EF & LinQ Cheat Sheet

[Query Syntax]
[Lamda Syntax]

Query operation	
var set1 = _Context.Products;	var set1 = _Context.Products;

Filtering operation	
Get a product by id	
var product =**from** p **in** _Context.Products **where** p.id == 1 **select** p;	var product = _Context.Products .**where**(x=>x.id == 1);
Get a list of products for price > 10	
var products =from p in _Context.Products **where** p.price > 10 **select** p;	var product s= _Context.Products .**where**(x=> x.price >10);
Get a list of products for price > 10 and qty =0	
var products =from p in _Context.Products **where** p.price > 10 **and** p.qty =0 **select** p;	var product s= _Context.Products .**where**(x=> x.price >10 && x.qty == 0);

Mapping Operation	
Get a list of anonymous type including attribute from product and a new information = price and qty as value_in_stock	

var result =from p in _Context.Products
select new {
name = p.name,
value_in_stock = p.price*p.qty
}
	var result= _Context.Products
	.**Select**(x => **new** {
	name = x.name,
	value_in_stock = x.price*x.qty
	}
	);

Sorting operation	
Get a list by ascending order	
var result =**from** p **in** _Context.Products **orderby** p.name ascending **select** p;	var result= _Context.Products .**OrderBy**(x=> x.name);
Get a list by descending order	
var result =**from** p **in** _Context.Products **orderby** p.name descending **select** p;	var result= _Context.Products .**OrderByDescending**(x=> x.name);
Reorder list by multiple properties	
var result =**from** p **in** _Context.Products **orderby** p.name descending **orderby** p.price **select** p;	var result= _Context.Products .**OrderByDescending**(x=> x.name) .**OrderBy**(x=>x.price);

Join Operation	
Join lists from two tables	

var result =**from** p **in** _Context.Products
join s **in** _Context.Suppliers
on
p.supplierId
equals
s.supplierId
select new {
name = p.name,
price = p.price,
supplier =s.name,
supplier_addr = s.address
}
	var result= _Context.Products
	.**Join**(_Context.Suppliers,
	P=> p.supplierId,
	s=> s.supplierId,
	(p,s)=> **new**
	{
	p.name,
	p.price,
	s.name,
	s.address
	}
	);

Grouping Operation	
Group a list by property	

var result =(**from** p **in** _Context.Products
group p **by** p.type into g
select new
{
Type = g.Key,
Count = g.Count()
}
);
	var result= _Context.Products
	.**GroupBy**(x=>x.type)
	.**Select**(g=> **new**
	{
	Type = g.Key,
	Count = g.Count()
	}
	);

Paging Operations (Take and Skip)	
Take the first three products where price > 10	
var product = (from p in _Context.Products **where** p.price > 10 **select** p) .**Take**(3););	var product = _Context.Products .**where**(x=>x.price >10) .**Take**(3);
Skip the first two and take the next three products where price >10	
var product = (from p in _Context.Products **where** p.price > 10 **select** p) .**Skip**(2) .**Take**(3)	var product = _Context.Products .**where**(x=>x.price >10) .**Skip**(2) .**Take**(3);

Element Operations (Single, Last, First, ElementAt, Defaults)	
Get single object. It will throws exception if no element found.	
var product = (**from** p **in** _Context.Products **where** p.price>10 **select** p) .**Single**(); //throws exception if no elements	var product = _Context.Products .**where**(x=>x.price >10) .**Single**();
Get single object. It will return **NULL** if no element found.	
var product = (**from** p **in** _Context.Products **where** p.price>10 **select** p) .**SingleOrDefault**(); //throws exception if no elements	var product = _Context.Products .**where**(x=>x.price>10) .**SingleOrDefault**();
Get the last objects.	
var product = (**from** p **in** _Context.Products **where** p.price>10 **orderby** p.price **select** p) .**Last**(); //First, Last, ElementAt used in same way	var product = _Context.Products .**where**(x=>x.price >10) .**OrderBy**(x=>x.price) .**Last**();

EF Sub-Query

Query with complex structure

```
public async Task<IActionResult> data(){
           var result = await _context.ProductOrders
          .Select(o=>new {
            orderid      = o.productOrderId,
            customer     = o.customer.customerName,
            order_details      = o.productOrderDetails
                                  .Select(x=> new {
                                        product_name = x.product.productName,
                                        qty          = x.qty,
                                        price        = x.price,
                                        total        = x.price*x.qty
                                  }),
             total       =  o.productOrderDetails.Sum(x=> x.qty*x.price)
          })
          .ToListAsync();
          return Json(result);
        }
```


```
[{"orderid":1,
"customer":{"customerId":1,"customerName":"andy","customerAddress":"address1"
},
"order_details":[
{"product_name":"p1","qty":10,"price":300.0,"total":3000.0},
{"product_name":"p2","qty":30,"price":200.0,"total":6000.0}],
"total":9000.0}]
```

EF Query with Include

Use Case: query table with foreign key

Table : ProductOrders [orderId as int | customer as Customer|posted_date as DateTime]
Customer[customerId as int | customerName string | customerAddress string]

```
public async Task<IActionResult> Index()
        {
            var mISDbContext = _context.ProductOrders.Include(p => p.customer);
            return View(await mISDbContext.ToListAsync());
        }
```


```
[{"productOrderId":1,
"posted_date":"2019-03-13T10:10:00",
"customer":{"customerId":1,"customerName":"andy","customerAddress"
:"address1"}}]
```