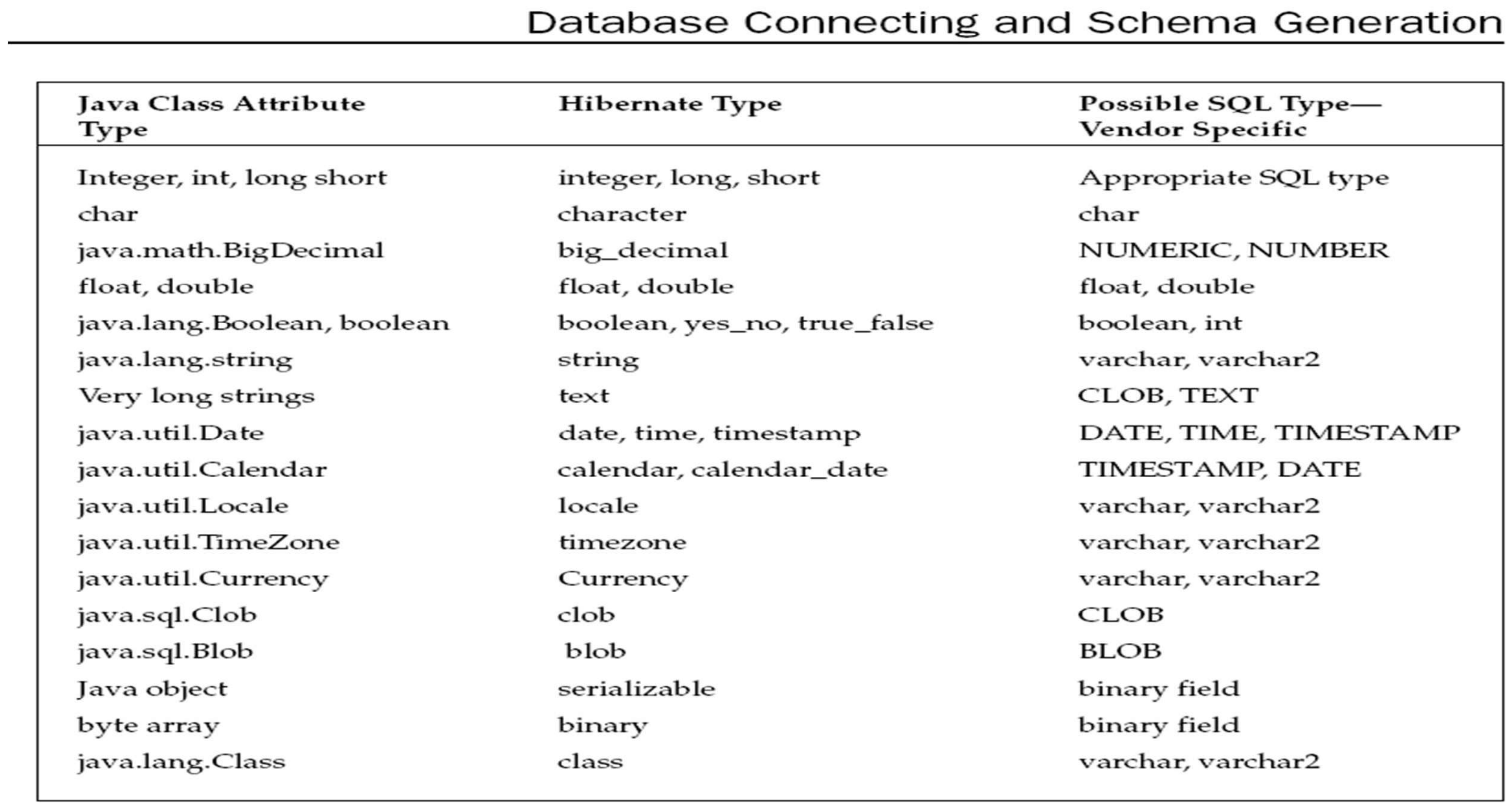
3

Object Lifecycle Persistence and Session Management

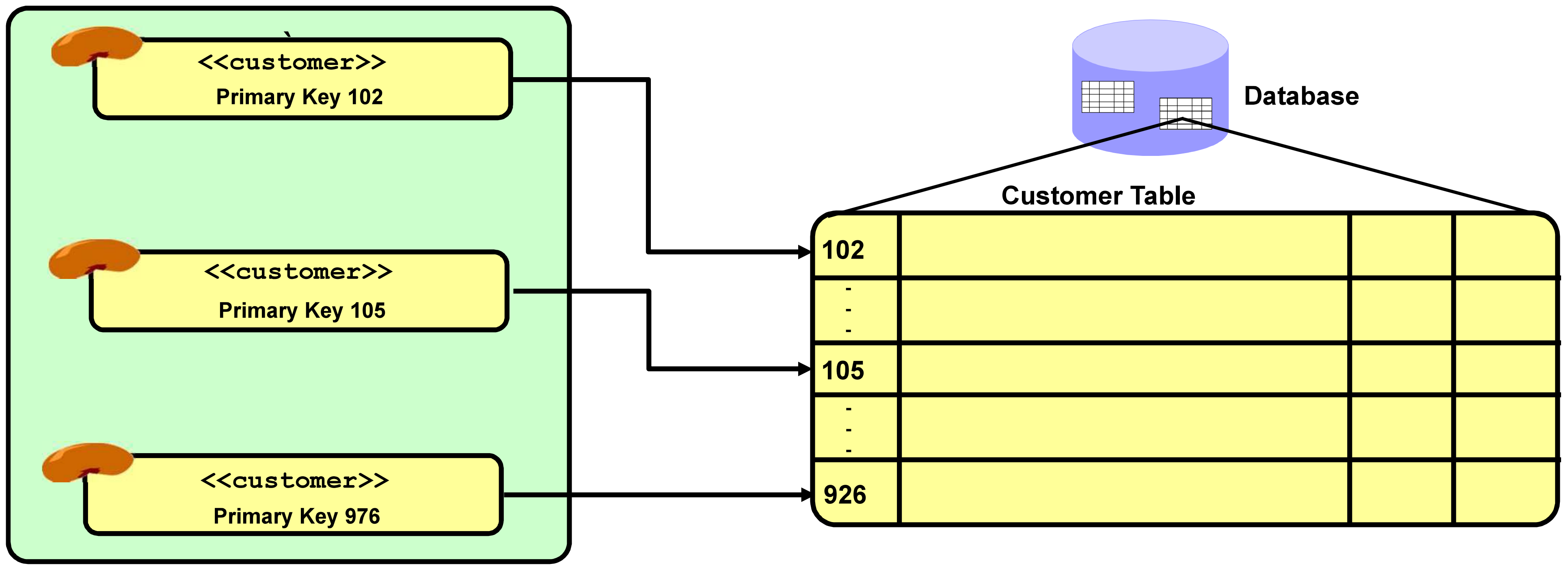
Objectives



Database Connecting and Schema Generation

Java Class Attribute Type	Hibernate Type	Possible SQL Type—Vendor Specific
Integer, int, long short	integer, long, short	Appropriate SQL type
char	character	char
java.math.BigDecimal	big_decimal	NUMERIC, NUMBER
float, double	float, double	float, double
java.lang.Boolean, boolean	boolean, yes_no, true_false	boolean, int
java.lang.string	string	varchar, varchar2
Very long strings	text	CLOB, TEXT
java.util.Date	date, time, timestamp	DATE, TIME, TIMESTAMP
java.util.Calendar	calendar, calendar_date	TIMESTAMP, DATE
java.util.Locale	locale	varchar, varchar2
java.util.TimeZone	timezone	varchar, varchar2
java.util.Currency	Currency	varchar, varchar2
java.sql.Clob	clob	CLOB
java.sql.Blob	blob	BLOB
Java object	serializable	binary field
byte array	binary	binary field
java.lang.Class	class	varchar, varchar2

Entity Component Primary Key Association



Hibernate Object Lifecycle

- **Hibernate considers objects it can manage to always be in one of four states**
  - Transient
  - Persistent
  - Removed
  - Detached
- **Objects transition from state to state through various method calls**

Transient State

- **All objects start off in the transient state**
  - Account account = new Account();
    - account is a transient object
- **Hibernate is not aware of the object instance**
- **Not related to database row**
  - No value for accountId
- **Garbage collected when no longer referenced by any other objects**

Persistent State

Hibernate is aware of, and managing, the object

Has a database id

- Already existing object retrieved from the database
- Formerly transient object about to be saved

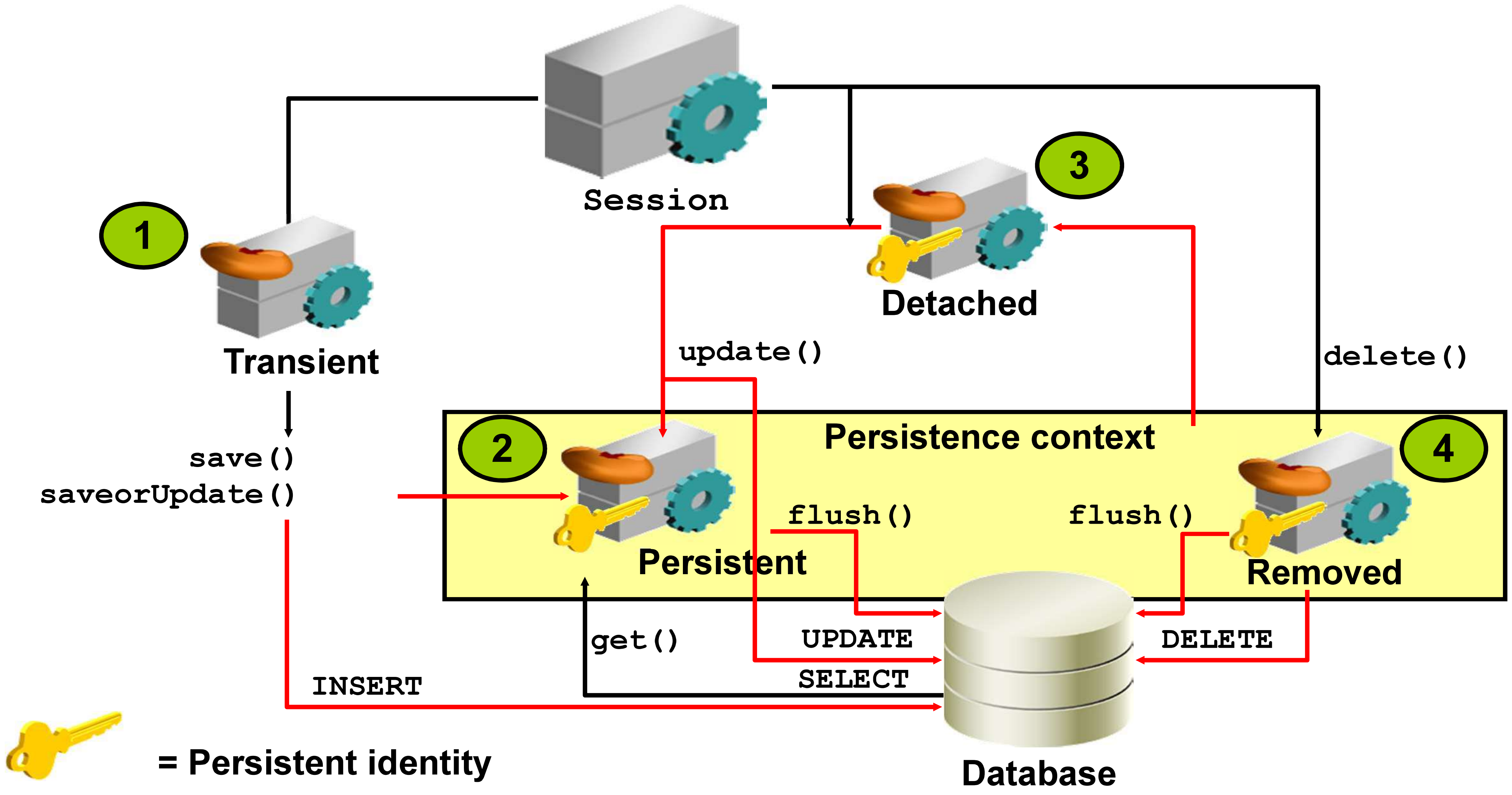
This is the only state where objects are saved to the database

- Modifications made in other states are NOT saved to the database while the object remains in that state
- Changes to objects in a persistent state are automatically saved to the database without invoking session persistence methods

Objects are made persistent through calls against the Hibernate session

- session.save(account);
- session.update(account);
- session.lock(account);
- session.merge(account);

Managing an Entity Life Cycle



Persistent State

```
Session session =
  SessionFactory.getCurrentSession();

// ‘transient’ state – Hibernate is NOT aware that it exists
Account account = new Account();

// transition to the ‘persistent’ state. Hibernate is NOW
// aware of the object and will save it to the database
session.saveOrUpdate(account);

// modification of the object will automatically be
// saved because the object is in the ‘persistent’ state
account.setBalance(500);

// commit the transaction
session.getTransaction().commit();
```

Removed State

- **A previously persistent object that is deleted from the database**
  - session.delete(account);
- **Java instance may still exist, but it is ignored by Hibernate**
  - Any changes made to the object are not saved to the database
  - Picked up for garbage collection once it falls out of scope
    - Hibernate does *not* null-out the in-memory object

Removed State

```
Session session = SessionFactory.getCurrentSession();

// retrieve account with id 1.  account is returned in a 'persistent' state
Account account  = session.get(Account.class, 1);

// transition to the 'removed' state. Hibernate deletes the
// database record, and no longer manages the object
session.delete(account);

// modification is ignored by Hibernate since it is in the 'removed' state
account.setBalance(500);

// commit the transaction
session.getTransaction().commit();

// notice the Java object is still alive, though deleted from the database.
// stays alive until developer sets to null, or goes out of scope
account.setBalance(1000);
```

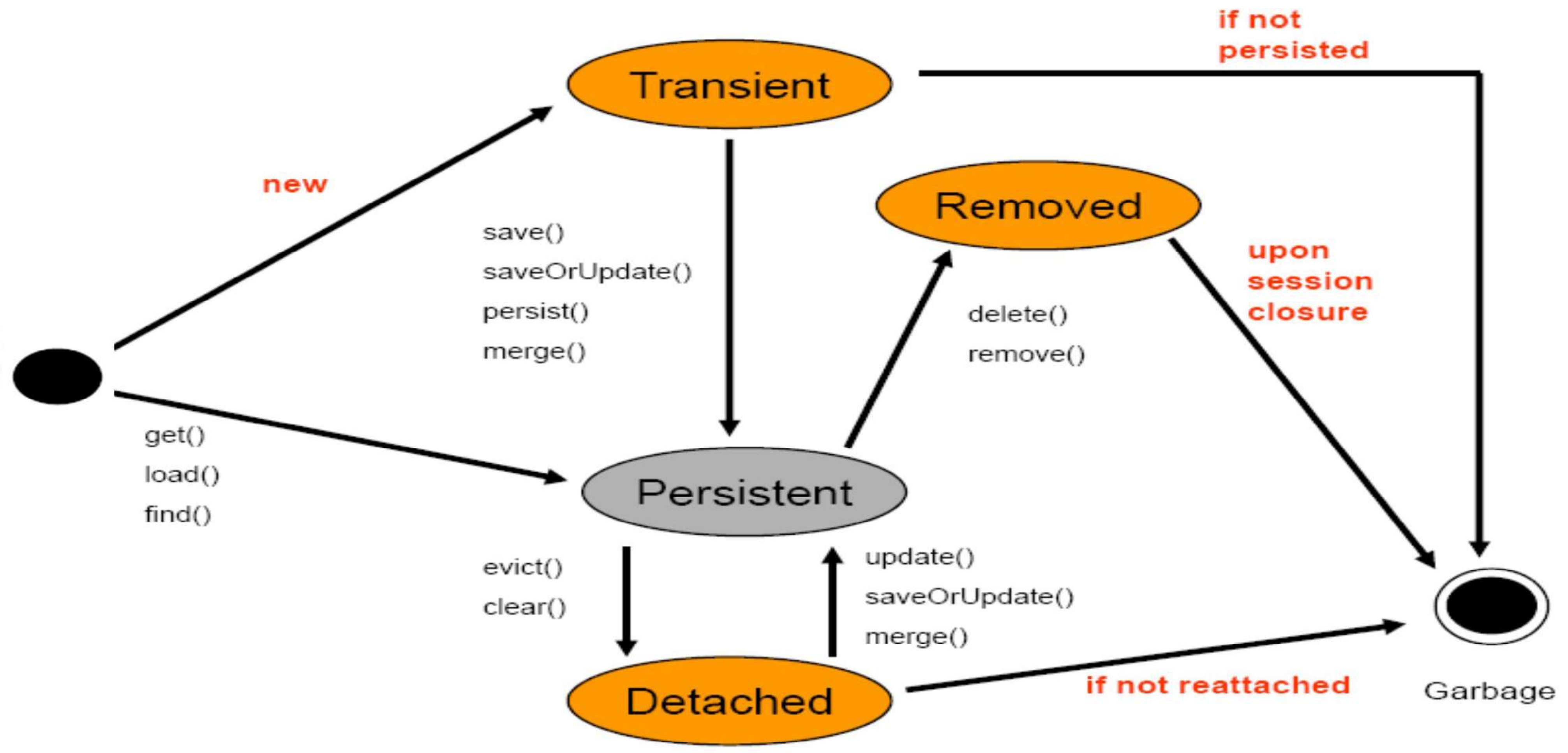
Detached State

- A persistent object that is still referenced after closure of the active session
  - session.close() changes object's state from persisted to detached
- Still represents a valid row in the database
- No longer managed by Hibernate
  - Changes made to detached objects are not saved to the database while object remains in the detached state
  - Can be reattached, returning it to the persistent state and causing it to save its state to the database
    - update();
    - merge();
    - lock(); // reattaches, but does not save state

Detached State

```
Session session1 = SessionFactory.getCurrentSession();

// retrieve account with id 1. account is returned in a 'persistent' state
Account account  = session1.get(Account.class, 1);

// transition to the 'detached' state. Hibernate no longer manages the object
session1.close();

// modification is ignored by Hibernate since it is in the 'detached'
// state, but the account still represents a row in the database
account.setBalance(500);

// re-attach the object to an open session, returning it to the 'persistent'
// state and allowing its changes to be saved to the database
Session session2 = SessionFactory.getCurrentSession();
session2.update(account);

// commit the transaction
session2.getTransaction().commit();
```

Hibernate Lifecycle With JPA Methods

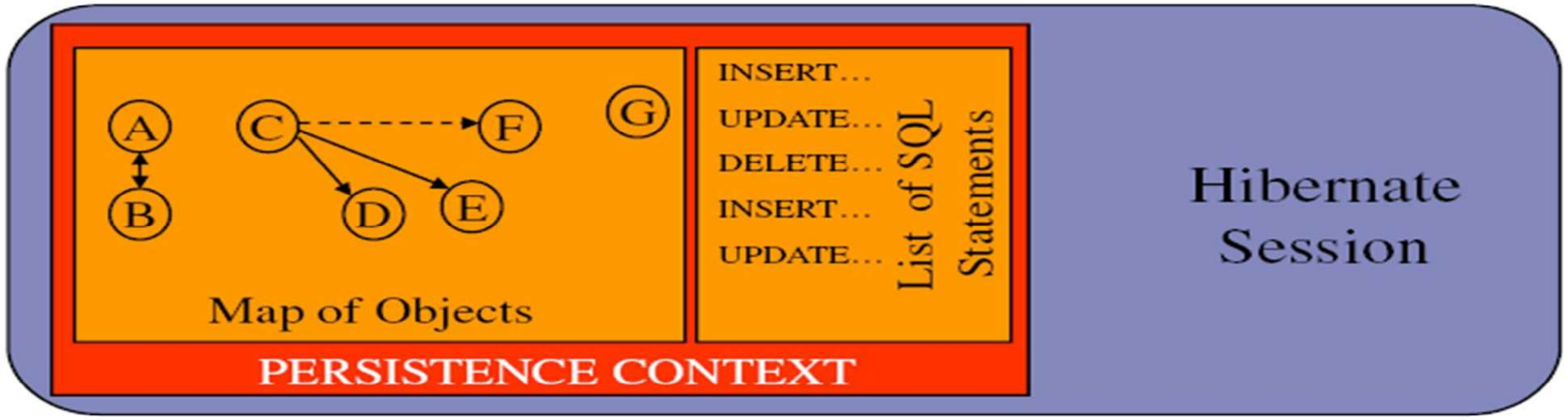


Saving Changes to the Database

- **Session methods do NOT save changes to the database**
  - save();
  - update();
  - delete();
- **These methods actually SCHEDULE changes to be made to the database**
- **Hibernate collects SQL statements to be issued**
- **• Statements are later *flushed* to the database**
  - Once submitted, modifications to the database are not permanent until a commit is issued
    - session.getTransaction().commit();

The Persistence Context

- **Managed object environment**
  - No API or concrete object to reference, just conceptual
  - Thought of as containing:
    - Graph of managed persistent instances
    - List of SQL statements to send to the database
- **• Each Hibernate session is said to have one ‘persistence context’**



Hibernate Session

- Single threaded non–shared object that represents a unit of work
- Used to retrieve objects from the database
- Contains the ‘persistence context’
- Used to *schedule* changes to be made in the database

Session API –

- **session.save(Object o)**
  - – Schedules insert statements to create the new object in the database
- **session.update(Object o)**
  - – Schedules update statements to modify the existing object in the database

Scheduled Changes

```
Session session = SessionFactory.getCurrentSession();

// 'transient' state – Hibernate is NOT aware that it exists
Account account = new Account();

// Transition to the 'persistent' state. Hibernate is NOW
// aware of the object and ~~will save it to the database~~
// schedules the insert statements to create the object in the database
session.saveOrUpdate(account);

// modification of the object will automatically be
//~~saved~~ scheduled because the object is in the 'persistent' state
// (actually alters the initial insert statement since it hasn't been sent yet)
account.setBalance(500);

// flushes changes to the database and commit the transaction
session.getTransaction().commit();
```

Session API –

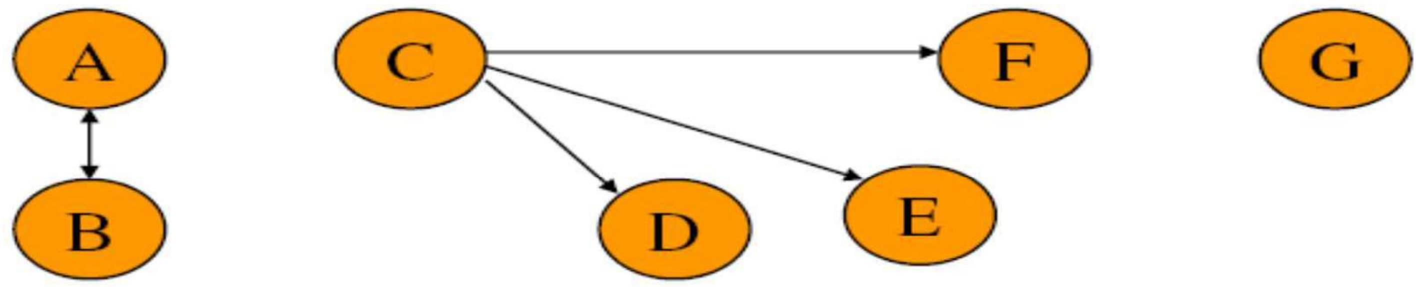
- **session.saveOrUpdate(Object o)**
  - Convenience method to determine if a ‘save’ or ‘update’ is required
- **session.merge(Object o)**
  - Retrieves a fresh version of the object from the database and based on that, as well as modifications made to the object being passed in, schedules update statements to modify the existing object in the database.

Session API –

- **session.get(Object.class, Identifier)**
  - Retrieves an object instance, or null if not found
- **session.lock(Object, LockMode)**
  - It is used to ‘lock’ records in the database
- **session.refresh(Object)**
  - Gets the latest version from the database
- **session.clear()**
  - Removes all objects from persistence context, changing all their states from persistent to detached
- **session.delete(Object)**
  - Schedule an object for removal from the database

Cascading Object Persistence

- Hibernate represents domain object relationships through a graph model



- Propagate the persistence action not only to the object *submitted*, but also to any objects *associated* with that object

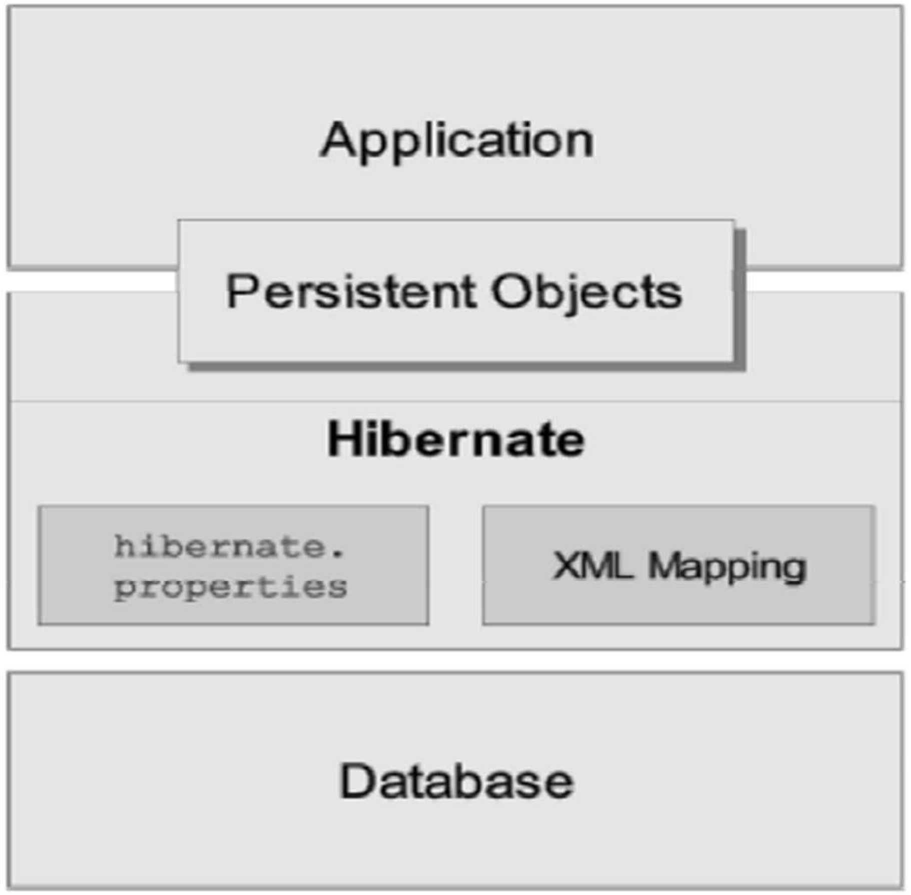
Cascade Attribute

- **none**
  - Default behavior
- **save-update**
  - Saves or updates associated objects
  - Associated objects can be transient or detached
- **delete**
  - Deletes associated persistent instances
- **delete-orphan**
  - Enables deletion of associated objects when they’re removed from a collection

Samples

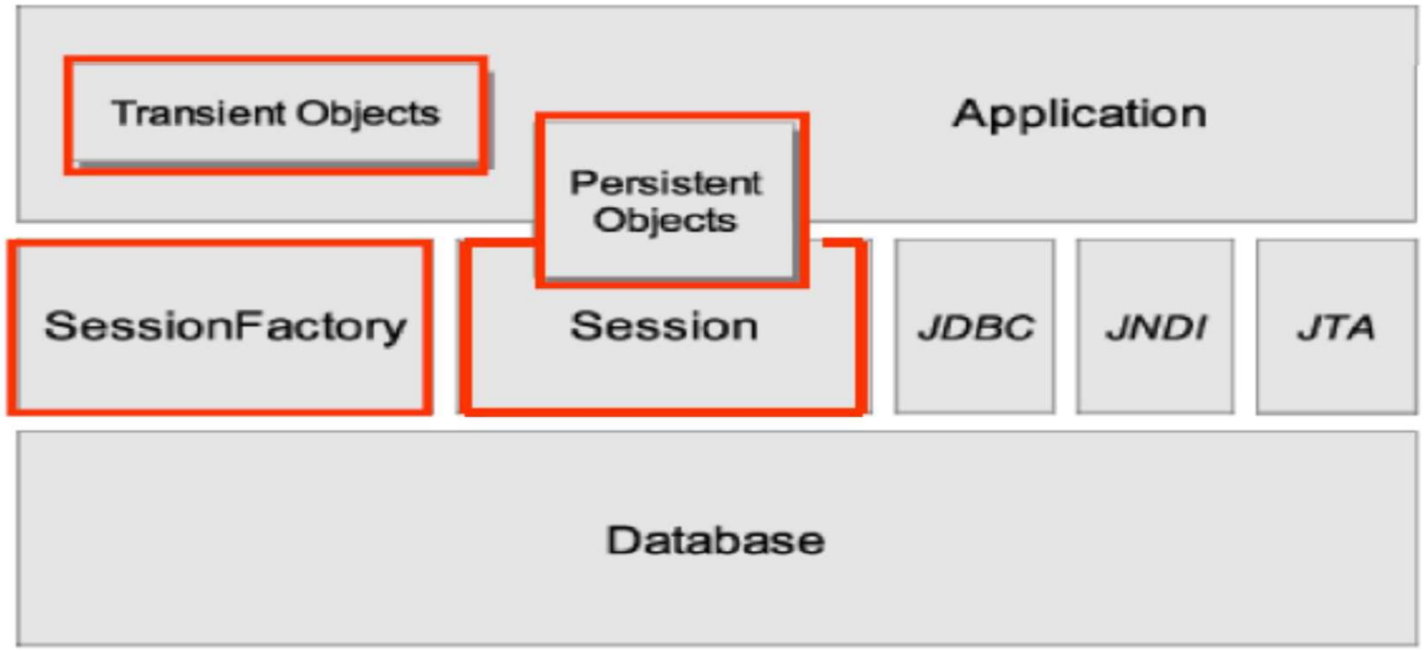
- < set name="ebills" cascade=“save-update” >
- <set name="ebills" cascade="delete“ >

Hibernate Architecture – High



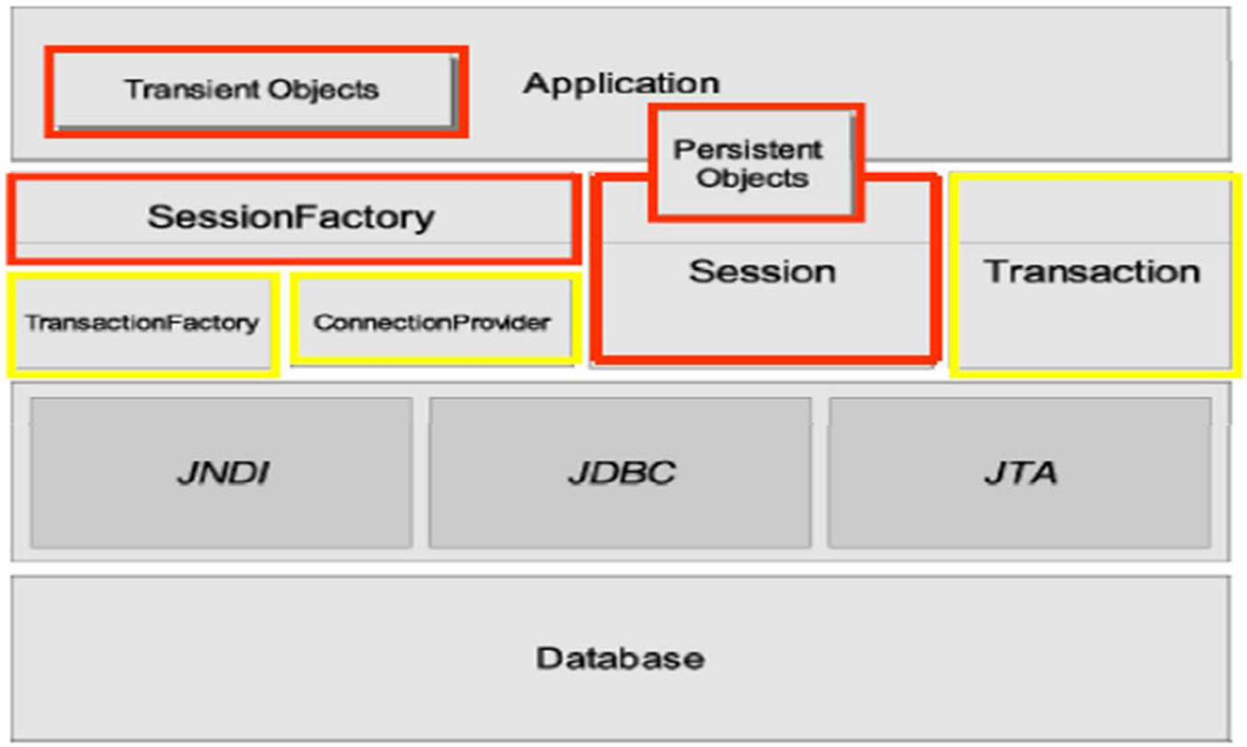
Hibernate Arch – “Lite”

- The “Lite" architecture has the application provide its own JDBC connections and manage its own transactions



Hibernate Arch – "Full Cream"

- The "full cream" architecture abstracts the applicationaway from the underlying JDBC and Java Transaction API (JTA) and lets Hibernate take care of the details



Obtaining the Session

- **SessionFactory.getCurrentSession()**
  - Reuses an existing session
    - If none exists, creates one and stores it for future use
  - Automatically flushed and closed when transaction.commit() is executed
- **SessionFactory.openSession()**
  - Always obtains a brand new session

Summary

- In this lesson, we have learned
  - Learn the lifecycle of Hibernate entities, when they transition from state–to–state, and how this affects development
  - Take a closer look at persistence, and discover how Hibernate manages dirty and related objects
  - See how Hibernate uses a Session to keep track of, and optimize, user activities