Сеньор.

# Производительность

## Кэширование

Кеш очень важная составляющая производительности проектов. Чем эффективнее кешируем, чем кеш легче и быстрее, тем меньше ресурсов надо на поддержание проекта.

```
GetID();
                }
                $cntStartCacheId =
__CLASS__.'::'.__FUNCTION__.'|'.SITE_ID.'|'.$userId;
                $cache = new \CXxxCache(
                        $cntStartCacheId.'sid0',
                        // увеличили время кеширования
                        604800,
                        // путь для ключей кеша сделали
зависимым от $userID

'user_data/'.substr(md5($userId),2,2).'/'.$userId
                );
                $this->userData = $cache->Init();
                if (null == $this->userData)
                {
                        $this-
>putUserData(array("ID"=>$userId));
                        // Выбираем только нужные поля
                        $this-
>putUserData(\CUser::GetList(...)->GetNext(true, false));
```

```
                        $this->putUserData(array("DEPARTMENT"
=> $this->getDepartment()));
                        $cache-
>registerTag('USER_NAME_'.$userId);
                        $cache->set($this->userData);
                }
        }
}
```

## Время кеширования (cache key and ttl)

Пример на компоненте, который отображает дни рождения. Установлено большое время кеширования и добавлен дополнительный параметр, который сам компонент никак не отрабатывает и его не обрабатывает шаблон. Но так как в параметры подставлен `date`, то в 0 часов получим новый ключ у кеша данного компонента.

```
IncludeComponent(
        "bitrix:intranet.structure.birthday.nearest",
        "widget",
        Array(
                "CACHE_TYPE" => "A",
                "CACHE_TIME" => "86450",
                "DATE_FORMAT" => "j F",
                "DETAIL_URL" =>
"#SITE_DIR#company/personal/user/#USER_ID#/",
                "DEPARTMENT" => "0",
                .....
                "CACHE_DATE" => date('dmy')
        )
```

```
);
```

## Как в API правильно подставлять ключи

В таком случае часто проставляют лишние параметры, что приводит к увеличению кеша. (например date без параметров приводит к обновлению кеша каждую секунду).

```
<?php
// Пример добавление в ключ кеша метки времени для корректного
переключения кеша. Метка может быть и не из времени.
$cache = Bitrix\Main\Data\Cache::createInstance();
if ($cache->initCache(86450, '/some_key/'.date('myd').'/',
'/some_dir/'))
{
        $var = $cache->getVars();
}
else
{
        // Получение данных
        $cache->startDataCache();
        $cache->endDataCache($var);
}
```

## Отключаем сброс ключей

Процесс импорта обычно приводит к сбросу кеша. Если мы выгружаем большой объем данных - это занимает продолжительное время и дает большую нагрузку на боевом проекте. В ряде случаев этого можно избежать. В частности при работе с инфоблоками:

1. Отключаем кеширование элементов (сбрасывание тегированного кеша) перед импортом;
2. Включаем его по окончании процесса импорта элементов;
3. Сбрасываем те теги инфоблока, которые сбрасывались в данном случае.

В таком варианте кеш сбросится один раз после полной загрузки, а не после загрузки каждого элемента.

Индексация фасетного индекса также может отключена перед импортом и включена по окончании.

```
<?php
// Отключение сброса тегированного кеша инфоблоков и пересчета
фасетного индекса, во время импорта.
Bitrix\Iblock\PropertyIndex\Manager::enableDeferredIndexing();
Bitrix\Catalog\Product\Sku::enableDeferredCalculation();
\CAllIBlock::disableTagCache($iblockID);
// Импорт элементов
\CAllIBlock::enableTagCache($iblockID);
\CAllIBlock::clearIblockTagCache($iblockID);
Bitrix\Catalog\Product\Sku::disableDeferredCalculation();
Bitrix\Catalog\Product\Sku::calculate();
Bitrix\Iblock\PropertyIndex\Manager::disableDeferredIndexing();
Bitrix\Iblock\PropertyIndex\Manager::runDeferredIndexing($ibloc
kID);
```

## Использование метода GetList

Задача:
необходимо для каких-либо элементов одного инфоблока (авторы книг) получить дополнительные свойства из другого инфоблока

(информация по авторам), например путем изменения шаблона или **result modifier**.

```
// информация по авторам выбирается одним запросом и только нужная
$avtorID = array();
foreach($arResult['ITEMS'] as $ikey => $ival)
{
	$aID = intval($ival['PROPERTIES']['AVTOR']['VALUE']);
	if($aID > 0)
	{
		$avtorID[] = $aID;
	}
}
$avtorID = array_unique($avtorID);
$rs = CIBlockElement::GetList(
	array('ID' => 'ASC'),
	array(
		'IBLOCK_ID' => XX,
		'ID' => $avtorID,
		'ACTIVE' => 'Y'
	),
	false,
	false,
	array('ID', 'NAME', 'PREVIEW_PICTURE')
);
while($ar = $rs->GetNext())
{
	$arResulr['AVTOR_INFO'][$ar['ID']] = $ar;
}
```

## Оптимизации запроса

Код: в цикле идет запрос к элементу списка

Можно сделать так:

```
foreach($arResult["ORDERS"] as $key => $val)
{
        foreach($val["BASKET_ITEMS"] as $vvval)
        {
                $rsEls = CIBlockElement::GetByID();
        }
}
```

Но это будет медленно, потому что это долго искать среди объектов айдишники.

Лучше сделать так:
и искать айди среди айди

```
$arIDs = array();
foreach($arResult["ORDERS"] as $key => $val)
        {
                foreach($val["BASKET_ITEMS"] as $vvval)
                {
                        $arIDs[] = $vvval["PRODUCT_ID"];
                }
        }
if(!empty($arIDs))
{
        $rsEls = CIBlockElement::GetList(array(), array("ID" =>
$arIDs));
        ...
```

```
}
foreach($arResult["ORDERS"] as $key => $val)
{
        foreach($val["BASKET_ITEMS"] as $vval)
        {
                //наполняем данные, налаживая соответствие ID-
ков
        }
}
```

## Работа с БД

```
<?php
class d7SQL extends CBitrixComponent
{
        var $connection;
        var $sqlHelper;
        var $sql;
        function __construct($component = null)
        {
                parent::__construct($component);
                $this->connection =
\Bitrix\Main\Application::getConnection();
                $this->sqlHelper = $this->connection-
>getSqlHelper();
                //Строка запроса. Выбираем все логины, активных
пользователей
                $this->sql = 'SELECT LOGIN FROM b_user WHERE
ACTIVE = \''.$this->sqlHelper->forSql('Y', 1).'\' ';
        }
        /*
        * Возвращаем все значения
```

```
        */
        function var1()
        {
                $recordset = $this->connection->query($this-
>sql);
                        while ($record = $recordset->fetch())
                {
                        $arResult[]=$record;
                }
                return $arResult;
        }
        /*
        * Возвращаем первые два значения
        */
        function var2()
        {
                $recordset = $this->connection->query($this-
>sql,2);
                while ($record = $recordset->fetch())
                {
                        $arResult[]=$record;
                }
                return $arResult;
        }
        /*
        * Возвращаем два значения, отступая два элемента от
начала
        */
        function var3()
        {
                $recordset = $this->connection->query($this-
>sql,2,2);
```

```
                while ($record = $recordset->fetch())
                {
                $arResult[]=$record;
                }
                return $arResult;
        }
        /*
        * Возвращаем сразу первый элемент из запроса
        */
        function var4()
        {
                $arResult = $this->connection-
>queryScalar($this->sql);
                return $arResult;
        }
        /*
        * Выполняем запрос, не возвращая результат, т. е.
INSERT, UPDATE, DELETE
        */
        function var5()
        {
                $this->connection->queryExecute('UPDATE b_user
SET ACTIVE = \'N\' WHERE LOGIN=\'test\' ');//Заменить на UPDATE
        }
        /*
        * Модифицируем результат
        */
        function var6()
        {
                $recordset = $this->connection->query($this-
>sql);
                $recordset->addFetchDataModifier(
```

```
                        function ($data)
                        {
                                $data["LOGIN"] .= ": Логин
пользователя";
                                return $data;
                        }
                );
                while ($record = $recordset->fetch())
                {
                        $arResult[]=$record;
                }
                return $arResult;
        }
        public function executeComponent()
        {
                //$this->arResult = $this->var1();
                //$this->arResult = $this->var2();
                //$this->arResult = $this->var3();
                //$this->arResult = $this->var4();
                //$this->var5();
                $this->arResult = $this->var6();
                $this->includeComponentTemplate();
        }
};
```

В коде объявлены три переменные:

1. `connection` - хранит подключение к базе данных;
2. `sqlHelper` - хранит объект конкретного класса формирования sql запросов;
3. `sql` - sql запрос.

В конструкторе класса получаем соединение через приложения, которые, кроме всего прочего, являются точкой входа.

Так же у нас здесь формируется строка запроса: выбираются из таблицы пользователей логины всех пользователей, которые активны, то есть поле **ACTIVE** установлено в `Y`. В строке запроса использован метод [forSql], который делает входные параметры безопасными. Так же он может ограничить длину строки. В нашем случае он показан для примера: передан `Y` и указано что длина не должна быть больше одного символа.

Функция **var1**: в ней осуществляется запрос и с помощью [fetch] получаются результаты. Типизированные данные возвращаются сразу в виде типа, а не в виде строк или чисел.

Функция **var2**. Здесь выполняется тот же самый запрос, но указывается лимит на количество получаемых элементов. В нашем случае 2.

Функция **var3**. Выполняется тот же самый запрос, но указываются два дополнительных параметра. Такая запись означает, то, что возвратятся два элемента. Это последний параметр. И эти элементы возвращаются нам начиная со второй позиции. Это второй параметр. То есть отступаем два элемента и отдаем два, начиная с третьего элемента.

Функция **var4** - скалярный запрос, то есть когда возвращается первый, единственный результат выборки.

Функция **var5** - выполнение запроса, без получения результата. Это нужно в случае **INSERT**, **UPDATE**, **DELETE**.

Функция **var6** - модификация результата. Смотрим . С помощью метода [addFetchDataModifier] объявляется функцию, которая на вход принимает массив результата для одного элемента и после модификации его возвращает. В нашем случае не сложный пример:

просто к полю логин после двоеточия добавляется текст `Логин пользователя`.

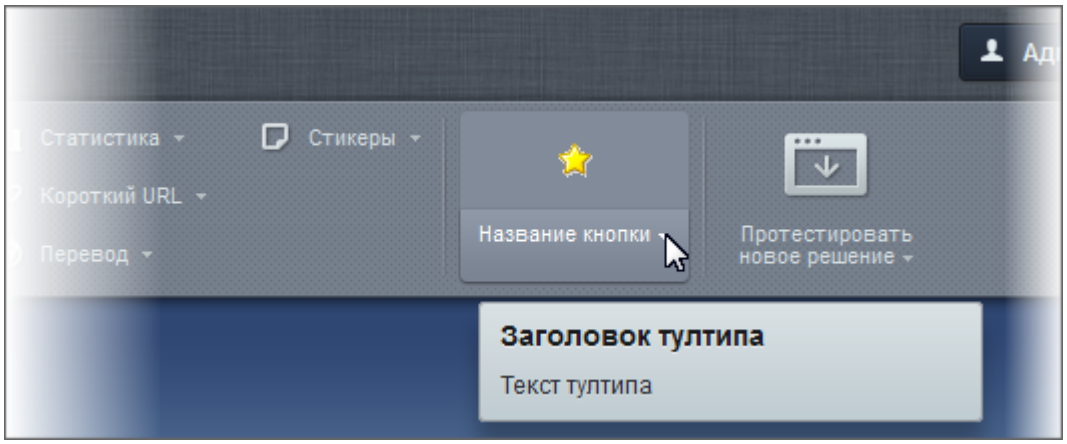
# Кастомизация админ. части

## Добавление кнопок на панель управления

```
        Array(
                "ID" => "ID кнопки", //определяет уникальность
кнопки
                "TEXT" => "Название кнопки",
                "TYPE" => "BIG", //BIG - большая кнопка, иначе
маленькая
                "MAIN_SORT" => 100, //индекс сортировки для
групп кнопок
                "SORT" => 10, //сортировка внутри группы
                "HREF" => "URL для перехода", //или
javascript:MyJSFunction())
                "ICON" => "icon-class", //название CSS-класса с
иконкой кнопки
                "SRC" => "путь к иконке кнопки",
                "ALT" => "Текст всплывающей подсказки", //
старый вариант
                "HINT" => array( //тултип кнопки
                        "TITLE" => "Заголовок тултипа",
                        "TEXT" => "Текст тултипа" //HTML
допускается
                ),
                "HINT_MENU" => array( //тултип кнопки
контекстного меню
                        "TITLE" => "Заголовок тултипа",
                        "TEXT" => "Текст тултипа" //HTML
допускается
                ),
                "MENU" => Array(
                        Array( //массив пунктов контекстного
```

```
меню
                                    "TEXT" => "название пункта",
                                    "TITLE" => "всплывающая
подсказака над пунктом",
                                    "SORT" => 10, //индекс
сортировки пункта
                                    "ICON" => "", //иконка пункта
                                    "ACTION" => "Javascript-код",
                                    "SEPARATOR" => true, //
определяет пункт-разделитель
                                    "DEFAULT" => true, //пункт по
умолчанию?
                                    "MENU" => Array() //массив
подменю
                                    )
                            )
                ),
        $bReplace = false //заменить существующую кнопку?
);
?>
```

Результат добавления в шаблон сайта:




# Добавление контекстного меню

```
$APPLICATION -> AddPanelButtonMenu($btnId, $arMenuItem)
```

где:

- `$btnId` – идентификатор кнопки;
- `$arMenuItem` – массив пунктов.