Fetching strategies in Hibernate

Hibernate uses a fetching strategy to retrieve associated objects if the application needs to navigate the association. Fetch strategies can be declared in the O/R mapping metadata, or over-ridden by a particular HQL or Criteria query.

Hibernate3 defines the following fetching strategies:

Join fetching:

Hibernate retrieves the associated instance or collection in the same SELECT, using an OUTER JOIN.

Select fetching:

a second SELECT is used to retrieve the associated entity or collection. Unless you explicitly disable lazy fetching by specifying lazy="false", this second select will only be executed when you access the association.

Subselect fetching:

a second SELECT is used to retrieve the associated collections for all entities retrieved in a previous query or fetch. Unless you explicitly disable lazy fetching by specifying lazy="false", this second select will only be executed when you access the association.

Batch fetching:

an optimization strategy for select fetching. Hibernate retrieves a batch of entity instances or collections in a single SELECT by specifying a list of primary or foreign keys.
Hibernate also distinguishes between:

Immediate fetching:

an association, collection or attribute is fetched immediately when the owner is loaded.

Lazy collection fetching:

a collection is fetched when the application invokes an operation upon that collection. This is the default for collections.
"Extra-lazy" collection fetching: individual elements of the collection are accessed from the database as needed. Hibernate tries not to fetch the whole collection into memory unless absolutely needed. It is suitable for large collections.

Proxy fetching:

a single-valued association is fetched when a method other than the identifier getter is invoked upon the associated object.
"No-proxy" fetching: a single-valued association is fetched when the instance variable is accessed. Compared to proxy fetching, this approach is less lazy; the association is fetched even when only the identifier is accessed. It is also more transparent, since no proxy is visible to the application. This approach requires buildtime bytecode instrumentation and is rarely necessary.
Lazy attribute fetching: an attribute or single valued association is fetched when the instance variable is accessed. This approach requires buildtime bytecode instrumentation and is rarely necessary.
We have two orthogonal notions here: when is the association fetched and how is it fetched. It is important that you do not confuse them. We use fetch to tune performance. We can use lazy to define a contract for what data is always available in any detached instance of a particular class.