Assignment 2 Solution

Database Systems

DataLab, CS, NTHU

Spring, 2018

Outline

- *UpdateItem* transaction (SP/JDBC implementations)
- *TestbedLoader* (JDBC implementation)
- *StatisticManager*

Outline

- *UpdateItem* transaction (SP/JDBC implementations)
- *TestbedLoader* (JDBC implementation)
- *StatisticManager*

Modified/Added Classes

- Shared class
  - *As2TransactionType*
- Client-side classes
  - *As2Rte*
  - *As2UpdateItemParamGen*
  - *As2JdbcExecutor*
  - *As2UpdateItemJob*
- Server-side classes
  - *As2StoredProcFactory*
  - *As2UpdateItemProcParamHelper*
  - *As2UpdateItemProc*
  - *As2Constants*

Modified/Added Classes

- Shared class
  - *As2TransactionType*
- Client-side classes
  - *As2Rte*
  - *As2UpdateItemParamGen*
  - *As2JdbcExecutor*
  - *As2UpdateItemJob*
- Server-side classes
  - *As2StoredProcFactory*
  - *As2UpdateItemProcParamHelper*
  - *As2UpdateItemProc*
  - *As2Constants*

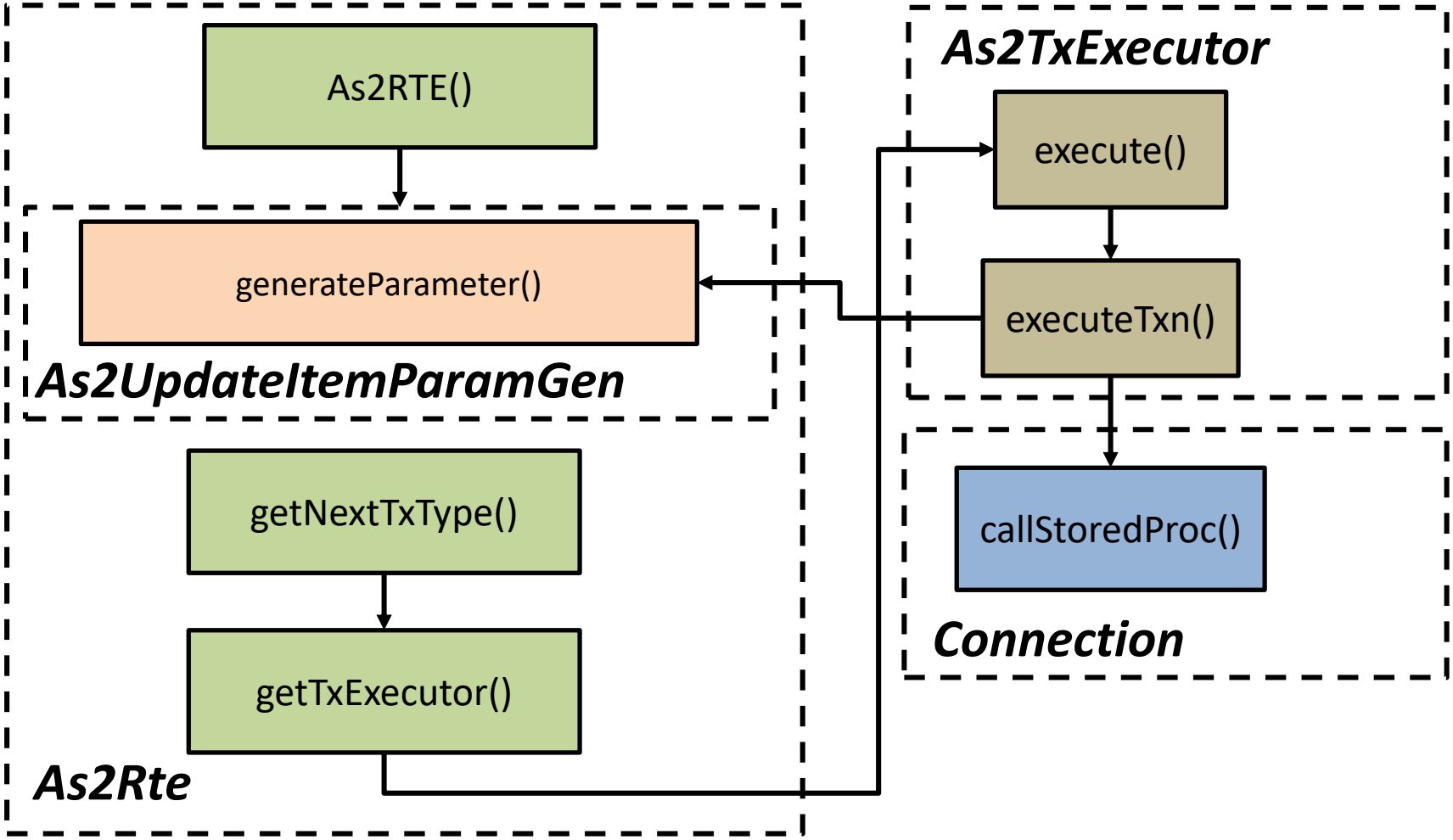
New Transaction Type

```
public enum As2TransactionType implements TransactionType {
        // Loading procedures
        SCHEMA_BUILDER, TESTBED_LOADER,

        // Main procedures
-       READ_ITEM;
+       READ_ITEM, UPDATE_ITEM;

        public static As2TransactionType fromProcedureId(int pid) {
                return As2TransactionType.values()[pid];
        }

        public int getProcedureId() {
                return this.ordinal();
        }

        public boolean isBenchmarkingTx() {
-               if (this == READ_ITEM)
+               if (this == READ_ITEM || this == UPDATE_ITEM)
                        return true;
                return false;
        }
}
```

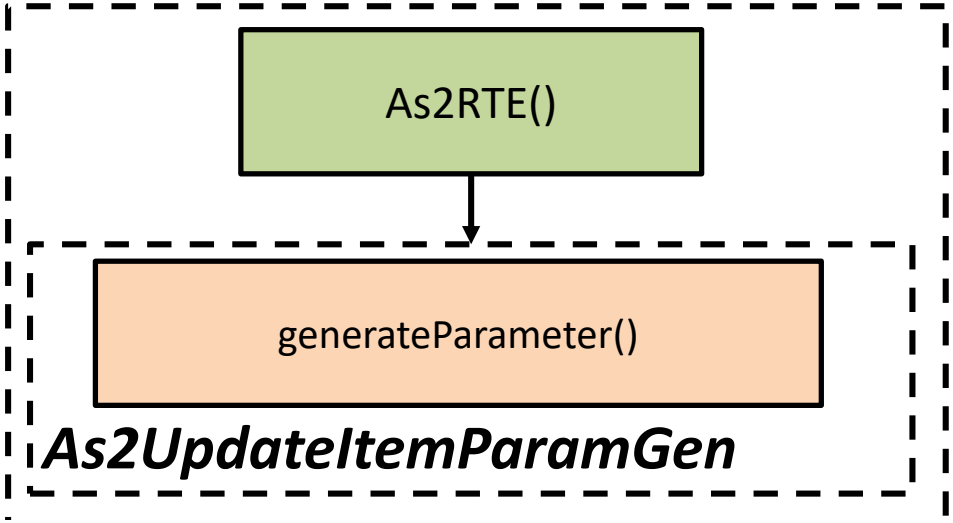
Modified/Added Classes (SP)

- Shared class
  - *As2TransactionType*
- Client-side classes
  - *As2Rte*
  - *As2UpdateItemParamGen*
  - *As2JdbcExecutor*
  - *As2UpdateItemJob*
- Server-side classes
  - *As2UpdateItemProc*
  - *As2StoredProcFactory*
  - *As2UpdateItemProcParamHelper*
  - *As2Constants*

Inquiry via Stored Procedure



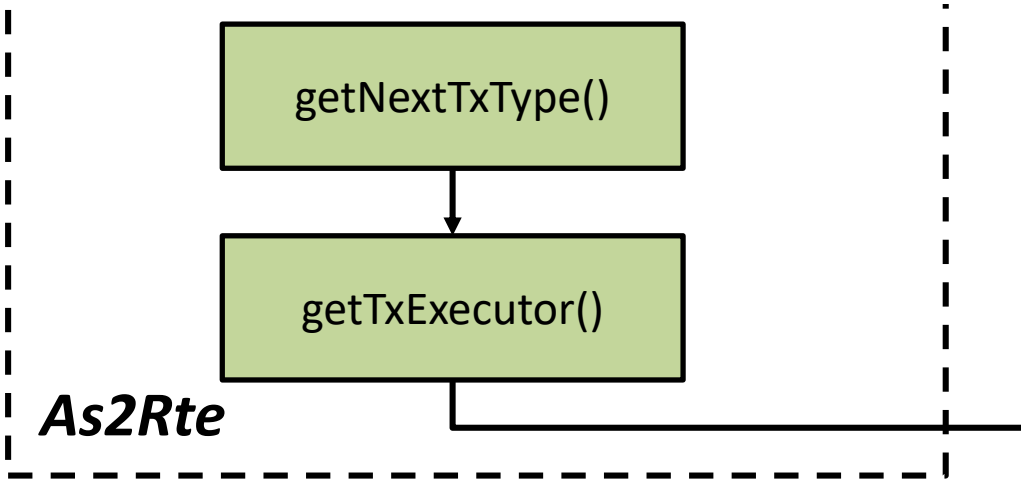
Initialize RTE



Initialize RTE

```
public As2Rte(SutConnection conn, StatisticMgr statMgr) {
        super(conn, statMgr);
-       executor = new As2TxExecutor(new As2ParamGen());
+       executors = new HashMap<As2TransactionType, As2TxExecutor>();
+       rvg = new RandomValueGenerator();
+       executors.put(As2TransactionType.READ_ITEM, new As2TxExecutor(new As2ReadItemParamGen()));
+       executors.put(As2TransactionType.UPDATE_ITEM, new As2TxExecutor(new As2UpdateItemParamGen()));
}
```

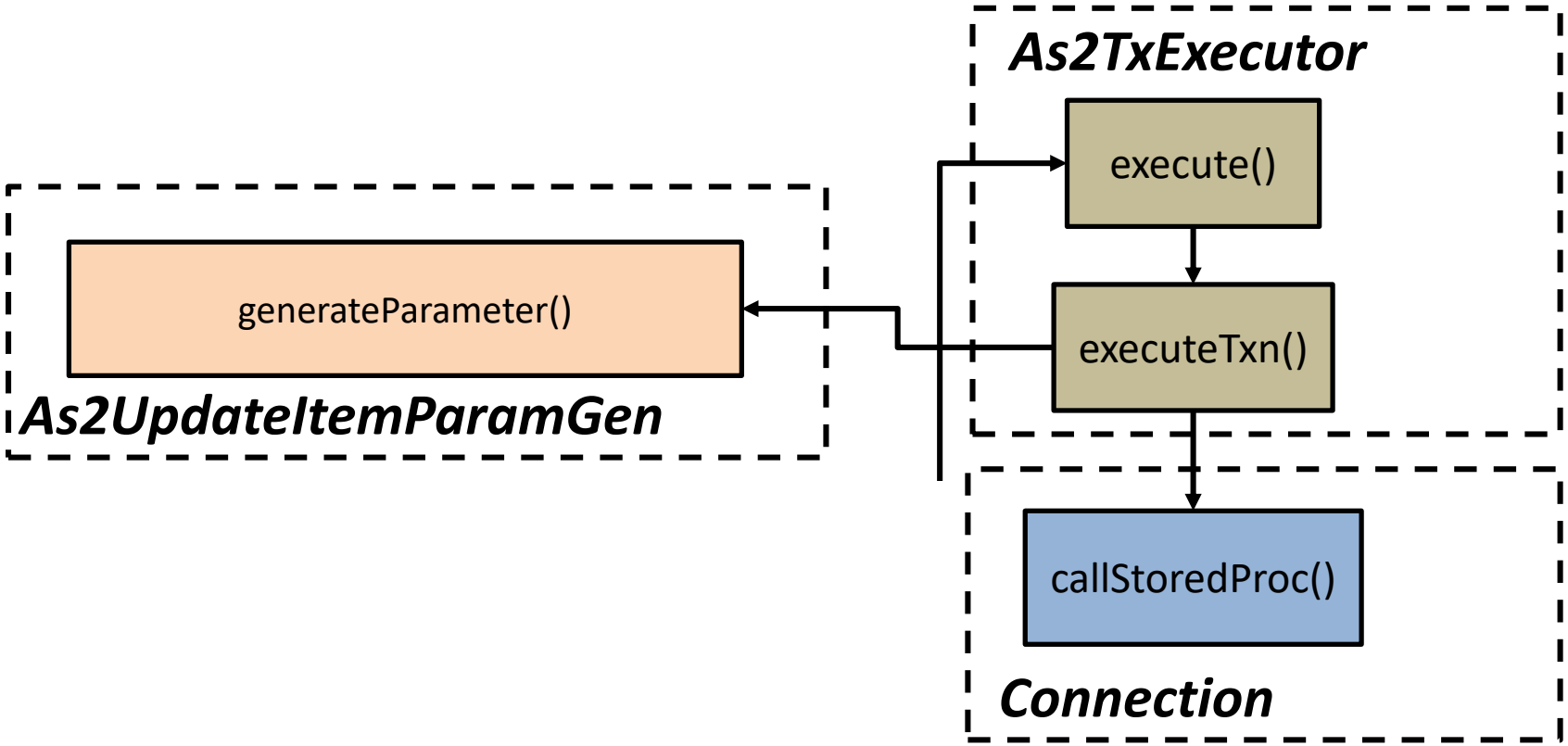
Choose a Transaction



Choose a Transaction

```
        protected As2TransactionType getNextTxType() {
-               return As2TransactionType.READ_ITEM;
+               if (rvg.number(1, 100) <= As2Constants.RATIO_WRITE)
+                       return As2TransactionType.UPDATE_ITEM;
+               else
+                       return As2TransactionType.READ_ITEM;
        }

        protected As2TxExecutor getTxExeutor(As2TransactionType type) {
-               return executor;
+               return executors.get(type);
        }
```

Generate and Send Parameters



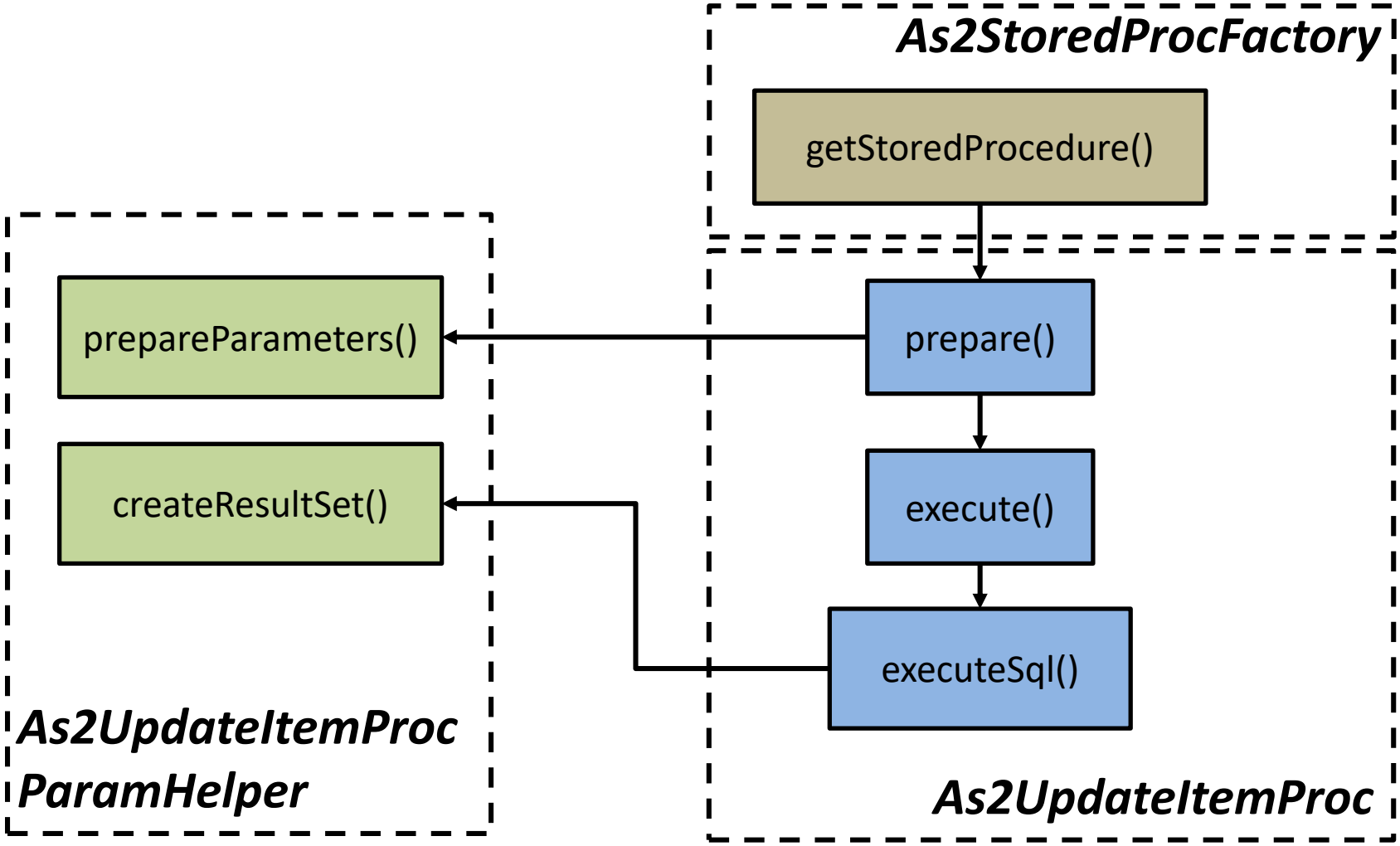
Generate Parameters

```
+       @Override
+       public Object[] generateParameter() {
+               RandomValueGenerator rvg = new RandomValueGenerator();
+               LinkedList<Object> paramList = new LinkedList<Object>();
+
+               paramList.add(UPDATE_COUNT);
+               for (int i = 0; i < UPDATE_COUNT; i++)
+                       paramList.add(rvg.number(1, As2Constants.NUM_ITEMS));
+               for (int i = 0; i < UPDATE_COUNT; i++)
+                       paramList.add(rvg.fixedDecimalNumber(2, As2Constants.MIN_PRICE, As2Constants.MAX_PRICE));
+
+               return paramList.toArray();
+       }
```

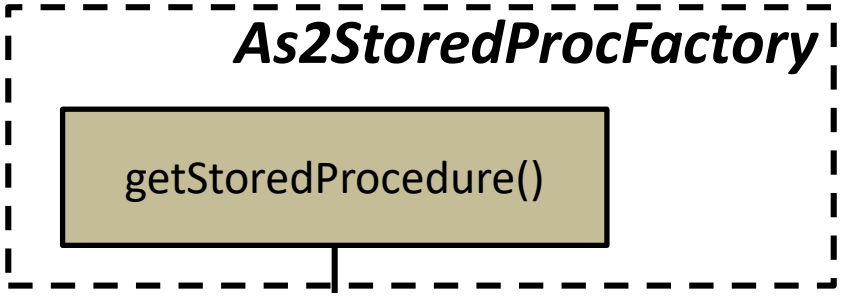
Modified/Added Classes (SP)

- Shared class
  - *As2TransactionType*
- Client-side classes
  - *As2Rte*
  - *As2UpdateItemParamGen*
  - *As2JdbcExecutor*
  - *As2UpdateItemJob*
- Server-side classes
  - *As2UpdateItemProc*
  - *As2StoredProcFactory*
  - *As2UpdateItemProcParamHelper*
  - *As2Constants*

Execute a Stored Procedure



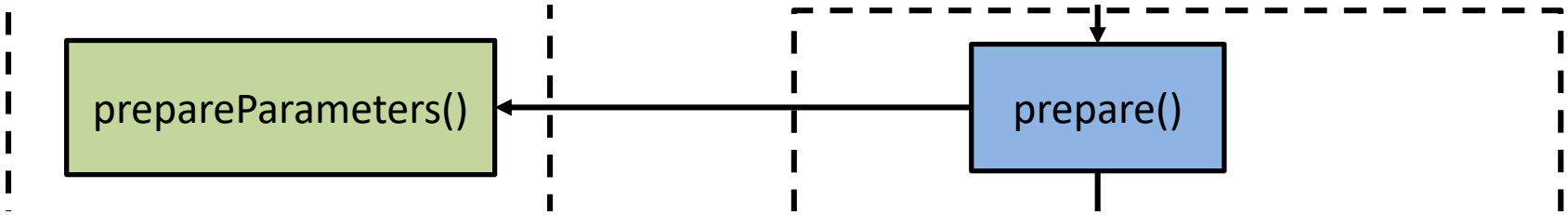
Get the Specified SP



Get the Specified SP

```
@Override
public StoredProcedure getStroredProcedure(int pid) {
        StoredProcedure sp;
        switch (As2TransactionType.fromProcedureId(pid)) {
        case SCHEMA_BUILDER:
                sp = new As2BuilderProc();
                break;
        case TESTBED_LOADER:
                sp = new As2TestbedLoaderProc();
                break;
        case READ_ITEM:
                sp = new As2ReadItemProc();
                break;
+       case UPDATE_ITEM:
+               sp = new As2UpdateItemProc();
+               break;
        default:
                sp = null;
        }
```

Preprocess Parameters

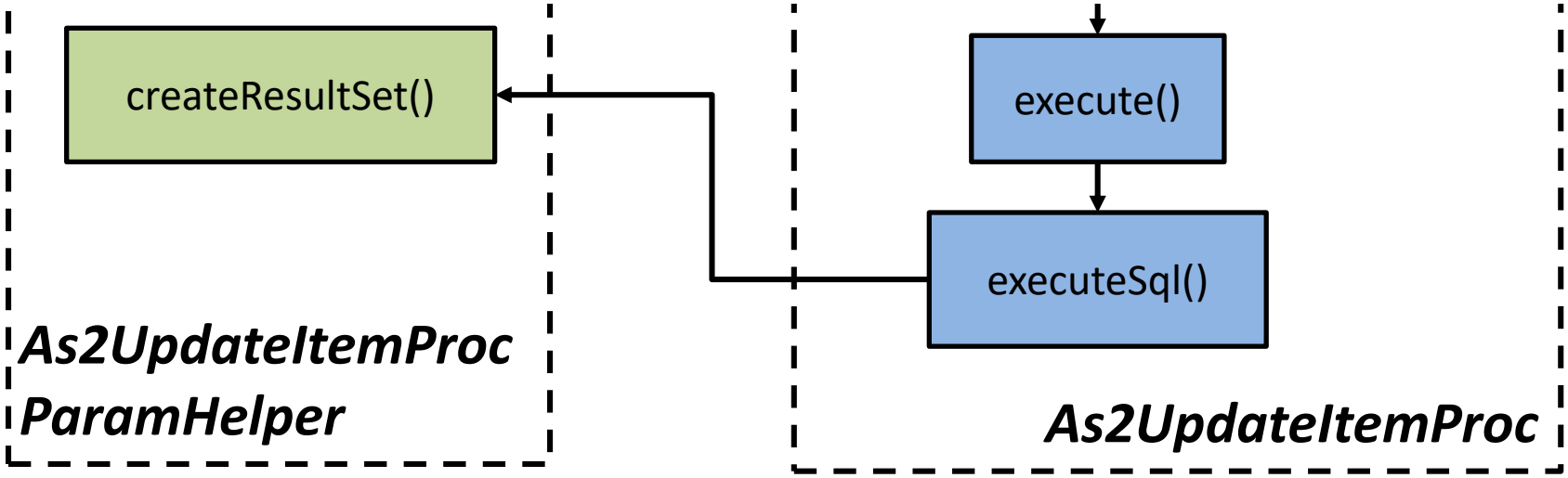


Preprocess Parameters

```
+       public double getUpdateItemPrice(int index) {
+               return updateItemPrice[index];
+       }
```

```
+       @Override
+       public void prepareParameters(Object... pars) {
+
+               // Show the contents of paramters
+               // System.out.println("Params: " + Arrays.toString(pars));
+
+               int indexCnt = 0;
+
+               updateCount = (Integer) pars[indexCnt++];
+               updateItemId = new int[updateCount];
+               updateItemPrice = new double[updateCount];
+               itemName = new String[updateCount];
+               itemPrice = new double[updateCount];
+
+               for (int i = 0; i < updateCount; i++)
+                       updateItemId[i] = (Integer) pars[indexCnt++];
+               for (int i = 0; i < updateCount; i++)
+                       updateItemPrice[i] = (Double) pars[indexCnt++];
+       }
```

Execute Queries



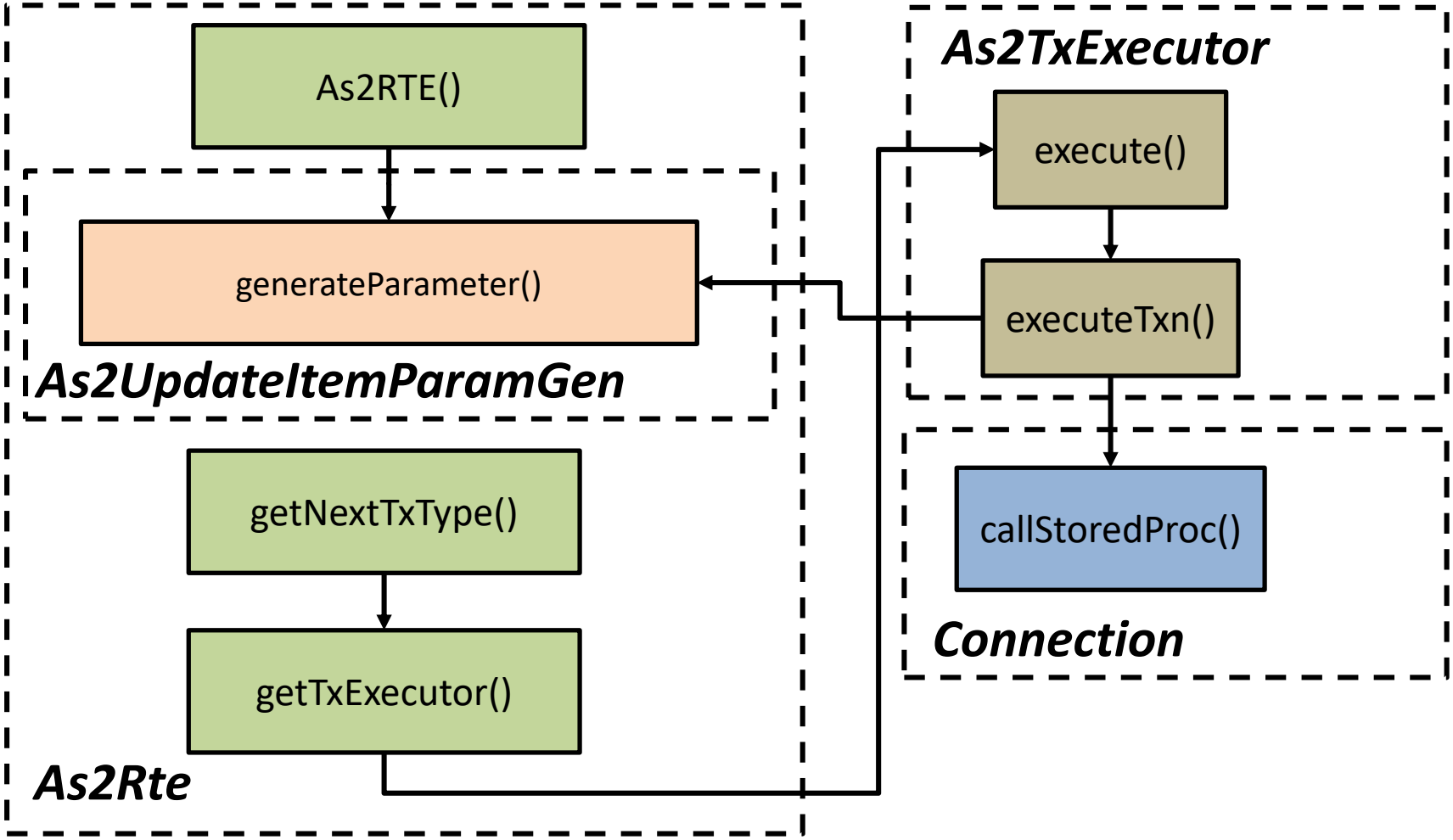
Execute Queries

```
+       @Override
+       protected void executeSql() {
+               for (int idx = 0; idx < paramHelper.getUpdateCount(); idx++) {
+                       int iid = paramHelper.getUpdateItemId(idx);
+                       Plan p = VanillaDb.newPlanner().createQueryPlan(
+                                       "SELECT i_name, i_price FROM item WHERE i_id = " + iid, tx);
+                       Scan s = p.open();
+                       s.beforeFirst();
+                       if (s.next()) {
+                               String name = (String) s.getVal("i_name").asJavaVal();
+                               double price = (Double) s.getVal("i_price").asJavaVal();
+
+                               paramHelper.setItemName(name, idx);
+                               paramHelper.setItemPrice(price, idx);
+
+                               double iprice = paramHelper.getUpdateItemPrice(idx);
+                               VanillaDb.newPlanner().executeUpdate(
+                                               "UPDATE item SET i_price = " + iprice + " WHERE i_id = " + iid, tx);
+                       } else
+                               throw new RuntimeException("Cloud not find item record with i_id = " + iid);
+
+                       s.close();
+               }
```

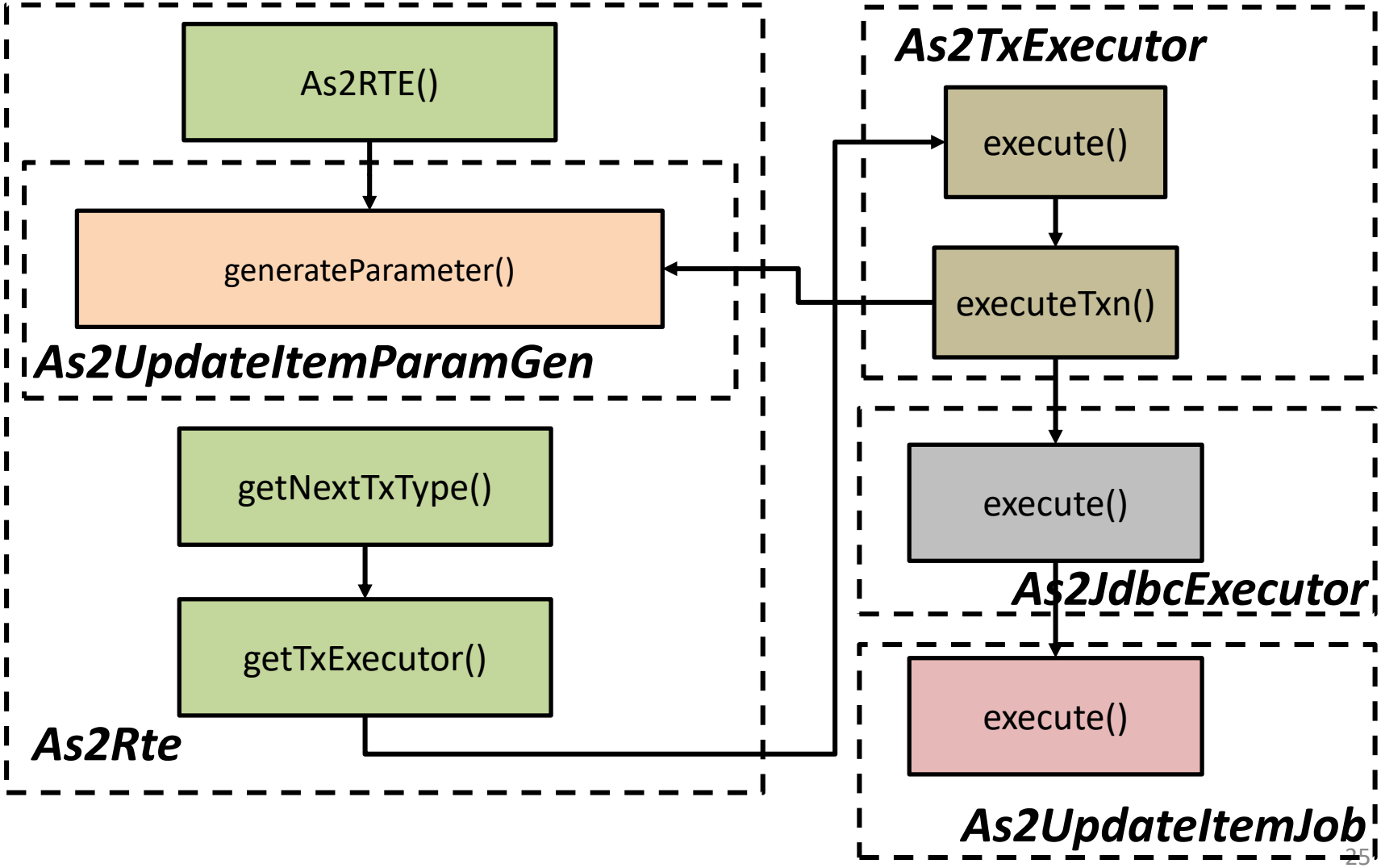
Modified/Added Classes (JDBC)

- Shared class
  - *As2TransactionType*
- Client-side classes
  - *As2Rte*
  - *As2UpdateItemParamGen*
  - *As2JdbcExecutor*
  - *As2UpdateItemJob*
- Server-side classes
  - *As2UpdateItemProc*
  - *As2StoredProcFactory*
  - *As2UpdateItemProcParamHelper*
  - *As2Constants*

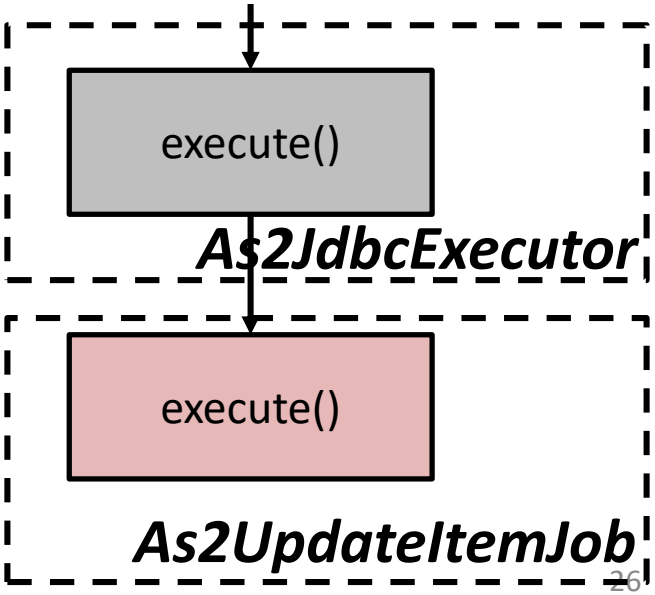
Inquiry via SP



Inquiry via JDBC



Inquiry via JDBC



Inquiry via JDBC

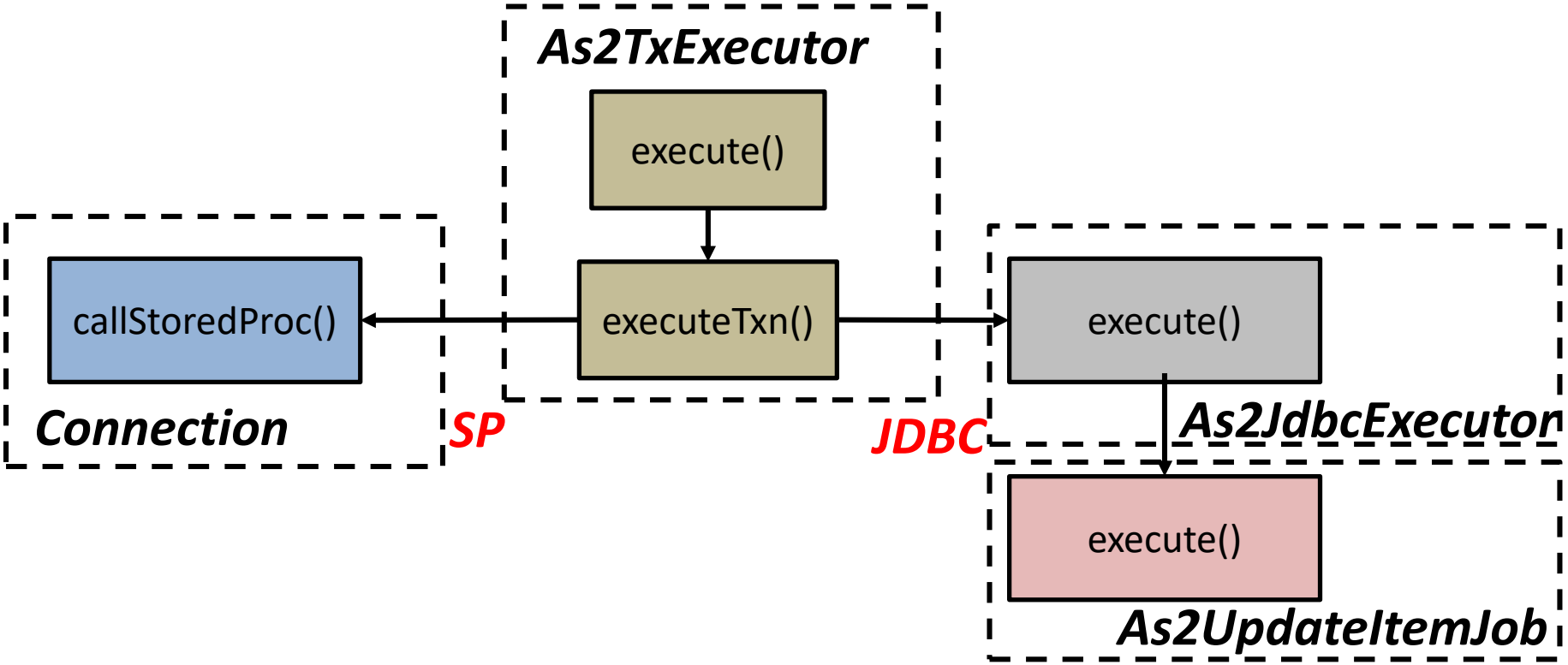
```
public class As2JdbcExecutor implements JdbcExecutor<As2TransactionType> {

        @Override
        public SutResultSet execute(Connection conn, As2TransactionType txType, Object[] pars)
                        throws SQLException {
                switch (txType) {
                case READ_ITEM:
                        return new As2ReadItemJob().execute(conn, pars);
+               case UPDATE_ITEM:
+                       return new As2UpdateItemJob().execute(conn, pars);
                default:
                        throw new UnsupportedOperationException(
                                        String.format("no JDCB implementation for '%s'", txType));
                }
        }

}
```

```
+       @Override
+       public SutResultSet execute(Connection conn, Object[] pars) throws SQLException {
+               // Parse parameters
+               int updateCount = (Integer) pars[0];
+               int[] itemIds = new int[updateCount];
+               double[] itemPrices = new double[updateCount];
+               for (int i = 0; i < updateCount; i++)
+                       itemIds[i] = (Integer) pars[i + 1];
+               for (int i = 0; i < updateCount; i++)
+                       itemPrices[i] = (Double) pars[i + updateCount + 1];
+
+               // Output message
+               StringBuilder outputMsg = new StringBuilder("[");
+
+               // Execute logic
+               try {
+                       Statement statement = conn.createStatement();
+                       ResultSet rs = null;
+                       for (int i = 0; i < 10; i++) {
+                               String sql = "SELECT i_name FROM item WHERE i_id = " + itemIds[i];
+                               rs = statement.executeQuery(sql);
+                               rs.beforeFirst();
+                               if (rs.next()) {
+                                       outputMsg.append(String.format("'%s', ", rs.getString("i_name")));
+                                       sql = "UPDATE item SET i_price = " + itemPrices[i] + " WHERE i_id = " + itemIds[i];
+                                       statement.executeUpdate(sql);
+                               } else
+                                       throw new RuntimeException("cannot find the record with i_id = " + itemIds[i]);
+                               rs.close();
+                       }
+                       conn.commit();
+
```

Outline

- *UpdateItem* transaction (SP/JDBC implementations)
- *TestbedLoader* (JDBC implementation)
- *StatisticManager*

Use Executor




```
protected void executeLoadingProcedure(SutConnection conn) throws SQLException {
-           conn.callStoredProc(As2TransactionType.SCHEMA_BUILDER.ordinal());
-           conn.callStoredProc(As2TransactionType.TESTBED_LOADER.ordinal());
```

As2TestbedLoaderJob

```
@Override
public SutResultSet execute(Connection conn, Object[] pars) throws SQLException {
        // Parse parameters
        paramHelper = new As2SchemaBuilderProcParamHelper();
        paramHelper.prepareParameters(pars);

        // Execute logic
        try {
                Statement statement = conn.createStatement();
                createSchemas(statement);
                generateItems(statement, 1, paramHelper.getNumberOfItems());

                conn.commit();

                return new VanillaDbJdbcResultSet(true, "Successfully load testbed.");
        } catch (Exception e) {
                if (logger.isLoggable(Level.WARNING))
                        logger.warning(e.toString());
                return new VanillaDbJdbcResultSet(false, "Fail loading testbed.");
        }
}
```

refer to *As2BuilderProc*

refer to *As2TestbedLoaderProc*

Outline

- *UpdateItem* transaction (SP/JDBC implementations)
- *TestbedLoader* (JDBC implementation)
- *StatisticManager*

Modified Class

- *StatisticMgr*

Latency History

```
public void addTxnLatency(TxnResultSet rs) {
        long t = TimeUnit.NANOSECONDS.toMillis(rs.getTxnEndTime() - benchStartTime);
        t = (((t - BenchmarkerParameters.WARM_UP_INTERVAL) / GRANULARITY) * GRANULARITY
                        + BenchmarkerParameters.WARM_UP_INTERVAL) / 1000;

        if (!latencyHistory.containsKey(t))
                latencyHistory.put(t, new ArrayList<Long>());

        latencyHistory.get(t).add(TimeUnit.NANOSECONDS.toMillis(rs.getTxnResponseTime()));
}
```

(0, [145, 27, 33, …])
(5, [23, 11, 150, …])
(10, [28, 16, 50, …])
…

```
+                        for (Map.Entry<Long, ArrayList<Long>> entry : latencyHistory.entrySet()) {
+                                List<Long> lats = entry.getValue();
+                                long stats[] = processLat(lats);
+                                long tp = lats.size() * 1000 / GRANULARITY;
+
+                                bwrFile.write(String.format("%d, %d, %d, %d, %d, %d, %d, %d\n",
+                                                entry.getKey(), tp, stats[0], stats[1],
+                                                stats[2], stats[3], stats[4], stats[5]));
+                        }
```

```
+       private long[] processLat(List<Long> lats) {
+               long[] stats = new long[6];
+               if (resultSets.size() == 0)
+                       return stats;
+               Collections.sort(lats);
+
+               stats[0] = findAvgLat(lats);
+               stats[1] = findMinLat(lats);
+               stats[2] = findMaxLat(lats);
+               stats[3] = findFirstQuartileLat(lats);
+               stats[4] = findSecondQuartileLat(lats);
+               stats[5] = findThirdQuartileLat(lats);
+
+               return stats;
+       }
```

(0, [145, 27, 33, …])

(5, [23, 11, 150, …])

(10, [28, 16, 50, …])

…