Tools for reading code

Calvin Loncaric

Code is **read** much more often than it is **written**.

Code is **read** much more often than it is **written**.

Reading code can be hard and boring!

Quick, what does this function do??

```
@Override
public void tupleBatchInsert(final RelationKey relationKey, final TupleBatch tupleBatch) throws DbException {
  LOGGER.debug("Inserting batch of size {}", tupleBatch.numTuples());
  Objects.requireNonNull(jdbcConnection, "jdbcConnection");

  Schema schema = tupleBatch.getSchema();

  boolean writeSucceeds = false;
  if (jdbcInfo.getDbms().equals(MyriaConstants.STORAGE_SYSTEM_POSTGRESQL)) {
    /*
     * There are bugs when using the COPY command to store doubles and floats into PostgreSQL. See
     * uwescience/myria-web#48
     */
    try {
      postgresCopyInsert(relationKey, schema, tupleBatch);
      writeSucceeds = true;
    } catch (DbException e) {
      LOGGER.error("Error inserting batch via PostgreSQL COPY", e);
      /*
       * TODO - should we do a VACUUM now? The bad rows will not be visible to the DB, however, so the write did not
       * partially happen.
       *
       * http://www.postgresql.org/docs/9.2/static/sql-copy.html
       */
    }
  }
  if (!writeSucceeds) {
    try {
      /* Set up and execute the query */
      final PreparedStatement statement =
          jdbcConnection.prepareStatement(insertStatementFromSchema(schema, relationKey));
      for (int row = 0; row < tupleBatch.numTuples(); ++row) {
        for (int col = 0; col < tupleBatch.numColumns(); ++col) {
          switch (schema.getColumnType(col)) {
            case BOOLEAN_TYPE:
              statement.setBoolean(col + 1, tupleBatch.getBoolean(col, row));
              break;
            case DATETIME_TYPE:
              statement.setTimestamp(col + 1, new Timestamp(tupleBatch.getDateTime(col, row).getMillis()));
              break;
            case DOUBLE_TYPE:
              statement.setDouble(col + 1, tupleBatch.getDouble(col, row));
              break;
            case FLOAT_TYPE:
              statement.setFloat(col + 1, tupleBatch.getFloat(col, row));
              break;
            case INT_TYPE:
              statement.setInt(col + 1, tupleBatch.getInt(col, row));
              break;
            case LONG_TYPE:
              statement.setLong(col + 1, tupleBatch.getLong(col, row));
              break;
            case STRING_TYPE:
              statement.setString(col + 1, tupleBatch.getString(col, row));
              break;
          }
        }
        statement.addBatch();
      }
      statement.executeBatch();
      statement.close();
    } catch (final SQLException e) {
      throw ErrorUtils.mergeSQLException(e);
    }
  }
  LOGGER.debug(".. done inserting batch of size {}", tupleBatch.numTuples());
}
```

Quick, what does this function do??

logging

```
@Override
public void tupleBatchInsert(final RelationKey relationKey, final TupleBatch tupleBatch) throws DbException {
  LOGGER.debug("Inserting batch of size {}", tupleBatch.numTuples());
  Objects.requireNonNull(jdbcConnection, "jdbcConnection");

  Schema schema = tupleBatch.getSchema();

  boolean writeSucceeds = false;
  if (jdbcInfo.getDbms().equals(MyriaConstants.STORAGE_SYSTEM_POSTGRESQL)) {
    /*
     * There are bugs when using the COPY command to store doubles and floats into PostgreSQL. See
     * uwescience/myria-web#48
     */
    try {
      postgresCopyInsert(relationKey, schema, tupleBatch);
      writeSucceeds = true;
    } catch (DbException e) {
      LOGGER.error("Error inserting batch via PostgreSQL COPY", e);
      /*
       * TODO - should we do a VACUUM now? The bad rows will not be visible to the DB, however, so the write did not
       * partially happen.
       *
       * http://www.postgresql.org/docs/9.2/static/sql-copy.html
       */
    }
  }
  if (!writeSucceeds) {
    try {
      /* Set up and execute the query */
      final PreparedStatement statement =
          jdbcConnection.prepareStatement(insertStatementFromSchema(schema, relationKey));
      for (int row = 0; row < tupleBatch.numTuples(); ++row) {
        for (int col = 0; col < tupleBatch.numColumns(); ++col) {
          switch (schema.getColumnType(col)) {
            case BOOLEAN_TYPE:
              statement.setBoolean(col + 1, tupleBatch.getBoolean(col, row));
              break;
            case DATETIME_TYPE:
              statement.setTimestamp(col + 1, new Timestamp(tupleBatch.getDateTime(col, row).getMillis()));
              break;
            case DOUBLE_TYPE:
              statement.setDouble(col + 1, tupleBatch.getDouble(col, row));
              break;
            case FLOAT_TYPE:
              statement.setFloat(col + 1, tupleBatch.getFloat(col, row));
              break;
            case INT_TYPE:
              statement.setInt(col + 1, tupleBatch.getInt(col, row));
              break;
            case LONG_TYPE:
              statement.setLong(col + 1, tupleBatch.getLong(col, row));
              break;
            case STRING_TYPE:
              statement.setString(col + 1, tupleBatch.getString(col, row));
              break;
          }
        }
        statement.addBatch();
      }
      statement.executeBatch();
      statement.close();
    } catch (final SQLException e) {
      throw ErrorUtils.mergeSQLException(e);
    }
  }
  LOGGER.debug(".. done inserting batch of size {}", tupleBatch.numTuples());
}
```

Quick, what does this function do??

```
@Override
public void tupleBatchInsert(final RelationKey relationKey, final TupleBatch tupleBatch) throws DbException {
  LOGGER.debug("Inserting batch of size {}", tupleBatch.numTuples());
  Objects.requireNonNull(jdbcConnection, "jdbcConnection");

  Schema schema = tupleBatch.getSchema();

  boolean writeSucceeds = false;
  if (jdbcInfo.getDbms().equals(MyriaConstants.STORAGE_SYSTEM_POSTGRESQL)) {
    /*
     * There are bugs when using the COPY command to store doubles and floats into PostgreSQL. See
     * uwescience/myria-web#48
     */
    try {
      postgresCopyInsert(relationKey, schema, tupleBatch);
      writeSucceeds = true;
    } catch (DbException e) {
      LOGGER.error("Error inserting batch via PostgreSQL COPY", e);
      /*
       * TODO - should we do a VACUUM now? The bad rows will not be visible to the DB, however, some were did not
       * partially happen.
       *
       * http://www.postgresql.org/docs/9.2/static/sql-copy.html
       */
    }
  }
  if (!writeSucceeds) {
    try {
      /* Set up and execute the query */
      final PreparedStatement statement =
          jdbcConnection.prepareStatement(insertStatementFromSchema(schema, relationKey));
      for (int row = 0; row < tupleBatch.numTuples(); ++row) {
        for (int col = 0; col < tupleBatch.numColumns(); ++col) {
          switch (schema.getColumnType(col)) {
            case BOOLEAN_TYPE:
              statement.setBoolean(col + 1, tupleBatch.getBoolean(col, row));
              break;
            case DATETIME_TYPE:
              statement.setTimestamp(col + 1, new Timestamp(tupleBatch.getDateTime(col, row).getMillis()));
              break;
            case DOUBLE_TYPE:
              statement.setDouble(col + 1, tupleBatch.getDouble(col, row));
              break;
            case FLOAT_TYPE:
              statement.setFloat(col + 1, tupleBatch.getFloat(col, row));
              break;
            case INT_TYPE:
              statement.setInt(col + 1, tupleBatch.getInt(col, row));
              break;
            case LONG_TYPE:
              statement.setLong(col + 1, tupleBatch.getLong(col, row));
              break;
            case STRING_TYPE:
              statement.setString(col + 1, tupleBatch.getString(col, row));
              break;
          }
        }
        statement.addBatch();
      }
      statement.executeBatch();
      statement.close();
    } catch (final SQLException e) {
      throw ErrorUtils.mergeSQLException(e);
    }
  }
  LOGGER.debug(".. done inserting batch of size {}", tupleBatch.numTuples());
}
```

logging

debugging

Quick, what does this function do??

```
@Override
public void tupleBatchInsert(final RelationKey relationKey, final TupleBatch tupleBatch) throws DbException {
  LOGGER.debug("Inserting batch of size {}", tupleBatch.numTuples());
  Objects.requireNonNull(jdbcConnection, "jdbcConnection");

  Schema schema = tupleBatch.getSchema();

  boolean writeSucceeds = false;
  if (jdbcInfo.getDbms().equals(MyriaConstants.STORAGE_SYSTEM_POSTGRESQL)) {
    /*
     * There are bugs when using the COPY command to store doubles and floats into PostgreSQL. See
     * uwescience/myria-web#48
     */
    try {
      postgresCopyInsert(relationKey, schema, tupleBatch);
      writeSucceeds = true;
    } catch (DbException e) {
      LOGGER.error("Error inserting batch via PostgreSQL COPY", e);
      /*
       * TODO - should we do a VACUUM now? The bad rows will not be visible to the DB, however, some rows may have
       * partially happen.
       *
       * http://www.postgresql.org/docs/9.2/static/sql-copy.html
       */
    }
  }
  if (!writeSucceeds) {
    try {
      /* Set up and execute the query */
      final PreparedStatement statement =
          jdbcConnection.prepareStatement(insertStatementFromSchema(schema, relationKey));
      for (int row = 0; row < tupleBatch.numTuples(); ++row) {
        for (int col = 0; col < tupleBatch.numColumns(); ++col) {
          switch (schema.getColumnType(col)) {
            case BOOLEAN_TYPE:
              statement.setBoolean(col + 1, tupleBatch.getBoolean(col, row));
              break;
            case DATETIME_TYPE:
              statement.setTimestamp(col + 1, new Timestamp(tupleBatch.getDateTime(col, row).getMillis()));
              break;
            case DOUBLE_TYPE:
              statement.setDouble(col + 1, tupleBatch.getDouble(col, row));
              break;
            case FLOAT_TYPE:
              statement.setFloat(col + 1, tupleBatch.getFloat(col, row));
              break;
            case INT_TYPE:
              statement.setInt(col + 1, tupleBatch.getInt(col, row));
              break;
            case LONG_TYPE:
              statement.setLong(col + 1, tupleBatch.getLong(col, row));
              break;
            case STRING_TYPE:
              statement.setString(col + 1, tupleBatch.getString(col, row));
              break;
          }
        }
        statement.addBatch();
      }
      statement.executeBatch();
      statement.close();
    } catch (final SQLException e) {
      throw ErrorUtils.mergeSQLException(e);
    }
  }
  LOGGER.debug(".. done inserting batch of size {}", tupleBatch.numTuples());
}
```

logging

debugging

error handling

Quick, what does this function do??

```
@Override
public void tupleBatchInsert(
      final RelationKey relationKey,
      final TupleBatch tupleBatch) throws DbException {
  Schema schema = tupleBatch.getSchema();

  if (jdbcInfo.getDbms().equals(MyriaConstants.STORAGE_SYSTEM_POSTGRESQL)) {
    /*
     * There are bugs when using the COPY command to store doubles and
     * floats into PostgreSQL. See uwescience/myria-web#48
     */
    postgresCopyInsert(relationKey, schema, tupleBatch);
  }

}
```

How can we figure out what code is less-relevant?

How can we figure out what code is less-relevant?

- Static properties
  - `try…catch…`
  - `org.slf4j.LoggerFactory`
  - `boolean success = …`

How can we figure out what code is less-relevant?

- Static properties
  - `try…catch…`
  - `org.slf4j.LoggerFactory`
  - `boolean success = …`
- Dynamic properties
  - What code runs more often?

How can we figure out what code is less-relevant?

- Static properties
  - `try…catch…`
  - `org.slf4j.LoggerFactory`
  - `boolean success = …`
- Dynamic properties
  - What code runs more often?
- Fuzzy reasoning
  - We don’t (yet) know how to reconstruct the programmer’s intent exactly… but maybe we can approximate it!