Assignment 2 Solution

Introduction to Database Systems

DataLab

CS, NTHU

Outline

- *UpdateItemPrice* transaction (SP/JDBC implementations)
- *StatisticManager*
- *An example of Experiment Results*

Outline

- *UpdateItemPrice* transaction (SP/JDBC implementations)
- *StatisticManager*
- *An example of Experiment Results*

Modified/Added Classes

- Shared class
  - *As2BenchConstants*
  - *As2BenchTransactionType*
- Client-side classes
  - *As2BenchmarkRte*
  - *As2UpdateItemPriceParamGen*
  - *As2BenchJdbcExecutor*
  - *UpdateItemPriceTxnJdbcJob*
- Server-side classes
  - *As2BenchStoredProcFactory*
  - *UpdateItemPriceProcParamHelper*
  - *UpdateItemPriceTxnProc*

Modified/Added Classes

- Shared class
  - *As2BenchConstants*
  - *As2BenchTransactionType*
- Client-side classes
  - *As2BenchmarkRte*
  - *As2UpdateItemPriceParamGen*
  - *As2BenchJdbcExecutor*
  - *UpdateItemPriceTxnJdbcJob*
- Server-side classes
  - *As2BenchStoredProcFactory*
  - *UpdateItemPriceProcParamHelper*
  - *UpdateItemPriceTxnProc*

READ_WRITE_TX_RATE

```
public class As2BenchConstants {

    public static final int NUM_ITEMS;
    public static final double READ_WRITE_TX_RATE;

    static {
        NUM_ITEMS = BenchProperties.getLoader().getPropertyAsInteger(
                As2BenchConstants.class.getName() + ".NUM_ITEMS", 100000);
        READ_WRITE_TX_RATE = BenchProperties.getLoader().getPropertyAsDouble(
                As2BenchConstants.class.getName() + ".READ_WRITE_TX_RATE", 1.00);
    }

    public static final int MIN_IM = 1;
    public static final int MAX_IM = 10000;
    public static final double MIN_PRICE = 1.00;
    public static final double MAX_PRICE = 100.00;
    public static final int MIN_I_NAME = 14;
    public static final int MAX_I_NAME = 24;
    public static final int MIN_I_DATA = 26;
    public static final int MAX_I_DATA = 50;
    public static final int MONEY_DECIMALS = 2;

}
```

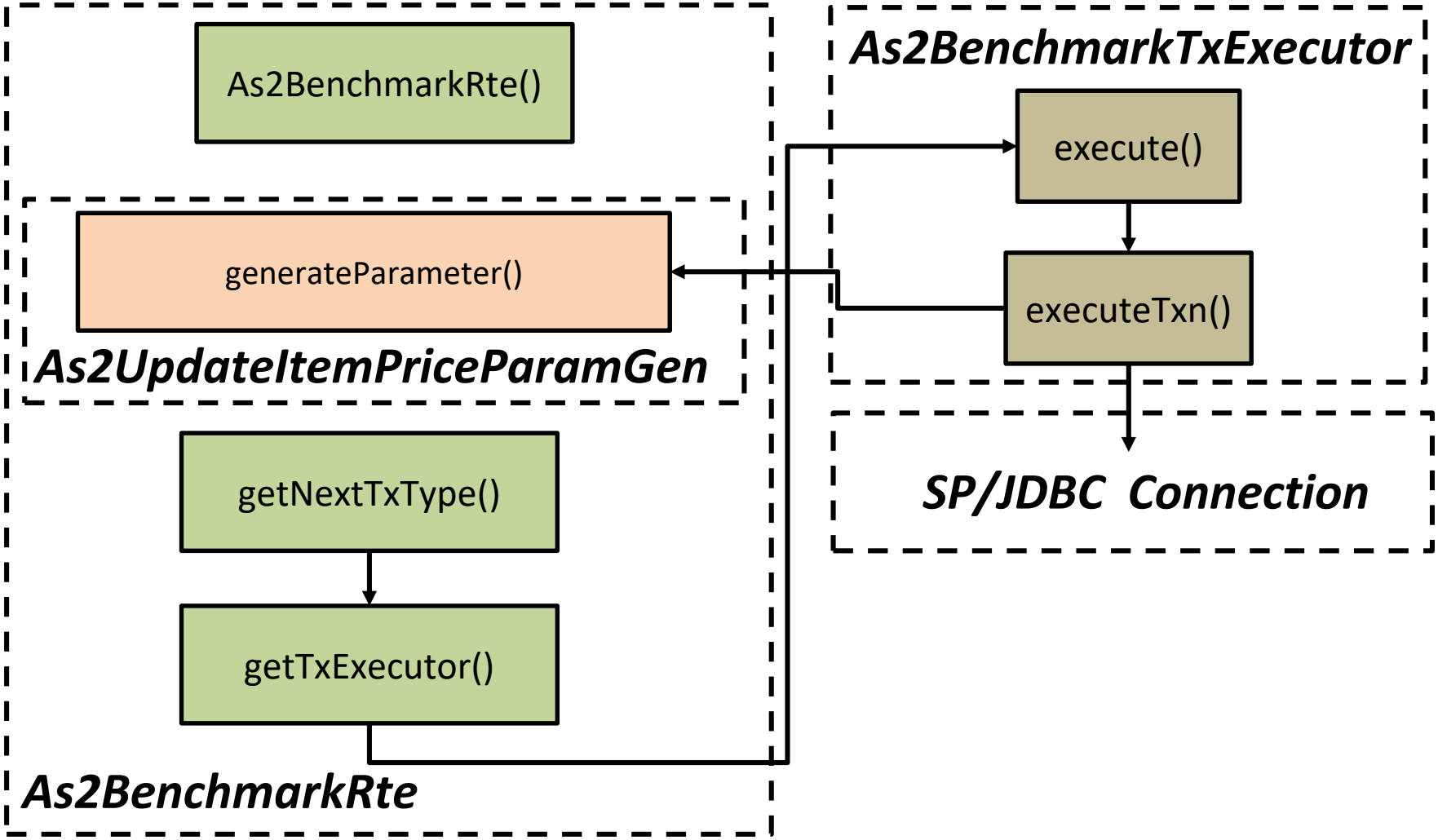
New Transaction Type

```
public enum As2BenchTransactionType implements BenchTransactionType {
    // Loading procedures
    TESTBED_LOADER(false),

    // Database checking procedures
    CHECK_DATABASE(false),

    // Benchmarking procedures
    READ_ITEM(true),

    UPDATE_ITEM_PRICE(true);

    public static As2BenchTransactionType fromProcedureId(int pid) {
        return As2BenchTransactionType.values()[pid];
    }

    private boolean isBenchProc;

    As2BenchTransactionType(boolean isBenchProc) {
        this.isBenchProc = isBenchProc;
    }

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

    @Override
    public boolean isBenchmarkingProcedure() {
        return isBenchProc;
    }
}
```

Modified/Added Classes (Shared)

- Shared class
  - *As2BenchConstants*
  - *As2BenchTransactionType*
- Client-side classes
  - *As2BenchmarkRte*
  - *As2UpdateItemPriceParamGen*
  - *As2BenchJdbcExecutor*
  - *UpdateItemPriceTxnJdbcJob*
- Server-side classes
  - *As2BenchStoredProcFactory*
  - *UpdateItemPriceProcParamHelper*
  - *UpdateItemPriceTxnProc*

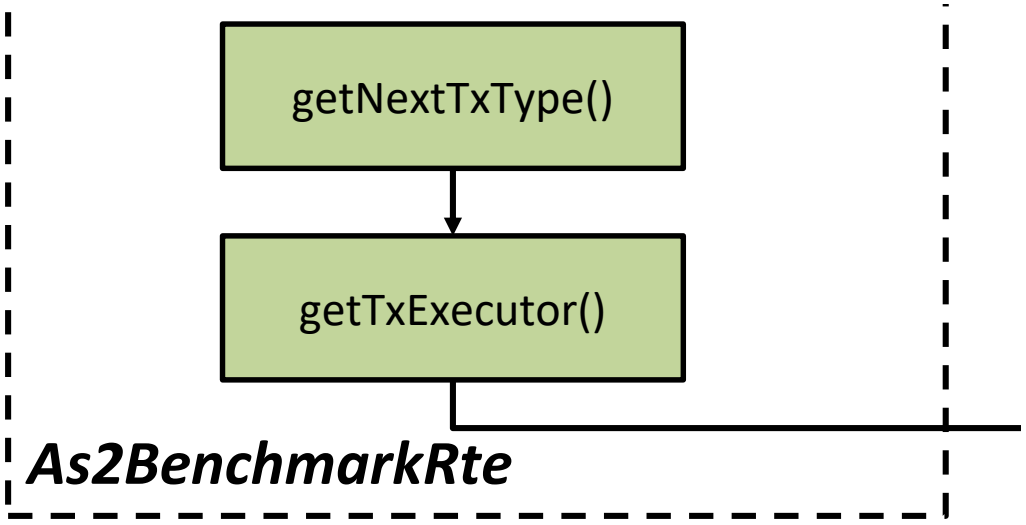
Workflow of As2BenchmarkRte



As2BenchmarkRte

```
public class As2BenchmarkRte extends RemoteTerminalEmulator<As2BenchTransactionType> {

    private As2BenchmarkTxExecutor executor;
    private static final int precision = 100;

    public As2BenchmarkRte(SutConnection conn, StatisticMgr statMgr) {
        super(conn, statMgr);
    }

    protected As2BenchTransactionType getNextTxType() {
        RandomValueGenerator rvg = new RandomValueGenerator();

        // flag would be 100 if READ_WRITE_TX_RATE is 1.0
        int flag = (int) (As2BenchConstants.READ_WRITE_TX_RATE * precision);

        if (rvg.number(0, precision - 1) < flag) {
            return As2BenchTransactionType.READ_ITEM;
        } else {
            return As2BenchTransactionType.UPDATE_ITEM_PRICE;
        }
    }

    protected As2BenchmarkTxExecutor getTxExeutor(As2BenchTransactionType type) {
        TxParamGenerator<As2BenchTransactionType> paraGen;
        switch (type) {
        case READ_ITEM:
            paraGen = new As2ReadItemParamGen();
            break;

        case UPDATE_ITEM_PRICE:
            paraGen = new As2UpdateItemPriceTxnParamGen();
            break;

        default:
            paraGen = new As2ReadItemParamGen();
            break;
        }
        executor = new As2BenchmarkTxExecutor(paraGen);
        return executor;
    }
}
```

Choose a Transaction

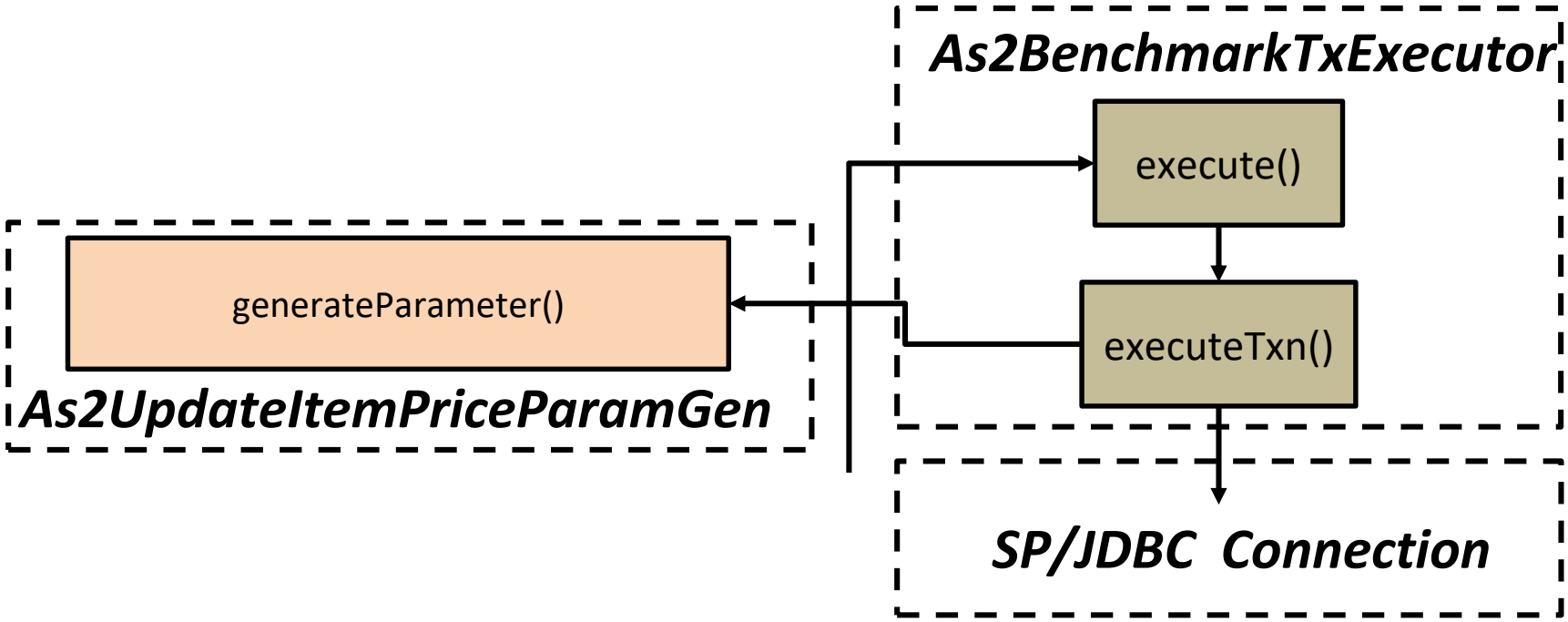


Choose a Transaction

```
protected As2BenchTransactionType getNextTxType() {
    return As2BenchTransactionType.READ_ITEM;
}
```

```
protected As2BenchTransactionType getNextTxType() {
    RandomValueGenerator rvg = new RandomValueGenerator();

    // flag would be 100 if READ_WRITE_TX_RATE is 1.0
    int flag = (int) (As2BenchConstants.READ_WRITE_TX_RATE * precision);

    if (rvg.number(0, precision - 1) < flag) {
        return As2BenchTransactionType.READ_ITEM;
    } else {
        return As2BenchTransactionType.UPDATE_ITEM_PRICE;
    }
}
```

Generate and Send Parameters



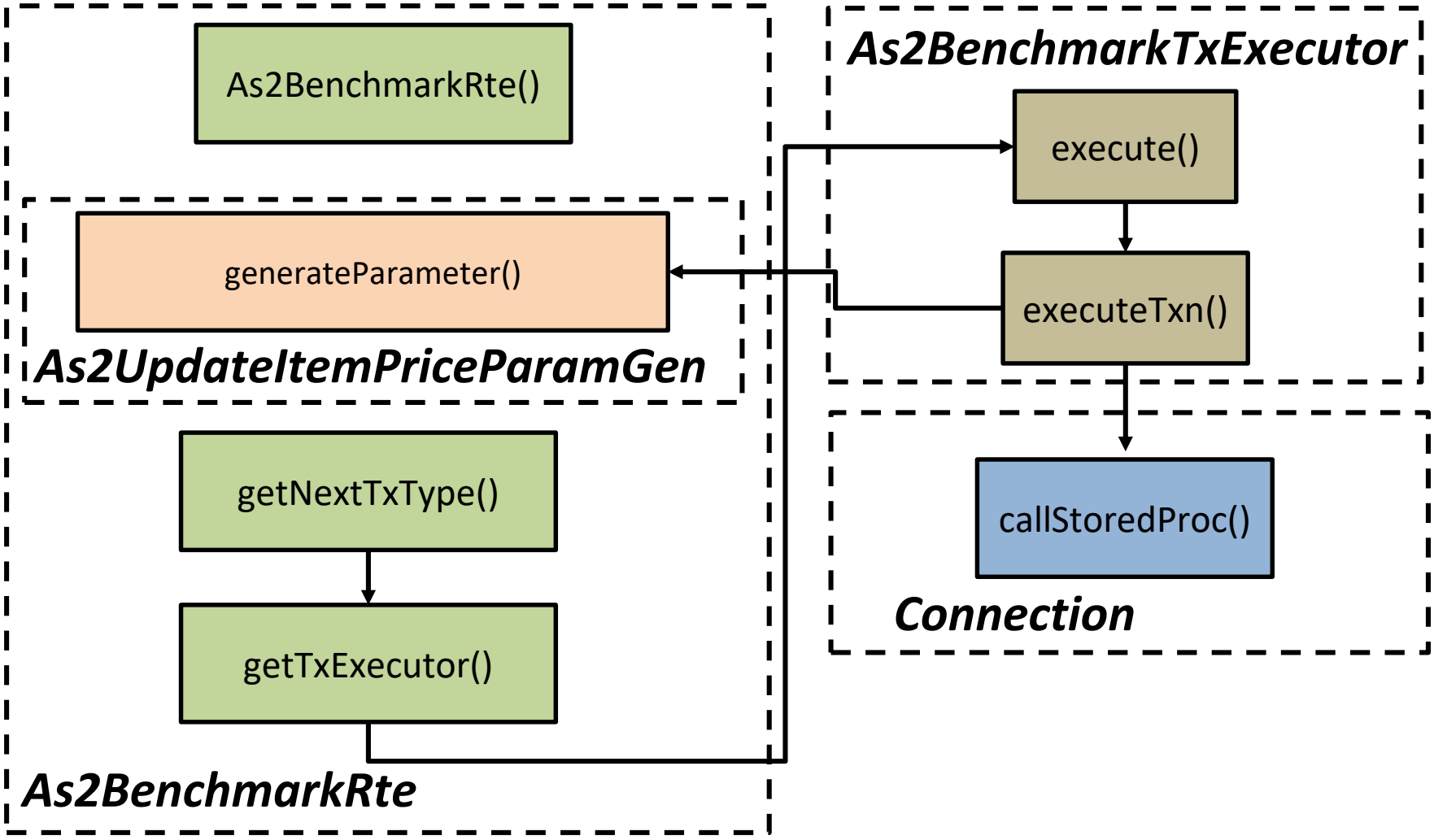
Generate Parameters

```
public class As2UpdateItemPriceTxnParamGen implements TxParamGenerator<As2BenchTransactionType> {
    private static final int WRITE_COUNT = 10;
    private static final int MAX_RAISE = 50;

    @Override
    public As2BenchTransactionType getTxnType() {
        return As2BenchTransactionType.UPDATE_ITEM_PRICE;
    }

    @Override
    public Object[] generateParameter() {
        RandomValueGenerator rvg = new RandomValueGenerator();
        LinkedList<Object> paramList = new LinkedList<Object>();

        paramList.add(WRITE_COUNT);

        for (int i = 0; i < WRITE_COUNT; i++) {
            int itemId = rvg.number(1, As2BenchConstants.NUM_ITEMS);
            double raise = ((double) rvg.number(0, MAX_RAISE)) / 10;

            paramList.add(new UpdateItemPriceTxnParam(itemId, raise));
        }

        return paramList.toArray();
    }
}
```

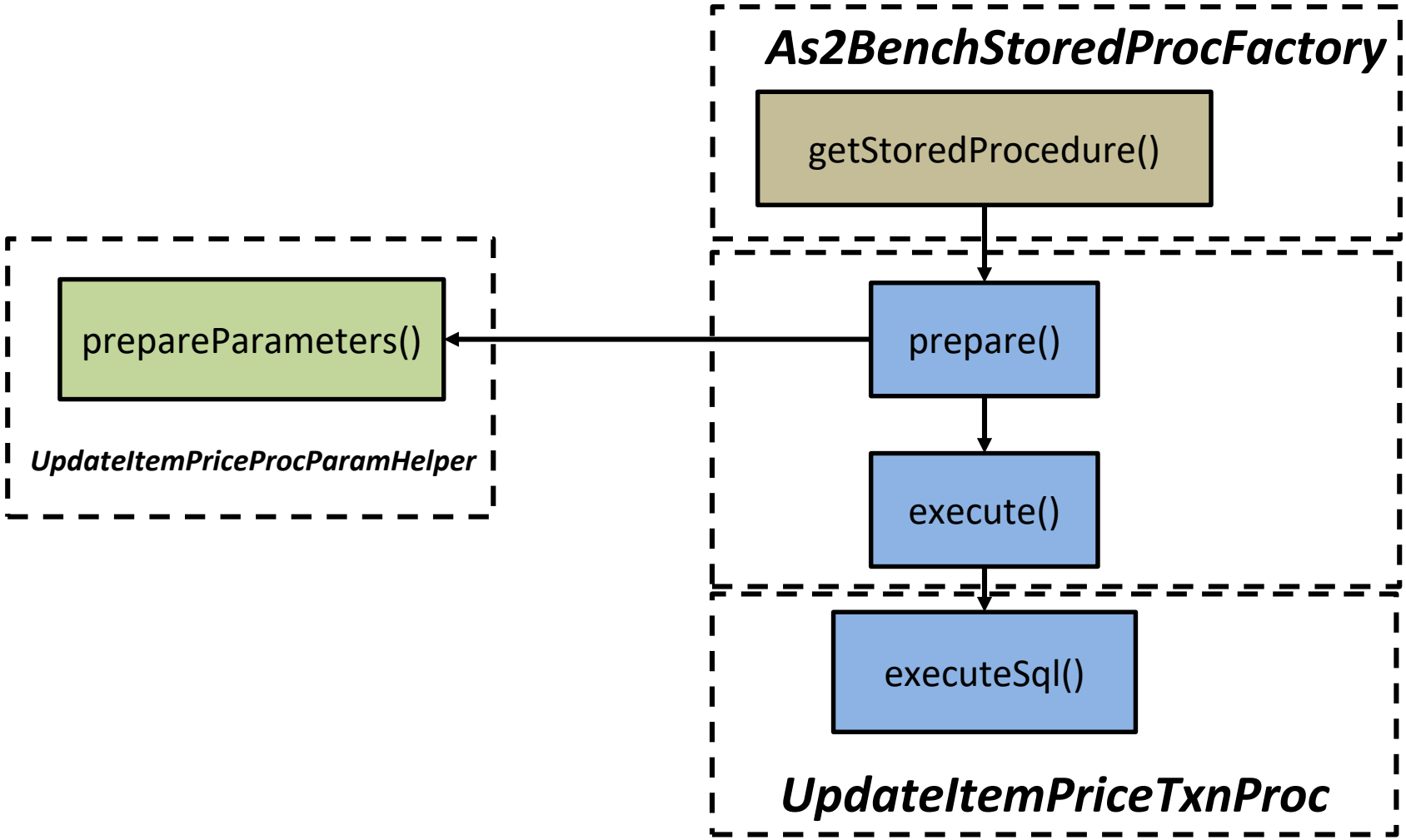
Modified/Added Classes (SP)

- Shared class
  - *As2BenchTxnType*
  - *As2BenchConstants*
- Client-side classes
  - *As2BenchRte*
  - *As2UpdateItemPriceParamGen*
  - *As2BenchJdbcExecutor*
  - *UpdateItemPriceTxnJdbcJob*
- Server-side classes
  - *As2BenchStoredProcFactory*
  - *UpdateItemPriceProcParamHelper*
  - *UpdateItemPriceTxnProc*

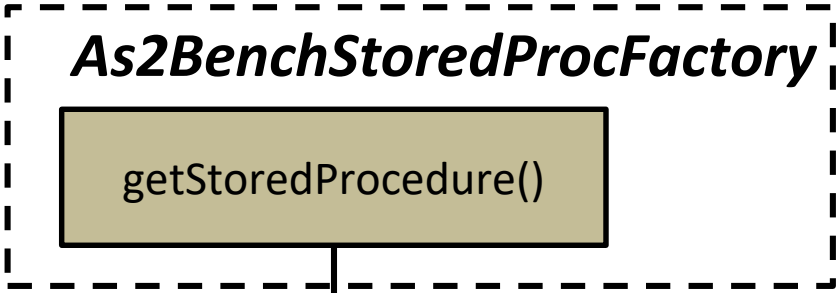
Inquiry via SP



Execute a Stored Procedure



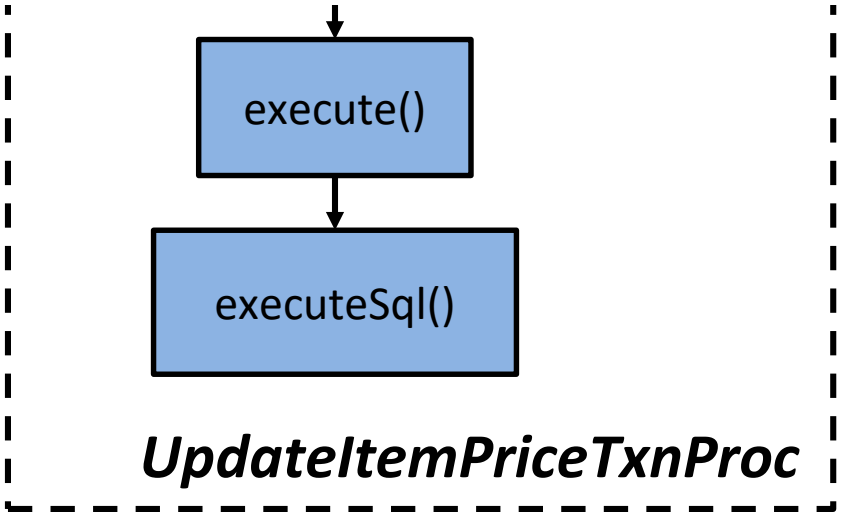
Get the Specified SP



Get the Specified SP

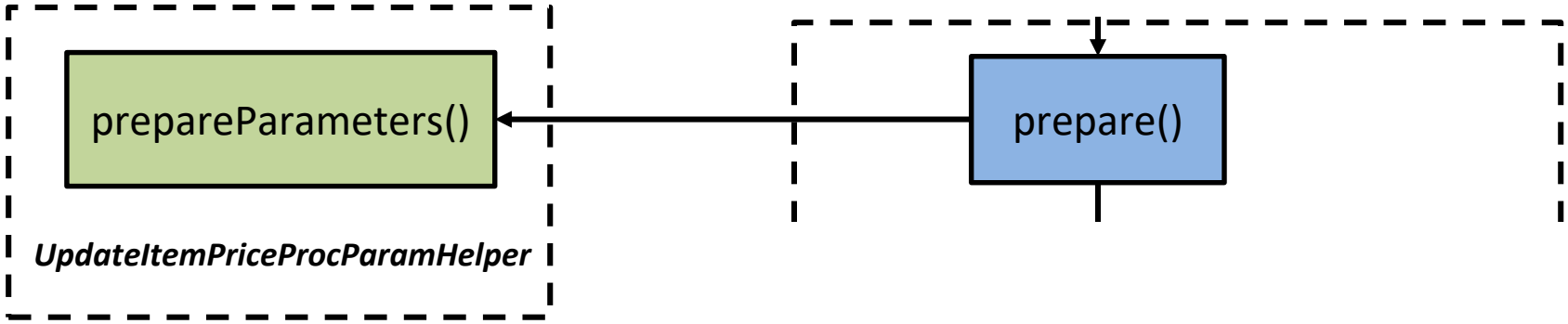
```
public class As2BenchStoredProcFactory implements StoredProcedureFactory {

    @Override
    public StoredProcedure<?> getStroredProcedure(int pid) {
        StoredProcedure<?> sp;
        switch (As2BenchTransactionType.fromProcedureId(pid)) {
        case TESTBED_LOADER:
            sp = new TestbedLoaderProc();
            break;
        case CHECK_DATABASE:
            sp = new As2CheckDatabaseProc();
            break;
        case READ_ITEM:
            sp = new ReadItemTxnProc();
            break;
        case UPDATE_ITEM_PRICE:
            sp = new UpdateItemPriceTxnProc();
            break;
        default:
            throw new UnsupportedOperationException("The benchmarker does not recognize procedure " + pid + "");
        }
        return sp;
    }
}
```

Execute Queries



```java
@Override
protected void executeSql() {
    UpdateItemPriceProcParamHelper paramHelper = getParamHelper();
    Transaction tx = getTransaction();

    for (int idx = 0; idx < paramHelper.getReadCount(); idx++) {
        int iid = paramHelper.getItemId(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);
        } else
            throw new RuntimeException("Cloud not find item record with i_id = " + iid);

        s.close();
        // Update part
        int result = VanillaDb.newPlanner()
                .executeUpdate("UPDATE item SET i_price = " + paramHelper.getUpdatedItemPrice(idx) + " WHERE i_id = " + iid, tx);
        if (result == 0) {
            throw new RuntimeException("Could not update item record with i_id = " + iid);
        }
    }
}
```

Preprocess Parameters



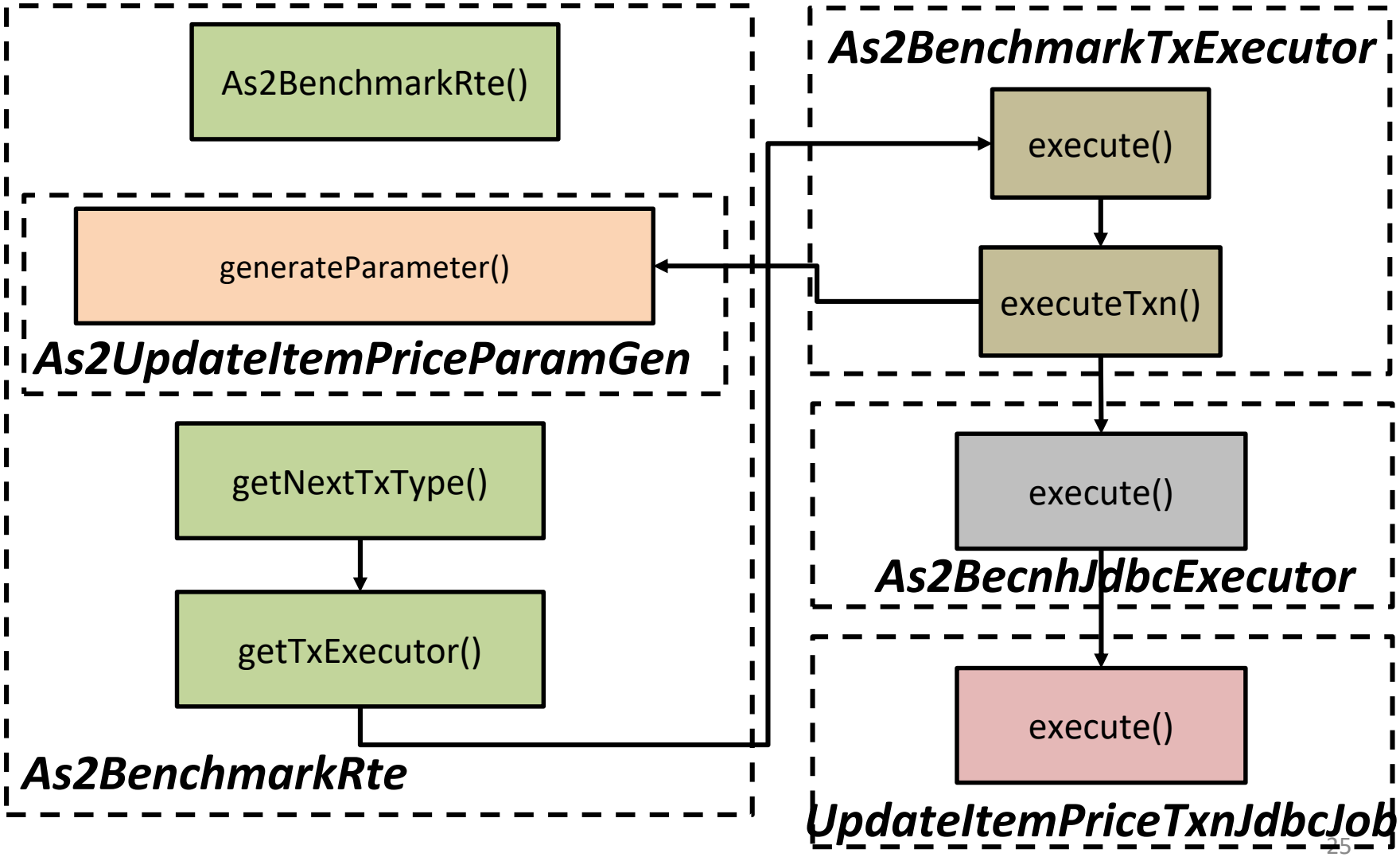
Preprocess Parameters

```java
public double getUpdatedItemPrice(int idx) {
    double updatedPrice = itemPrices[idx] + raises[idx];
    return (Double) (updatedPrice > As2BenchConstants.MAX_PRICE ? As2BenchConstants.MIN_PRICE : updatedPrice);
}

@Override
public void prepareParameters(Object... pars) {

    // Show the contents of paramters
    // System.out.println("Params: " + Arrays.toString(pars));

    int indexCnt = 0;

    readCount = (Integer) pars[indexCnt++];
    itemIds = new int[readCount];
    itemNames = new String[readCount];
    itemPrices = new double[readCount];
    raises = new double[readCount];

    for (int i = 0; i < readCount; i++) {
        itemIds[i] = (Integer) (((UpdateItemPriceTxnParam) pars[indexCnt]).itemId);
        raises[i] = (Double) (((UpdateItemPriceTxnParam) pars[indexCnt]).raise);
        indexCnt++;
    }
}
```

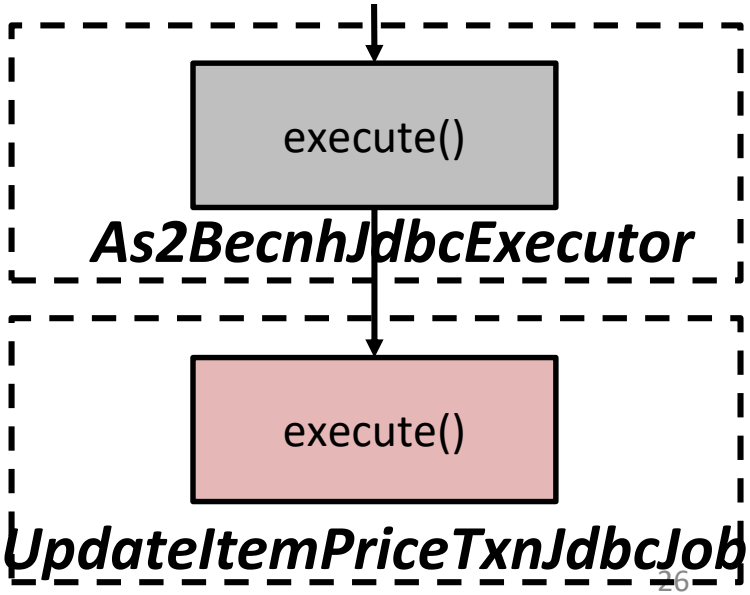
Modified/Added Classes (JDBC)

- Shared class
  - *As2BenchTxnType*
  - *As2BenchConstants*
- Client-side classes
  - *As2BenchRte*
  - *As2UpdateItemPriceParamGen*
  - *As2BenchJdbcExecutor*
  - *UpdateItemPriceTxnJdbcJob*
- Server-side classes
  - *As2BenchStoredProcFactory*
  - *UpdateItemPriceProcParamHelper*
  - *UpdateItemPriceTxnProc*

Inquiry via JDBC



Inquiry via JDBC



Inquiry via JDBC

```
public class As2BenchJdbcExecutor implements JdbcExecutor<As2BenchTransactionType> {

    @Override
    public SutResultSet execute(Connection conn, As2BenchTransactionType txType, Object[] pars)
            throws SQLException {
        switch (txType) {
        case TESTBED_LOADER:
            return new LoadingTestbedJdbcJob().execute(conn, pars);
        case CHECK_DATABASE:
            return new CheckDatabaseJdbcJob().execute(conn, pars);
        case READ_ITEM:
            return new ReadItemTxnJdbcJob().execute(conn, pars);
        case UPDATE_ITEM_PRICE:
            return new UpdateItemPriceTxnJdbcJob().execute(conn, pars);
        default:
            throw new UnsupportedOperationException(
                    String.format("no JDCB implementation for '%s'", txType));
        }
    }
}
```

```java
@Override
public SutResultSet execute(Connection conn, Object[] pars) throws SQLException {
    // Parse parameters
    int readCount = (Integer) pars[0];
    int[] itemIds = new int[readCount];
    double[] raises = new double[readCount];

    for (int i = 0; i < readCount; i++) {
        itemIds[i] = (Integer) (((UpdateItemPriceTxnParam) pars[i + 1]).itemId);
        raises[i] = (Double) (((UpdateItemPriceTxnParam) pars[i + 1]).raise);
    }

Statement statement = conn.createStatement();
ResultSet rs = null;

for (int i = 0; i < 10; i++) {
    double price;

    String sql = "SELECT i_name, i_price 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")));
        price = rs.getDouble("i_price");
    } else
        throw new RuntimeException("cannot find the record with i_id = " + itemIds[i]);
    rs.close();

    Double updatedPrice = updatePrice(price, raises[i]);
    sql = "UPDATE item SET i_price = " + updatedPrice + " WHERE i_id = " + itemIds[i];

    int result = statement.executeUpdate(sql);
    if (result == 0) {
        throw new RuntimeException("cannot update the record with i_id = " + itemIds[i]);
    }

}
conn.commit();

private Double updatePrice(double originalPrice, double raise) {
    return (Double) (originalPrice > As2BenchConstants.MAX_PRICE ? As2BenchConstants.MIN_PRICE : originalPrice + raise);
}
```

Outline

- *UpdateItemPrice* transaction (SP/JDBC implementations)
- *StatisticManager*
- *An example of Experiment Results*

Modified Class

- *StatisticMgr*

```
public synchronized void outputReport() {
    try {
        SimpleDateFormat formatter = new SimpleDateFormat("yyyyMMdd-HHmmss"); // E.g. "20200524-200824"
        String fileName = formatter.format(Calendar.getInstance().getTime());

        if (fileNamePostfix != null && !fileNamePostfix.isEmpty())
            fileName += "-" + fileNamePostfix; // E.g. "20200524-200824-postfix"

        outputDetailReport(fileName + "-detail");

        // output As2 required report
        outputAs2Report(fileName);

    } catch (IOException e) {
        e.printStackTrace();
    }

    if (logger.isLoggable(Level.INFO))
        logger.info("Finnish creating tpcc benchmark report");
}
```

Add Method

```
private void addTxnLatency(TxnResultSet rs) {
    long elapsedTime = TimeUnit.NANOSECONDS.toSeconds(rs.getTxnEndTime() - recordStartTime);
    long timeSlotBoundary = (elapsedTime / granularity) * granularity ;

    ArrayList<Long> timeSlot = latencyHistory.get(timeSlotBoundary );
    if (timeSlot == null) {
        timeSlot = new ArrayList<Long>();
        latencyHistory.put(timeSlotBoundary, timeSlot);
    }
    timeSlot.add(TimeUnit.NANOSECONDS.toMillis(rs.getTxnResponseTime()));
}
```

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

```
private void outputAs2Report(String fileName) throws IOException {
    try (BufferedWriter writer = new BufferedWriter(new FileWriter(new File(OUTPUT_DIR, fileName + ".csv")))) {
        writer.write(
                "time(sec), throughput(txs), avg_latency(ms), min(ms), max(ms), 25th_lat(ms), median_lat(ms), 75th_lat(ms)");
        writer.newLine();

        int timeAdvance = granularity;
        for (long timeBound = 0, outCount = 0; outCount < latencyHistory.size(); timeBound += timeAdvance) {
            List<Long> slot = latencyHistory.get(timeBound);
            if (slot != null) {
                writer.write(makeStatString(timeBound, slot));
                outCount++;
            } else
                writer.write(String.format("%d, 0, NaN, NaN, NaN, NaN, NaN, NaN", timeBound));
            writer.newLine();
        }
    }
}
```

```java
private String makeStatString(long timeSlotBoundary, List<Long> timeSlot) {
    Collections.sort(timeSlot);

    // Transfer it to unmodifiable in order to prevent modification
    // when we use a sublist to access it.
    timeSlot = Collections.unmodifiableList(timeSlot);

    int count = timeSlot.size();
    int middleOffset = timeSlot.size() / 2;
    long lowerQ, upperQ, median;
    double mean;

    median = calcMedian(timeSlot);
    mean = calcMean(timeSlot);

    if (count < 2) { // Boundary case: there is only one number in the list
        lowerQ = median;
        upperQ = median;
    } else if (count % 2 == 0) { // Even
        lowerQ = calcMedian(timeSlot.subList(0, middleOffset));
        upperQ = calcMedian(timeSlot.subList(middleOffset, count));
    } else { // Odd
        lowerQ = calcMedian(timeSlot.subList(0, middleOffset));
        upperQ = calcMedian(timeSlot.subList(middleOffset + 1, count));
    }
    Long min = Collections.min(timeSlot);
    Long max = Collections.max(timeSlot);

    return String.format("%d, %d, %f, %d, %d, %d, %d, %d",
            timeSlotBoundary, count, mean, min, max, lowerQ, median, upperQ);
}
```

Outline

- *UpdateItemPrice* transaction (SP/JDBC implementations)
- *StatisticManager*
- *An example of Experiment Results*

An Example of Experiments