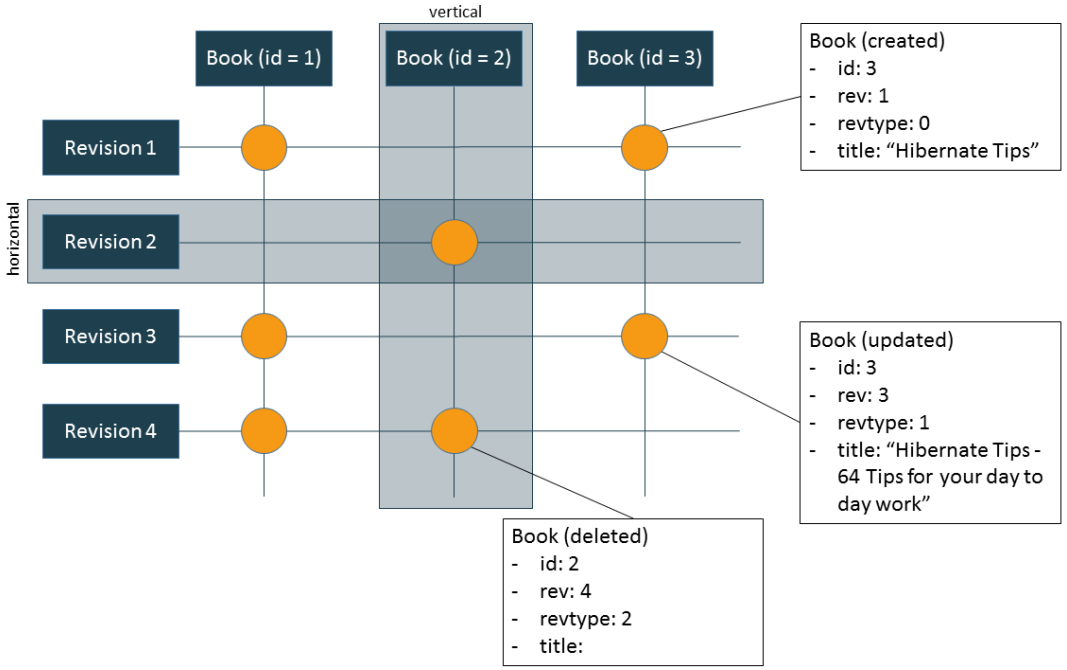
The 2 dimensions of audit information

Hibernate Envers creates a new revision for each transaction that creates, updates or deletes an audited entity and stores it in the database. That adds a 2nd dimension to your data structure as you can see in the following graphic.



You can use 2 different perspectives when you look at your audit log. The vertical perspective looks at an entity instance and shows you in which revisions it was created, edited or deleted. The horizontal perspective looks at a revision and shows you the information stored in your database at that point in time.

Create a vertical query

Vertical queries allow you to retrieve the revisions in which an entity instance was created, edited or deleted. You can create such a query by calling the *forRevisionsOfEntity(Class c, boolean selectedEntitiesOnly, boolean selectDeletedEntities)* method.

The *selectedEntitiesOnly* parameter defines if you want to retrieve a list of entities which changed at the selected revisions or if you want to retrieve a list of arrays with an instance of the affected entity, a revision entity, and a *RevisionType* enum. This parameter has no effect when you define a projection for your query.

The effect of the *selectDeletedEntities* is pretty obvious. When you set it to *true*, your query will also return deleted entities. All attributes of a deleted entity except its *id* are *null*.

The following code snippt returns the number of the first revision in which the Book entity with a given id had the title “Hibernate Tips – 64 Tips for your day to day work”.

```
AuditQuery q =
auditReader.createQuery().forRevisionsOfEntity(Book.class,
false, true);

q.addProjection(AuditEntity.revisionNumber().min());

q.add(AuditEntity.id().eq(b.getId()));

q.add(AuditEntity.property(“title”).eq(“Hibernate Tips – 64
Tips for your day to day work”));

Number revision = (Number) q.getSingleResult();
```

Create a horizontal query

You define horizontal queries in a similar way as the vertical queries I showed you before. You just need to call the *forEntitiesAtRevision* method instead of the *forRevisionsOfEntity* when you create the AuditQuery.

The following code snippet shows an *AuditQuery* which returns all *Book* entities in *revision* 2 which *title* contained “JPA” or “Hibernate” and which were published by a *Publisher* which *name* contained “Manning”.

```
AuditQuery q =
auditReader.createQuery().forEntitiesAtRevision(Book.class,
2);

q.traverseRelation(“publisher”, JoinType.LEFT, “p”);

q.add(AuditEntity.and(

AuditEntity.or(AuditEntity.property(“title”).ilike(“JPA”,
MatchMode.ANYWHERE),
AuditEntity.property(“title”).ilike(“Hibernate”,
MatchMode.ANYWHERE)),

AuditEntity.property(“p”, “name”).ilike(“Manning”)));

q.addOrder(AuditEntity.property(“title”).asc());

List<Book> audit = q.getResultList();
```

Get active revision at a given date

If you just want to get an entity that was active at a given time, you can call the find method of the AuditReader and provide a java.util.Date instead of a revision number. You can see an example of it in the following code snippet.

```
AuditReader auditReader = AuditReaderFactory.get(em);


Book auditedBook = auditReader
        .find(Book.class, b.getId(), created);
log.info("Book ["+auditedBook+"] at ["+created+"].");
```