Разработка LINQ-запросов на выборку

Действие	LINQ
Выборка / проекция (SQL: select)	**данные.Select(переменная => возвращаемое значение)** возвращаемое значение может быть определенного или анонимного типа **var result = games.Select(g => g.category);** **var result = games.Select(g => new { g.name, g.price });**
Фильтрация (SQL: where)	**данные.Where(переменная => условие)** условие составляется по правилам C# **var result = games.Where(g => g.category == "аркада");** **var result = games.Where(g => g.description != null);** **var result = games.Where(g => g.price > 200 && g.price < 500);** **var result = games.Where(g => g.name.StartsWith("M"));**
Пропуск / взятие значений (SQL: fetch-offset)	**данные.Take(количество требуемых строк)** **данные.Skip(количество пропускаемых строк)** **данные.TakeWhile(переменная => условие взятия строк)** – до первого несоответствия **данные.SkipWhile(переменная => условие пропуска строк)** – до первого несоответствия **var result = games.Take(3);** **var result = games.Skip(10);** **int count = 2;** // количество требуемых строк **int page = 1;** // номер страницы (с нулевой) **var result = games.Skip(count * page).Take(count);**
Сортировка (SQL: order by)	**данные.OrderBy(переменная => столбец сортировки)** – по возрастанию **данные.OrderByDescending(переменная => столбец сортировки)** – по убыванию **ThenBy / ThenByDescending** используются, если нужно сортировать по 2 и более столбцам **var result = games.OrderBy(g => g.category);** **var result = games.OrderByDescending(g => g.category);** **var result = games.OrderBy(g => g.category).ThenByDescending(g => g.price);**

Агрегатные функции	**данные.Min(); данные.Max(); данные.Average(); данные.Sum(); данные.Count();**
	
	в параметрах у агрегатных функций можно указать поле для вычисления и условие отбора:
	int count = games.Count(g => g.price > 500); // количество игр с ценой > 500
	
	данные.Aggregate() – применяет к последовательности указанное действие
	int[] numbers = { 1, 2, 3, 4, 5};
	int query = numbers.Aggregate((x,y)=> x + y); // аналогично 1 + 2 + 3 + 4 + 5

Группировка (SQL: group by)	**данные.GroupBy(переменная => столбец группировки)** **var result = games.GroupBy(g => g.category);** Группа записывается в поле Key, после группировки можно применять агрегатные функции **var result = games.GroupBy(g => g.category)** **.Select(gr => new { Category = gr.Key, MaxPrice = gr.Max(g => g.price) });**
Соединение (SQL: join)	**данные1.Join(данные2,** **x1 => соединитель из 1 набора,** **x2 => соединитель из 2 набора,** **(x1, x2) => результат)** **var result = games // первый набор** **.Join(categories, // второй набор** **g => g.category, // свойство-селектор объекта из первого набора** **c => c.name, // свойство-селектор объекта из второго набора** **(g, c) => new { Name = g.name, Price = g.price, Category = c.description }); // результат**
Соединение + группировка	**данные1.GroupJoin(те же параметры, что у Join)** позволяет применять агрегатные функции в результате
Без дубликатов (SQL: distinct))	**данные.Distinct()** **var result = games.Select(g => new { g.category }).Distinct();**
Объединение результатов	**данные1.Concat(данные2)** // union all – все, включая дубликаты **данные1.Union(данные2)** // union – все без дубликатов **данные1.Intersect(данные2)** // intersect – пересечение множеств **данные1.Except(данные2)** // except – исключение второго множества У объединяемых множеств должны совпадать названия и количество свойств **var result = games.Where(g => g.price < 1000).Intersect(games.Where(g => g.category == "RPG"));**
Условие ANY / ALL	**bool result = данные.All(условие);** // условие выполнилось для всех **bool result = данные.Any(условие);** // условие выполнилось для хотя бы одной записи
Первая / последняя запись	**First()** - первая запись. Если нет – исключение **| FirstOrDefault()** - первая или по умолчанию **Last()** - последняя запись. Если нет – исключение **| LastOrDefault()** - последняя или по умолчанию **var result = games.First();** **var result = games.FirstOrDefault(g => g.price > 10000);**
Приведение типов	**ToList()** список **| ToArray()** массив **| ToDictionary()** словарь **| AsEnumerable()** перечисляемый тип var resultList = result.ToList();