1) Выпадающие списки там, где потенциально может содержаться множество данных, нужно заменять текстовыми полями, так как при большом объёме информации загрузка всех записей таблицы приводит к печальным последствиям в виде огромного нечитабельного списка. Например, для списка лошадей вместо

<div class="form-group">

<div class="col-md-4 control-label">

@Html.LabelFor(m => m.horse.HorseName)

</div>

<div class="col-md-6">

@Html.DropDownListFor(

m => m.horse.horse.HorseName,

Html.CreateSelectListFromLookup(ViewBag.Horses as IEnumerable<RtLookupBase>),

new { @class = "form-control" })

</div>

</div>

Следует использовать

<div class="form-group">

<div class="col-md-4 control-label">

@Html.LabelFor(m => m.horse.Name)

</div>

<div class="col-md-6">

@Html.TextBoxFor(m => m.horse.Name, new {@class = "form-control"})

</div>

</div>

2) Для списков следует использовать не тэг <select>, а стандартные методы разметки Razor. Это позволяет записывать выбранные пользователем значения непосредственно в модель, а не передавать их через параметры запроса. Следует заметить, что в Razor в классе HtmlHelper имеются методы для создания практически всех основных элементов html-разметки, например, checkbox, radiobutton, input и многих других.

Вместо

<label>Gender</label>

<select id="HorseGender" name="HorseGender">

<option selected></option>

@foreach (var gender in ViewBag.Lookups[LookupType.Gender])

{

<option value="@gender.Name">@gender.Name</option>

}

</select>

Использовать

<div class="form-group">

<div class="col-md-4 control-label">

@Html.LabelFor(m => m.horse.GenderId)

</div>

<div class="col-md-6">

@Html.DropDownListFor(

m => m.horse.GenderId,

Html.CreateSelectListFromLookup(ViewBag.Lookups[LookupType.Gender] as IEnumerable<RtLookupBase>),

new { @class = "form-control" })

</div>

</div>

3) Рекомендуется использовать атрибут ValidateAntiforgeryToken для противодействия подделке межсайтовых запросов, производя верификацию токенов при обращении к методу действия. Для подключения данного атрибута во View используется метод @Html.AntiForgeryToken().

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<ActionResult> RaceDetails(RaceDto race)

{

.....

}

4) Преимущетсва Eager и Lazy load-ов

Eager-load лучше использовать, когда требуется частое обращение к базе данных. Все данные загружаются непосредственно при обращении к сущности, обычно это делается с помощью одного запроса join, возвращающего все данные. Используется с помощью метода Include:

protected override IQueryable<Horse> EagerLoadItems()

{

return \_table

.Include(x => x.Father)

.Include(x => x.Mother)

.Include(x => x.Owner)

.Include(x => x.BirthOwner)

.Include(x => x.HorseBreed)

.Include(x => x.HorseColor)

.Include(x => x.Gender)

.Include(x => x.RacetrackStats);

}

В данном случае вместо 8 запросов для каждой из таблиц будет выполнен один общий.

При Lazy-load данные загружаются только тогда, когда в этом возникает непосредственная необходимость. Если обращение к этим данным в ходе работы и не потребуется, то таким образом будет достигнут выигрыш во времени благодаря тому, что ресурсы не тратились на лишние операции. Однако если же такая необходимость возникнет, то вместо одного запроса, как в случае Eager-load, потребуется совершать отдельный запрос для подключения каждой из связанных таблиц.