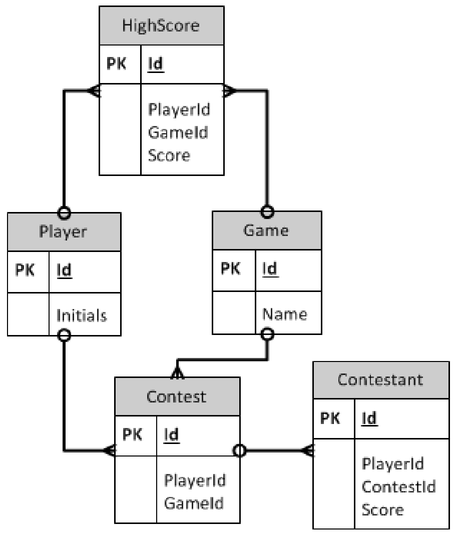
NHIBERNATE VS. ACTIVERECORD QUERY & MAPPING SYNTAX

This comparison references this simple database layout.



While NHibernate enables several alternative syntax approaches these examples choose only a single, common pattern for display.

Each general form of the syntax will be followed by an example that references the simple database diagram provided using this style.

```
General Syntax
Specific example
```

In cases where an analogous syntax does not exist the relevant box is left blank.

Query Operation	NHIBERNATE	RAILS

Return a single record, passing the primary key value	`Session.Get<Model>(id)`
	
	`var p = Session.Get<Player>(2);`
		`Model.find(id)`
		
		`p = Player.find(2)`

Return a single record, passing something other than the primary key value to find it	`Session.Query<Model>()`
	`   .Where(`*`conditional`*`)`
	`   .FirstOrDefault();`
	
	`var centipede = Session.Query<Game>()`
	`  .Where(x => x.Name == 'Centipede')`
	`  .FirstOrDefault();`
		`Model.find_by(field: value)`
		
		`centipede = Game.find_by(name: 'Centipede')`

Return a single record from session cache preferentially, passing the primary key value	`Session.Load<Model>(id)`
	
	`var p = Session.Load<Player>(2);`	

Return only the first n number of records	`Session.Query<Model>()`
	`   .Take(`*`n`*`);`
	
	`var players = Session.Query<Player>()`
	`  .Take(2)`
	`  .ToList();`
		`Model.take(`*`n`*`)`
		
		`players = Player.take(2)`

Skip the first n number of records before returning the remaining	`Session.Query<Model>()` `.Skip(n);` `var players = Session.Query<Player>()` `.Skip(10)` `.ToList();`	`Model.offset(n)` `players = Player.order(:initials).offset(10)`
Sort results	`Session.Query<Model>()` `.OrderBy(x => x.field)` `.ToList();` `Session.Query<Model>()` `.OrderByDescending(x => x.field)` `.ToList();` `var players = Session.Query<Player>()` `.OrderBy(x => x.Initials)` `.ToList();`	`Model.order(:field1, field2: :desc)` `players = Player.order(:initials)`
Filter results	`Session.Query<Model>()` `.Where(x => conditional)` `.ToList();` `var games = Session.Query<Game>()` `.Where(x => x.Name.Contains("C"))` `.ToList();`	`Model.where(["sql string", replacement1, replacement2])` `Model.where("sql string")` `Model.where({field1: value1, field2: value2})` `games = Game.where(["name like ?", "%A%"])`
Project select fields from results	`Session.Query<Model>()` `.Select(x => new {` `Field1 = x.Field1,` `Field2 = x.Field2` `}).ToList();` `var playerDtos = Session.Query<Player>()` `.Select(x => {` `Id = x.Id,` `Initials = x.Initials` `}).ToList();`	`Model.pluck(:field1, :field2)` `Model.pluck('field1 as alias')` `Model.select(:field1, :field2)` `Model.select('field1 as alias', 'field2 as alias2')` `Model.select([:field1, :field2])` `playersHashes = Player.pluck(:id, :initials)` `players = Player.select(:id, :initials)` *** select() returns an array of full models while pluck returns an array of hashes containing only the fields specified. select() must be used with map() to yield the same results as pluck()*

Perform a must match join between 2 tables (INNER)	(from x in Session.Query<Model1>()
	  join y in Session.Query<Model2>() on x.*Field* equals y.*Field*
	  select x)
	  .ToList();
	
	var players = (from hs in Session.Query<HighScore>()
	  join p in Session.Query<Player>() on hs.PlayerId equals p.Id
	  join g in Session.Query<Game>() on hs.GameId equals g.Id
	  select p)
	  .ToList();
		Model1.joins(*:model2*)
		
		Model1.joins(*:model2, :model3*)
		
		Model1.joins(*model2: [model3: model4]*)
		
		Model.joins("SQL String")
		
		players = Player.joins(contestants: [contest: :game])
		  .where('games.name = ?', 'Pac-Man')
		
		*** use joins() only to construct a logical query for filtering. No objects are included in the results with this so includes() is a better choice if that's your use case.*

Perform a maybe match join between 2 tables (LEFT)	Model1 alias1 = null;
	Model2 alias2 = null;
	Session.QueryOver<Model1>(() => alias1)
	  .JoinAlias(() => alias1.Model2Property, () => alias2, JoinType.LeftOuterJoin)
	  .SelectList(list => list
	    .Select(() => alias1.Field)
	    .Select(() => alias2.Field2)
	  ).List<object[]>();
	
	Game gameAlias = null;
	HighScore hsAlias = null;
	var gamesAndHighScores = session.QueryOver<Game>(() => gameAlias)
	  .JoinAlias(() => gameAlias.HighScores, () => hsAlias, JoinType.LeftOuterJoin)
	  .SelectList(list => list
	    .SelectGroup(() => gameAlias.Name)
	    .SelectCount(() => hsAlias.Id)
	  ).List<object[]>();
		Model1.left_joins(*:model2*)
		
		Model1.left_joins(*:model2, :model3*)
		
		games = Game.left_joins(:high_scores)
		  .where('high_scores.score > 5000')

Group results	(from x in Session.Query<Model>()
	  group x by x.Field into g
	  select g)
	  .ToList();
	
	var gameHighScores = (from gm in Session.Query<Game>()
	  join hs in Session.Query<HighScore>() on gm.Id equals hs.GameId
	  group hs by new { gm.Id, gm.Name } into g
	  select new {
	    GameId = g.Key.Id,
	    GameName = g.Key.Name,
	    HighScores = g.ToList()
	  }).ToList();
		Model.group(*:field*)
		
		Model.group('sql expression')
		
		high_score_groups = HighScore.group(:player_id).count(:player_id)

Cull to unique results	Session.Query<Model>()
	  .Select(x => *projection*)
	  .Distinct()
	  .ToList();
	
	var highScorePlayers = Session.Query<HighScore>()
	  .Select(x => x.Player)
	  .Distinct()
	  .ToList();
		Model.select(*projection*).distinct()
		
		high_score_players = HighScore.select(:player_id).distinct
		
		***note that only player_id may be referenced in each of the HighScore models in the list returned. Attempts to access any other attribute will result in a 'missing attribute' error EVEN THOUGH the attribute is clearly defined on the model!*

Execute a raw SQL statement	`Session.CreateSQLQuery(“sql”)`
	`  .SetResultTransformer(`
	`Transformers.AliasToBean(typeof(Model)))`
	`  .ToList();`
	
	`playerList = Session.CreateSQLQuery("select * from Player p")`
	`.SetResultTransformer(Transformers.AliasToBean(typeof(Player)))`
	`.List<Player>();`
		`ActiveRecord::Base.connection.select_all(`
		`ActiveRecord::Base.send(:sanitize_sql_array, [sql]))`
		
		`ActiveRecord::Base.connection.select_all(`
		`ActiveRecord::Base.send(:sanitize_sql_array,`
		`[sql_with_parameters, replacement1, replacement2]))`
		`sql = 'select * from player'`
		`players = ActiveRecord::Base.connection.select_all(`
		`ActiveRecord::Base.send(:sanitize_sql_array, [sql]))`

Execute a stored procedure and return results	`Session.CreateSQLQuery(“exec spName :param1”)`
	`  .SetParameter(“param1”, param1Value)`
	`  .SetResultTransformer(`
	`Transformers.AliasToBean(typeof(Model)))`
	`  .ToList();`
	`var contestResults = Session.CreateSQLQuery(“exec spGetContestResult :contestId”)`
	`  .SetParameter(“contestId”, 3)`
	`  .SetResultTransformer(`
	`Transformers.AliasToBean(typeof(ContestResultDto)))`
	`  .ToList();`
		`ActiveRecord::Base.connection.execute(`
		`ActiveRecord::Base.send(:sanitize_sql_array,`
		`[‘sp SQL’, replacement1, replacement2]))`
		
		`contest_result = ActiveRecord::Base.connection.execute(`
		`  ActiveRecord::Base.send(:sanitize_sql_array,`
		`  [‘call spGetContestResult ?’, contest_id]))`

Eager fetch joined data to reduce round trips	`Session.Query<Model>()`
	`  .Fetch(x => x.Model2)`
	`  .ToList();`
	
	`Session.Query<Model>()`
	`  .FetchMany(x => x.Model2)`
	`  .ToList();`
	`gameList = session.Query<Game>()`
	`  .FetchMany(x => x.Contests)`
	`  .ToList()`
	`contests = session.Query<Contest>()`
	`  .Fetch(x => x.Game)`
	`  .FetchMany(x => x.Contestants)`
	`  .ToList()`
		`Model1.includes(:model2)`
		
		`Model1.preload(:model2)`
		
		`Model1.eager_load(:model2)`
		
		`Game.includes(contests: :players).take(2)`
		`Game.preload(:contests).take(3)`
		`Game.eager_load(:contests).take(4)`
		*** includes() will either generate a secondary parameterized query and return the results seamlessly OR a left join and do the same.*
		If you filter on an included model you MUST specify the related model as well with a references() clause. For instance, to filter on contests held after 1/1/2018 you’d need to do this:
		`Game.includes(contests: :players).where(‘players.initials = ?’, ‘ART’).references(:players)`
		eager_load() will try to force a left join to load the specified models (think “optimized for 1 query”)
		preload() will execute a dedicated query to fetch and populated the specified models, but that secondary query will be filtered if appropriate (think “optimized for 1 query PER model”)
		Remember that joins() never has any impact on objects returned.

Mapping Type	NHIBERNATE	RAILS

Property	`Map(x => x.Field)`
	
	`Map(x => x.Field).Nullable()`
	
	`Map(x => x.Initials)`
		`** not listed in the model`

Primary key	`Id(x => x.`*`PKFieldName`*`)`
	
	`Id(x => x.Id);`
		`** not listed in the model`

Parent table	`References(x => x.`*`ParentModel`*`)`
	`.Column("`*`ChildFKField`*`")`
	`.ForeignKey("`*`ParentPKFIeld`*`")`
	
	`References(x => x.Player)`
	`.Column("PlayerId")`
	`.ForeignKey("Id");`
		`belongs_to :parent_model`
		
		`belongs_to :player`

Child table rows list property	`HasMany(x => x.`*`ChildListProperty`*`)`
	`.KeyColumn("`*`ChildFKField`*`")`
	`.Cascade.None()`
	`.Inverse();`
	
	`HasMany(x => x.Contestants)`
	`.KeyColumn("ContestId")`
	`.Cascade.None()`
	`.Inverse();`
	
	`** cascade none and inverse tell NH to ignore objects added or removed from the collection when considering the need to put those changes back to the database`
		`has_many :`*`plural_child_model`*
		
		`has_many :contestants`

Child table rows list property using cross reference table	`HasManyToMany(x => x.`*`ChildListProperty`*`)`
	`.Table("`*`CrossReferenceTable`*`")`
	`.ParentKeyColumn("`*`FKToTHISTable`*`")`
	`.ChildKeyColumn("`*`FKToChildTable`*`")`
	`.Cascade.None()`
	`.Inverse();`
	
	`HasManyToMany(x => x.Players)`
	`.Table("Contestant")`
	`.ParentKeyColumn("ContestId")`
	`.ChildKeyColumn("PlayerId")`
	`.Cascade.None()`
	`.Inverse();`
		`has_many :`*`plural_child_model`*`, through: :`*`cross_reference_table`*
		
		`has_many :high_score_games, through: :high_scores`