**[Different types of JDBC drivers in Java - Quick overview](http://javarevisited.blogspot.sg/2012/05/different-types-of-jdbc-drivers-in-java.html" \o "Different types of JDBC drivers in Java - Quick overview)**

**How many types of JDBC drivers in Java** is a classical **JDBC interview question**, though I have not see this question recently but it was very popular during 2006 - 2008 period and still asked mostly on Junior programmer level interviews. There are mainly 4 types of JDBC drivers in Java, those are referred as type 1 to type 4 jdbc drivers. I agree its easy to remember them by type rather than with there actual name, Which I have yet to get in memory except plain old JDBC-ODBC bridge driver. By the way here are there full names :

Type 1 JDBC Driver is called JDBC-ODBC Bridge driver (bridge driver)

Type 2 JDBC Driver is referred as Native-API/partly Java driver (native driver)

Type 3 JDBC Driver is called AllJava/Net-protocol driver (middleware driver)

Type 4 JDBC Driver is called All Java/Native-protocol driver (Pure java driver)

This JDBC tutorial is in continuation of my earlier tutorials in JDBC like [How to connect to Oracle database using JDBC](http://javarevisited.blogspot.com/2012/04/java-program-to-connect-oracle-database.html)  
and  [4 tips to improve performance of JDBC applications](http://javarevisited.blogspot.com/2012/01/improve-performance-java-database.html). If you are new here and haven't read them already, Its worth looking.  Anyway out of all those 4 types, **JDBC-ODBC Bridge driver** is most common for connecting **SQL Server**, **MS Access** and mostly on training and development. here are quick review of all these four types of JDBC drivers. Also there has been some speculation of type 5 JDBC driver, I have to yet to see it.

JDBC ODBC Bridge Driver or Type 1 JDBC driver

[types of JDBC drivers in Java ](http://javarevisited.blogspot.com/2012/03/difference-between-transient-and.html)In case of JDBC ODBC bridge driver all JDBC calls doesn't directly goes to database instead they go via ODBC driver. JDBC-ODBC driver translates JDBC calls into ODBC callas and send them to ODBC driver for passing to database. Since type 1 driver act as bridge between JDBC and ODBC and that's why its called JDBC-ODBC bridge driver. This driver is not fast and good for production use mainly because of **several layer of translation** on back and fourth database traffic but it has  advantage in terms of of availability and can be your last choice.

Native-API/partly Java driver or Type 2 JDBC driver

This is also called **type 2 driver** and its slightly better than type 1 JDBC driver. type 2 JDBC driver convert JDBC calls into database calls by using native API provided by database. This driver is database specific so once you switch from one database to another you need to change type 2 JDBC driver. performance is better than *JDBC-ODBC bridge driver* since communication layer is reduced. type 2 JDBC driver requires database native library to be available on client but it poses several version and compatibility issue. This was liked by Database vendors though because they can reuse there existing native libraries.

AllJava/Net-protocol driver or Type 3 JDBC driver

both type 1 and type 2 JDBC drivers were not written in Java so there was need for pure Java JDBC driver to resolve portability issue. type 3 JDBC driver comes with pure java implementation (that's why All Java word ) but it uses 3 tier architecture where you have a Java client and Java Server which talk with Net protocol and Server speaking to database. type 3 JDBC driver never get popular among database vendors as it was costly for them to rewrite there existing native database library which was mainly on C and C++.

All Java/Native-protocol driver or Type 4 JDBC driver

type 4 JDBC driver is most popular among all *four types of JDBC driver*. it has not only implemented in Java but also incorporates all database call in single driver. It was pretty easy to use and deploy as well just include driver's [jar](http://javarevisited.blogspot.com/2012/03/how-to-create-and-execute-jar-file-in.html) in [classpath](http://javarevisited.blogspot.com/2011/01/how-classpath-work-in-java.html) and you are ready. It also removes 3 tier architecture of type 3 JDBC driver which makes it faster than type 3.  Major development happens on type 4 JDBC driver when database upgrade themselves, though some of them still upgrade native database library or type 2 driver.

That's all on **quick overview of different types of JDBC drivers in Java**. JDBC drivers has evolved from JDBC ODBC bridge driver to type 4 JDBC driver, which is clean and portable. There has been some buzz around **JDBC driver 5 on Java** community which may include some advanced functionality. let us know if you come across some news on JDBC 5 driver.

Read more: <http://javarevisited.blogspot.com/2012/05/different-types-of-jdbc-drivers-in-java.html#ixzz3KkWUTm9i>

[**Top 10 JDBC Best Practices for Java Programmer**](http://javarevisited.blogspot.sg/2012/08/top-10-jdbc-best-practices-for-java.html)

**Java JDBC Best practices**

JDBC Best Practices are some coding practices which Java programmer should follow while writing JDBC code. As discussed in [how to connect to Oracle database from Java](http://javarevisited.blogspot.sg/2012/04/java-program-to-connect-oracle-database.html), JDBC API is used to connect and interact with a Database management System.  We have touched some of the JDBC best practices in our last article [4 JDBC Performance tips](http://javarevisited.blogspot.sg/2012/01/improve-performance-java-database.html), On which we have discussed simple tips to improve performance of Java application with database. By using JDBC you can execute DDL, DML and Stored Procedures. JDBC Best practices is probably most significant set of coding practices in Java because it significantly affect performance of Java application. I have seen substantial performance gain by simply following common *JDBC best practices* like running queries with auto commit mode disable. One of the query which we used in our example of JDBC Batch update was taking almost 30 second to finish with auto commit mode enabled but it just took under one second with auto commit mode disable and using explicit commit. This JDBC tutorial is collection of such practices which help you to write better JDBC code and in most cases result in improved performance.

 

10 JDBC Best pratices in Java

[JDBC Best practices Java coding database](http://3.bp.blogspot.com/-K6q0DQ1v-tw/TWu8owBtc2I/AAAAAAAAADA/oBoHDBiJ8ag/s1600/17.jpg)Here is my list of top 10 JDBC best practices in Java which helps to avoid potential error, to get better performance and helps to write robust Java database connection code.

**JDBC Best Practice #1: Use PreparedStatement**

This is by far most popular JDBC practices suggested by everyone who has worked in JDBC API in Java. Indeed PreparedStatement deserve that admiration because of useful services it provides like *prevention from SQL injection*, Precompiled SQL queries and use of bind variables as discussed in  [why Use PreparedStatement in Java](http://javarevisited.blogspot.sg/2012/03/why-use-preparedstatement-in-java-jdbc.html)

**JDBC Best Practice #2: Use ConnectionPool**

ConnectionPool as JDBC best practice has already gained recognition and it even become standard now days. Several framework provides in built connection Pool facility like [Database Connection Pool in Spring](http://javarevisited.blogspot.sg/2012/06/jdbc-database-connection-pool-in-spring.html), DBCP and if you are running in managed environment like [J2EE Application Server](http://javarevisited.blogspot.sg/2012/05/5-difference-between-application-server.html) e.g. WAS or JBOSS, Server will provide Connection Pool facility. rational behind this JDBC best practices is that Creating JDBC connection take relatively longer time which can increase overall response time, by caching JDBC connection in pool application can immediately access database.

**JDBC Best Practice #3: Disable auto commit mode**

This is one of those JDBC best practices which provided substantial performance gain in our JDBC batch update example. Its recommended to run SQL query with auto commit mode disable. Rational behind this JDBC best practice is that with auto commit mode disabled you can [group SQL Statement in one transaction](http://javarevisited.blogspot.sg/2011/11/database-transaction-tutorial-example.html) while in case of auto commit mode every SQL statement runs in its own transaction and committed as soon as it finishes. So always run queries with auto commit mode disabled

**JDBC Best Practice #4: Use JDBC Batch Update**

This is another JDBC best practice which is very popular. JDBC API provides addBatch() method to add [SQL queries](http://javarevisited.blogspot.sg/2012/07/subquery-example-in-sql-correlated-vs.html) into batch and executeBatch() to send batch queries for execution. Rational behind this JDBC best practices is that, JDBC batch update potentially reduce number of database roundtrip which result in significant performance gain. So always Use JDBC batch update for insertion and update queries.

**JDBC Best Practice #5: Access ResultSet using column name to avoid invalidColumIndexError**

JDBC API allows to access data returned by [SELECT query](http://javarevisited.blogspot.sg/2011/10/selct-command-sql-query-example.html) using ResultSet, which can further be accessed using either column name or column index. This JDBC best practice suggest using column name over column index in order to [avoid InvalidColumnIndexException](http://javarevisited.blogspot.sg/2012/01/javasqlsqlexception-invalid-column.html) which comes if index of column is incorrect, most common of them is 0, since ResultSet column Index starts from 1, zero is invalid. Also you don't need to change your JDBC access code if order of column changed in SELECT SQL query, which is a major maintenance gain and a robust way to write JDBC code. Some Java programmer may argue that accessing column using index is faster than name, which is true but if you look in terms of maintenance, robustness and readability, I prefer accessing column using name in ResultSet [Iterator](http://javarevisited.blogspot.sg/2011/10/java-iterator-tutorial-example-list.html).

**JDBC Best Practice #6: Use Bind variables instead of String concatenation**

In JDBC Best Practice #1 we have suggest to [use PreparedStatement in Java](http://javarevisited.blogspot.sg/2012/03/why-use-preparedstatement-in-java-jdbc.html) because of better performance. But performance can only be improved if you use *bind variables* denoted by *? or place holders*. which allows database to run same query with different parameter. This JDBC best practices also result in better performance and also provide protection against SQL injection.

**JDBC Best Practice #7: Always close Statement, PreparedStatement and Connection.**

Nothing new on this JDBC Best practice. Its common Java coding practice to close any resource in finally block as soon as you are done with that. JDBC Connection and other JDBC classes are costly resource and should be closed in finally block to ensure release of connection even in case of any SQLException. From Java 7 onwards you can use [Automatic Resource Management (ARM) Block](http://javarevisited.blogspot.sg/2011/09/arm-automatic-resource-management-in.html) to close resources automatically.

**JDBC Best Practice #8: Choose suitable JDBC driver for your application**

There are 4 typs of JDBC driver in Java and it can directly affect the performance of DAO layer. always use latest JDBC Driver if available and prefer [type 4 native JDBC Drivers](http://javarevisited.blogspot.sg/2012/05/different-types-of-jdbc-drivers-in-java.html).

**JDBC Best Practice #9: Use standard SQL statement and avoid using db specific query until necessary**

This is another JDBC best practice in Java which ensures writing portable code. Since most of JDBC code is filled up with [SQL query](http://javarevisited.blogspot.sg/2012/07/subquery-example-in-sql-correlated-vs.html) its easy to start using Database specific feature which may present in [MySQL](http://javarevisited.blogspot.sg/2010/10/frequently-used-mysql-commands-part-3.html) but not in Oracle etc. By using ANSI SQL or by not using DB specific SQL you ensure minimal change in your DAO layer in case you switch to another database.

**JDBC Best Practice #10: Use correct getXXX() method**

This is the last JDBC best practice in this article which suggest using correct getter while getting data from ResultSet to avoid data conversion even though JDBC allows to get any data type using getString()or getObject().

That's all on *JDBC best practices for Java Programmer*, I am sure there are many more JDBC best practices around but these are most common practices which I can think of. let us know if you are familiar with any other JDBC best practice.

Read more: <http://javarevisited.blogspot.com/2012/08/top-10-jdbc-best-practices-for-java.html#ixzz3KkH52LA0>

# [Java Program to connect Oracle Database with Example - JDBC Tutorial Sample Code](http://javarevisited.blogspot.sg/2012/04/java-program-to-connect-oracle-database.html)

**How to connect to Oracle database from Java Program using JDBC API** is common need for many Java programmer, though there are lot of framework available which has simplified JDBC development e.g hibernate, Spring JdbcTempate and many more, but creating **Java program to connect to oracle database** from plain old Java is still most easy and quickest method for testing and debugging database connectivity. Database connection program is also a common Java programming exercise in many Java programming courses on school, colleges and various training institutes. We have been exploring  some advanced concepts and best practices on JDBC in my previous articles like  [Why should you use PreparedStatement in Java](http://javarevisited.blogspot.com/2012/03/why-use-preparedstatement-in-java-jdbc.html) and [4 JDBC performance tips for Java](http://javarevisited.blogspot.com/2012/01/improve-performance-java-database.html) application , which you may like if you are on more advanced level. This simple java program is intended for beginners in Java which have just started learning JDBC API.

If you like to read tutorials on database then you may find [difference between truncate and delete](http://javarevisited.blogspot.com/2011/10/how-to-use-truncate-and-delete-command.html) ,  [10 example of SQL SELECT queries](http://javarevisited.blogspot.com/2011/10/selct-command-sql-query-example.html) and [how to manage database transaction](http://javarevisited.blogspot.com/2011/11/database-transaction-tutorial-example.html) useful and interesting.

## How to connect Oracle Database from Java Program using JDBC - code example

[How to connect Oracle database form Java program with example](http://javarevisited.blogspot.com/2012/03/simpledateformat-in-java-is-not-thread.html)Here is a complete **code example of Java program to connect Oracle database using JDBC**. This Java program will connect  to Oracle database and print current date from "**dual**" system table. By the way before [running this Java program](http://javarevisited.blogspot.com/2011/11/run-java-program-from-command-prompt.html) to connect Oracle database make sure your Oracle database server is running and you have JDBC thin driver in your [classpath](http://javarevisited.blogspot.com/2011/01/how-classpath-work-in-java.html) like **ojdbc6.jar or ojdbc6\_g.jar**.

**import** java.sql.Connection;  
**import** java.sql.DriverManager;  
**import** java.sql.PreparedStatement;  
**import** java.sql.ResultSet;  
**import** java.sql.SQLException;  
**import** java.util.Properties;  
  
***/\*\*  
 \* Simple Java Program to connect Oracle database by using Oracle JDBC thin driver  
 \* Make sure you have Oracle JDBC thin driver in your classpath before running this program  
 \* @author  
 \*/***  
**public** **class** OracleJdbcExample {  
  
    **public** **static** **void** main(**String** args[]) **throws** **SQLException** {  
        *//URL of Oracle database server*  
        **String** url = "jdbc:oracle:thin:@localhost:1632:DEVROOT32";   
        
        *//properties for creating connection to Oracle database*  
        **Properties** props = **new** **Properties**();  
        props.setProperty("user", "scott");  
        props.setProperty("password", "tiger");  
        
        *//creating connection to Oracle database using JDBC*  
        **Connection** conn = **DriverManager**.getConnection(url,props);  
  
        **String** sql ="select sysdate as current\_day from dual";  
  
        *//creating PreparedStatement object to execute query*  
        **PreparedStatement** preStatement = conn.prepareStatement(sql);  
      
        **ResultSet** result = preStatement.executeQuery();  
        
        **while**(result.next()){  
            **System**.out.println("Current Date from Oracle : " +         result.getString("current\_day"));  
        }  
        **System**.out.println("done");  
        
    }  
}  
  
Output:  
**Current** **Date** from Oracle : 2012-04-12 17:13:49  
done

## Error and Exception while connecting Oracle Database from Java Program:

1) **Invalid Username and Password**

Exception in thread "main" java.sql.SQLException: ORA-01017: invalid username/password; logon denied

        at oracle.jdbc.driver.T4CTTIoer.processError(T4CTTIoer.java:452)

This Error comes when username and password provided to Java program connecting to Oracle database is not correct.

**2)No suitable driver found**

**Exception in thread "main" [java.sql.SQLException: No suitable driver found for](http://javarevisited.blogspot.com/2012/03/jdbc-javalangclassnotfoundexception.html)**

jdbc:oracle:thin:@localhost:1632:DEVROOT32

        at java.sql.DriverManager.getConnection(DriverManager.java:602)

        at java.sql.DriverManager.getConnection(DriverManager.java:154)

This Error comes **when JDBC thin driver for relevant Oracle version is not in Classpath**. e.g. **ojdbc6.jar or ojdbc6\_g.jar** (compiled with javac -g with debug information) for Oracle 11g.

Also while reading data from ResultSet ensure that you are using proper column index to avoid [**Exception in thread "main" java.sql.SQLException: Invalid column index**](http://javarevisited.blogspot.com/2012/01/javasqlsqlexception-invalid-column.html)which comes when you use invalid index like zero to access ResultSet on various get and set method.

In this Java program Example we have seen **how to connect to Oracle database using JDBC thin driver**, with thin driver its much easier to *connect to oracle database* as you don’t need to create data-sources like you do if you use JDBC ODBC Driver. Let me know if you face any issue while *connecting to Oracle database form Java Program*. Another worth noting point is connecting Oracle database using SSL from Java Program, which may see in another java tutorial.

Read more: <http://javarevisited.blogspot.com/2012/04/java-program-to-connect-oracle-database.html#ixzz3KkL9i4pj>

# [JDBC Performance Tips - 4 Tips to improve performance of Java application with database](http://javarevisited.blogspot.sg/2012/01/improve-performance-java-database.html)

**JDBC Performance tips**are collection of some tried and tested way of coding and applying process which improves performance of JDBC code. **Performance of core java application** or **J2EE web application** is very important, especially if its using database in back end which tend to slow down performance drastically. do you experience your java j2ee web application to be very slow (taking few seconds to process simple requests which involves database access, paging, sorting etc) than below tips may *improve performance of your Java application*. these tips are simple in nature and can be applied to other programming language application which uses [database](http://javarevisited.blogspot.com/2011/11/database-transaction-tutorial-example.html) as back-end.

## Improve performance Java application with database

### 4 JDBC Performance Tips

[how to improve performance Java database application ](http://javarevisited.blogspot.com/2011/09/find-hostname-from-ip-address-to.html)Here are four JDBC performance tips, not really super cool or something you never heard and I rather say fundamentals but in practice many programmers  just missed these, you may also called this **database performance tips** but I prefer to keep them as Java because I mostly used this when I access database from Java application.

**JDBC Performance Tips 1: Use Cache**

Find out how many database calls you are making and minimize those believe it or not if you see performance in seconds than in most cases culprit is database access code. since connecting to database requires connections to be prepared, network round trip and processing on database side, its best to avoid database call if you can work with cached value. even if your application has quite dynamic data having a **short time cache** can save many database round trip which can boost your java application performance by almost 20-50% based on how many calls got reduced. In order to find out database calls just put [logging](http://javarevisited.blogspot.com/2011/05/top-10-tips-on-logging-in-java.html) for each db call in DAO layer, even better if you log in and out time which gives you idea which call is taking how much time.

**Java database performance tips 2: Use Database Index**

Check whether your database has indexed on columns if you are reading from database and your query is taking longer than expected than first thing you should check is whether you have index on columns which you are using for search (in where clause of query). this is most common error programmers make and believe me there is huge difference than [querying a database](http://javarevisited.blogspot.com/2011/10/selct-command-sql-query-example.html) which is indexed and the one which is not. This tip can boost your performance by more than 100% but as I said its mistake now having proper indexes in your tables so don't do that in first place. Another point which is worth noting is that **too many indexes slows insert and update operation**so be careful with indexes and always go on suitable and practical numbers like having indexes on fields which most often used for searching like id, category, class etc.

**JDBC performance tips 3: Use PreparedStatement**

Use **PreparedStatement** or Stored Procedure for executing query Prepared Statements are much faster than normal Statement object as database can pre-compile them and also cache there query plan. so always use **parametric form of Prepared Statement** like "select \* from table where id=?" , don't use "select \* from table where id='" + id "'" which is still a prepared Statement but not parametrized. you won't get performance benefit of preparestatment by using second form. see here for more [advantages of PreparedStatement in Java](http://javarevisited.blogspot.com/2012/03/why-use-preparedstatement-in-java-jdbc.html) like prevention from SQL Injection.

**Java database performance tips 4:Use Database Connection Pool**

Use **Connection Pool** for holding Database Connections. Creating Database connections are slow process in Java and take long time. So if you are creating connection on each request than obviously your response time will be lot more. Instead if you use Connection pools with adequate number of connections based upon your traffic or number of concurrent request to make to database you can minimize this time. Even with Connection pooling few of first request may take little longer to execute till your connection gets created and cached in pool.

**JDBC performance tips 5:Use JDBC Batch Update**

using JDBC batch update can improve performance of Java database application significantly. you should always execute your insert and update queries on Java using batch. You can execute batch queries in Java by using either Statement or PreparedStatement. Prepared Statement is preferred because of other advantages. Use executeBatch() method to execute batch queries

**JDBC performance tips 6:Disable auto commit**

This is one of those JDBC performance tips which provides substantial benefit by small change. One of the better ways to improve performance of Java database application is running queries with setAutoCommit(false). By default new JDBC connection has there auto commit mode ON, which means every individual SQL Statement will be executed in its own transaction. while without auto commit you can group SQL statement into logical transaction, which can either be committed or rolled back by calling commit() or rollback(). Also its significant performance gain when you commit() explicitly. try running same query number of times with and without auto-commit and you can see how much different it make  
  
  
These **Java database application performance tips** are very simple in nature and most of advanced Java developers already employ these while [writing production code](http://javarevisited.blogspot.com/2011/09/how-to-write-production-quality-code.html), but same time I have seen many java programmers which doesn't put so much attention until they found there java application is very slow. So geeks and expert may not get anything new but for beginners this is something worth remembering and applying. You can also use this Java performance tips as [code review checklist](http://javarevisited.blogspot.com/2011/09/code-review-checklist-best-practice.html) of what not to do while writing Java applications which uses database in back-end.

That's all on **how to improve performance of java programs with database**. let me know if you have some other java or database tips which is helpful to boost performance of java database applications.

Read more: <http://javarevisited.blogspot.com/2012/01/improve-performance-java-database.html#ixzz3KkOwJKSO>

## Improve performance Java application with database

### 4 JDBC Performance Tips

[how to improve performance Java database application ](http://javarevisited.blogspot.com/2011/09/find-hostname-from-ip-address-to.html)Here are four JDBC performance tips, not really super cool or something you never heard and I rather say fundamentals but in practice many programmers  just missed these, you may also called this **database performance tips** but I prefer to keep them as Java because I mostly used this when I access database from Java application.

**JDBC Performance Tips 1: Use Cache**

Find out how many database calls you are making and minimize those believe it or not if you see performance in seconds than in most cases culprit is database access code. since connecting to database requires connections to be prepared, network round trip and processing on database side, its best to avoid database call if you can work with cached value. even if your application has quite dynamic data having a **short time cache** can save many database round trip which can boost your java application performance by almost 20-50% based on how many calls got reduced. In order to find out database calls just put [logging](http://javarevisited.blogspot.com/2011/05/top-10-tips-on-logging-in-java.html) for each db call in DAO layer, even better if you log in and out time which gives you idea which call is taking how much time.

**Java database performance tips 2: Use Database Index**

Check whether your database has indexed on columns if you are reading from database and your query is taking longer than expected than first thing you should check is whether you have index on columns which you are using for search (in where clause of query). this is most common error programmers make and believe me there is huge difference than [querying a database](http://javarevisited.blogspot.com/2011/10/selct-command-sql-query-example.html) which is indexed and the one which is not. This tip can boost your performance by more than 100% but as I said its mistake now having proper indexes in your tables so don't do that in first place. Another point which is worth noting is that **too many indexes slows insert and update operation**so be careful with indexes and always go on suitable and practical numbers like having indexes on fields which most often used for searching like id, category, class etc.

**JDBC performance tips 3: Use PreparedStatement**

Use **PreparedStatement** or Stored Procedure for executing query Prepared Statements are much faster than normal Statement object as database can pre-compile them and also cache there query plan. so always use **parametric form of Prepared Statement** like "select \* from table where id=?" , don't use "select \* from table where id='" + id "'" which is still a prepared Statement but not parametrized. you won't get performance benefit of preparestatment by using second form. see here for more [advantages of PreparedStatement in Java](http://javarevisited.blogspot.com/2012/03/why-use-preparedstatement-in-java-jdbc.html) like prevention from SQL Injection.

**Java database performance tips 4:Use Database Connection Pool**

Use **Connection Pool** for holding Database Connections. Creating Database connections are slow process in Java and take long time. So if you are creating connection on each request than obviously your response time will be lot more. Instead if you use Connection pools with adequate number of connections based upon your traffic or number of concurrent request to make to database you can minimize this time. Even with Connection pooling few of first request may take little longer to execute till your connection gets created and cached in pool.

**JDBC performance tips 5:Use JDBC Batch Update**

using JDBC batch update can improve performance of Java database application significantly. you should always execute your insert and update queries on Java using batch. You can execute batch queries in Java by using either Statement or PreparedStatement. Prepared Statement is preferred because of other advantages. Use executeBatch() method to execute batch queries

**JDBC performance tips 6:Disable auto commit**

This is one of those JDBC performance tips which provides substantial benefit by small change. One of the better ways to improve performance of Java database application is running queries with setAutoCommit(false). By default new JDBC connection has there auto commit mode ON, which means every individual SQL Statement will be executed in its own transaction. while without auto commit you can group SQL statement into logical transaction, which can either be committed or rolled back by callingcommit() or rollback(). Also its significant performance gain when you commit() explicitly. try running same query number of times with and without auto-commit and you can see how much different it make  
  
  
These **Java database application performance tips** are very simple in nature and most of advanced Java developers already employ these while [writing production code](http://javarevisited.blogspot.com/2011/09/how-to-write-production-quality-code.html), but same time I have seen many java programmers which doesn't put so much attention until they found there java application is very slow. So geeks and expert may not get anything new but for beginners this is something worth remembering and applying. You can also use this Java performance tips as [code review checklist](http://javarevisited.blogspot.com/2011/09/code-review-checklist-best-practice.html) of what not to do while writing Java applications which uses database in back-end.

That's all on **how to improve performance of java programs with database**. let me know if you have some other java or database tips which is helpful to boost performance of java database applications.

Read more: <http://javarevisited.blogspot.com/2012/01/improve-performance-java-database.html#ixzz3KkQzd9Og>

# [Why use PreparedStatement in Java JDBC – Example Tutorial](http://javarevisited.blogspot.com/2012/03/why-use-preparedstatement-in-java-jdbc.html)

**PreparedStatement in Java** is one of several ways to execute SQL queries using JDBC API. Java provides Statement,

PreparedStatement and CallableStatement for executing queries. Out of these three, Statement is used for general purpose queries, PreparedStatement is used for executing parametric query and CallableStatement is used for executing Stored Procedures. PreparedStatement is also a popular topic in java interviews. Questions like **Difference between Statement and PreparedStatement in Java** and **How to prevent SQL Injection attacks in Java** are [popular java interview questions](http://javarevisited.blogspot.com/2011/04/top-20-core-java-interview-questions.html). In this Java JDBC tutorial we will see why should you use usePreparedStatement in Java, What are the major advantages of using PreparedStatement in Java and how PreparedStatement prevents SQL Injection attacks in Java.

This article is in continuation of my earlier post on database and java like [4 tips to improve performance of Java application with database](http://javarevisited.blogspot.com/2012/01/improve-performance-java-database.html) and [Difference between truncate and delete in SQL](http://javarevisited.blogspot.com/2011/10/how-to-use-truncate-and-delete-command.html).If you haven’t read  them already you may found  those tutorial useful and interesting.

## What is PreparedStatement in Java

[How to use PreparedStatement in Java JDBC – Example Tutorial](http://javarevisited.blogspot.com/2012/03/why-character-array-is-better-than.html)

PreparedStatement is a class in java.sql package and allows Java programmer to execute SQL queries by using JDBC package. You can get PreparedStatement object by calling connection.prepareStatement() method.SQL queries passed to this method goes to [Database](http://javarevisited.blogspot.com/2011/11/database-transaction-tutorial-example.html) for pre-compilation if JDBC driver supports it. If it doesn't than pre-compilation occurs when you execute prepared queries. Prepared Statement queries are **pre-compiled on database** and there access plan will be reused to execute further queries which allows them to execute much quicker than normal queries generated by Statement object. Here is an **example of how to use PreparedStatement in Java**:

**public** **class** PreparedStmtExample {  
  
    **public** **static** **void** main(**String** args[]) **throws** **SQLException** {  
        **Connection** conn = **DriverManager**.getConnection("mysql:**\\**localhost:1520", "root", "root");  
        **[PreparedStatement](http://java.sun.com/j2se/1.5.0/docs/api/java/sql/PreparedStatement.html)** preStatement = conn.prepareStatement("select distinct loan\_type from loan where bank=?");  
        preStatement.setString(1, "Citibank");  
      
        **ResultSet** result = preStatement.executeQuery();  
        
        while(result.next()){  
            **System**.out.println("Loan Type: " + result.getString("loan\_type"));  
        }         
    }  
}   
**Output:**  
Loan **Type**: Personal Loan  
Loan **Type**: Auto Loan  
Loan **Type**: Home Loan  
Loan **Type**: Gold Loan

In this example of PreparedStatement same query and access path will be used if you pass a different parameter  e.g.

"Standard Charted" or "HSBC". ResultSet returned by prepared statement execution is of "TYPE\_FORWARD\_ONLY" but can be customized by using [overloaded method](http://javarevisited.blogspot.com/2011/12/method-overloading-vs-method-overriding.html) of prepareStatement().

## Benefits of Java Prepared Statement

PreparedStatement in Java JDBC offers several benefits and it’s a recommended way to execute SQL queries in any enterprise Java application or in production code. Here are few ***advantages of using PreparedStatement in Java***:

1. **PreparedStatement allows you to write dynamic and parametric query**.

By using PreparedStatement in Java you can write parametrized sql queries and send different parameters by using same sql queries which is lot better than creating different queries. Here is an example of parametric query written using PreparedStatement in java:

**select** interest\_rate **from** loan where loan\_type=?

Now you can run this query for any loan type e.g. "personal loan”, "home loan" or "gold loan". This [example of SELECT query](http://javarevisited.blogspot.com/2011/10/selct-command-sql-query-example.html) is called parametric or parametrized query because it can be invoked with different parameter. Here “?” is used as place holder for parameter.

2. **PreparedStatement is faster than Statement in Java**

One of the major benefits of using PreparedStatement is better performance. PreparedStatement gets pre compiled

In database and there access plan is also cached in database, which allows database to execute parametric query written using prepared statement much faster than normal query because it has less work to do. You should always try to use PreparedStatement in production JDBC code to reduce load on database. In order to get performance benefit its worth noting to **use only parametrized version of sql query** and not with [string concatenation](http://javarevisited.blogspot.com/2011/07/string-vs-stringbuffer-vs-stringbuilder.html). Out of following two examples of SELECT queries, first example of SELECT query  will not offer any performance benefit:

SQL Query 1: PreparedStatement with String concatenation

**String** loanType = getLoanType();  
**PreparedStatement** prestmt = conn.prepareStatement("select banks from loan where loan\_type=" + loanType);

SQL Query 2: Parameterized query using PreparedStatement

**PreparedStatement** prestmt = conn.prepareStatement("select banks from loan where loan\_type=?");  
prestmt.setString(1,loanType);

Second SQL query is correct use of PreparedStatement in Java and give better performance than SQL query1.

3. **PreparedStatement prevents SQL Injection attacks in Java**

If you have been working in Java web application you must be familiar with infamous SQL Injection attacks, last year Sony got victim of SQL injection and compromised several Sony play station user data. In SQL Injection attack, malicious user pass **SQL meta-data** combined with input which allowed them to *execute sql query of there choice,* If not validated or prevented before sending query to database. By using parametric queries and **PreparedStatement you prevent many forms of SQL injection** because all the parameters passed as part of place-holder will be escaped automatically by JDBC Driver. Though It’s worth remembering that in above example of two PreparedStatement only second example will prevent SQL injection attacks and first example is not secure with SQL injection.

4. At last PreparedStatement queries are more readable and secure than cluttered string concatenated queries.

### Limitation of Java PreparedStatement

Despite of being very useful PreparedStatement also has few limitations:

1. In order to prevent SQL Injection attacks in Java, PreparedStatement doesn't allow multiple values for one placeholder (?) who makes it tricky to execute SQL query with IN clause. Following example of SQL query with IN clause using prepared Statement will not work in Java:

select \* from loan where loan\_type IN ( ?)  
preparedSatement.setString(1, "'personal loan', 'home loan', 'gold loan'");

Though there are some workarounds and ways to **execute IN queries using PreparedStatement** but those are

rather tricky or have performance impact.

## Important points on PreparedStatement in Java

Here are few important points about PreparedStatement Class in Java, worth remembering:

1. PreparedStatement in Java allows you to write parametrized query which gives better performance than Statement [class in Java](http://javarevisited.blogspot.com/2011/10/class-in-java-programming-general.html).

2. In case of PreparedStatement, Database use an already compiled and defined access plan, this allows prepared statement query to run faster than normal query.

3. Parametrized query written using PreparedStatement in Java prevents many common SQL Injection attacks.

4. PreparedStatement allows you to write dynamic query in Java.

5. PreparedStatement are associated with java.sql.Connection object, once you drop a connection all PreparedStatement  associated with that connection will be dropped by Database.

6. "?" is also called placeholder or IN parameter in Java.

7. PreparedStatement query return FORWARD\_ONLY ResultSet, so you can only move in one direction Also concurrency level of ResultSet would be "CONCUR\_READ\_ONLY".

8. All JDBC Driver doesn't support pre compilation of SQL query in that case query is not sent to database when you call prepareStatement(..) method instead they would be sent to database when you execute PreparedStatement query.

9. Index of placeholder or parameter starts with "1" and not with "0", which is common cause of [java.sql.SQLException: Invalid column index](http://javarevisited.blogspot.com/2012/01/javasqlsqlexception-invalid-column.html)**.** So in a PreparedStatement t of two placeholder, first will be referred by index 1 and second will be reference by index 2.

These were the reasons Why PreparedStatement in java is very popular and useful. You can still use Statement object for test programmers but consider PreparedStatement before moving to production.

Read more: <http://javarevisited.blogspot.com/2012/03/why-use-preparedstatement-in-java-jdbc.html#ixzz3KkR7MPpG>

# [Difference between Truncate and Delete command in SQL - Interview Questions with Example](http://javarevisited.blogspot.com/2011/10/how-to-use-truncate-and-delete-command.html)

## Delete and truncate command in SQL Truncate and delete in SQL are two commands which is used to remove or delete data from table. Though quite basic in nature both sql commands can create lot of trouble until you are familiar with details before using it. Difference between Truncate and delete are not just important to understand perspective but also a very popular [SQL interview topic](http://javarevisited.blogspot.com/2012/12/how-to-find-second-highest-or-maximum-salary-sql.html) which in my opinion a definite worthy topic. What makes them tricky is amount of data. Since most of Electronic trading system stores large amount of transactional data and some even maintain historical data, good understanding of *delete and truncate command* is required to effectively work on those environment.I have still seen people firing delete command just to empty a table with millions of records which eventually lock the whole table for doing anything and take ages to complete or Simply blew log segment or hang the machine.

[](http://javarevisited.blogspot.com/2011/04/symbolic-link-or-symlink-in-unix-linux.html)

Most of **enterprise stock trading system** maintains two kind of database one transactional and other static. Transactional data is for day by day records which need to be purge at end of data or moved to historical data so that application can make a fresh start another day. If you need to work on such large set of data, my advice is to get clear and complete knowledge of delete and truncate command, along with there differences and when to use which command to remove data or purge tables.

In this article we will see **where to use truncate in SQL** and where to use delete in SQL, How to use truncate or delete and what danger or harm they can create if not used carefully along with *difference between truncate and delete in SQL*.

### What is Truncate command in SQL

Use truncate table if you need to delete all rows, since truncate doesn't allow you to specify WHERE clause. truncate removes data by deallocating space used by table which removes lot of overhead in terms of logging and locking and that's why truncate is faster than delete.What you need to take care is rollback, data deleted by truncate can not be rolled back until data server specifically supports it e.g. MSSQL Server which allows to *commit or rollback truncate table* statement transactional. Another caveat with truncate table statement is that it doesn't fire a trigger and you can not truncate a table when a foreign key references any column to the table to be truncated. Only situation I see which is perfect for using truncate is purging tables with huge data, though there is another solution exists to drop table and recreated it if that make sense.

**Example of truncate command in SQL**

**truncate table Orders;**  //Order table shouldn't have a column which is foreign key on other table

### What is Delete command in SQL

Delete is another sql command available for removing records from table. Delete is even more flexible than truncate like it provides support to WHERE Clause which can be use to remove selective data. It logs each row which allows operation to be rolled back and it also fires triggers. One disadvantage of using delete is speed and locking. Delete acquires lock on table and its also very slow operation because of logging, which makes it unsuitable for removing records from large tables. One workaround for this is batch-delete in which you remove batch of records instead on one record at a time. Delete is most suitable fore removing selective data and use it where you want to [**rollback transaction in database**](http://javarevisited.blogspot.com/2011/11/database-transaction-tutorial-example.html). It’s not useful to purge large amount of data from tables and should not be used, otherwise it could lock the table for very long time, blew log segment and can take ages to complete.

**Example of delete command in SQL**

**delete  \* from Orders;** //delete all row from Orders, should not be used if Orders is large

**delete  \* from Orders where Symbol="MSFT.NQ"** //delete all orders where symbol is MSFT.NQ

### Difference between truncate and delete command in SQL

This is an important point to understand before using truncate or delete on production environment, or writing any script which purges data from tables.

1.**truncate is fast delete is slow.**

2. truncate doesn't do logging delete logs on per row basis.

3. rollback is possible with delete not with truncate until specifically supported by vendor.

4. truncate doesn't fire trigger, delete does.

5. Don't delete, truncate it when it comes to purge tables.

6. truncate reset identity column in table if any, delete doesn't.

7. truncate is DDL while delete is DML (use this when you are writing exam)

8. truncate doesn't support where clause, delete does.

So finally if you have table with huge data and want to empty it **don’t Delete, truncate it**

### Interview questions on truncate and delete in SQL

Truncate and delete both are popular interview topics and there is always some question on these commands in SQL interview. Here I am listing some of [SQL interview questions](http://java67.blogspot.com/2013/04/10-frequently-asked-sql-query-interview-questions-answers-database.html) based on delete and truncate command in SQL, you can find answer in this article itself or by google.

1) If you have table which contains large amount of data which command will you use for removing data, truncate or delete?

**2) What are differences between truncate and delete?**

3) Which one is fast truncate or delete?

4) What is disadvantage of using truncate in sql?

**5) How will you delete data if truncate is not supported and log segment is also not big enough to support complete delete?**

6) Is there any way to remove data other than truncate and delete in SQL?

Read more: <http://javarevisited.blogspot.com/2011/10/how-to-use-truncate-and-delete-command.html#ixzz3KkTmJ5O8>

# [Database Transaction Tutorial in SQL with Example for Beginners](http://javarevisited.blogspot.com/2011/11/database-transaction-tutorial-example.html)

**Database transaction** is an important concept to understand while working in database and SQL. Transaction in database is required to protect data and keep it consistent when multiple users access the database at same time.  In this **database transaction tutorial** we will learn what is transaction in database, why do you need *transaction in database*, ACID properties of database transaction and an example of database transaction along with commit and rollback.   Almost all vendors like Oracle, MySQL, SQL Server or Sybase provide transaction facility though MySQL only provide it for certain storage engine like InnoDB and BDB and not for MyISAM.

### What is transaction in database?

[database transaction tutorial example in sql](http://javarevisited.blogspot.com/2011/04/symbolic-link-or-symlink-in-unix-linux.html)Database transaction is collection of SQL queries which forms a logical one task. For transaction to be completed successfully all SQL queries has to run successfully. Database transaction executes either all or none, so for example if your database transaction contains 4 SQL queries and one of them fails then change made by other 3 queries will be rolled back. This way your database always remain consistent whether transaction succeeded or failed. Transaction is implemented in database using **SQL keyword transaction, commit and rollback**. Commit writes the changes made by transaction into database and rollback removes temporary changes logged in transaction log by database transaction.

## Database Transaction tutorial

### Why transaction is required in database

Database is used to store data required by real life application e.g. Banking, Healthcare, Finance etc. All your money stored in banks is stored in database, all your shares of DMAT account is stored in database and many application constantly work on these data. In order to protect data and keep it consistent any changes in this data needs to be done in transaction so that even in case of failure data remain in previous state before start of transaction. Consider a Classical example of ATM (Automated Tailor Machine); we all use to withdraw and transfer money by using ATM. If you break withdrawal operation into individual steps you will find:

1) Verify account details.

2) Accept withdrawal request

3) Check balance

4) Update balance

4) Dispense money

Suppose your account balance is 1000$ and you make a withdrawal request of 900$. At fourth step your balance is updated to 900$ and ATM machine stops working due to power outage

Once power comes back and you again tried to withdraw money you surprised by seeing your balance just 100$ instead of 1000$. This is not acceptable by any person in the world :) so we need transaction to perform such task. If SQL statements would have been executed inside transaction in database balance would be either 100$ until money has been dispensed or 1000$ if money has not been dispensed.

### ACID Properties of database transaction

There are four important **properties of database transactions** these are represented by acronym ACID and also called **ACID properties or database transaction** where:

A stands for **Atomicity**, Atom is considered to be smallest particle which can not be broken into further pieces.database transaction has to be atomic means either all steps of transaction completes or none of them.

C stands for **Consistency**, transaction must leave database in consistent state even if it succeed or rollback.

I is for **Isolation**

Two database transactions happening at same time should not affect each other and has consistent view of database. This is achieved by using isolation levels in database.

D stands for **Durability**

Data has to be persisted successfully in database once transaction completed successfully and it has to be saved from power outage or other threats. This is achieved by saving data related to transaction in more than one places along with database.

### When to use database transaction

Whenever any operation falls under ACID criteria you should use transactions. Many real world scenarios require transaction mostly in banking, finance and trading domain.

**How to implement transaction in SQL**

*Database transaction is implemented in SQL* using three keywords start transaction, commit and rollback.once you type start transaction, database starts a transaction and execute all subsequent SQL statements in transaction and keep track of all of them to either commit or rollback changes. **Commit** keywords saves then changes made by transaction into database and after commit change is normally visible to other transaction though is subject to **isolation level**. In case you encountered any error while executing individual sql statements inside database transaction, you can rollback all your changes by executing "**rollback**" command.

### Database Transaction Example

To understand database transaction better let's see a real life example of transaction in database. For this example we will assume we have an Account table which represent a Bank Account and we will transfer money from one account to another account

Request: transfer 900$ from Account 9001 to 9002

**start transaction**

select balance from Account where Account\_Number='9001';

select balance from Account where Account\_Number='9002';

update Account set balance=balance-900 here Account\_Number='9001' ;

update Account set balance=balance+900 here Account\_Number='9002' ;

**commit;** //if all sql queries succed

**rollback;** //if any of Sql queries failed or error

### Database transaction in MySQL

In my previous [mysql command tutorials](http://javarevisited.blogspot.com/2010/10/frequently-used-mysql-commands-part-3.html) I have talked aobut different databse storage engines available in mysql e.g. myISAM or InnoDB. Not all mysql engines supports transaction in order to make transaction works in mysql you either need to use InnoDB or BDB Engine. You can specify engige while creating table in mysql or you can also change your engine in mysql by using ALTER keyword. For example **"ALTER TABLE tablename TYPE=InnoDB;**

### Important point about database transaction

1. *Database transaction* is nothing but a set of SQL statement.

2. Transaction in database is either all or none means either all SQL statement success or none.

3. Its good practice to execute sql query inside transaction and commit or rollback based on result but you need to be little careful with transaction log. To faciliate rollback and commit every sql query which executed inside database transaction is written into transaction log and size of transaction log can grow significantly if don't commit or rollback for longtime.

4. Effect of two simulteneous database transaction into data is controlled by using Isolation level. Isolation level is used to separate one database transaction with other and currently there are four databse isolation levels:

1**) Read Uncommited**

This is lowest level of databse isolation level in this one database transaction can see changes made by other databse transaction which is not yet commited. This can allow you dirty read so quite dangerous.

2) **Read Commited**

This is sligltly better where one database transaction only sees commited changes by other database transaction. But this is also not safe and can lead you to non-repeatable reads problem.

3) **Repeatable Reads**

4) **Serializable**

Highest level of database isolation level. In this all **database transactions** are totally isolated with other database transaction.though this is safe but this safety can cause significant performance hit.

5. MyISAM storage engine in MySQL doesn't support transaction. In order to make transaction works in MySQL use InnoDB.

6. Databse transaction should follow ACID properties.

That’s all for now on database transaction tutorial, I will add more useful points about transaction in databse as I come across or recall, you can also provide your input and issues face during transaction in database on different RDBMS e.g. Oracle, MySQL, MSSQL Server or Sybase etc.

Read more: <http://javarevisited.blogspot.com/2011/11/database-transaction-tutorial-example.html#ixzz3KkVTHGiD>