

OVERVIEW

This project includes creating a database using MySQL and writing a Java or C# program to access the database. Batch processing is used – i.e., the queries (data retrieval and update requests) are read from a Transaction file, and output responses are sent to a Log file.

BASIC STEPS

1. Download & install MySQL.
2. Download & install appropriate connector for MySQL which allows Java or C# to interact with a MySQL database. *[C# people need Visual Studio, not just Microsoft C# Express Edition – see staff in room C-208 for help on this since CS Department no longer has an MSDN license, but CEAS does].*
3. SETUP: Use MySQL and appropriate script files to create WorldDB consisting of Country and CountryLanguage tables (though not City table). *[No Java/C# program is involved here].*
4. USERAPP – a program with additional class(es)
 - Write a C# or Java program which does batch processing (using the “Sequential Stream Processing Algorithm” *[i.e., loop til EOF {read 1 transaction, then deal with it completely}].*
 - Handling of the Trans file and Log file may be done in the main program or a UI class.
 - The 4 transaction handler methods (RetrieveData, InsertData, DeleteData, UpdateData) may be in the main program or a separate class.
 - The actual SQL DB handling methods are in a separate class (in a physically separate file). There could be EITHER:
 - 4 separate DB access methods (one each for handling SELECTs, INSERTs, DELETEs, UPDATEs)

- OR 2 separate DB access methods (one for handling SELECTs and a generic one for handling INSERTs/DELETEs/UPDATEs)
- But there MUST NOT be multiple methods of each type of QUERY (which my example program used for demonstration purposes only)!!!

LOG FILE (.txt)

For each transaction:

1. Echo the original transaction request from the transaction file
2. For INSERTs and DELETEs (but not SELECTs or UPDATEs), display the actual SQL statement which your program built and sent to the DBS for processing
3. Show the results from executing the “query” – i.e.,
 - a. a “table” for a SELECT
 - b. a reassurance message for an INSERTs / DELETEs / UPDATEs:
OK, INSERT WORKED
OK, DELETE WORKED
OK, UPDATE WORKED
4. Write out a blank line.

TRANSACTION FILE (.txt)

Each transaction starts on a new line, where the 1st column is the tranCode:

S, I, D, U (always caps).

S(elect) transactions (RETRIEVE data from DB)

- transaction data is an **actual SQL statement** to be used “as is”.
[This is not a common/proper programming approach, but time is short...].
- Allow for 0 or 1 or Many ROWS to be returned to the program from the DBS.
- **rdr.FieldCount** gives the number of COLUMNS there’ll be for a result set
- “table” printed to Log file does **NOT NEED TO**:
 - be in a box (e.g., like a typical interactive result in the command window)
 - have column headings (but if it does,
DON’T use what’s in my demo program&
DON’T hardcode the heads (as my demo program does)
- be perfectly aligned since this **SINGLE GENERIC METHOD** doesn’t know what data type **rdr[0]** or **rdr[1]** or the other columns are.
[Given more time for this asgn, then yes, the program could gather the necessary column names & column data types & column data sizes from the DB itself to be able to create a “nice” output report].

U(pdate) transactions (CHANGE data in DB)

- transaction data is an **actual SQL statement** to be used “as is”

D(elete) transactions (CHANGE data in DB)

- transaction data is **NOT an SQL statement**

- your **program has to construct it from the “parameters” supplied**

- Basic format for a simple DELETE SQL statement:

```
DELETE FROM Country WHERE Name = 'Disneyland'
```

Transaction data (i.e., the parameters) would look like:

```
D Country|Name| 'Disneyland'
```

I(nsert) transactions (CHANGE data in DB)

- transaction data is **NOT an SQL statement**

- your **program has to construct it from the “parameters” supplied**
using various string-handling methods (e.g., Split, +, etc.)

[See sample C# code which does part of this: StringHandling.cs].

- 2 basic formats for simple SQL INSERT statements:

1) **all-columns** INSERT (so column names NOT specified in SQL statement):

```
INSERT INTO CountryLanguage VALUES ('USA', 'C#', 'F', 0.01)
```

Transaction data (parameters) looks like:

```
I CountryLanguage| 'USA', 'C#', 'F', 0.01
```

2) **some-columns** INSERT (so column names MUST BE specified in SQL stmt):

```
INSERT INTO Country(Code, Name) VALUES ('HEX', 'Hexland')
```

Transaction data (parameters) looks like:

```
I Country|(Code, Name)| 'HEX', 'Hexland'
```

NOTES

1. The project MUST be modular, using good descriptive naming (as shown in specs), a comment-line-of-*s between modules, etc.
2. My demo programs use hard-coded queries, hard-coded column-headings, multiple retrieval methods, etc. just to demonstrate various C#/Java language concepts regarding connecting and interacting with an existing DB.

DO NOT DO THIS FOR THE ASGN.

3. My examples used MySQL 5.1. Newer versions should work the same.
4. I've used the term “Transaction” (e.g., Tran file, TranCode) thus far in CS3310 (and in this Asgn) in a **data FILE** context. It has a somewhat different meaning

in a **DBS context** - i.e., A “*Transaction*” is an ordered set of DB operations (a series of SQL SELECT/INSERT/DELETE/UPDATE statements, perhaps embedded in some procedural program code) that are considered as a **SINGLE UNIT**, where **ALL** the DB-changing operations have to work successfully in order for the transaction to succeed. If ANY ONE of the SQL DB-changing operations fails to work, all earlier SQL operations in this current transaction are **ROLLED BACK** to **UNDO** the changes they made to the DB.

5. I've used the term “Query” thus far in CS3310 to mean just RETRIEVAL of a SINGLE data item. In a DBS context, Query (e.g., SQL, Structured Query Language) includes retrieval as well as update operations (UPDATE, INSERT, DELETE), and retrieval may be of a single data item (e.g., data for just USA) or multiple data items (e.g., data for all countries in Europe).
6. Use caution when transferring or viewing/saving the transaction file so that you don't introduce extra <CR><LF>'s (or <LF>'s for Linux) in the middle of a single transaction. Some of the S transactions are quite long –but I only put a single <CR><LF> at the END of the whole statement - though wrap-around makes it appear as if there are internal ones as well. “Save Link As...” from the course website should be fine and not introduce extraneous <CR><LF>'s. But if you email it or ftp it or..., then, depending on the client program you're using to ship the file, it might insert extra ones. If you're having problems, check it using a HexEditor or od.
7. When should there be a **semi-colon** at the end of an sql statement?

The SQL string/command:

- in a C# program does NOT have an ending ; (see my sample program)
- in a script file to be run in mysql's command window DOES have an ending ;
- entered MANUALLY at the mysql> prompt DOES have an ending ;

The LINES in the transaction FILE do NOT have an ending ; since they're just DATA (strings), and not SQL commands, per se.

8. Restore a table's original data by doing the following in MySQL
(for the Country table, for example):

```
DELETE FROM Country;          (removes ALL DATA from table, keeping table itself)
SOURCE c:/../InsertCountryData.sql    (specifying YOUR path instead)
```

- (This is the same SOURCE statement as in the WorldDriver.sql file)

9. Restore the entire DB (e.g., after the programs causes INSERTs/DELETEs/UPDATEs to change the data in the DB) by running the appropriate command files as you used when you initially created the World DB.

OVERVIEW

This project includes creating a database using MySQL and writing a Java or C# program to access the database. Batch processing is used – i.e., the queries (data retrieval and update requests) are read from a Transaction file, and output responses are sent to a Log file.

BASIC STEPS

1. Download & install MySQL.
2. Download & install appropriate connector for MySQL which allows Java or C# to interact with a MySQL database. *[C# people need Visual Studio, not just Microsoft C# Express Edition – see staff in room C-208 for help on this since CS Department no longer has an MSDN license, but CEAS does].*
3. SETUP: Use MySQL and appropriate script files to create WorldDB consisting of Country and CountryLanguage tables (though not City table). *[No Java/C# program is involved here].*
4. USERAPP – a program with additional class(es)
 - Write a C# or Java program which does batch processing (using the “Sequential Stream Processing Algorithm” *[i.e., loop til EOF {read 1 transaction, then deal with it completely}].*
 - Handling of the Trans file and Log file may be done in the main program or a UI class.
 - The 4 transaction handler methods (RetrieveData, InsertData, DeleteData, UpdateData) may be in the main program or a separate class.
 - The actual SQL DB handling methods are in a separate class (in a physically separate file). There could be EITHER:
 - 4 separate DB access methods (one each for handling SELECTs, INSERTs, DELETEs, UPDATEs)

- OR 2 separate DB access methods (one for handling SELECTs and a generic one for handling INSERTs/DELETEs/UPDATEs)
- But there MUST NOT be multiple methods of each type of QUERY (which my example program used for demonstration purposes only)!!!

LOG FILE (.txt)

For each transaction:

1. Echo the original transaction request from the transaction file
2. For INSERTs and DELETEs (but not SELECTs or UPDATEs), display the actual SQL statement which your program built and sent to the DBS for processing
3. Show the results from executing the “query” – i.e.,
 - a. a “table” for a SELECT
 - b. a reassurance message for an INSERTs / DELETEs / UPDATEs:
OK, INSERT WORKED
OK, DELETE WORKED
OK, UPDATE WORKED
4. Write out a blank line.

TRANSACTION FILE (.txt)

Each transaction starts on a new line, where the 1st column is the tranCode:

S, I, D, U (always caps).

S(elect) transactions (RETRIEVE data from DB)

- transaction data is an **actual SQL statement** to be used “as is”.

[This is not a common/proper programming approach, but time is short...].

- Allow for 0 or 1 or Many ROWS to be returned to the program from the DBS.

- **rdr.FieldCount** gives the number of COLUMNS there’ll be for a result set

- “table” printed to Log file does **NOT NEED TO**:

- be in a box (e.g., like a typical interactive result in the command window)

- have column headings (but if it does,

DON’T use what’s in my demo program&

DON’T hardcode the heads (as my demo program does)

- be perfectly aligned since this **SINGLE GENERIC METHOD** doesn’t know what data type **rdr[0]** or **rdr[1]** or the other columns are.

[Given more time for this asgn, then yes, the program could gather the necessary column names & column data types & column data sizes from the DB itself to be able to create a “nice” output report].

U(pdate) transactions (CHANGE data in DB)

- transaction data is an **actual SQL statement** to be used "as is"

D(elete) transactions (CHANGE data in DB)

- transaction data is **NOT an SQL statement**

- your **program has to construct it from the "parameters" supplied**

- Basic format for a simple DELETE SQL statement:

```
DELETE FROM Country WHERE Name = 'Disneyland'
```

Transaction data (i.e., the parameters) would look like:

```
D Country|Name| 'Disneyland'
```

I(nsert) transactions (CHANGE data in DB)

- transaction data is **NOT an SQL statement**

- your **program has to construct it from the "parameters" supplied**
using various string-handling methods (e.g., Split, +, etc.)

[See sample C# code which does part of this: StringHandling.cs].

- 2 basic formats for simple SQL INSERT statements:

1) **all-columns** INSERT (so column names NOT specified in SQL statement):

```
INSERT INTO CountryLanguage VALUES ('USA', 'C#', 'F', 0.01)
```

Transaction data (parameters) looks like:

```
I CountryLanguage| 'USA', 'C#', 'F', 0.01
```

2) **some-columns** INSERT (so column names MUST BE specified in SQL stmt):

```
INSERT INTO Country(Code, Name) VALUES ('HEX', 'Hexland')
```

Transaction data (parameters) looks like:

```
I Country|(Code, Name)| 'HEX', 'Hexland'
```

NOTES

1. The project MUST be modular, using good descriptive naming (as shown in specs), a comment-line-of-*s between modules, etc.
2. My demo programs use hard-coded queries, hard-coded column-headings, multiple retrieval methods, etc. just to demonstrate various C#/Java language concepts regarding connecting and interacting with an existing DB.

DO NOT DO THIS FOR THE ASGN.

3. My examples used MySQL 5.1. Newer versions should work the same.
4. I've used the term "Transaction" (e.g., Tran file, TranCode) thus far in CS3310 (and in this Asgn) in a **data FILE** context. It has a somewhat different meaning

in a **DBS context** - i.e., A "Transaction" is an ordered set of DB operations (a series of SQL SELECT/INSERT/DELETE/UPDATE statements, perhaps embedded in some procedural program code) that are considered as a **SINGLE UNIT**, where **ALL** the DB-changing operations have to work successfully in order for the transaction to succeed. If ANY ONE of the SQL DB-changing operations fails to work, all earlier SQL operations in this current transaction are **ROLLED BACK** to **UNDO** the changes they made to the DB.

5. I've used the term "Query" thus far in CS3310 to mean just RETRIEVAL of a SINGLE data item. In a DBS context, Query (e.g., SQL, Structured Query Language) includes retrieval as well as update operations (UPDATE, INSERT, DELETE), and retrieval may be of a single data item (e.g., data for just USA) or multiple data items (e.g., data for all countries in Europe).
6. Use caution when transferring or viewing/saving the transaction file so that you don't introduce extra <CR><LF>'s (or <LF>'s for Linux) in the middle of a single transaction. Some of the S transactions are quite long –but I only put a single <CR><LF> at the END of the whole statement - though wrap-around makes it appear as if there are internal ones as well. "Save Link As..." from the course website should be fine and not introduce extraneous <CR><LF>'s. But if you email it or ftp it or..., then, depending on the client program you're using to ship the file, it might insert extra ones. If you're having problems, check it using a HexEditor or od.
7. When should there be a **semi-colon** at the end of an sql statement?

The SQL string/command:

- in a C# program does NOT have an ending ; (see my sample program)
- in a script file to be run in mysql's command window DOES have an ending ;
- entered MANUALLY at the mysql> prompt DOES have an ending ;

The LINES in the transaction FILE do NOT have an ending ; since they're just DATA (strings), and not SQL commands, per se.

8. Restore a table's original data by doing the following in MySQL
(for the Country table, for example):

```
DELETE FROM Country;           (removes ALL DATA from table, keeping table itself)
SOURCE c:/../InsertCountryData.sql    (specifying YOUR path instead)
```

- (This is the same SOURCE statement as in the WorldDriver.sql file)

9. Restore the entire DB (e.g., after the programs causes INSERTs/DELETEs/UPDATEs to change the data in the DB) by running the appropriate command files as you used when you initially created the World DB.