JDBC Performance from the Inside

July 2017

Introduction

- Dave Cramer
- Work for OpenSCG supporting PostgreSQL
- Maintainer for the JDBC driver since 1999
- There are many options for connecting
- Many of them I didn't totally understand
- This talk hopes to unveil some of the more interesting ones
- And explain how they work.

Overview

- History of the driver
- Connecting to the driver
- Under utilized features
- Performance tips
- Latest Release major features

History

- Originally written by Peter Mount in 1997
- Supported JDBC 1.2
- 1997 JDBC 1.2 Java 1.1
- 1999 JDBC 2.1 Java 1.2
- 2001 JDBC 3.0 Java 1.4
- 2006 JDBC 4.0 Java 6
- 2011 JDBC 4.1 Java 7
- 2014 JDBC 4.2 Java 8
- 2017 JDBC 4.3 Java 9 (Maybe ?)
- Each one of these were incremental additions to the interface
- Requiring additional concrete implementations of the spec to be implemented

Connecting to the server

URL options

- jdbc:postgresql:
    - Connects to localhost, port 5432, database specified in user
- jdbc:postgresql://host/
    - Connects to <host>, port 5432, and database specified in user
- jdbc:postgresql://host:port/
    - Connects to <host><port> and database specified in user
- jdbc:postgresql:database
- jdbc:postgresql://host:port/database
- jdbc:postgresql://host1:port, host2:port/database

Connection Properties

- PG_DBNAME
- PG_DBHOST
- PG_DBPORT

These can be used in the following manner

```
Properties props = new Properties();
props.setProperty(PGProperty.PG_DBNAME.getName(),"test");
props.setProperty(PGProperty.PG_HOST.getName(),"localhost");
props.setProperty(PGProperty.PG_PORT.getName(),"5432");

props.setProperty("user","davec");
props.setProperty("password", "");
Connection connection = DriverManager.getConnection("jdbc:postgresql:”, props);
```

OpenSCG BIGSQL

Logging

- loggerLevel = OFF|DEBUG|TRACE
  - Enables java.util.logging.Logger DEBUG=FINE, TRACE=FINEST
  - Not intended for SQL logging but rather to debug the driver
- loggerFile=<filename> the file to output the log to. If this is not set then the output will be written to the console.

Logging continued

- We will honour DriverManager.setLogStream or DriverManager.setLogWriter
- Parent logger is org.postgresql
- Since we are using java.util.Logging, we can use a properties file to configure logging
- handlers=java.util.logging.FileHandler
- org.postgresql.level=FINEST
- java –Djava.util.logging.config.file=…

Logging continued

- logUnclosedConnections=boolean
- Provides an easy way to find connection leaks
- If this is turned on we track connection opening. If the finalizer is reached and the connection is still open the stacktrace message is printed out.

Autosave

- autosave = never | always | conservative
- PostgreSQL transaction semantics all or nothing. This is not always desirable
- autosave=always will create a savepoint for every statement in a transaction.
- The effect of which means that if you do
- Insert into invoice_header …
- Insert into invoice_lineitem …
- If the insert into invoice lineitem fails the header will still be valid.
- In conservative mode if the driver determines that reparsing the query will work then it will be reparsed and retried.

Binary Transfer

- binaryTransferEnable=comma separated list of oid's or names
- binaryTransferDisable
- Currently the driver will use binary mode for most built-in types.

preferQueryMode

- simple
  - Fewer round trips to db no bind, no parse
  - Required for replication connection
- extended
  - Default creates a server prepared statement, uses parse, bind and execute.
  - Protects against sql injection
  - Possible to re-use the statement

preferQueryMode

- extendedForPrepared
  - Does not use extended for statements, only prepared statements
  - Potentially faster execution of statements
- extendedCacheEverything
  - Uses extended and caches even simple statements such as ‘select a from tbl’ which is normally not cached

defaultRowFetchSize=int

- Default is 0 which means fetch all rows
  - This is sometimes surprising and can result in out of memory errors
- If set *AND* autocommit=false THEN will limit the number of rows per fetc
- Potentially significant performance boost

stringtype=varchar|unspecified

- The default is varchar, which tells the server that strings are actually strings!
- You can use stringtype='unspecified'
  - Usefull if you have an existing application that uses setString('1234') to set an integer column.
  - Server will attempt to cast the "string" to the appropriate type.

OpenSCG BIGSQL

ApplicationName=String

- sets the application name
- Servers version 9.0 and greater
- Useful for logging and seeing which connections are yours in pg_stat_activity, etc.

OpenSCG BIGSQL

readOnly=boolean

- The default is false
- True sends **SET SESSION CHARACTERISTICS AS TRANSACTION READ ONLY** to the server.
- This blocks any writes to persistent tables, interestingly you can still write to a temporary table.

disableColumnSanitizer=boolean

- columnSanitizer folds column names to lower case.
- Column names like FirstName become firstname.
- Resultset.getInt("firstname")
- default is to sanitize names

assumeMinServerVersion=String

- Currently there are only 2 use cases
  - 9.0 which will enable
    - ApplicationName=ApplicationName (defaults to PostgreSQL JDBC Driver)
    - sets extra float digits to 3
  - 9.4 necessary for replication connections

OpenSCG BIGSQL

currentSchema=String

- by default the current schema will be “public”
- If you want to refer to a table in a different schema it would have to be specified by schema.table
- If you set this connection property to “audit” for example instead of “select * from audit.log” you could use select * from log;

reWriteBatchedInserts=true

- Enables the driver to optimize batch inserts by changing multiple insert statements into one insert statement.
- Multiple statements such as “insert into tab1 values (1,2,3);”
- Rewritten as “insert into tab1 values (1,2,3), (4,5,6)

OpenSCG BIGSQL

Connection Failover

- Specify multiple hosts in the connection string
- "jdbc:postgresql://host1:port1,host2:port2/database"
- By default this will attempt to make connections to each host until it succeeds

OpenSCG BIGSQL

Connection Failover tuning

- targetServerType=master, slave, preferSlave
  - Observes if server allows writes to pick
  - preferSlave will try slaves first then fall back to master
- loadBalanceHosts=boolean will randomly pick from suitable candidates
- hostRecheckSeconds=number of seconds between checking status (read or write) of hosts default is 10 seconds

replication=database, true

- Tells the backend to go into walsender mode
- Simple query mode, subset of commands
- Setting to database enables logical replication for that database
- Must be accompanied by assumMinServerVersion="9.4" and preferQueryMode="simple"

Performance tricks

- setFetchSize
- rewriteBatchInserts

OpenSCG BIGSQL

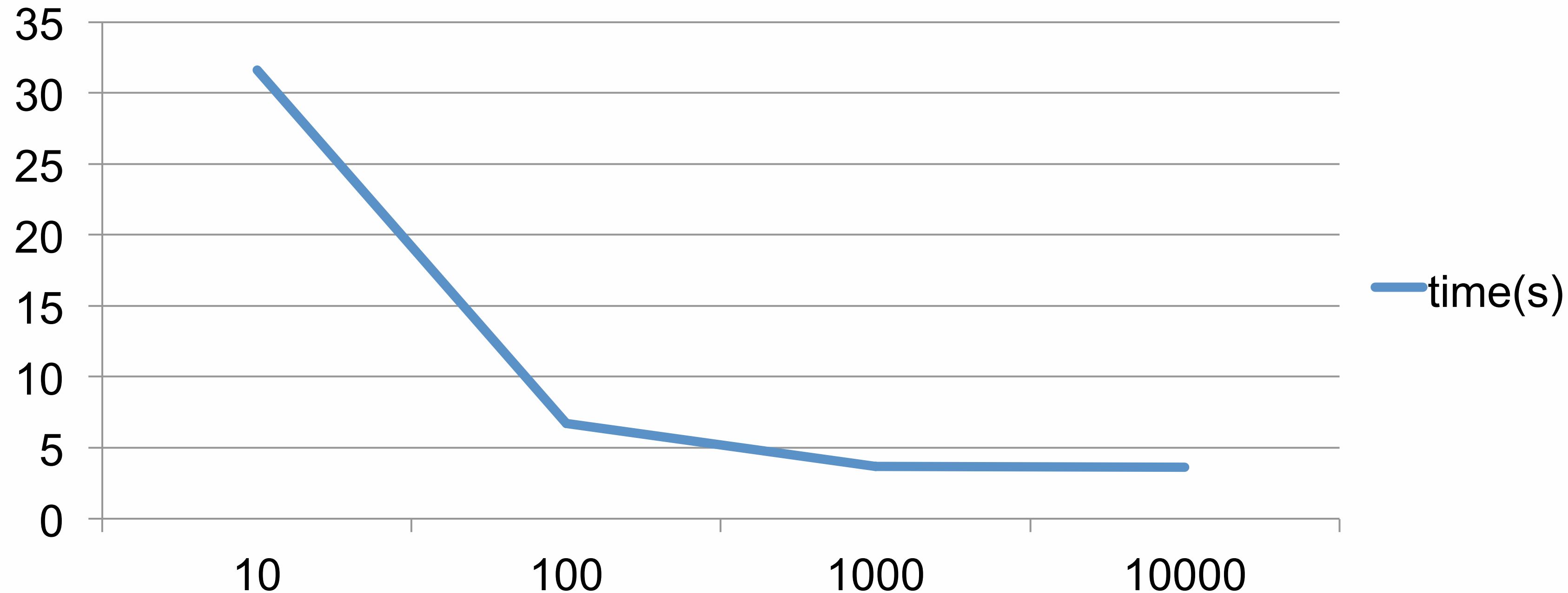
Set FetchSize performance

- Fetch a large amount of data with different fetch sizes

```
public static final String QUERY = "SELECT t FROM number";

@Benchmark
public void test(Blackhole blackhole, PgStatStatements pgStatStatements) throws SQLException {

pgStatStatements.setTestName(QueryBenchmarks.JMHTestNameFromClass(_6_String_NoAutocommit.class));

    QueryUtil.executeProcessQueryNoAutocommit(QUERY, resultSet -> {
            while (resultSet.next()) {
                blackhole.consume(resultSet.getString(1));
            }
    });
}

// Used to fetch rows in batches from the db. Will only work if the connection does not use
AutoCommit
PGProperty.DEFAULT_ROW_FETCH_SIZE.set(properties, FETCH_SIZE);
```

OpenSCG BIGSQL

Time it takes to fetch 1M rows



OpenSCG

BIGSQL

What are the options for inserting lots of data

- For each row insertExecute this is the slowest
- For each row insertBatch this would be ideal
- Insert into foo (i,j) values (1,’one’), (2,’two’) …. (n,’n’) hand rolled code
- Copy into foo from stdin…

JDBC micro benchmark suite

- Java 1.8_60
- Core i7 2.8GHz
- PostgreSQL 9.6
- https://github.com/pgjdbc/pgjdbc/tree/master/ubenchmark
- create table batch_perf_test(a int4, b varchar(100), c int4)

```
Table "public.batch_perf_test"

Column |         Type
--------+-----------------------------------
   a    | integer
   b    | character varying(100)
   c    | integer
```

OpenSCG BIGSQL

INSERT Batch 1 row at a time

- For each row Insert into perf (a,b,c) values (?,?,?)
- After N rows executeBatch
- Normal mode this executes N inserts, not any faster than
- Looping over N inserts without batch mode

INSERT Batch N rows_at_a_time

- For each row Insert into perf (a,b,c) values (?,?,?), (?,?,?), (?,?,?), (?,?,?)
- After N/ rows_at_a_time rows executeBatch
- Given 1000 (N) rows if we insert them 100(rows_at_a_time) , end up inserting 10 rows 100 wide
- More data inserted per statement, less statements

INSERT Batch with insertRewrite

- For each row Insert into perf (a,b,c) values (?,?,?)
- After N rows executeBatch
- Same as last slide except we set the connection parameter insertRewrite=true
- As of version 1209 this is has been enabled
- Same as insert into foo (i,j) values (1,'one'), (2,'two') …. (n,'n') except the driver does it for you.

Copy

- Loop over the rows creating the input string in memory
- Build a string in memory which looks like 0\ts0\t0\n1\ts1\t1\n….
- The string will end up being nrows / rows_at_a_time long
- Use the copy API to copy this into the table

Hand rolled insert struct N structs at a time

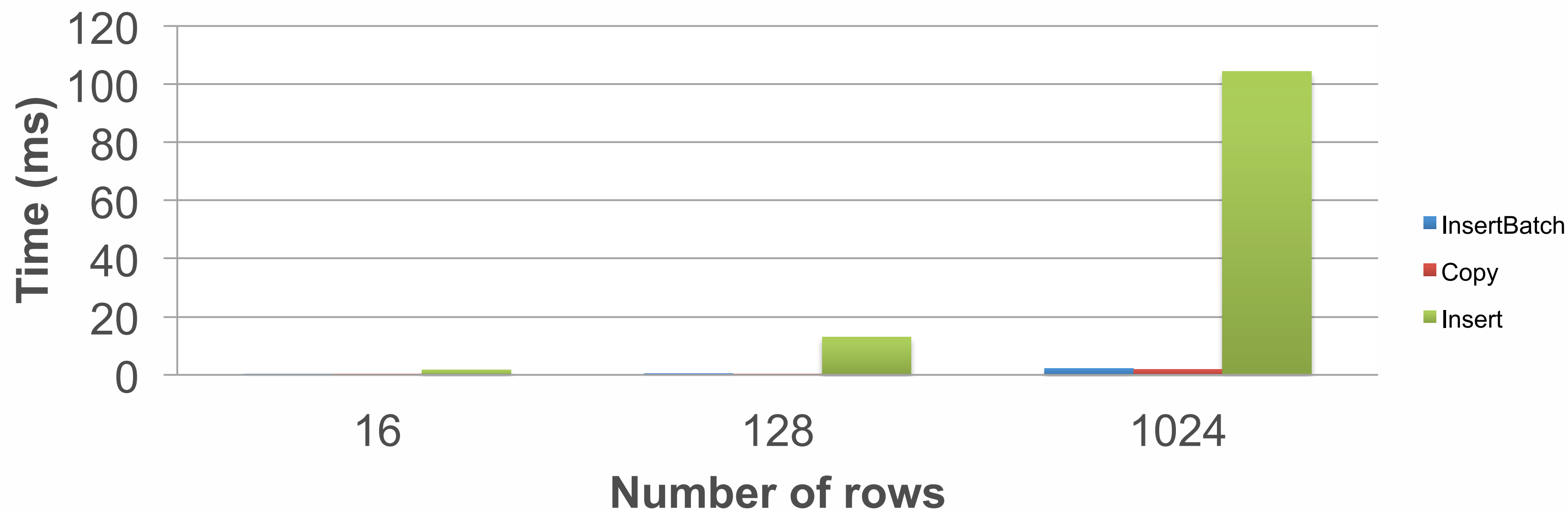
- Insert into batch_perf_test select * from unnest (?::batch_perf_test[])
- For N rows setString to ‘{“(1,s1,1)”,”(2,s2,2)”,”(3,s3,3)”}’
- Add Batch
- executeBatch
- The query that gets executes look like:

Insert into batch_perf_test select *

from unnest (‘{“(1,s1,1)”,”(2,s2,2)”,”(3,s3,3)”}’::batch_perf_test[])

OpenSCG BIGSQL

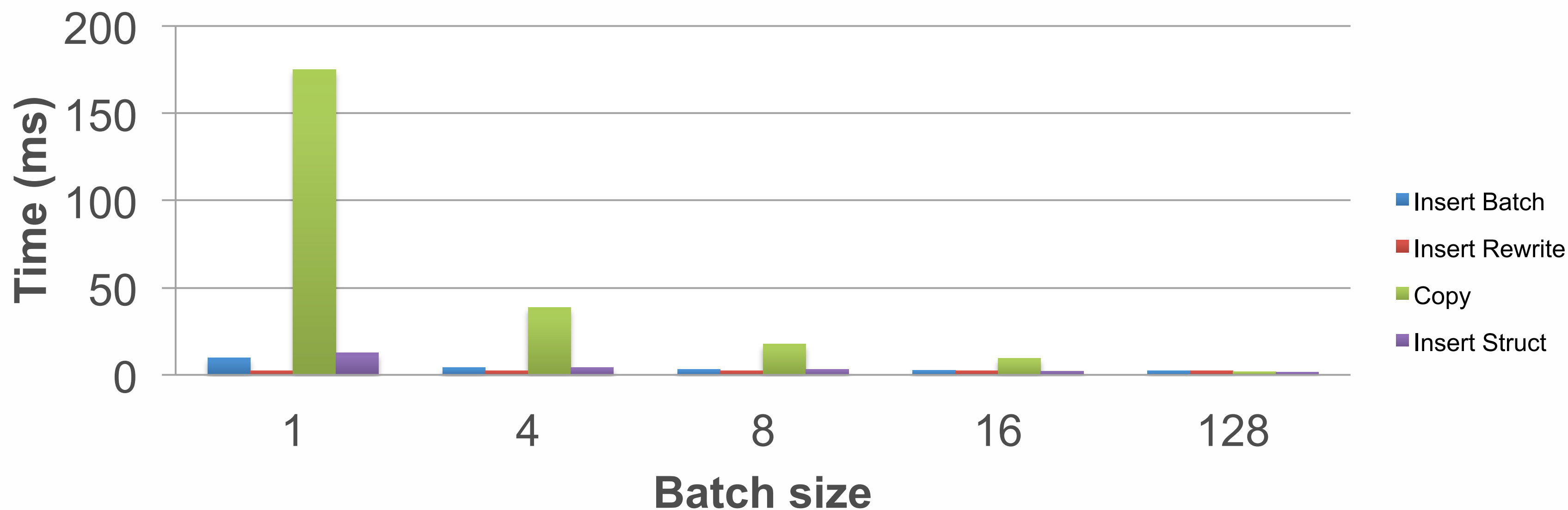
Results



Conclusion

- Compared to batch inserts, plain inserts are very slow for large amounts of data

Results



OpenSCG BIGSQL

How not to use JDBC (unfortunately typical)

- Open connection
- Prepare statement ‘select * from foo where id=?’
- preparedStatment.executeQuery()
- preparedStatement.close()
- Close Connection
- Without a pool connection creation is a heavyweight operation. PostgreSQL uses processes so each connection is a process
- Does not take advantage of caching

OpenSCG BIGSQL

Better solution

- Open connection
- Prepare statement ‘select * from foo where id=?’
- By default after 5 executions will create a named statement PARSE S_1 as ‘select * from foo where id=?’
- Multiple preparedStatment.executeQuery() BIND/EXEC instead of PARSE/BIND/EXEC

- Never close the statement if possible

OpenSCG BIGSQL

Query cache best practices

- Client side query cache only works in 9.4.1203 and up
- Do not use generated queries, as they generate new server side prepared statement
- Things like executeUpdate('**insert into foo (i,l,f,d) values (1,2,3,4)'**) will never use a named statement
- Do not change the type of a parameter as this leads to DEALLOCATE/PREPARE
- Pstmt.setInt(1,1)
- Pstmt.setNull(1,Types.VARCHAR) this will cause the prepared statement to be deallocated

OpenSCG BIGSQL

Less obvious issues

- Server Prepare activated after 5 executions
- There is a configuration parameter called prepareThreshold (default 5)
- PGStatement.isUseServerPrepare() can be used to check
- After 5 executions of the same prepared statement we change from unnamed statements to named
- Named statements will use binary mode where possible;
- binary mode is faster when we have to parse things like timestamps
- Named statements are only parsed once on the server then bind/execute

  operations on the server

setFetchSize

- If we don't use a fetch size we will read the entire response into memory then process
- Optimizing the data sent at one time reduces memory usage and GC
- Only works with in a transaction
- Make sure fetch size is above 100
- If you have a lot of data this is really the only way to read it in without an Out Of Memory Exception

Performance enhancements review

- Cache parsed statements across PrepareStatement calls now don't have to parse the statement in java each time
- Execute Batch changed to not execute statement by statement bug in code disabled batching
- Rewrite Batched inserts rewrites inserts from multiple insert into foo (a,b,c) values (1,2,3) to insert into foo (a,b,c) values (1,2,3), (4,5,6) this provides 2x-3x speed up
- Avoid Calendar cloning provides 4x speed increase for setTimestamp pr 376

Conclusions

- Using insert rewrite gives us a 2-3x performance increase for batch inserts
- Makes sense as it is one trip
- Use setFetchSize(100) or greater and use transactions
- Don’t close prepared statements.

New Release Numbering 42.0.0

- Wanted to divorce ourselves from the server release schedule
- Wanted to reduce confusion as to which version to use. Previously the numbers 9.x were in the version number.
- Introduce semantic versioning
- 42 more or less at random, but also the answer to the question.

Notable changes

- Support dropped for versions before 8.2
- Replace hand written logger with java.util.logging
- Replication protocol API was added.

Logical Replication Overview

- Reads the WAL logs and outputs them in any format you want
- Read changes
- Send confirmation of changes read
- GOTO read more changes

Logical Replication High level Steps

- Create a replication connection
- Create a logical replication slot
- Read changes
- Send confirmation of changes read
- GOTO read more changes

Create a Replication Connection

```
String url = "jdbc:postgresql://localhost:5432/postgres";
 Properties props = new Properties();
 PGProperty.USER.set(props, "postgres");
 PGProperty.PASSWORD.set(props, "postgres");
 PGProperty.ASSUME_MIN_SERVER_VERSION.set(props, "9.4");
 PGProperty.REPLICATION.set(props, "database");
 PGProperty.PREFER_QUERY_MODE.set(props, "simple");
 Connection con = DriverManager.getConnection(url, props);
 PGConnection replConnection = con.unwrap(PGConnection.class);
```

- PGProperty.REPLICATION set to "database" instructs the walsender to connect to the database in the url and allow the connection to be used for logical replication.
- PREFER_QUERY_MODE needs to be set to simple as replication does not allow the use of the extended query mode

Create a Logical Replication Slot

```
String outputPlugin =  'test_decode';
try (PreparedStatement preparedStatement =
              connection.prepareStatement("SELECT * FROM
pg_create_logical_replication_slot(?, ?)"))
{
    preparedStatement.setString(1, slotName);
    preparedStatement.setString(2, outputPlugin);
    preparedStatement.executeQuery())

}
```

- Slots require a name and an output plugin
- Any unique name will work
- The output plugin is a previously compiled C library which formats the logical WAL

OpenSCG BIGSQL

Create a replication stream

```
PGReplicationStream stream =
        pgConnection
                .getReplicationAPI()
                .replicationStream()
                .logical()
                .withSlotName(SLOT_NAME)
                .withStartPosition(lsn)
                .withSlotOption("include-xids", true)
                .start();
```

- Open a PGReplicationStream with the same slot name
- Start position can be an existing LSN or InvalidLSN
- SlotOptions are sent to the logical decoder and are decoder specific

Read Changes from database

```
while (true) {
     //non blocking receive message
     ByteBuffer msg = stream.readPending();
     if (msg == null) {
       TimeUnit.MILLISECONDS.sleep(10L);
       continue;
     }
     int offset = msg.arrayOffset();
     byte[] source = msg.array();
     int length = source.length - offset;
     System.out.println(new String(source, offset, length));
     //feedback
     stream.setAppliedLSN(stream.getLastReceiveLSN());
     stream.setFlushedLSN(stream.getLastReceiveLSN());
   }
```

- Read from the stream, data will be in a ByteBuffer
- After reading the data send confirmation messages
- github.com:davecramer/LogicalDecode.git

https://github.com/pgjdbc/pgjdbc

- Credit where credit is due:
- Much of the optimization work on the driver was done by Vladimir Sitnikov
- Much (if not all) of the work to convert the build to Maven was done by Stephen Nelson
- Rewriting batch statements thanks to Jeremy Whiting
- Replication support was provided by Vladimir Gordiychuk
- Questions ?