

DB Programming

Database Systems

Presented by Yael Amsterdamer

Agenda

- × Basic MySQL Usage
- × Little More Complex MySQL stuff..
- × JDBC
- × Coding Tips

MySQL Data Types

There are 3 main groups of types:

- × Numeric
- × Date
- × String

× <http://dev.mysql.com/doc/refman/5.6/en/data-types.html>

MySQL Data Types - Numeric

× Integers

Type	Storage (Bytes)	Minimum Value (Signed/Unsigned)	Maximum Value Signed/Unsigned)
<u>TINYINT</u>	1	-128	127
		0	255
<u>SMALLINT</u>	2	-32768	32767
		0	65535
<u>MEDIUMINT</u>	3	-8388608	8388607
		0	16777215
<u>INT</u>	4	-2147483648	2147483647
		0	4294967295
<u>BIGINT</u>	8	-9223372036854775808	9223372036854775807
		0	18446744073709551615

× INT(M) – sets the display width.

Numeric (Floating-Point)

Approximate Value

- × Float/Double
- × Float(M, D) – M =#digits, D =#digits after “.”
 - Float(7,4) will look like -999.9999

Exact-Value

- × Decimal (==Numeric)
 - Decimal(5,2) range from -999.99 to 999.99

Numeric (Bit)

- × $\text{Bit}(M)$ – number of bits..
- × $\text{Bit} = \text{Bit}(1)$

MySQL Data Types – Date/Time

- ✖ Date - range is '1000-01-01' to '9999-12-31'
- ✖ DateTime - 'YYYY-MM-DD HH:MM:SS'
- ✖ Timestamp - range is '1970-01-01 00:00:01' to '2038-01-19 03:14:07'
(epoch time)

MySQL Data Types – Date/Time

× Zero values

Data Type	“Zero” Value
<u>DATETIME</u>	'0000-00-00 00:00:00'
<u>DATE</u>	'0000-00-00'
<u>TIMESTAMP</u>	'0000-00-00 00:00:00'
<u>TIME</u>	'00:00:00'
<u>YEAR</u>	0000

× ODBC can't handle 0 → convert to null

× (Use the table for the types..)

MySQL Data Types – Date/Time

× Storage

Data Type	Storage Required Before MySQL 5.6.4	Storage Required as of MySQL 5.6.4
<u>YEAR</u>	1 byte	1 byte
<u>DATE</u>	3 bytes	3 bytes
<u>TIME</u>	3 bytes	3 bytes + fractional seconds storage
<u>DATETIME</u>	8 bytes	5 bytes + fractional seconds storage
<u>TIMESTAMP</u>	4 bytes	4 bytes + fractional seconds storage

MySQL Data Types – Date/Time

× Important Functions

Date_format, Datediff, Dayname.....

<http://dev.mysql.com/doc/refman/5.6/en/date-and-time-functions.html>

MySQL Data Types - String

- × Char and Varchar are similar but differ in:
 - Storage – Chars are “padded”
 - Max length: char(255), varchar(65535)

Value	CHAR (4)	Storage Required	VARCHAR (4)	Storage Required
' '	' '	4 bytes	' '	1 byte
'ab'	'ab '	4 bytes	'ab'	3 bytes
'abcd'	'abcd'	4 bytes	'abcd'	5 bytes
'abcdefgh'	'abcd'	4 bytes	'abcd'	5 bytes

MySQL Data Types - String

- ✖ For larger size use **Blob** and **Text**
- ✖ **Blob** - binary strings (byte strings). They have no character set..
- ✖ **Text** - They have a character set, and values are sorted and compared based on the character set.

MySQL Data Types - String

× Blob - TINYBLOB
 BLOB
 MEDIUMBLOB
 LONGBLOB

× Text - TINYTEXT
 TEXT
 MEDIUMTEXT
 LONGTEXT

MySQL Data Types - String

Data Type	Storage Required
<code>CHAR (M)</code>	$M \times w$ bytes, $0 \leq M \leq 255$, where w is the number of bytes required for the maximum-length character in the character set
<code>BINARY (M)</code>	M bytes, $0 \leq M \leq 255$
<code>VARCHAR (M)</code> , <code>VARBINARY (M)</code>	$L + 1$ bytes if column values require 0 – 255 bytes, $L + 2$ bytes if values may require more than 255 bytes
<u><code>TINYBLOB</code></u> , <u><code>TINYTEXT</code></u>	$L + 1$ bytes, where $L < 2^8$
<u><code>BLOB</code></u> , <u><code>TEXT</code></u>	$L + 2$ bytes, where $L < 2^{16}$
<u><code>MEDIUMBLOB</code></u> , <u><code>MEDIUMTEXT</code></u>	$L + 3$ bytes, where $L < 2^{24}$
<u><code>LONGBLOB</code></u> , <u><code>LONGTEXT</code></u>	$L + 4$ bytes, where $L < 2^{32}$

Define Foreign keys

- ✖ Don't forget to define the primary key on the other table..
- ✖ What happens when you delete the “key record” from the “primary table”?
 - Restrict
 - Cascade
 - Set null

Define Foreign keys

student

Foreign Key Name	Referenced Table	Column	Referenced C...
fk_1	`test`.`city`	☐ student_id	
		☐ student_name	
		☒ city_id	city_id

Foreign Key Options

On Update: RESTRICT

On Delete: RESTRICT

☐ Skip in SQL generation

Foreign Key Comment

Table Columns Indexes Foreign Keys Triggers Partitioning Options

DBMS feedback messages will go here upon applying changes.

Apply Revert Close

Basic workbench usage - Demo

× Demo..

- create table (data types)
- define primary key
- define foreign keys (insert / delete data)

Agenda

- × Basic MySQL Usage
- × (A bit) More Complex MySQL stuff..
- × JDBC
- × Coding Tips

Index

- ✖ Index improves the speed of operations on a table
- ✖ Can be created using one or more fields
- ✖ You will later learn more..
- ✖ But don't forget, its important

Index - HowTo

student

Index Name	Type
PRIMARY	PRIMARY
fk_1	INDEX
index_1	INDEX

Column	#	Order	Len...
☐ student_id		ASC	
☐ student_name		ASC	
☒ city_id	1	ASC	

Index Options

Storage:

Key Block: 0

Parser:

Index Comment:

Table Columns Indexes Foreign Keys Triggers Partitioning Options

Changes have been applied OK

Apply Revert Close

“AutoNumber”

ID	NAME
1	Rubi
2	Tova
3	Itay
4	Dvir
...	...

✖ How do you know how to assign an ID??

“AutoNumber” – Algorithm?

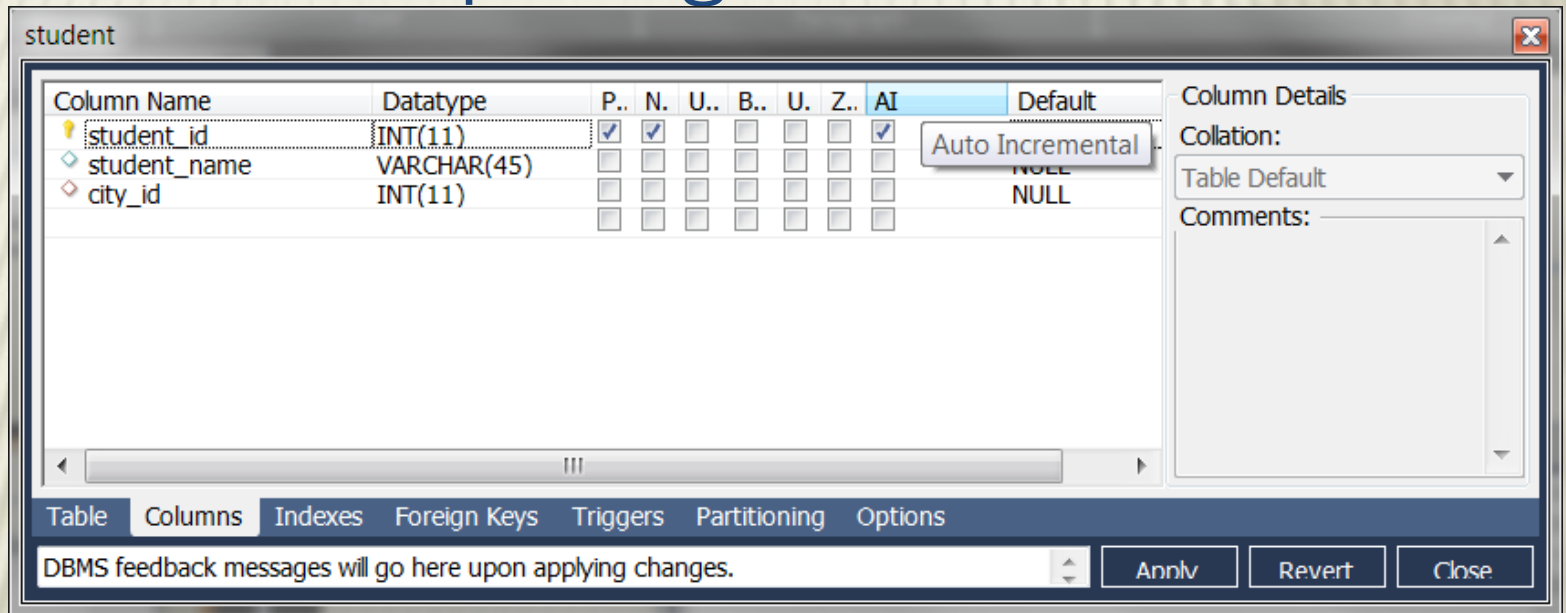
Lock table

```
new_id      =      1 + select max id from table  
insert into table values(new_id, "Rubi");
```

Unlock table

MySQL – Life is easy...

✖ Just mark a simple flag..



✖ In Oracle you need to define a “Sequence” and to use it via a “Trigger”..

Triggers

× A database trigger is procedural code that is automatically executed in response to certain events on a particular table

× Events:

BEFORE INSERT

AFTER INSERT

BEFORE UPDATE

AFTER UPDATE

BEFORE DELETE

AFTER DELETE

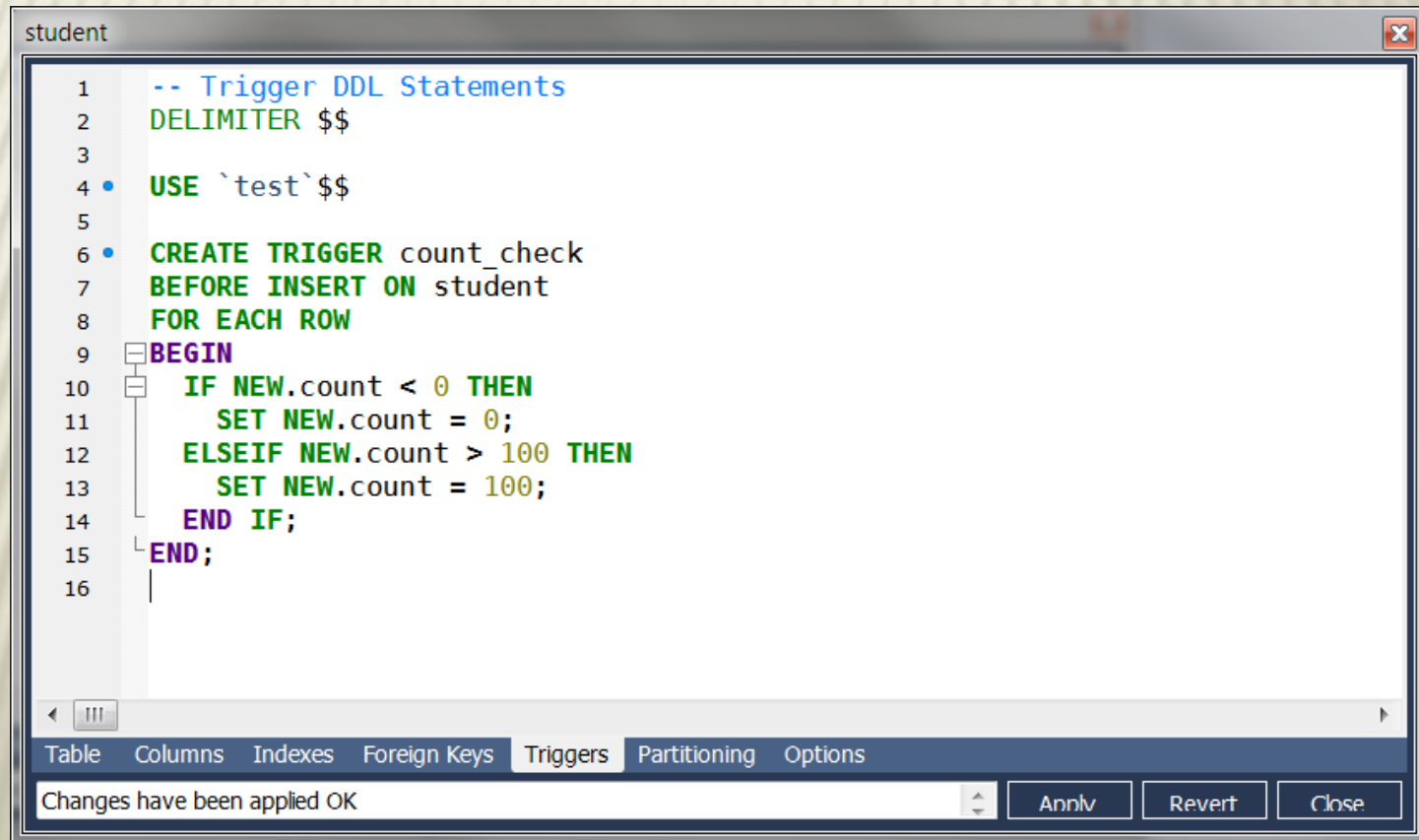
Triggers – Row Level

× Occurs for each row

```
CREATE OR REPLACE TRIGGER <trigger_name>
<BEFORE | AFTER> <ACTION> ON <table_name>
FOR EACH ROW
    BEGIN
        <trigger_code>
    END;
```

Triggers – Row Level – Example

- ✗ You can not “just use the GUI” - you need to “code” the trigger



The screenshot shows a MySQL GUI window titled 'student'. The main text area contains the following SQL code:

```
1  -- Trigger DDL Statements
2  DELIMITER $$
3
4  •  USE `test`$$
5
6  •  CREATE TRIGGER count_check
7  BEFORE INSERT ON student
8  FOR EACH ROW
9  BEGIN
10     IF NEW.count < 0 THEN
11         SET NEW.count = 0;
12     ELSEIF NEW.count > 100 THEN
13         SET NEW.count = 100;
14     END IF;
15 END;
16
```

The code is color-coded: comments are blue, keywords are green, and identifiers are black. The GUI has a tabbed interface at the bottom with tabs for 'Table', 'Columns', 'Indexes', 'Foreign Keys', 'Triggers' (which is selected), 'Partitioning', and 'Options'. Below the tabs is a status bar that says 'Changes have been applied OK' and three buttons: 'Apply', 'Revert', and 'Close'.

Triggers – Row Level – Example

- ✗ Use “NEW” to refer to the row

```
CREATE TRIGGER count_check
BEFORE INSERT ON student
FOR EACH ROW
BEGIN
    IF NEW.count < 0 THEN
        SET NEW.count = 0;
    ELSEIF NEW.count > 100 THEN
        SET NEW.count = 100;
    END IF;
END;
```

Limit the Results

- ✖ What if your query returns 1,000,000 results?
- ✖ How to return the TOP n results
- ✖ How to return the results from n to m

MySQL's Limit

- ✖ Very simple... just use the “Limit” keyword

`LIMIT [offset,] row_count`

- ✖ `SELECT * FROM sakila.film limit 10,5`

Oracle's Rownum – NOT THAT SIMPLE!

- ✖ Its assigned BEFORE sorting or aggregation
- ✖ ROWNUM value is incremented only after it is assigned

Little More Complex MySQL Stuff

- × Demo..
 - Create index
 - Create “Autonumber”:
 - Create Sequence
 - Create Trigger
 - Create Trigger
 - Limit the results..

Table Engine – InnoDB vs MyISAM

- ✖ A schema can contain tables of different storage engines
- ✖ Depends on the usage.

<http://www.kavoir.com/2009/09/mysql-engines-innodb-vs-mysam-a-comparison-of-pros-and-cons.html>

InnoDB Advantages

- ✖ strict in **data integrity**
 - supports foreign keys
 - supports transactions(MyISAM does not..)
- ✖ Row-level lock for insert and update
(MyISAM is Table-level)
- ✖ Better crash recovery

MyISAM Advantages

- × Full-text Indexing!

(InnoDB does not..)

- × Faster...

- Reads are more efficient

- When a single user use the system (y?),
batch inserts are faster

How to choose?

Not so simple... but here are a few rules:

- ✖ If you need foreign keys → InnoDB
- ✖ If you need transactions → InnoDB
- ✖ If you need Fulltext Index → MyISAM
- ✖ More speed → MyISAM
BUT only if not used by users simultaneously

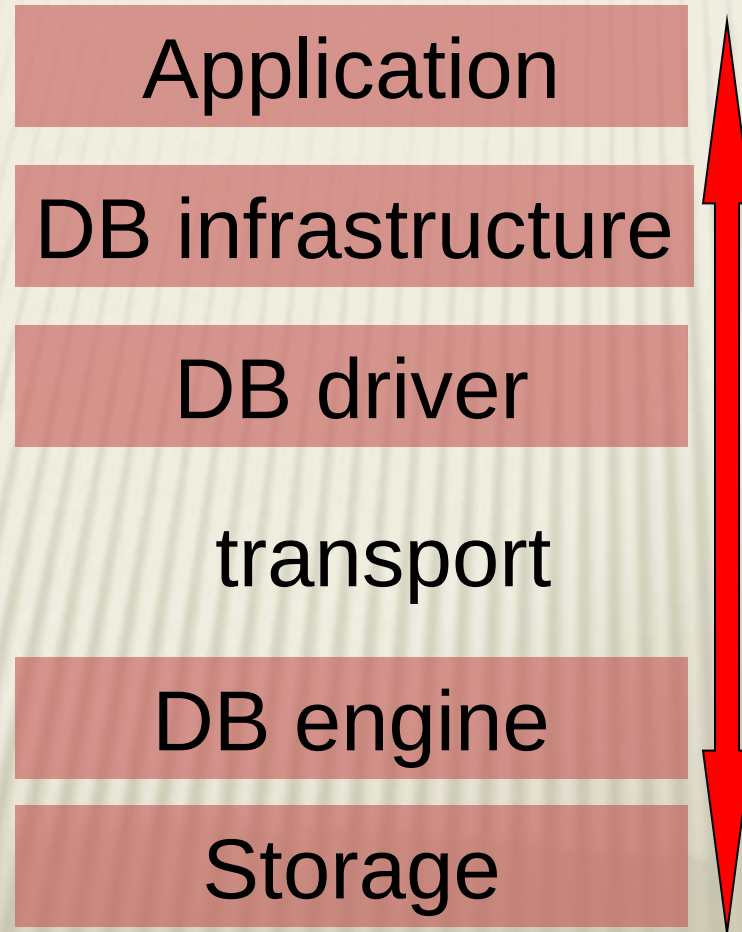
Important Tip

- ✖ Most tables in the project should be InnoDB
- ✖ BUT, if you need (and there is a good chance for that) FULL-TEXT-INDEX you have to create a table with MyISAM
- ✖ (you can create a specific table with only the field you want to index (and the key), and leave all other fields on a different InnoDB table)

Agenda

- × Basic MySQL Usage
- × Little More Complex MySQL stuff..
- × JDBC
- × Coding Tips

During the last episode...



Concepts vs APIs

Concepts

APIs/Language

Connection

Connection pooling

Error Handling

Fetching results

Rowset

Prepared statements

Batch processing

X

ODBC

JDBC

OCI/OC CI

ADO.NET

ODBC – Open Database Connectivity API

- × Pros:

- + Cross platform and cross databases
- + Easy to use

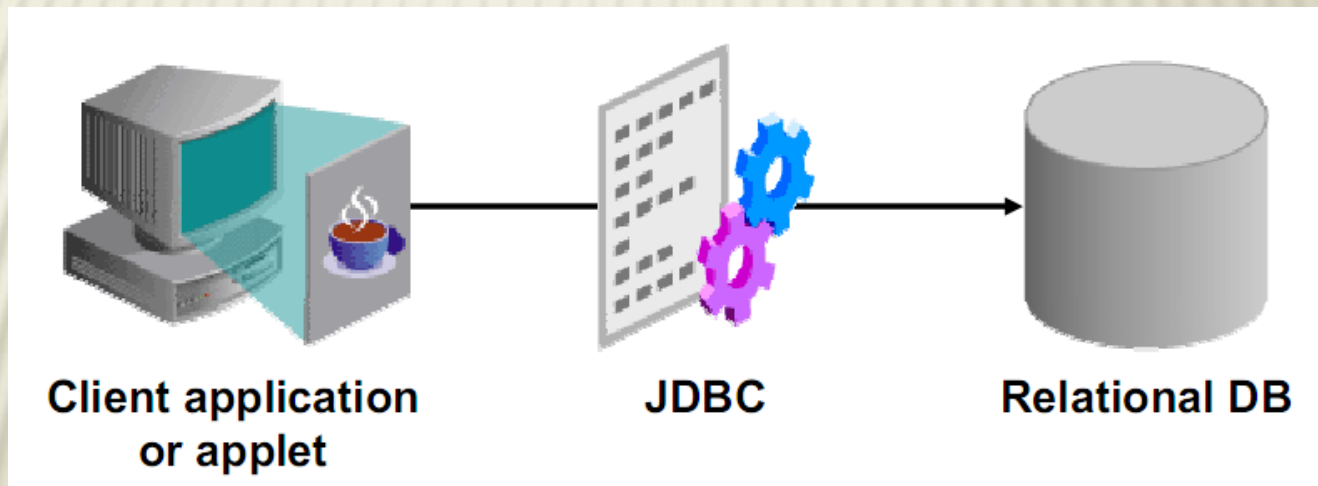
- × Cons:

- + Too low level

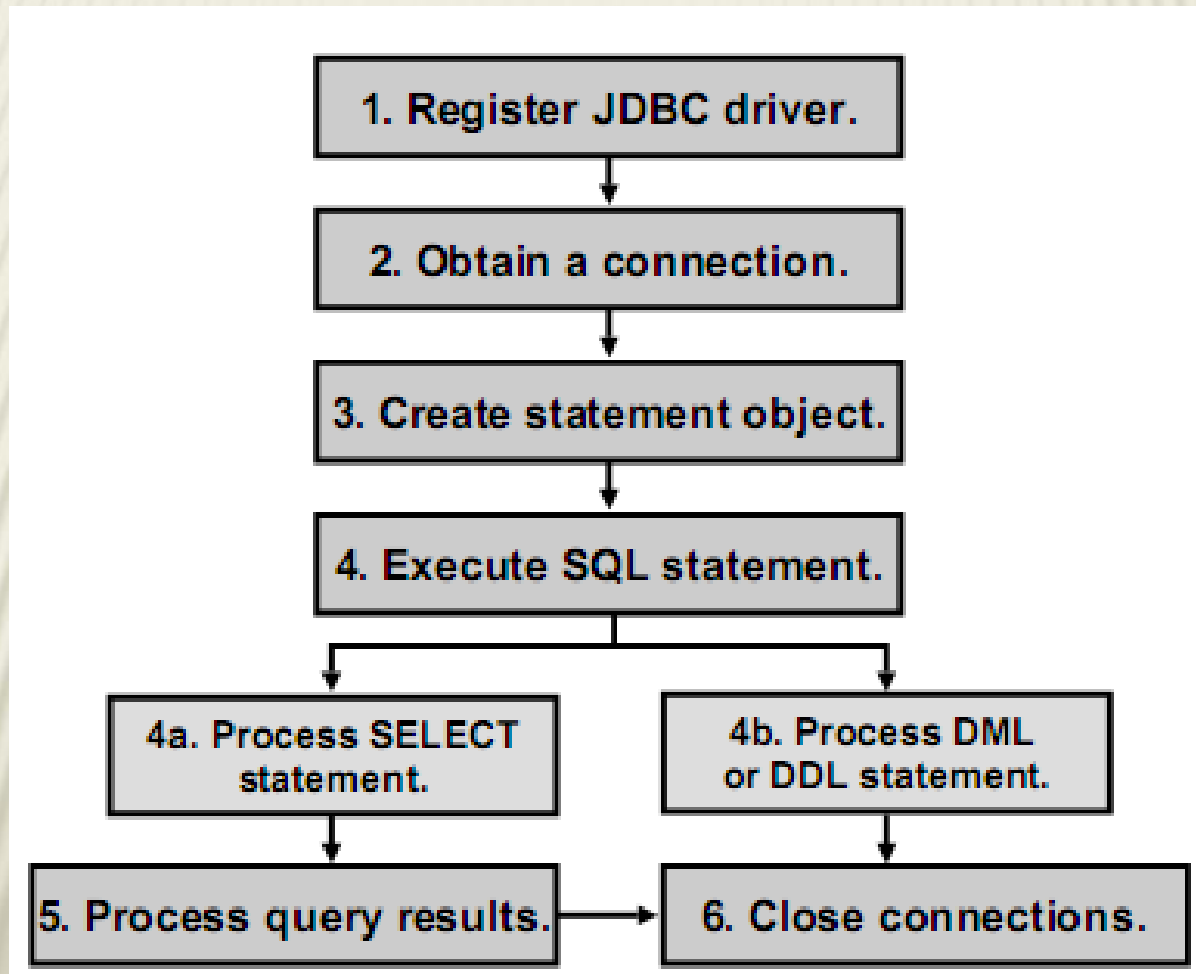
- × We won't use it, But it's very common

JDBC

- ✧ JDBC is a standard interface for connecting to relational databases from Java



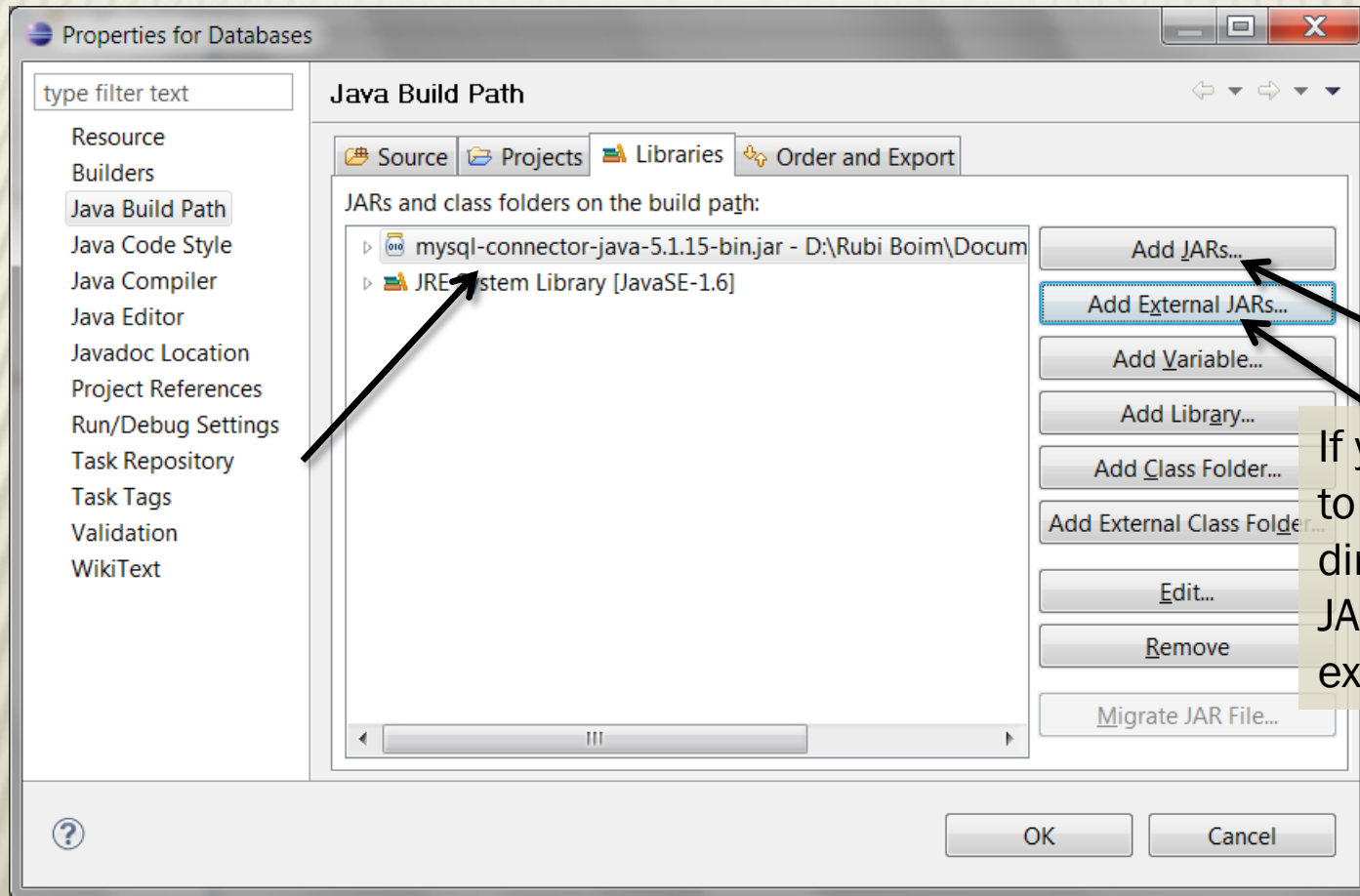
How to execute SQL using JDBC



Preparing the Environment 1

- × Download MySQL's JDBC driver:
<http://www.mysql.com/downloads/connector/j/>
- × Can also be found at the course page
- × Setup Eclipse:
 - add “mysql-connector-java-xx.xx.xx-bin.jar” to the project

Preparing the Environment 2



If you copy the jar file to the project directory, press “add JAR”. Otherwise, “Add external JAR”

Preparing the Environment 3

- ✖ `import java.sql.*` (JDBC API)
- ✖ Register the driver in the code:
`Class.forName("com.mysql.jdbc.Driver");`

Opening a Connection

- ✗ Connection class - java.sql.Connection
- ✗ use the DriverManager with JDBC URL

```
conn = DriverManager.getConnection(  
    "jdbc:mysql://host_addr:port/schema_name"  
    "username",  
    "password");
```

Creating a Statement

- × Created from the connection object

```
Statement stmt = conn.createStatement();
```

Using a Statement

Three different methods:

- ✖ `executeQuery(String)` for `SELECT` statements
returns `ResultSet`
- ✖ `executeUpdate(String)` for `DML/DDL`
returns `int`
- ✖ `execute(String)` for any `SQL` statement
returns `boolean`

executeQuery & ResultSet

ResultSet:

- ✖ Maintains a cursor to its current row
- ✖ Provides methods for retrieving values:
getInt(), getDate(), getString()..
- ✖ Fields can be identified by name or order:
getXXX("Name")
getXXX(2)

executeQuery & ResultSet

- ✖ Initially the cursor is positioned before the first row

```
stmt = conn.createStatement();  
rs    = stmt.executeQuery(  
        "SELECT * FROM employees");  
while (rs.next() == true)  
    System.out.println(rs.getString("field"));
```


executeUpdate

- ✖ Again, via the statement
- ✖ Execute DDL or DML
- ✖ Returns Int for DML, 0 for DDL

executeUpdate

```
stmt=conn.createStatement();  
result=stmt.executeUpdate(  
    "DELETE FROM demo");
```

× Demo..

execute

- ✖ Executes any command for the DB
- ✖ Returns boolean (success/failure)

Closing Connections

- × Important! So don't forget..
- × `ResultSet.close()`
- × `Statement.close()`
- × `Connection.close()`

Transactions

- ✖ By default, connections are “autocommit”
- ✖ Can be disabled by:
`conn.setAutoCommit(false)`
- ✖ Commit a transaction: `conn.commit()`
- ✖ Rollback a transaction: `conn.rollback()`

Transactions – When to use?

- ✖ In general, in any logical operations that involve more than one call:
insert/update/delete in several tables
- ✖ Inconsistent data is unacceptable!
- ✖ Don't forget to use!

PreparedStatement

- ✖ Prevents reparsing of SQL statements
- ✖ Used for statements executed more than once
- ✖ Saves time
- ✖ Nicer code

PreparedStatement - how

- × Specify a variable by “?”

```
PreparedStatement pstmt = conn.prepareStatement(  
    "INSERT INTO demo(fname, lname) VALUES(?, ?)");
```

- × Supply values for the variables:

```
pstmt.setXXX(index, value)
```

- × Execute the statement

```
pstmt.executeUpdate();
```

PreparedStatement - example

```
PreparedStatement pstmt = conn.prepareStatement(  
    "INSERT INTO demo(fname, lname) VALUES(?, ?)");
```

```
pstmt.setString(1, "Rubi");  
pstmt.setString(2, "Boim");  
pstmt.executeUpdate();
```

```
pstmt.setString(1, "Tova");  
pstmt.setString(2, "Milo");  
pstmt.executeUpdate();
```

× Demo..

Batch PreparedStatement

- ✖ PreparedStatement can be slow for long calls
- ✖ Batch together all the calls!
- ✖ I.e., instead of 50,000 calls, do one call with 50,000 parameters
- ✖ Improves performance

Batch PreparedStatement - how

- ✖ Instead of `pstmt.executeUpdate()`
do `pstmt.addBatch()`
- ✖ After all statement are added to the batch:
`int[] = pstmt.executeBatch()`
- ✖ TIP: don't batch too much together
- ✖ Demo..

The effect of table engines

- ✗ Don't forget the difference between the table engines...

InnoDB vs. MyISAM

(InnoDB = Foreign keys...
MyISAM = Super speed (for single user..))

Major Improvement

- ✖ You can insert massive data into InnoDB table with MyISAM speeds:
- ✖ Currently – each insert is a single transaction
- ✖ Instead – create one big transaction for the entire batch

```
conn.setAutoCommit(false);
```

```
batch code....
```

```
conn.commit();
```

How to insert with AutoNumber

- ✖ Assuming you created a trigger similar to the one showed before..
(MySQL == Built-in..)
- ✖ Specify the exact fields in the “Insert”

ID	NAME
1	Yonni
2	Tova
3	Dvir

```
INSERT INTO test(name) VALUES('Rubi');
```

Retrieving the AutoNumber Generated

- ✖ When calling “executeUpdate”, you can specify which fields you want to “get back”
- ✖ After executing, use getGeneratedKeys() to retrieve a ResultSet with the returned fields

```
stmt.executeUpdate("INSERT INTO  
    demo(fname, lname)  
    VALUES('Rubi','Boim')",  
        new String[]{"ID"});  
rs=stmt.getGeneratedKeys();  
rs.next();  
id=rs.getInt(1);
```

Retrieving the AutoNumber Generated

- × Demo.. (I.E. there is an example code ☺)

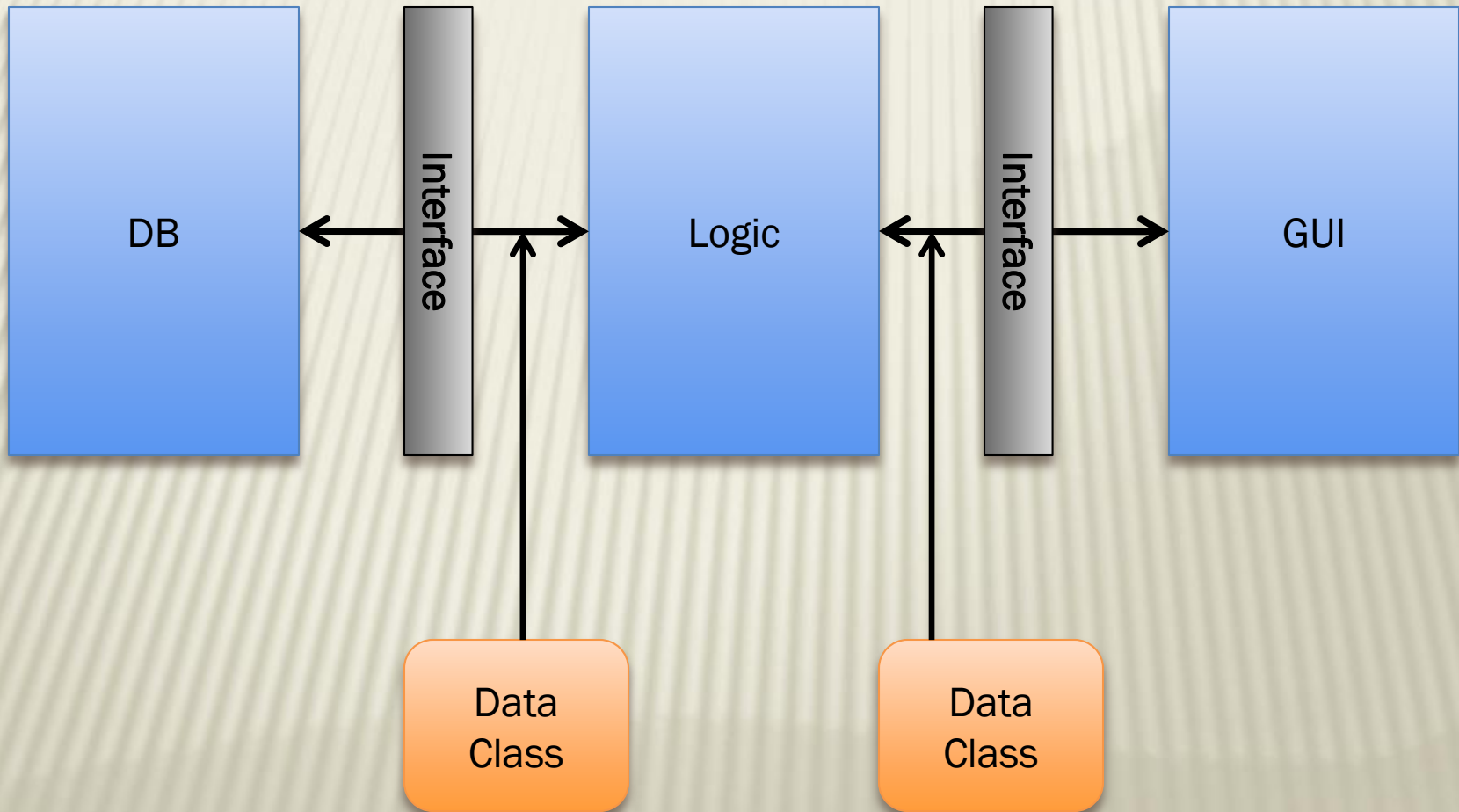
Agenda

- × Basic MySQL Usage
- × Little More Complex MySQL stuff..
- × JDBC
- × Coding Tips

Layering

- × Separate the GUI!
- × Separate the DB!
- × Use classes to describe entities
- × Use interfaces!

Layering



Reuse & Encapsulation

- × Identify main processes
- × Abstract implementation
- × Reuse..
- × NO COPY PASTE CODE

Don't create too many functions

- × Search for movies:
searchMovieByName()
searchMovieByDate()
..
- × It's the same query! just different “where” →
manipulate the “where” in the function:
SearchMovie(searchOptions?)
- × Not so easy on some parameters..
searchMovieByActors()
searchMovieByActorsAndDate()

Configuration

- ✖ Your program will have several (many) variables:
 - server address
 - textfile location
 - number of connections/threads
 -
- ✖ Do not “hard code” them
- ✖ *.ini file, easy GUI,

Schema

- ✖ Well, you should be an expert by now. 😊
- ✖ Primary Key - ALWAYS an integer!
- ✖ Use indexes to speed up (but not on every field)

Testing

- × Obvious, no?
- × Try installing / running your program on different computers
- × Connection drops
- × Validate user input (date, number, special chars..)
- × Your program should never fail!!

Good questions...

- × Managing Database Connections
- × Managing Security
- × Managing Threads
- × Error handling

How to insert Strings

- ✖ In an SQL Query, strings are surrounded by ‘
- ✖ But what if we want to insert the char ‘?

INSERT INTO test VALUES('It's a test');

- ✖ Simply add another ‘

INSERT INTO test VALUES('It''s a test');

Important Tip for Manipulating Data

- ✗ Maybe your program uses DD/MM/YYYY.. You need to flip it...
- ✗ What if tomorrow you switch to MySQL??
- ✗ Create your own functions for adjusting types (not just dates)

```
String fixDate(String old_date) {  
    return yourFlipDateFormatFunc(old_date);    //mysql  
    //return "to_date('" + old_date + "', 'dd/mm/yyyy')"    // oracle  
}
```

```
stmt.executeUpdate(  
    "INSERT INTO demo(fname, lname, mydate) VALUES('Rubi',  
        'Boim'," + fixDate('13/12/2008') + ")");
```

Connection Pooling

- ✗ Opening a connection is “expensive”
- ✗ Multi tasks requires multi connections
- ✗ You should open a connection only when you need it (I.E. when a task asks for connection and there is no one available)
- ✗ When the task is done, do not close the connection but return it to the pool for future use

Connection Pooling – example

- ✖ Example of what might it look..

```
MyConn conn = cManager.poolConn();  
conn.getJDBCConn.executeQuery(..);
```

```
conn.returnConnection();      OR  
cManager.returnConn(conn)
```

- ✖ Implement it your own way, but be sure to use
“synchronized”

Thread Pooling

- ✖ If you build your application correctly, the GUI should be separate from the “program”
- ✖ Same concept as the Connection Pooling
- ✖ More about it when we talk about the GUI

Coding tips

- ✖ The following next slides are EXAMPLES for what **NOT-TO-DO** in the project. Basically they are based on last years submissions, which were altered to express important points.

Database Design

ID	Number
CD_NAME	NVARCHAR(50)
ARTIST_NAME	NVARCHAR(50)
GENRE	NVARCHAR(50)
YEAR	DATE

- ✗ Don't forget to normalize the DB
- ✗ Use the right types

Usability

- ✗ “non-refreshed” windows
- ✗ “Hanging” windows
- ✗ “Please wait...Your query may take a couple of minutes...”

Usability II

New search

Disc search

Title

Year

Genre

On sale

Track search

Title

connect

close

search

Result table:

Thank you