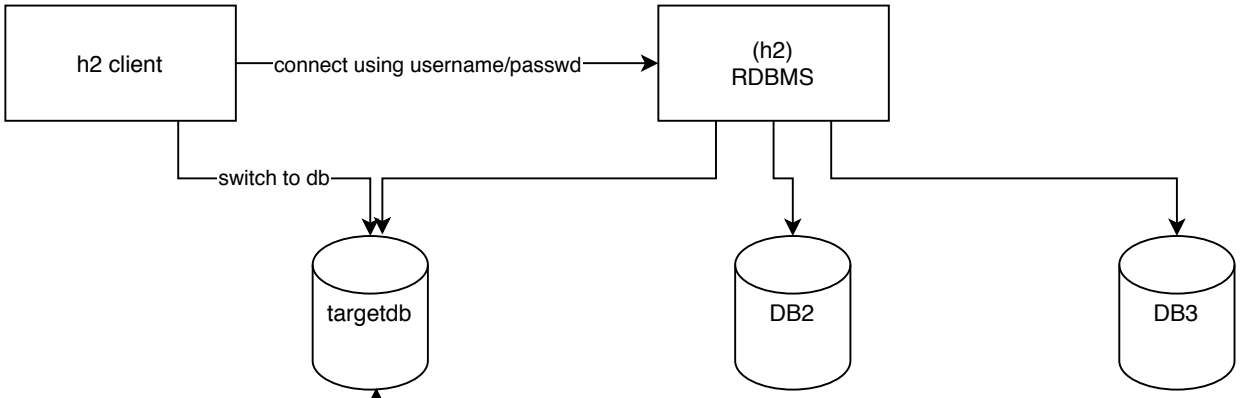
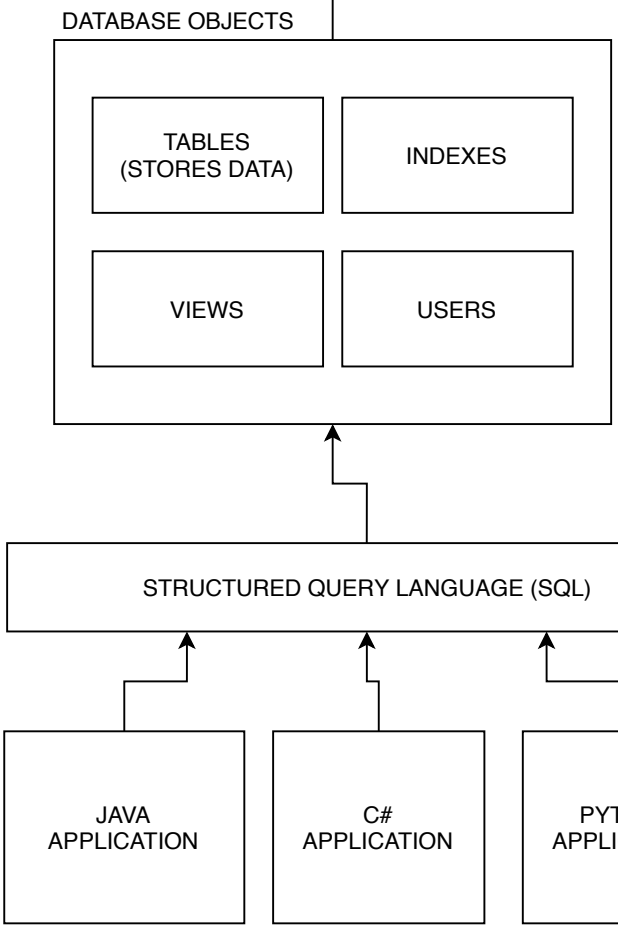


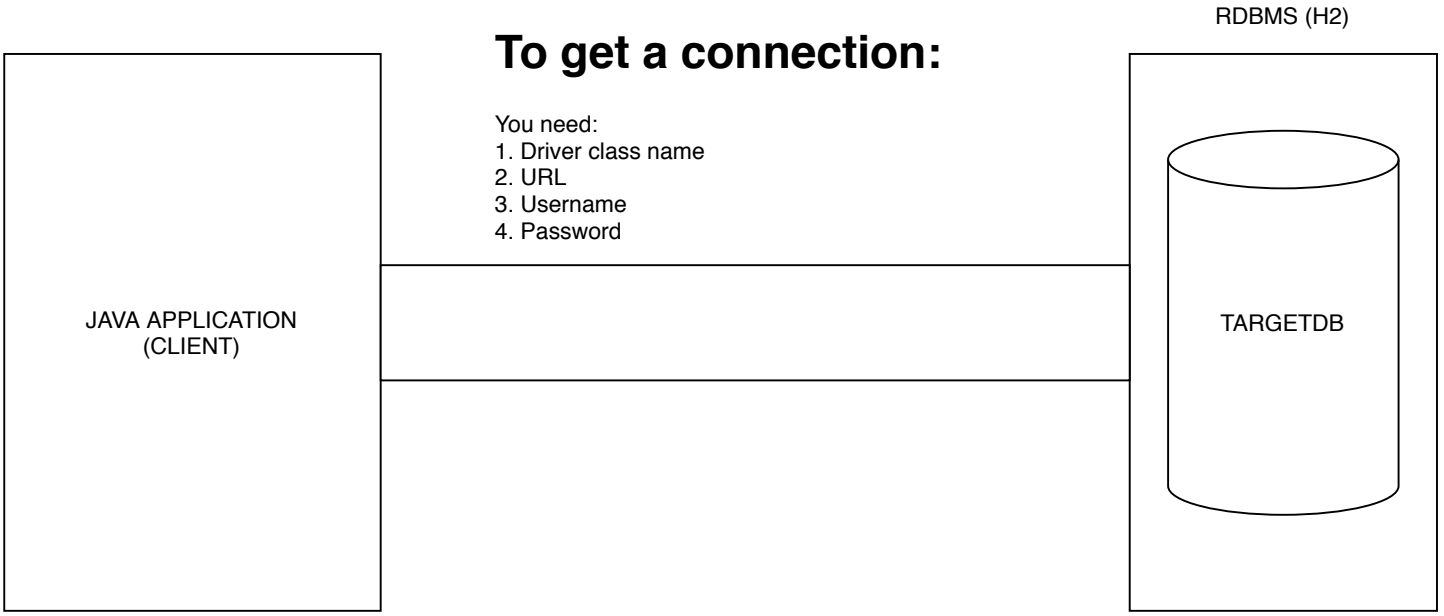


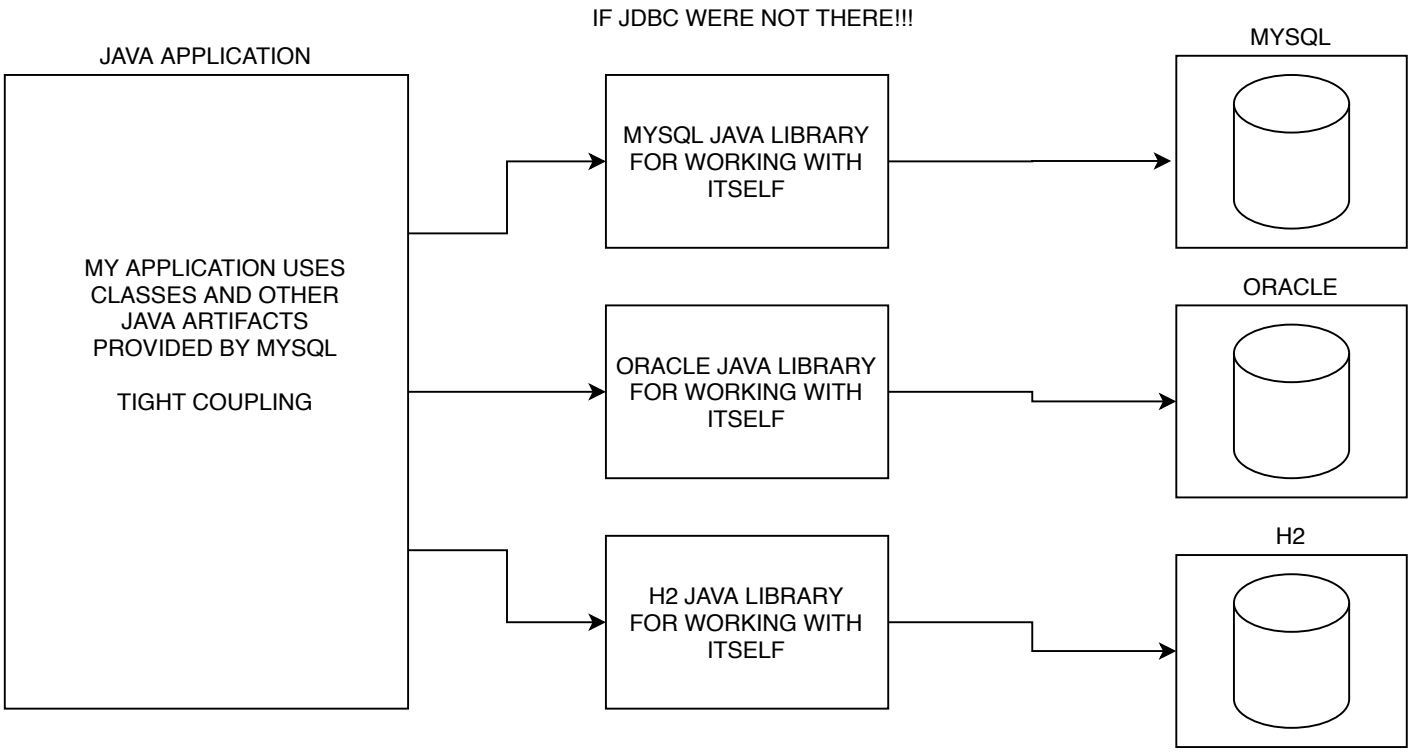
SQL:

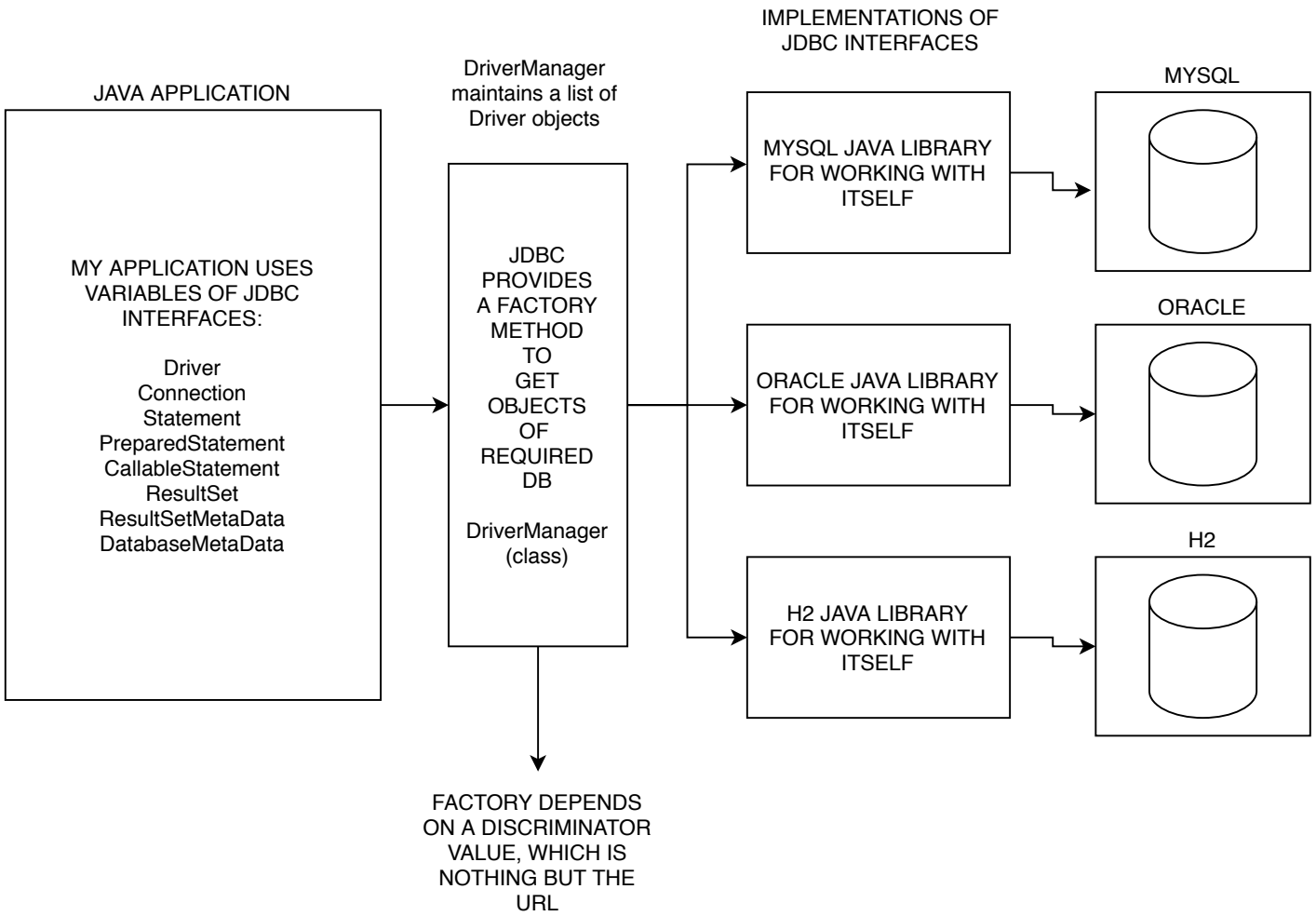
1. DDL -->CREATE / ALTER / DROP
2. DML --> INSERT / UPDATE /DELETE / SELECT
3. TCL --> COMMIT / ROLLBACK

To get a connection:

You need:
1. Driver class name
2. URL
3. Username
4. Password



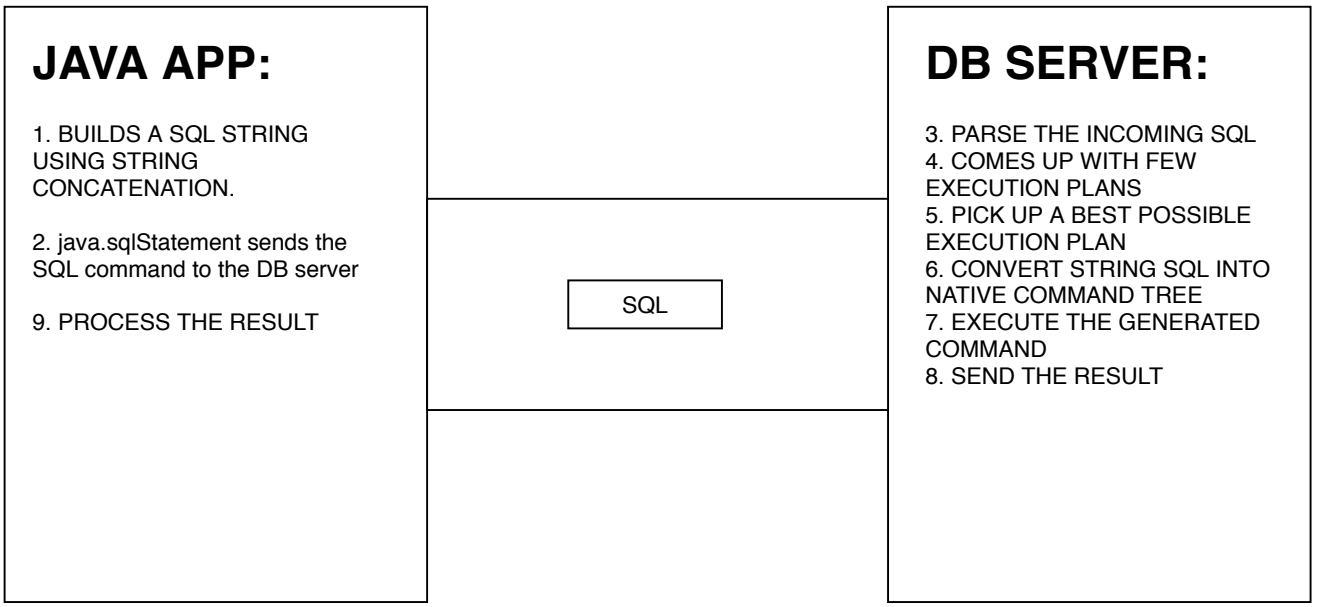


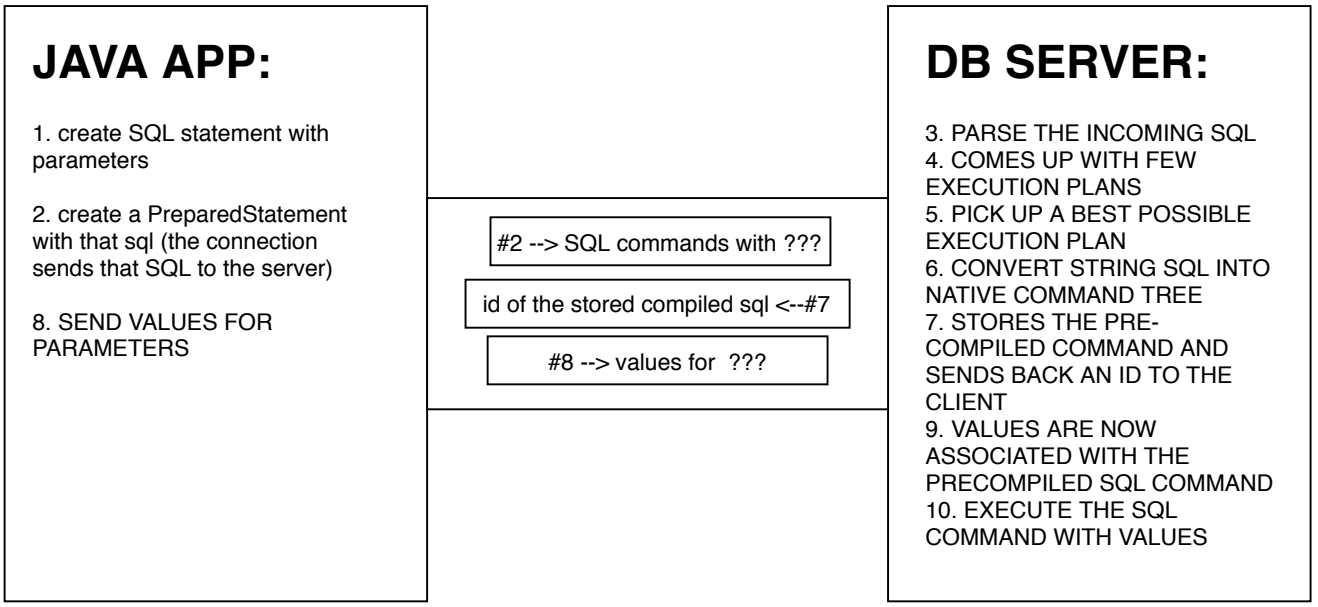


DriverManager

Does not create a connection object by itself, but uses an instanceof a Driver (interface) and asks that driver to create a connection object.

Driver object may be an instanceof one of the following (depending on DB):
1. For H2 db --> org.h2.Driver
2. For MySQL --> com.mysql.cj.jdbc.Driver
3. For Oracle --> oracle.jdbc.driver.JdbcDriver
4. For MS-SQL Server --> com.microsoft.sqlserver.jdbc.SQLServerDriver
5. For Postgres --> `org.postgresql.Driver`


JAVA APP:
1. BUILDS A SQL STRING USING STRING CONCATENATION.
2. java.sqlStatement sends the SQL command to the DB server
9. PROCESS THE RESULT
SQL
DB SERVER:
3. PARSE THE INCOMING SQL
4. COMES UP WITH FEW EXECUTION PLANS
5. PICK UP A BEST POSSIBLE EXECUTION PLAN
6. CONVERT STRING SQL INTO NATIVE COMMAND TREE
7. EXECUTE THE GENERATED COMMAND
8. SEND THE RESULT


JAVA APP:
1. create SQL statement with parameters
2. create a PreparedStatement with that sql (the connection sends that SQL to the server)
8. SEND VALUES FOR PARAMETERS
#2 --> SQL commands with ???
id of the stored compiled sql <--#7
#8 --> values for ???
DB SERVER:
3. PARSE THE INCOMING SQL
4. COMES UP WITH FEW EXECUTION PLANS
5. PICK UP A BEST POSSIBLE EXECUTION PLAN
6. CONVERT STRING SQL INTO NATIVE COMMAND TREE
7. STORES THE PRE-COMPILED COMMAND AND SENDS BACK AN ID TO THE CLIENT
9. VALUES ARE NOW ASSOCIATED WITH THE PRECOMPILED SQL COMMAND
10. EXECUTE THE SQL COMMAND WITH VALUES