

# Граница между логикой в СУБД и на сервере приложений

Мария Щекочихина

Архитектор приложений

## Аннотация

Все приложения работают с данными. Пока объем данных не слишком большой и приложение простое, не принципиально, где размещать бизнес-логику: в СУБД или на сервере приложений. С ростом объема данных и усложнением приложений появляется вопрос, где граница между логикой в СУБД и на сервере приложений. Когда C#-разработчику нужно звать на помощь SQL-разработчика? Всегда ли можно обойтись своими силами?

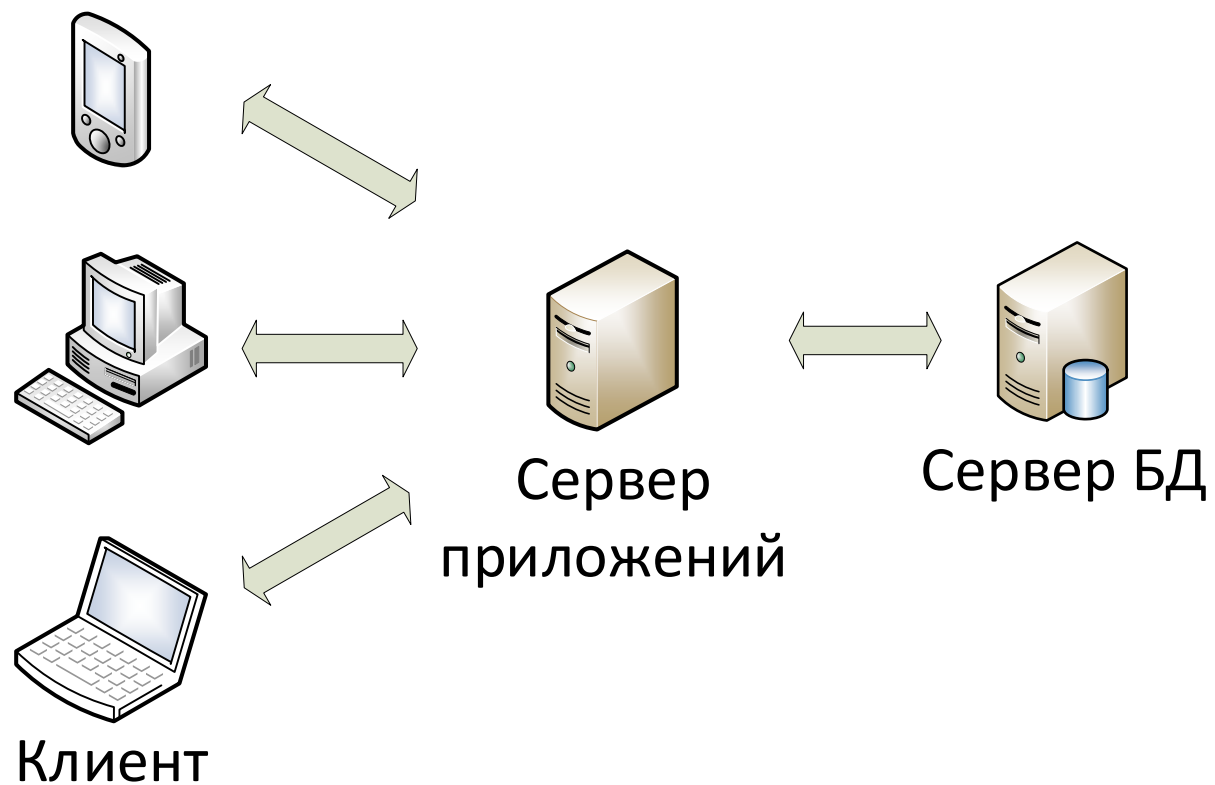
# План

- Контекст
- Где размещать бизнес-логику
- Критерии для анализа
- Смешанный вариант «и там и там»
- Выводы

# Контекст

- Что мы обычно имеем?

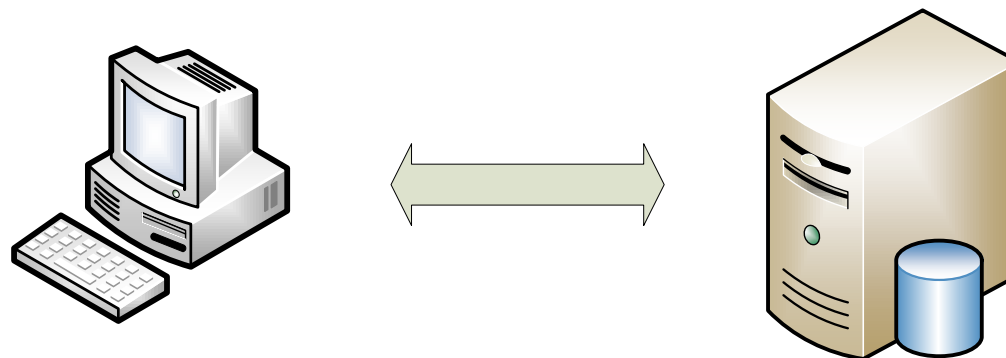
Трехзвенная  
архитектура



# Историческая справка

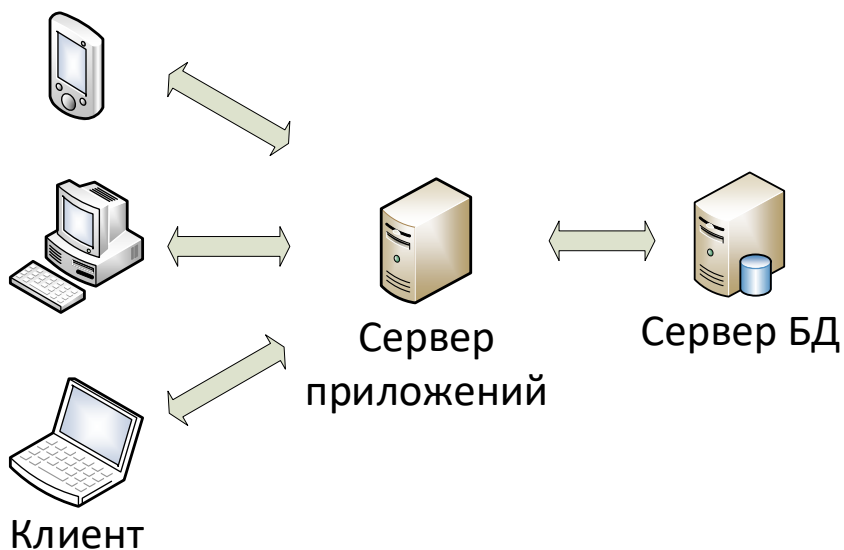
## Двухзвенная архитектура «Клиент – Сервер»

- Большая часть бизнес-логики на клиенте
- БД не масштабируется, клиент становится сложным и тяжелым, проблемы с версиями клиентов
- Появилась необходимость в третьем слое ради масштабирования и балансирования нагрузки на слои

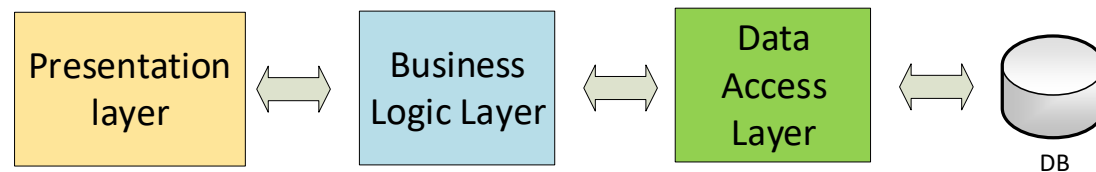


# Layer и tier

Tier – физические слои:  
сервер приложений, клиент, СУБД



Layer – логические слои: DAL (data access), BLL (business logic), presentation layer. Это способ организации кода



Layer может быть распределен по нескольким физическим tier.  
В частности, сервер БД может содержать выделенный слой BLL

## Пример расположения layer и tier

- Веб-приложение без своей БД
- Является оберткой к SAP-системе
- Слой BLL полностью расположен в SAP
- Данные в веб-приложение попадают из ФМ (похоже на ХП)
- Веб-приложение без слоя BLL, только DAL и клиентская логика

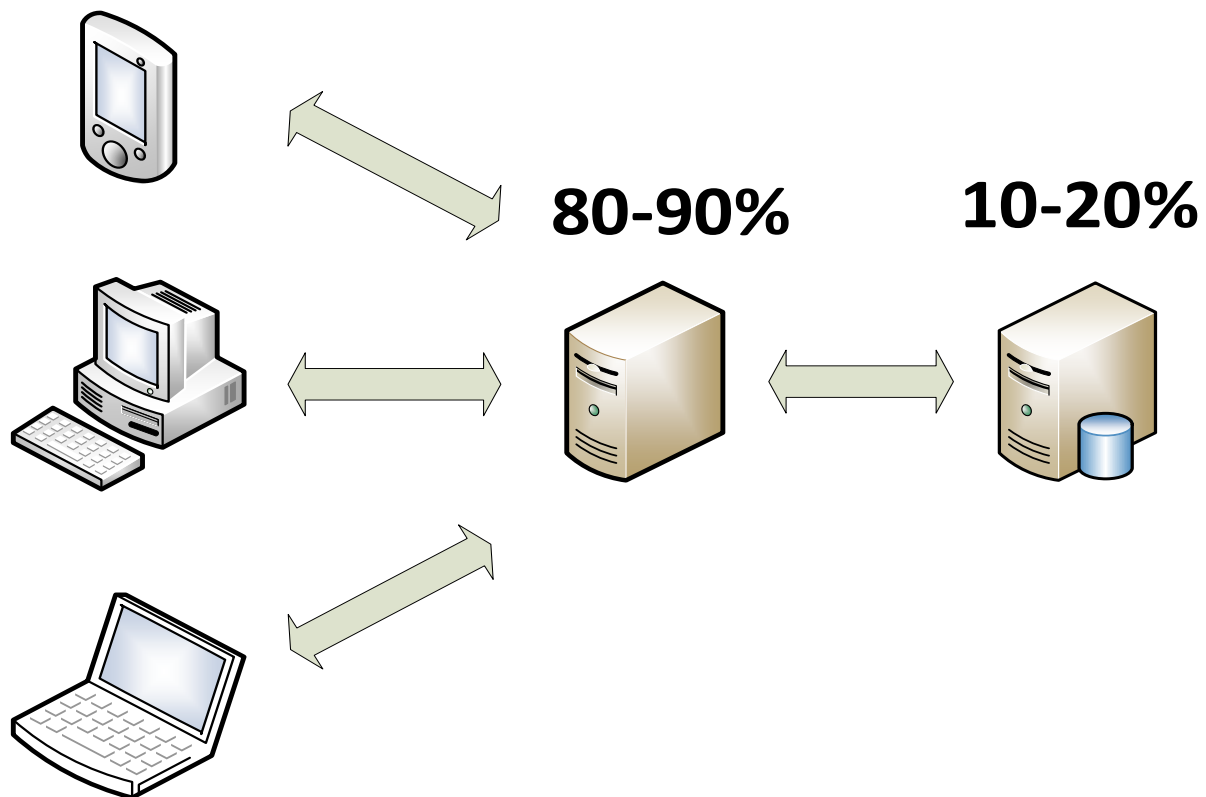
# План

- Контекст
- Где размещать бизнес-логику
- Критерии для анализа
- Смешанный вариант «и там и там»
- Выводы



# Бизнес-логика в трехзвенке

**10-15%**



## Где размещать бизнес-логику?

- Сервер приложений
- СУБД
- Сервер приложений и СУБД


## Пример. Только в БД

### Системы отчетности

- Интерфейс к БД, где нужно делать различные выборки
- Могут иметь промежуточный слой, в котором таблицы БД объединяются в логические объекты

# Проектирование

- На уровне архитектуры заложить, где будут находиться слои с бизнес-логикой
- Определить назначение каждого из этих слоев



Это задача  
архитектора

## Решение конкретных задач

- Опускаемся на уровень конкретной функции или задачи системы. Как мы можем решить задачу?
- Делаем оценку параметров задачи

# План

- Контекст
- Где размещать бизнес-логику
- Критерии для анализа
- Смешанный вариант «и там и там»
- Выводы

## Что оценивать


- Объем данных, передаваемых между слоями
- Потенциальный рост объема данных
- Сложность алгоритма получения и обработки
- Компетенции команды
- Наличие интеграционного слоя
- Процесс развертывания

## Объем данных

- Оценить объемы данных, которые будут считываться из БД
- Оценить потенциальный рост объема данных

## Зачем?

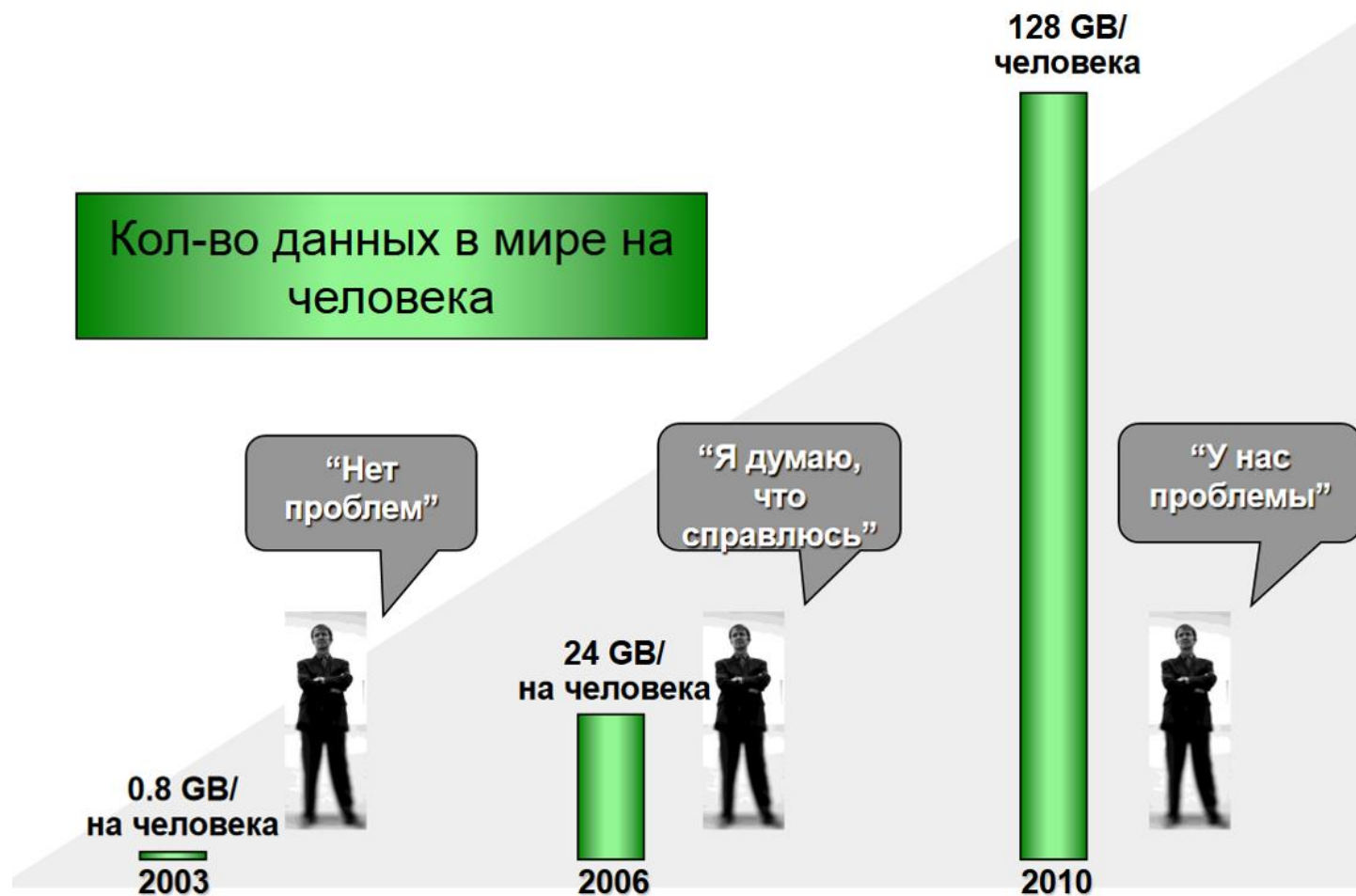
- Обработка 100 записей и обработка 1 млн записей – это разные задачи
- От объемов зависят физические мощности серверов (RAM, CPU, объемы дисков)



Оценку  
всегда  
проводить  
на реальных  
данных!

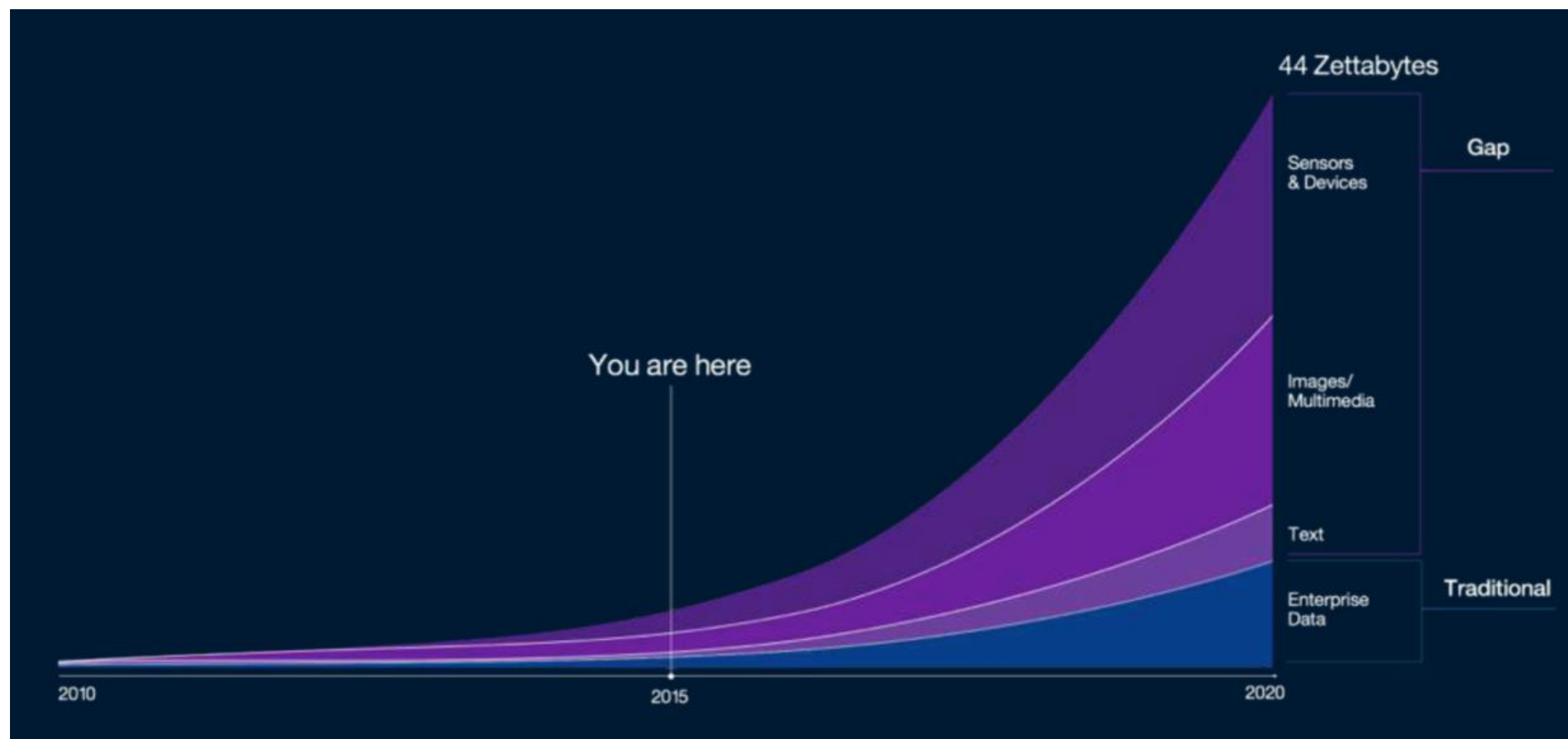


# Как быстро растут данные?



# Прогноз роста данных до 2020 года

CUSTIS®



## Сложность алгоритма получения и обработки

- Насколько сложные преобразования нужно выполнить, сколько их, какого типа операции: выборка, группировка, какие-то сложные правила?
- Как их эффективнее выполнять?

Для каких задач  
эффективнее  
использовать C#,  
а для каких – SQL?

## Разница между SQL и C#

- SQL – декларативный язык,  
позволяет использовать оптимизацию
- C# – императивный

## Пример

- Требуется по запросу выполнять расчет показателей по данным из базы
- Алгоритм расчета зависит от настроек, хранящихся в базе. Алгоритм получения «готовых» настроек нетривиальный
- Объем настроек в «сыром» виде ~1 млн записей (на момент формирования требований)
- Жестких требований по времени расчета нет

Какой вариант решения вы бы выбрали: в БД, в приложении или смешанный?

## Пример. Как было у нас

Задача полностью решалась на сервере приложений, перенесли в БД

### Было

- Считывали полностью таблицу, отдавали на сервер приложений, там делали обработку в поисках подходящих записей – ЦИКЛАМИ
- Объем увеличился с 1 млн строк до 12 млн
- Время работы увеличилось с 40 мин. до 3 часов

### Стало

- Из БД сразу отдавали только нужный объем
- Время работы: 3 мин

Почему сразу  
так не сделали?

## Компетенции команды

Если в команде нет SQL-разработчика, то не стоит делать задачи на SQL. Даже те, которые кажутся «простыми»

### Последствия

- Недооценка сложности задачи
- Глупые ошибки
- Конструкции вида «view вызывает view»

Что плохого  
в конструкции «view  
вызывает view»?

## View вызывает view

**Основная ошибка:** использовать view для инкапсуляции и переиспользования кода

- Такой метод нужно применять очень аккуратно
- Для использования view в запросе нужно понимать, что лежит в основе view на всю глубину

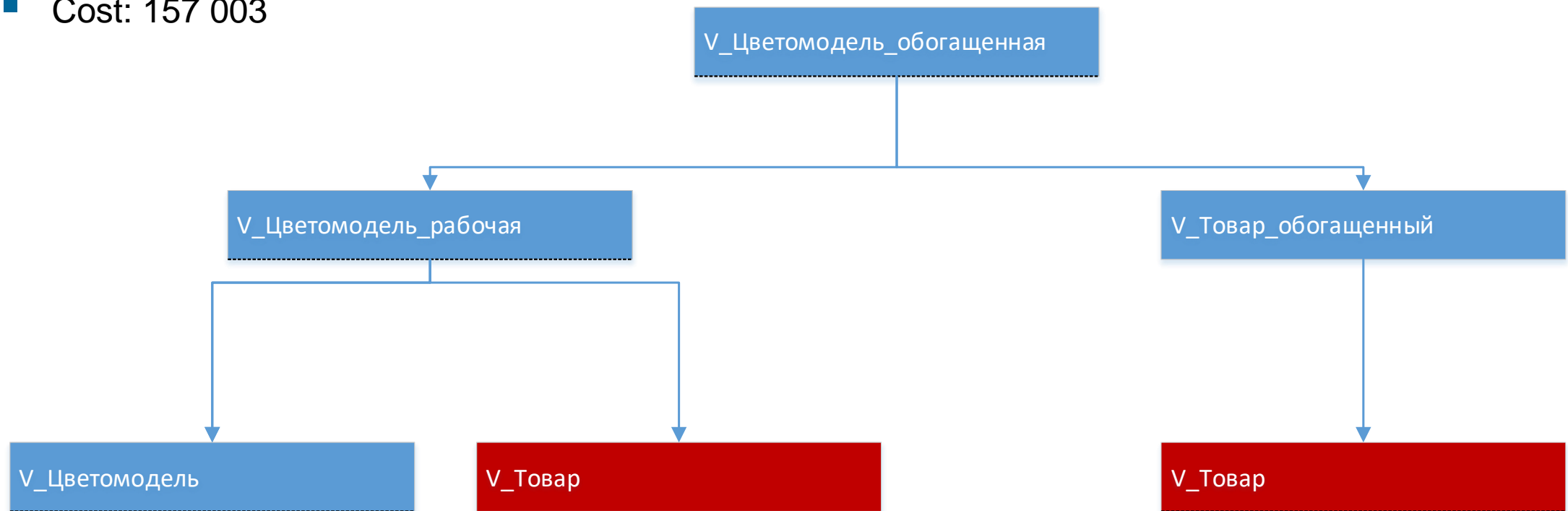
## Последствия

- Сложности в анализе работы view
- Появление неочевидных зависимостей



## Как не надо делать: пример

- Цепочка зависимостей view V\_Цветомодель\_обогащенная
- Cost: 157 003

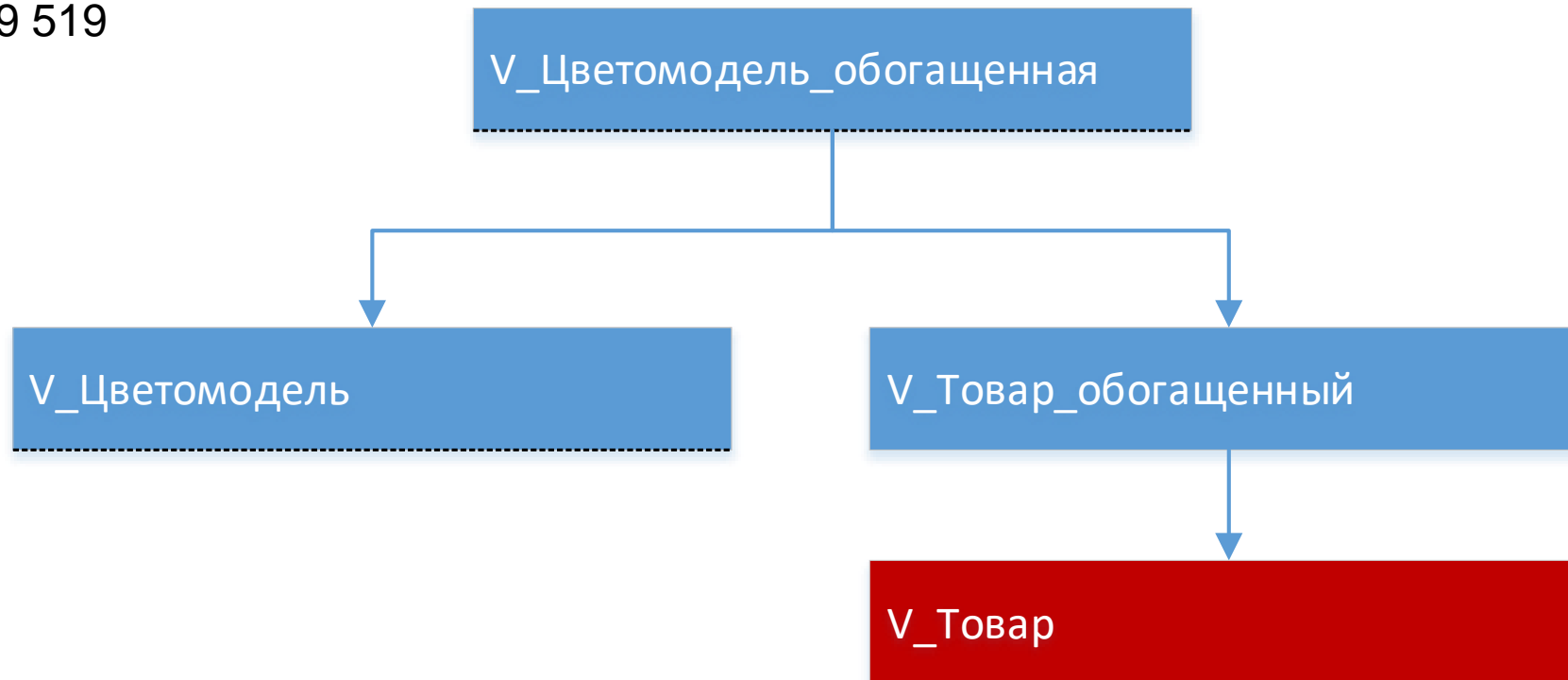


# Как не надо делать: план запроса

Optimizer goal: All rows					
Tree HTML Text XML					
Description	Object owner	Object name	Cost	Cardinality	Bytes
SELECT STATEMENT, GOAL = ALL_ROWS			157 003	9 600	20 390 400
MERGE JOIN OUTER			157 003	9 600	20 390 400
NESTED LOOPS			70 327	9 600	931 200
HASH JOIN			2 843	9 600	931 200
TABLE ACCESS FULL			68	9 600	729 600
TABLE ACCESS FULL			2 770	1 091 511	22 921 731
VIEW			7	1	
VIEW			7	1	13
WINDOW BUFFER PUSHED RANK			7	1	30
NESTED LOOPS SEMI			7	1	30
HASH JOIN			7	2	48
TABLE ACCESS BY INDEX ROWID			3	2	26
INDEX RANGE SCAN			2	2	
TABLE ACCESS FULL			4	55	605
INDEX RANGE SCAN			0	1	6
BUFFER SORT			156 996	1	2 027
VIEW			9	1	2 027
VIEW			9	1	2 027
VIEW			9	1	2 040
WINDOW BUFFER PUSHED RANK			9	1	116
HASH JOIN			9	1	116
NESTED LOOPS			6	3	66
NESTED LOOPS			6	3	66
SORT UNIQUE			3	3	24
TABLE ACCESS FULL			3	3	24
INDEX UNIQUE SCAN			0	1	
TABLE ACCESS BY INDEX ROWID			1	1	14
TABLE ACCESS BY INDEX ROWID			3	2	188
INDEX RANGE SCAN			2	2	

## Как исправить

- Избавляемся от двойного обращения к V\_Товар
- Cost: 89 519



# Как исправить

Optimizer goal: All rows					
Tree HTML Text XML					
Description	Object owner	Object name	Cost	Cardinality	Bytes
SELECT STATEMENT, GOAL = ALL_ROWS			89 519	9 600	20 390 400
NESTED LOOPS			89 519	9 600	20 390 400
HASH JOIN			2 843	9 600	931 200
TABLE ACCESS FULL			68	9 600	729 600
TABLE ACCESS FULL			2 770	1 091 511	22 921 731
VIEW			9	1	2 027
VIEW			9	1	2 040
WINDOW BUFFER PUSHED RANK			9	1	116
HASH JOIN			9	1	116
NESTED LOOPS			6	3	66
NESTED LOOPS			6	3	66
SORT UNIQUE			3	3	24
TABLE ACCESS FULL			3	3	24
INDEX UNIQUE SCAN			0	1	
TABLE ACCESS BY INDEX ROWID			1	1	14
TABLE ACCESS BY INDEX ROWID			3	2	188
INDEX RANGE SCAN			2	2	

## Наличие интеграционного слоя

Нужно ли интегрировать приложение с другими?  
Каким образом?

- Интеграция через БД. Например, вызовы процедур или view → располагаем слой BLL в БД
- Интеграция через веб-сервисы → располагаем слой BLL на сервере приложений

## Процесс развертывания

- Куда проще внести изменения:  
в БД или в приложение?

## Пример

- Веб-приложение, интерфейс с большим количеством бизнес-правил
- Объемы данных до 100 тысяч
- Задача: на основе данных формировать xml по утвержденной форме

Какой вариант решения вы бы выбрали: в БД, в приложении или смешанный?

## Пример. Как было у нас

- Команда 2 разработка: js+c# и чистый sql
- Очень ограниченные сроки
- Формирование xml сделали в виде хранимой процедуры



# План

- Контекст
- Где размещать бизнес-логику
- Критерии для анализа
- Смешанный вариант «и там и там»
- Выводы

## Зачем может понадобиться перенос логики из приложения в СУБД?

- Сокращение передаваемого объема данных между СУБД и сервером приложений
- Сокращение объема данных, которые попадают на обработку в приложение
- Работа с множествами
- Раскладка логики на блоки, которые выполняются наиболее подходящим инструментом

## Аргументы против смешанного варианта

- Усложняется отладка: когда все в одном месте, отладка проще
- Логика размывается, мы не всегда знаем, где искать



Что на это  
можно ответить?

## Что ответить

- Универсального решения в этом случае нет
- Всегда нужно анализировать ситуацию:  
смотреть на задачу, на данные, на объемы и т. д.
- По итогу принимать взвешенное решение,  
оптимальное для текущей ситуации

## Отладка

- Усложнение отладки зависит от квалификации участников и процесса разработки
- Если мы раскладываем логику на слои, то мы можем отлаживать эти слои независимо друг от друга. Таким образом упрощается поиск проблемных мест

## Размытие логики

- Нет ничего плохого в размытии логики, если мы понимаем, какие проблемы возникают при подходе «все в одном месте»
- Можно использовать подход с Linq, таким образом код будет «в одном месте»
- Нужно помнить, что реально выполнение разделено на 2 части: сервер приложения и запрос в БД

## Linq и ORM

- Разработчик полностью отгораживается от БД
- Нет прозрачности запроса: мы не знаем, какой именно запрос будет отправлен в БД
- Запрос неявный, нет плана запроса, соответственно не можем отладить запрос – получаем непредсказуемый результат
- Нет явной границы между моментом вычитки данных (lazyload)

## С ORM или без ORM

- С ORM – до 100 тыс. записей (выгружаемых на сервер приложения), объекты второго – третьего уровня вложенности
- Без ORM (или с микро-ORM) – все остальное



## Пример

- Пользовательский интерфейс, грид больше 20 колонок
- Информация собирается из 10+ таблиц (справочная информация, различные показатели со связанных объектов)
- Данные хранятся на низком уровне детализации, порядок 1 млн, делается агрегация до более высокого уровня, в результате на экране 3–4 тыс.строк
- Ограничение по времени открытия интерфейса – 3 минуты

Какой вариант решения вы бы выбрали: в БД, в приложении или смешанный?

## Пример. Как было у нас

- Интерфейс полностью построен на view
- View оптимизированы под выдачу экранов с конкретными наборами параметров
- Для ускорения выдачи результатов часть агрегации выполняется заранее

# План

- Контекст
- Где размещать бизнес-логику
- Критерии для анализа
- Смешанный вариант «и там и там»
- Выводы

## Когда звать SQL-разработчика

- Вычитка и работа с коллекциями больше 1 млн записей
- Для получения нужного набора данных требуется больше 7–10 соединений (join)
- Преобразование коллекций одного типа в другой
- View вызывает view

## Итог

- Если благодаря SQL мы можем сократить объем данных, передаваемых в приложение, это нужно делать
- Важно всегда делать оценку реальных объемов данных
- Выгодно, когда C#-разработчик понимает SQL, работа всегда с данными, объемы данных и сложность растут по экспоненте
- Не существует универсального метода решения задач – в каждой ситуации нужно взвешивать все за и против

# Спасибо за внимание!

Мария Щекочихина,  
архитектор приложений  
[mschekochikhina@custis.ru](mailto:mschekochikhina@custis.ru)

