Database - Bug #11831

Persistence context creation timing depends on metadata configuration (MetaLock / MetadataSecurityOps)

09/04/2026 07:47 AM - Alexandru Lungu

Status:	New	**Start date:**	
Priority:	Normal	**Due date:**	
Assignee:		**% Done:**	0%
Category:		**Estimated time:**	0.00 hour
Target version:			
billable:	No	**reviewer:**	
vendor_id:	GCD	**production:**	No
case_num:		**env_name:**	
version_reported:		**topics:**	
version_resolved:			

Description

History

#1 - 09/04/2026 08:19 AM - Alexandru Lungu

- Subject changed from Eager persistence context creation for unused databases to Persistence context creation timing depends on metadata configuration (MetaLock / MetadataSecurityOps)

While working on #8388 / #11771 I found that the moment a database's Persistence.Context is created is not determined by that database at all, but by whether the application happens to have certain **metadata** artifacts. This is invisible in normal operation and it silently decides which bugs are reachable in a given project. I hit it while debugging regressions on the ChUI suite, which **has no meta tables**.

Two separate layers

These have different gates and it is worth keeping them apart.

1. The META context is gated on metadata being configured at all:

```
DatabaseManager.java:1696    useMeta = MetadataManager.initialize()
DatabaseManager.java:2043      if (useMeta) activateAndRegister(sessionId, metaDB, () -> activateMetaDb(...))
DatabaseManager.java:2112        initMetaDb -> MetadataManager.populateDatabase(metaDB)
MetadataManager.java:680-681       PersistenceFactory.getInstance(metaDb)
                                   p.beginTransaction(Persistence.META_CTX)   --> META context
```

2. The application (PRIMARY) context -- the one that actually matters here -- is gated on one of two things, both of which sit *inside* if (useMeta).

MetaLock on the classpath, to set up shareLockTableUpdater:

```
DatabaseManager.java:1700-1714   Class.forName("<dmoPkg>._meta.MetaLock") -> lockTableUpdaters = new HashMap<>
()
DatabaseManager.java:1987-1991   if (lockTableUpdaters != null) mPrim.shareLockTableUpdater(primaryDB)
Persistence.java:4777              tenant.lockTableUpdater = pPrim.getContext(...)  --> PRIMARY context
```

userSecurityOpsClass = com.goldencode.p2j.util.MetadataSecurityOps, to resolve the default userid at connect:

```
ConnectionManager.java:4290-4295   putConnected -> SecurityOps.getDefaultUserid(ldbName)
SecurityOps.java:506-508             customSecurityOps.getAuthLevel(ldbname)
MetadataSecurityOps.java:321-323       -> ConnectionManager.getAuthLevel
ConnectionManager.java:2287-2292         getPersistence(ldbname).getSingleSQLResult(query, META_CTX, null)
```

```
Persistence.java:3113                    Context local = getContext(sharedDb)   --> PRIMARY context
```

So: with metadata configured but **neither** MetaLock **nor** MetadataSecurityOps, you get an eager META context and a **lazy** PRIMARY one. The default SecurityManagerSecurityOps.getAuthLevel (SecurityManagerSecurityOps.java:167-170) just returns FREE_ACCESS and touches no database, so the switch is the **directory setting**, not the presence of _User as such.

Two further notes: this is per **user session**, since activateAndRegister is keyed on the session id and short-circuits on connected.isActivated() (DatabaseManager.java:2063-2081); and it applies to **connected** databases only -- one that is merely defined in directory.xml but neither auto-connected nor connected at runtime gets nothing.

Why this severely affects testing

A project either pre-builds every connected database's context at session start, or builds each one lazily on first use. Those are two genuinely different runtime orderings, and which one you get is decided by metadata configuration that has nothing to do with the code under test.

A stock testcases project has **both** drivers: a <metadata> block including _Lock, plus userSecurityOpsClass = MetadataSecurityOps. It is therefore **structurally incapable** of exercising the lazy ordering. The ChUI suite, having no meta tables, is currently the only place that ordering ever runs.

This is exactly why #8388's record-nursery underflow survived the whole regression suite while crashing a customer application on a routine report save. Several candidate reproducers were written and all passed -- not because the shapes were wrong, but because in that project the fault could not occur at all.

How #8388 was affected

TxWrapper.WorkArea.beginTx (TxWrapper.java:1262-1300) builds a TxWrapper for every **connected** database (ConnectionManager.getActiveDatabases, :3002-3022) -- used or not. The constructor obtains the database's EmbeddedTransactionalChangesManager (TxWrapper.java:170) but **not** its Persistence.Context, which waits for activate() (:463). registerTransactionalChangesManagers (:432-438) then pushes a transaction scope on that manager while it still has no listeners. The nursery -- and its two RecordNurseryUndoLog listeners -- is only built later, by activate() -> persistence.getContext(...) -> Persistence.Context.initialize (Persistence.java:5584). Registered late against an already-scoped manager, an undo log then received an endTx for a scope it never saw the beginTx for, and died in ScopedList.popScope with IndexOutOfBoundsException: Index -1 out of bounds for length 0.

Worth stating precisely: this needs **one** database, not two. A transaction opens, its own database's manager gets a scope pushed because it is connected, and the first use of one of its tables inside that transaction creates the context. It only **looked** like a second-database problem in the field.

The seam itself is now fixed on 8388a -- TransientDatabaseManager.addListener primes a late listener with one beginTx per already-open scope -- so this class of bug no longer depends on creation order. **This ticket should not be closed by making contexts eager**: that would hide the common case and leave the tenant path (below) untouched.

What to decide

1. **Pre-create the context eagerly for every connected database**, so that metadata stops driving this inconsistently and non-deterministically. I am against it, and not only because of redundant DB connections: the invariant is unachievable. Persistence.tenantChanged builds a fresh nursery mid-flight by design -- it creates the private tenant context (Persistence.java:5292) and re-initializes an existing one (:5321), and initialize's own javadoc says it runs "when the tenant is changed". So "a persistence context is always bootstrapped at the start of a session" can never be true while multi-tenancy exists, and relying on eagerness for correctness leaves precisely the least-tested path broken.
2. **Defer the meta work and the persistence context creation until first use.** This matches the existing lazy design.
    1. shareLockTableUpdater: defer until the first time it is actually needed. This way, if a database is never used in the user session, we never bootstrap it and never create a persistence context for it.
    2. Auth level: the check genuinely must happen at connect time -- it decides the connect userid -- so **when** it runs cannot be deferred. But it does not need a Persistence.Context, only a query. The rest of the connect path already avoids the context by using a raw Session (reportUDFVersion, verifySettings, MetadataManager.prepareDatabase, setupIdentityManager). Switching ConnectionManager.getAuthLevel from getPersistence(ldbname).getSingleSQLResult(...) to a raw session would remove the eager context without changing when the check happens.