Assignment 5 Solution

Introduction to Databases

DataLab

CS, NTHU

Outline

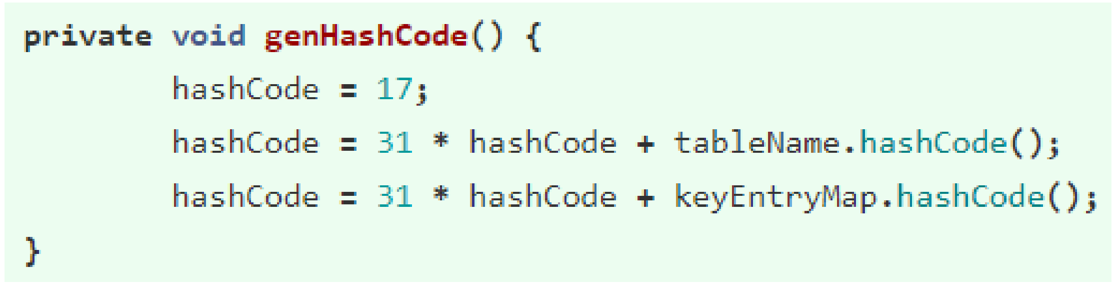
- Solution
  - PrimaryKey
  - StoreProcedure
  - ConservativeConcurrencyMgr
  - ConservativeLockTable
- Chanllenge on TPC-C
- Questions

Outline

- Solution
  - PrimaryKey
  - StoreProcedure
  - ConservativeConcurrencyMgr
  - ConservativeLockTable
- Chanllenge on TPC-C
- Questions

Solution

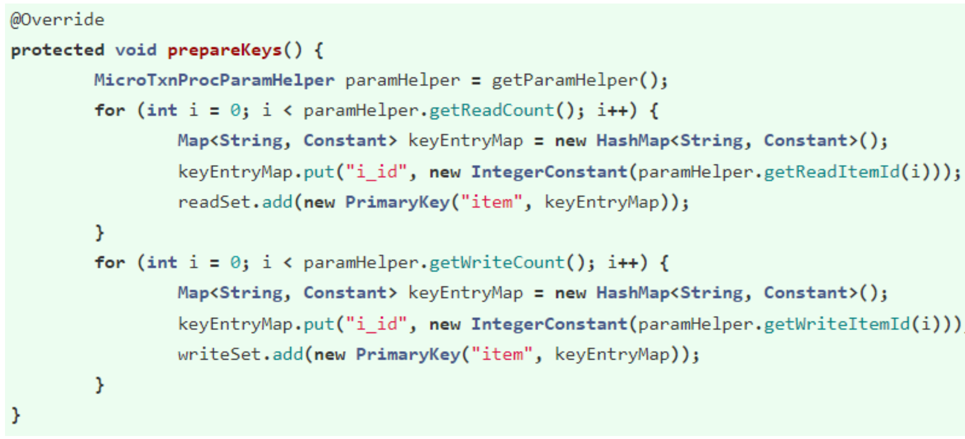
- PrimaryKey
  - An object as a lock in `ConservativeLockTable`
  - Hash tablename and keyentrymap into a hashcode to represent this object



```java
private void genHashCode() {
        hashCode = 17;
        hashCode = 31 * hashCode + tableName.hashCode();
        hashCode = 31 * hashCode + keyEntryMap.hashCode();
}
```

Solution

- StoreProcedure
  - Abstract function `prepareKeys()`
    - prepare Read Write set of txn
    - `E.g. MicroTxnProc`



```
@Override
protected void prepareKeys() {
        MicroTxnProcParamHelper paramHelper = getParamHelper();
        for (int i = 0; i < paramHelper.getReadCount(); i++) {
                Map<String, Constant> keyEntryMap = new HashMap<String, Constant>();
                keyEntryMap.put("i_id", new IntegerConstant(paramHelper.getReadItemId(i)));
                readSet.add(new PrimaryKey("item", keyEntryMap));
        }
        for (int i = 0; i < paramHelper.getWriteCount(); i++) {
                Map<String, Constant> keyEntryMap = new HashMap<String, Constant>();
                keyEntryMap.put("i_id", new IntegerConstant(paramHelper.getWriteItemId(i)))
                writeSet.add(new PrimaryKey("item", keyEntryMap));
        }
}
```

  - `scheduleTransactionSerially()`
    - Deterministic ordering

Solution

- ConservativeConcurrencyMgr
  - bookReadKey/ bookWriteKey
  - acquireBookLocks
  - releaseLocks

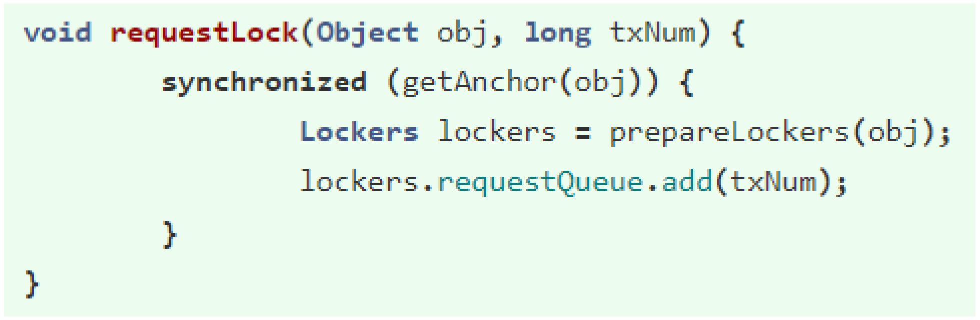
```java
public void bookReadKey(PrimaryKey key) {
        if (key != null) {
                // The key needs to be booked only once.
                if (!bookedObjs.contains(key))
                        lockTbl.requestLock(key, txNum);

                bookedObjs.add(key);
                readObjs.add(key);
        }
}
```

```java
public void acquireBookedLocks() {
        bookedObjs.clear();

        for (Object obj : writeObjs)
                lockTbl.xLock(obj, txNum);

        for (Object obj : readObjs)
                if (!writeObjs.contains(obj))
                        lockTbl.sLock(obj, txNum);
}
```

```java
private void releaseLocks() {
        for (Object obj : writeObjs)
            lockTbl.release(obj, txNum, LockType.X_LOCK);

        for (Object obj : readObjs)
            if (!writeObjs.contains(obj))
                lockTbl.release(obj, txNum, LockType.S_LOCK);

        readObjs.clear();
        writeObjs.clear();
}
```

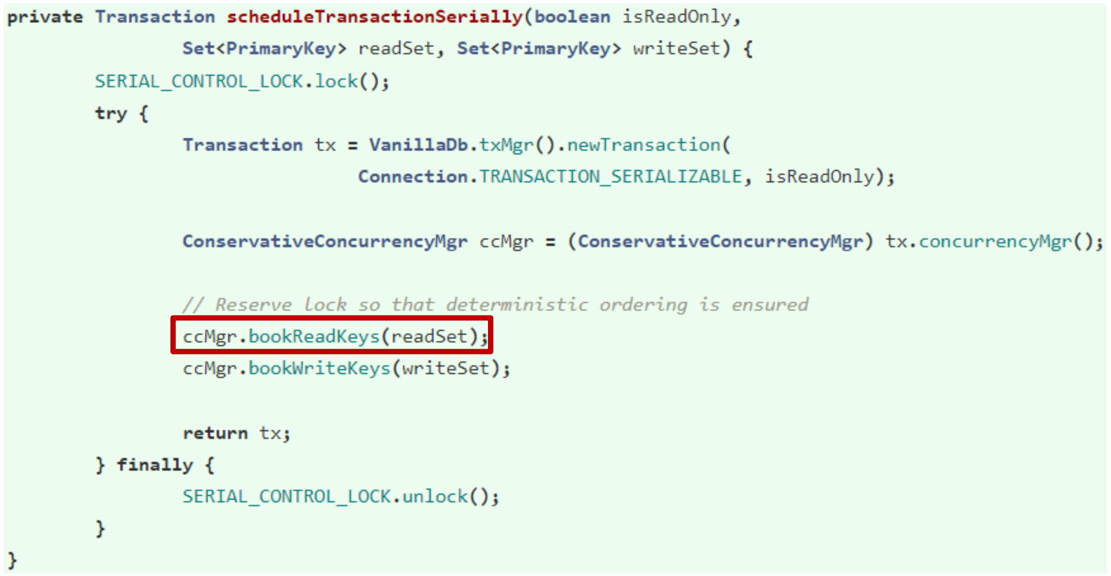
Solution

- ConservativeLockTable
  - requestQueue maintain deterministic property

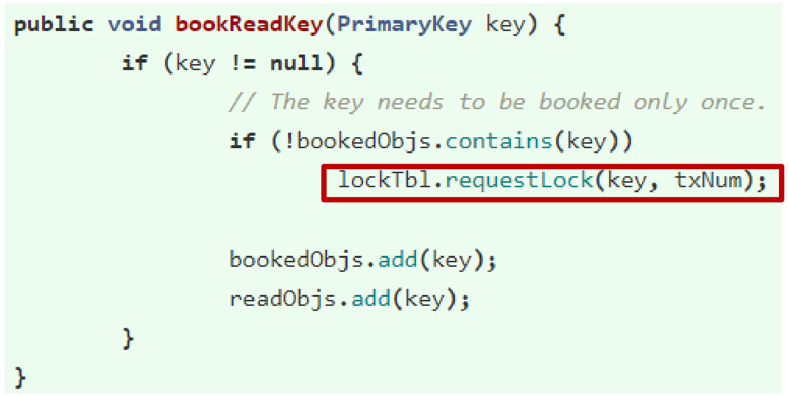


```
void requestLock(Object obj, long txNum) {
        synchronized (getAnchor(obj)) {
                Lockers lockers = prepareLockers(obj);
                lockers.requestQueue.add(txNum);
        }
}
```

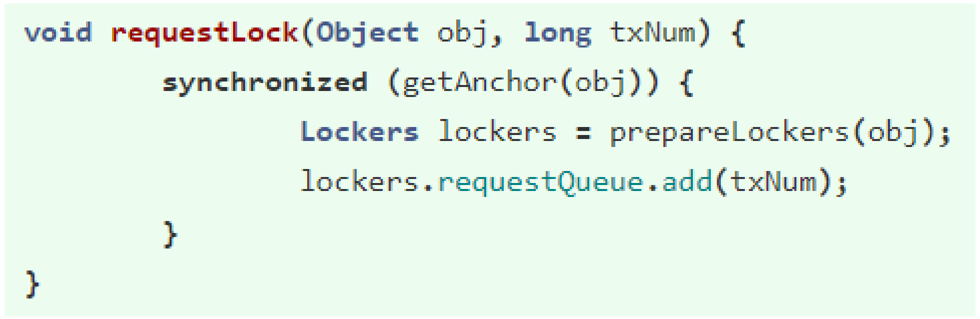
Deterministic Order



```java
private Transaction scheduleTransactionSerially(boolean isReadOnly,
                Set<PrimaryKey> readSet, Set<PrimaryKey> writeSet) {
        SERIAL_CONTROL_LOCK.lock();
        try {
                Transaction tx = VanillaDb.txMgr().newTransaction(
                                Connection.TRANSACTION_SERIALIZABLE, isReadOnly);

                ConservativeConcurrencyMgr ccMgr = (ConservativeConcurrencyMgr) tx.concurrencyMgr();

                // Reserve lock so that deterministic ordering is ensured
                ccMgr.bookReadKeys(readSet);
                ccMgr.bookWriteKeys(writeSet);

                return tx;
        } finally {
                SERIAL_CONTROL_LOCK.unlock();
        }
}
```



```java
public void bookReadKey(PrimaryKey key) {
        if (key != null) {
                // The key needs to be booked only once.
                if (!bookedObjs.contains(key))
                        lockTbl.requestLock(key, txNum);

                bookedObjs.add(key);
                readObjs.add(key);
        }
}
```

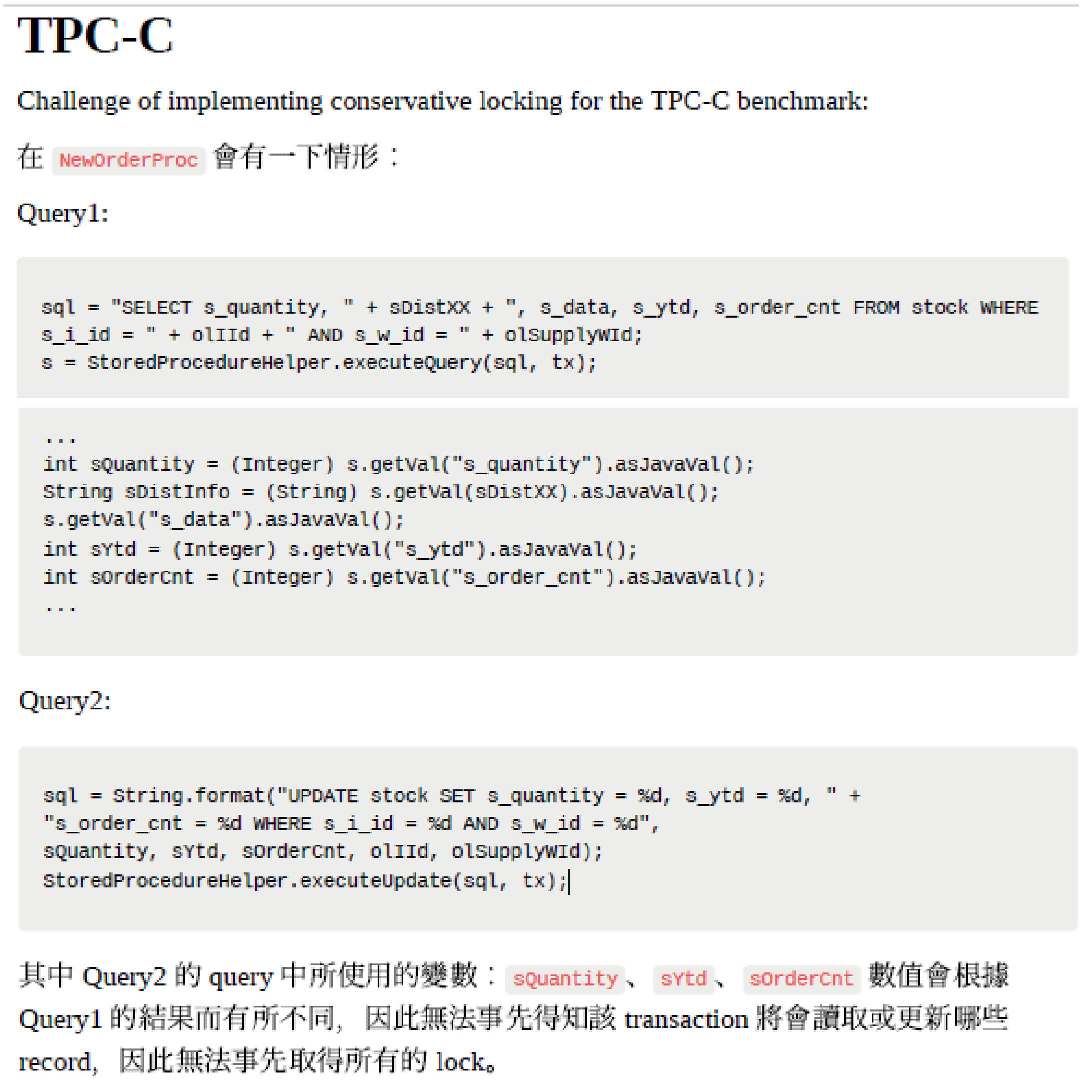


```java
void requestLock(Object obj, long txNum) {
        synchronized (getAnchor(obj)) {
                Lockers lockers = prepareLockers(obj);
                lockers.requestQueue.add(txNum);
        }
}
```

Outline

- Solution
  - PrimaryKey
  - StoreProcedure
  - ConservativeConcurrencyMgr
  - ConservativeLockTable
- Chanllenge on TPC-C
- Questions

Challenge of TPC-C



TPC-C

Challenge of implementing conservative locking for the TPC-C benchmark:

在 `NewOrderProc` 會有一下情形：

Query1:

```
sql = "SELECT s_quantity, " + sDistXX + ", s_data, s_ytd, s_order_cnt FROM stock WHERE
s_i_id = " + olIId + " AND s_w_id = " + olSupplyWId;
s = StoredProcedureHelper.executeQuery(sql, tx);
```

```
...
int sQuantity = (Integer) s.getVal("s_quantity").asJavaVal();
String sDistInfo = (String) s.getVal(sDistXX).asJavaVal();
s.getVal("s_data").asJavaVal();
int sYtd = (Integer) s.getVal("s_ytd").asJavaVal();
int sOrderCnt = (Integer) s.getVal("s_order_cnt").asJavaVal();
...
```

Query2:

```
sql = String.format("UPDATE stock SET s_quantity = %d, s_ytd = %d, " +
"s_order_cnt = %d WHERE s_i_id = %d AND s_w_id = %d",
sQuantity, sYtd, sOrderCnt, olIId, olSupplyWId);
StoredProcedureHelper.executeUpdate(sql, tx);
```

其中 Query2 的 query 中所使用的變數：`sQuantity`、 `sYtd`、 `sOrderCnt` 數值會根據 Query1 的結果而有所不同，因此無法事先得知該 transaction 將會讀取或更新哪些 record，因此無法事先取得所有的 lock。