every record has unique entry in primary key field

# The Books Database


Lines represent relationship

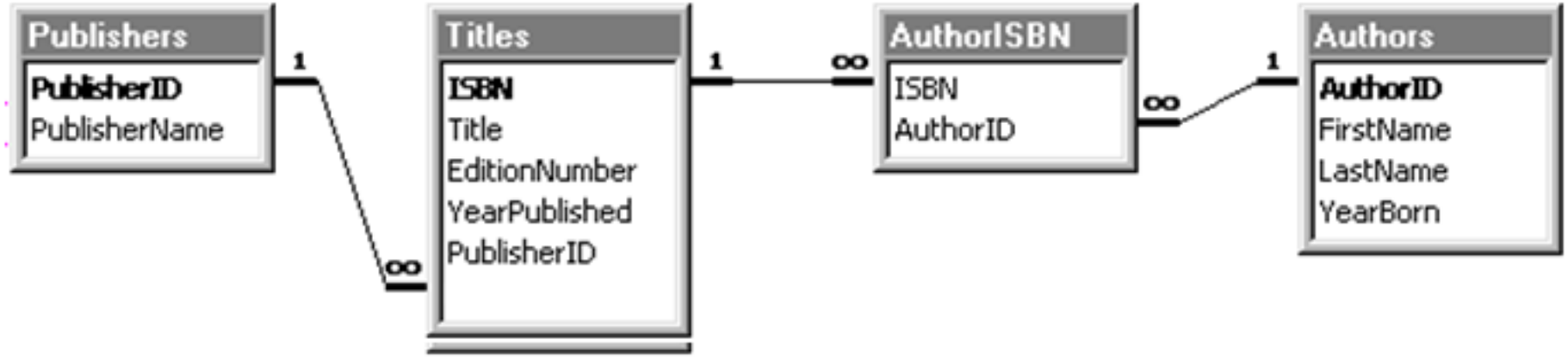
- Line between **Publishers** and **Titles**
  - One to many relationship
  - Every **PublisherID** can appear many times in **Titles** table

Foreign key

- Field in table that is primary field of another table
  - Maintains Rule of Referential Integrity
  - Used to join tables
  - One to many relationship between primary key and corresponding foreign key
- **PublisherID** is foreign key in **Titles** table

# The Books Database

Other relationships:




- One **AuthorID** can appear many times in **AuthorISBN** table
  - Author wrote many books
- One **ISBN** can appear many times in **AuthorISBN**
  - Book had multiple authors

# Structured Query Language - SQL

SQL is a language for storing, manipulating and retrieving data in database.

Important SQL keywords:

*SELECT, FROM, WHERE, GROUP BY, HAVING, ORDER BY, INSERT INTO, UPDATE, DELETE.*

For an excellent SQL tutorial consult W3Schools:

# Notes on the Examples

Examples 1-4 are located in a package called "sd3.com.UsingResultSets". These four examples all use ResultSets.

Examples 5-7 are located in a package called "sd3.com.UsingRowSets". These four examples all use RowSets.

All examples use a MySQL database called books. The Netbeans project for this lecture comes with a SQL script called books.sql. Run this script on your local machine to create the Database and its four tables.

When you run the script it will also create a user account on the DB (the *user* name is *sduser* and the *password* is *pass*)

# Example 1: View All Records In authors Table

Code: sd3.com.UsingResultSets.DisplayRecords

Query: SELECT * FROM authors

Output:

Output - Database Source (run)

```
run:
AuthorID        FirstName       LastName
1               Harvey          Deitel
2               Paul            Deitel
3               Tem             Nieto
```

# Example 2: Insert a Record Into The Authors Table

Code: sd3.com.UsingResultSets.InsertRecord

Query: INSERT INTO authors(AuthorID, FirstName, LastName, YearBorn) VALUES ('4', 'Gerry', Guinane', '1953')

Result:

Output - Database Source (run)

```
run:
1 row inserted

AuthorID        FirstName       LastName
1               Harvey          Deitel
2               Paul            Deitel
3               Tem             Nieto
4               Gerry           Guinane
```

# Example 3: Update A Record In The Authors Table

Code: sd3.com.UsingResultSets.UpdateRecord

Query: UPDATE authors SET FirstName = 'Brendan', LastName = 'Watson', YearBorn = '1967' WHERE LastName = 'Guinane'

Result:

Output - Database Source (run)

```
run:
1 row updated

AuthorID        FirstName       LastName
1               Harvey          Deitel
2               Paul            Deitel
3               Tem             Nieto
4               Brendan         Watson
```

# Example 4: Deleting A Record From The Authors Table

Code: sd3.com.UsingResultSets.DeleteRecord

Query: Delete from authors WHERE LastName = 'Watson'

Result:

Output - Database Source (run)

```
run:
1 row deleted

AuthorID        FirstName       LastName
1               Harvey          Deitel
2               Paul            Deitel
3               Tem             Nieto
```

# The Prepared Statement Interface

The `Statement` interface is used to execute static SQL statements that contain no parameters.

The `PreparedStatement` interface, extending `Statement`, can be used to execute a precompiled SQL statement with parameters.

A `PreparedStatement` object is created using the preparedStatement method in the `Connection` interface. E.G.

```
PreparedStatement pstmt =
connection.prepareStatement("insert into
                Authors(FirstName, LastName) values (?, ?)");
```

# The Prepared Statement Interface

The insert statement has two question marks as placeholders for parameters representing values for FirstName and LastName in a record of the Authors table.

`PreparedStatement` inherits all the methods defined in `Statement`.

It also provides methods for setting the parameters in a `PreparedStatement` object.

These methods are used to set the values for the parameters before executing statements or procedures. In general the methods have the following signature:

```
setX(int parameterIndex, x value);
```

Where X is the type of parameter and parameterIndex is the index of the parameter in the statement.

# The Prepared Statement Interface

The following statements pass the parameters “Alan”, “Ryan” to the placeholders for FirstName and LastName in the PreparedStatement pstmt.

```
pstmt.setString(1, “Alan”);
pstmt.setString(2, “Ryan”);
```

Instead of hard-coding the values of “Alan” and “Ryan” into your program, you can also pass variables name (ones that you have defined in your program) to the `setString` method. For example:

```
pstmt.setString(1, fname);
pstmt.setString(2, lname);
```

# The Prepared Statement Interface

After setting the parameters you can execute the prepared statement by invoking the `executeQuery` method. For example:

```
ResultSet rset = pstmt.executeQuery();
```

Consult the API for information on the `PreparedStatement` interface and the methods it contains. Here are a number of other useful methods.

```
void setDouble(int parameterIndex, double x);
```

=> Sets a double at a specified index.

```
void setInt(int parameterIndex, int x);
```

=> Sets an int at a specified index.

```
void setBoolean(int parameterIndex, boolean x);
```

=> Sets a boolean at a specified index.

# The RowSet Interface

In previous examples you learned to query a database by explicitly establishing a Connection, preparing a Statement, executing a query and then manipulating the resultant `ResultSet`.

With JDBC 2, Sun introduced a new interface called `RowSet` with the desire to simplify database programming.

The RowSet interface inherits from the `ResultSet` interface so it has a lot of similar functionality (as well as additional functionality) to `ResultSet`.

The RowSet interface can be considered as a combination of `Connection`, `Statement` and `ResultSet` into one interface.

# RowSet Basics

There are two types of `RowSet` objects – connected and disconnected.

A connected `RowSet` object connects to the database once and remains connected while the object is in use.

A disconnected `RowSet` object connects to the database, executes a query to retrieve the data (from the database) and then closes the connection. A program may then change the data in a disconnected `RowSet` while it is disconnected from the database. Modified data can be updated in the database later on when a disconnected `RowSet` re-establishes the connection with the database.

# Example 5: Displaying all the records from the authors table using a RowSet

Code: sd3.com.UsingRowSets.DisplayRecordsUsingRowSet

Query: SELECT * FROM authors

Result:

Output - Database Source (run)

```
run:
AuthorID        FirstName       LastName
1               Harvey          Deitel
2               Paul            Deitel
3               Tem             Nieto
```

# PreparedStatements and RowSets

As we saw earlier, `PreparedStatement` introduced the processing of paramaterised SQL statements.

The `RowSet` interface has the capability to support paramaterised SQL statements.

The “set methods” for setting parameter values in `PreparedStatement` are implemented in `RowSet`.

You can use these methods to set parameter values for a paramaterised SQL command.

# Example 6: PreparedStatements & RowSets

Code: sd3.com.UsingRowSets.RowSetAndPreparedStatement

Query: SELECT * FROM authors where lastname = ? and YearBorn < ?

```
rowSet.setString(1, "Deitel");
rowSet.setInt(2, 1960);
```

Result:

```
Output - Database Source (run)
run:
AuthorID        FirstName       LastName
1               Harvey          Deitel
```

# Scrolling and Updating RowSet

By default a `ResultSet` object is not scrollable and updatable.

However, a `RowSet` object is scrollable and updatable.

It is easier to navigate through a `RowSet` than a `ResultSet`.

Using a `RowSet`, methods such as `absolute(int)` can be used to move the cursor to a specific row, while `delete()`, `updateRow()` and `insertRow()` can be used to update the underlying database.

# Example 7: Scrolling and Updating RowSets

Code: sd3.com.UsingRowSets.CompleteRowSetExample

Query: Various: Select + Delete + Update + Insert

Result:

Output - Database Source (run)

```
run:
Before update: Display All Records
1               Harvey          Deitel
2               Paul            Deitel
3               Tem             Nieto

After delete: Display All Records
1               Harvey          Deitel
3               Tem             Nieto

After update: Display All Records
1               Rex             Deitel
3               Tem             Nieto

After insertion: Display All Records
1               Rex             Deitel
3               Tem             Nieto
14              Bertie          Ahern
```

# Batch Updates

A batch update is a batch of updates grouped together, and sent to the database in one "batch", rather than sending the updates one by one.

Sending a batch of updates to the database in one go, is faster than sending them one by one, waiting for each one to finish.

There is less network traffic involved in sending one batch of updates (only 1 round trip), and the database might be able to execute some of the updates in parallel.

The speed up compared to executing the updates one by one, can be quite big.

You can batch both SQL inserts, updates and deletes. It does not make sense to batch select statements.

# Batch Updates

There are two ways to execute batch updates:

1. Using a `Statement`
2. Using a `PreparedStatement`

# Batch Updates Using a Statement

```
30        Statement statement = connection.createStatement();
31
32        statement.addBatch("UPDATE authors set FirstName = 'Pat' WHERE    AuthorID = 1");
33
34        statement.addBatch("UPDATE authors set FirstName = 'John' WHERE AuthorID = 2");
35
36        statement.addBatch("INSERT INTO authors(AuthorID, FirstName, LastName, YearBorn)VALUES ('8', 'Tim', 'Joyce', '1951')");
37
38        statement.addBatch("Delete FROM authors WHERE FirstName = 'Paul'");
39
40        int[] recordsAffected = statement.executeBatch();
41
42        System.out.println("Number of records updated = " + recordsAffected.length);
43
44    }
```

# Batch Updates Using a PreparedStatement

```
28          String sql = "UPDATE authors set FirstName=? , LastName=? where AuthorID=?";
29
            PreparedStatement preparedStatement = null;
31
32          preparedStatement = connection.prepareStatement(sql);
33
34          preparedStatement.setString(1, "Gary");
35          preparedStatement.setString(2, "Larson");
36          preparedStatement.setInt(3, 1);
37          preparedStatement.addBatch();
38
39          preparedStatement.setString(1, "Stan");
40          preparedStatement.setString(2, "Lee");
41          preparedStatement.setInt(3, 4);
42          preparedStatement.addBatch();
43
44          int[] recordsAffected = preparedStatement.executeBatch();
45
46          System.out.println("Number of records updated = " + recordsAffected.length);
```

# Batch Updates and Transactions

It is important to keep in mind, that each update added to a `Statement` or `PreparedStatement` is executed separately by the database.

That means, that some of them may succeed before one of them fails.

All the statements that have succeeded are now applied to the database, but the rest of the updates may not be.

This can result in an inconsistent data in the database.

To avoid this, you can execute the batch update inside a transaction.

When executed inside a transaction you can make sure that either all updates are executed, or none are.

Any successful updates can be rolled back, in case one of the updates fail.

# Batch Updates and Transactions

A transaction is a set of actions to be carried out as a single, atomic action.

Either all of the actions are carried out, or none of them are.

The classic example of when transactions are necessary is the example of bank accounts.

You need to transfer €100 from one account to the other. You do so by subtracting €100 from the first account, and adding €100 to the second account. If this process fails after you have subtracted the $100 from the first bank account, the €100 are never added to the second bank account. The money is lost in cyber space.

To solve this problem the subtraction & addition of the €100 are grouped into a transaction.

If the subtraction succeeds, but the addition fails, you can "rollback" the fist subtraction. That way the database is left in the same state as before the subtraction was executed.

# Batch Updates and Transactions

You start a transaction by this invocation:

```
connection.setAutoCommit(false);
```

You continue to perform database queries and updates. All these actions are part of the transaction.

If any action attempted within the transaction fails, you should rollback the transaction. This is done like this:

```
connection.rollback();
```

If all actions succeed, you should commit the transaction. Committing the transaction makes the actions permanent in the database. Once committed, there is no going back.

```
connection.commit();
```

# Batch Updates and Transactions

```
...

Connection connection = ...

try{

  connection.setAutoCommit(false);

  // create and execute statements etc.

  connection.commit();


} catch(Exception e) {

  connection.rollback();

}
……
```

# Stored Procedures

A `java.sql.CallableStatement` is used to call stored procedures in a database.

A stored procedure is like a function/method in a class, except it lives inside the database.

Some database heavy operations may benefit performance-wise from being executed inside the same memory space as the database server, as a stored procedure.

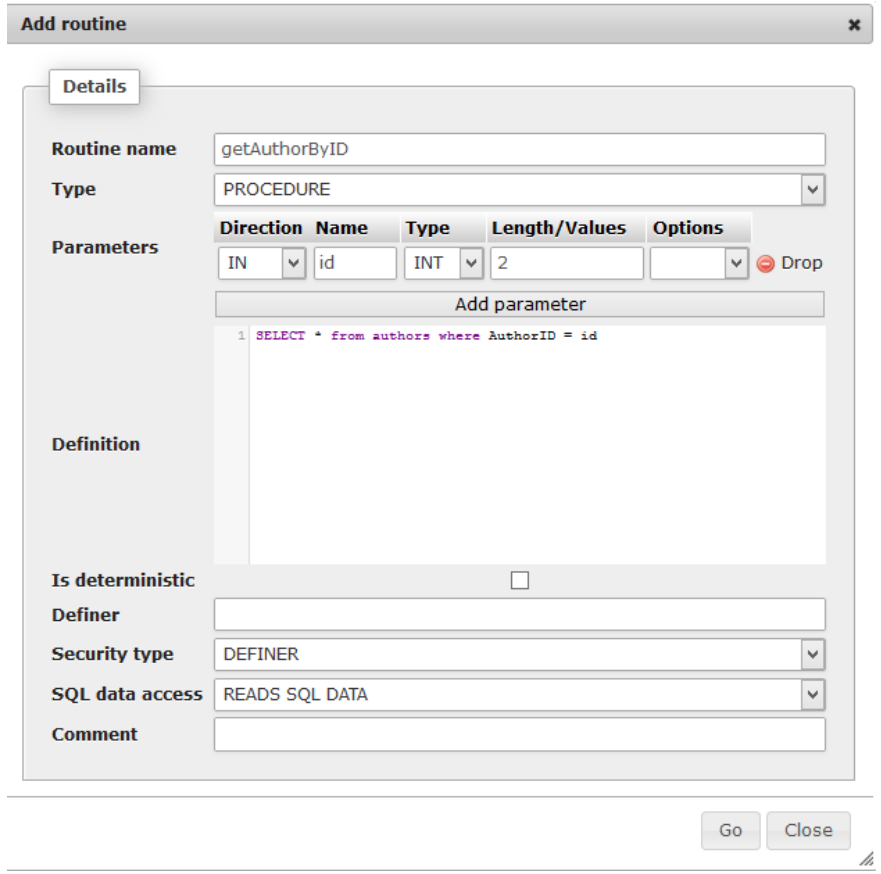
You create an instance of a `CallableStatement` by calling the `prepareCall()` method on a connection object.

```
CallableStatement callableStatement =
       connection.prepareCall("{call getAllAuthors()}");

ResultSet resultSet = callableStatement.executeQuery();
```

# Stored Procedures

Using phpmyadmin its very easy to create a stored procedure. Below is a screenshot of the tool which can be used to create a stored procedure (*getAuthorByID*).



# Stored Procedures

The procedure `getAuthorsByID` is slightly different from the get **`getAllAuthors`** procedure as it accepts a `AuthorID` and uses it to query the authors table.

This has an implication on how its called from within your program.

```
CallableStatement callableStatement =
      connection.prepareCall("{call getAuthorByID(?)}");

callableStatement.setInt   (1, 2);

ResultSet resultSet = callableStatement.executeQuery();
```

The `executeQuery()` method is used if the stored procedure returns a `ResultSet`.

If the stored procedure just updates the database, you can call the `executeUpdate()` method instead.

# Stored Procedures

```
CallableStatement callableStatement =
 connection.prepareCall("{call calculateStatistics(?,
 ?)}");

callableStatement.setString(1, "param1");
callableStatement.setInt (2, 123);
callableStatement.addBatch();

callableStatement.setString(1, "param2");
callableStatement.setInt (2, 456);
callableStatement.addBatch();

int[] updateCounts = callableStatement.executeBatch();
```

# References

Y. Daniel Liang (2014) *Introduction to Java Programming, Comprehensive Version*, 10th edn. Pearson.

Deitel P.J, Deitel H.M. (2014) *Java How To Program (Early Objects)*, 10th edn. Pearson.

https://www.w3schools.com/sql/default.asp

http://tutorials.jenkov.com/jdbc/index.html