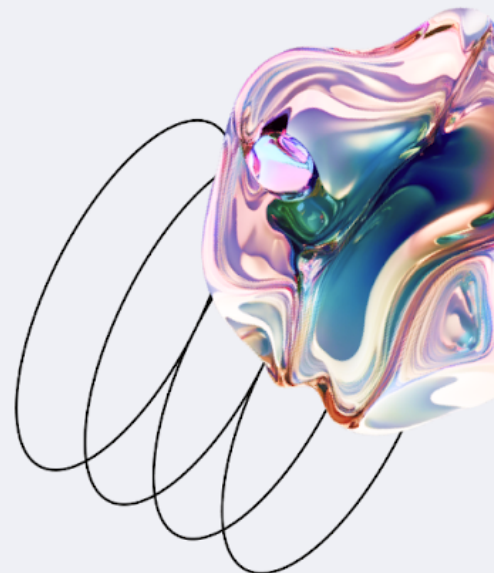


Получение денормализованных таблиц из нормализованных.

ETL: автоматизация
подготовки данных



Оглавление

Введение	2
Термины, используемые в лекции	2
Получение денормализованных таблиц из нормализованных.	3
Заключение	15
Что можно почитать еще?	16

Введение

Всем привет! Это наша третья лекция на курсе ETL и автоматизация подготовки данных.

На первом уроке мы с вами рассмотрели основные аспекты процесса ETL и поговорили о лучших практиках применения. Также мы обсудили существующие модели данных и немного углубились в реляционную модель. Нами был рассмотрен процесс нормализации таблиц. Мы поговорили о существующих нормальных формах и способах перехода из одной формы в другую. В предыдущей лекции мы обсудили что такое хранилище данных и затронули схему хранилища данных звезда.

На втором уроке мы поговорим о том, что же такое бизнес-аналитика и для чего она нужна, а также обсудим процесс подготовки данных. Кроме этого, вернемся к таблицам из схемы звезда (таблица фактов и таблица измерений) и погрузимся в особенности каждой из них.

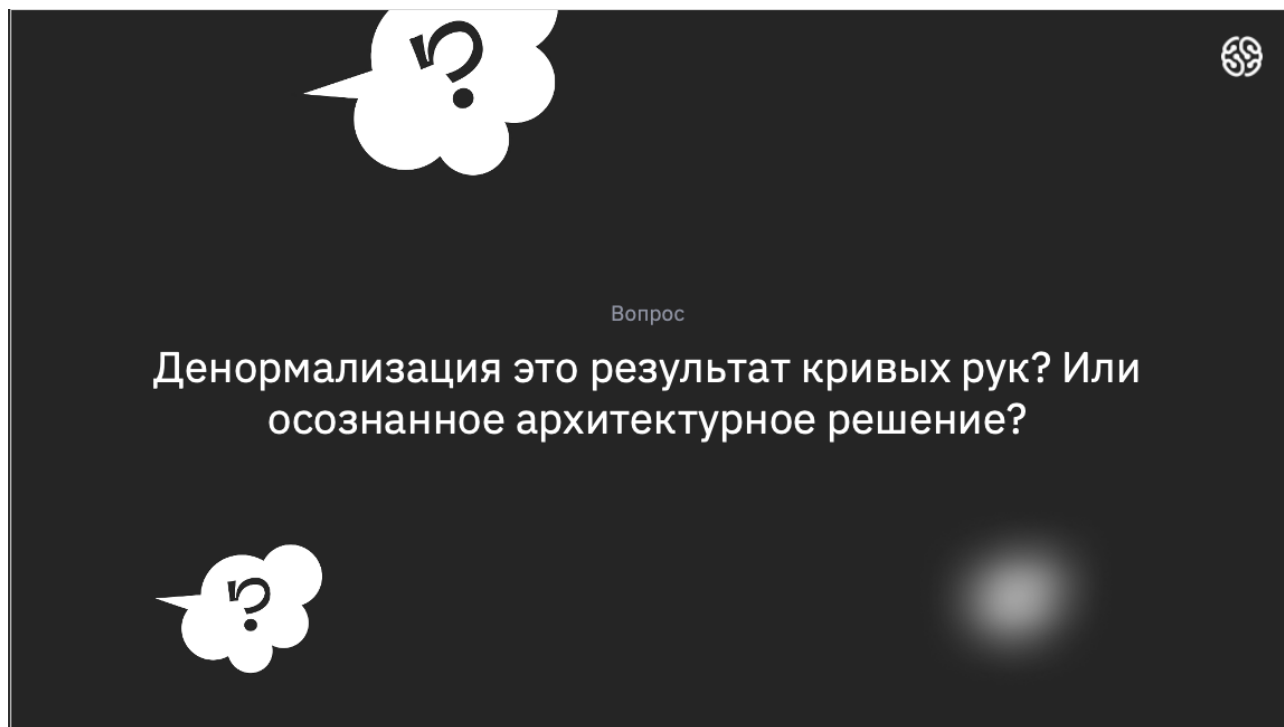
Сегодня на уроке мы разберем процесс денормализации таблиц, обсудим для чего нужна и когда применяется денормализация и каким образом можно получить денормализованные таблицы из нормализованных.

Термины, используемые в лекции

Денормализация (англ. denormalization) — намеренное приведение структуры базы данных в состояние, не соответствующее критериям нормализации, обычно проводимое с целью ускорения операций чтения из базы за счет добавления избыточных данных.

Получение денормализованных таблиц из нормализованных.

Предлагаю сегодняшнюю лекцию начать с вопроса



Вопрос: Денормализация это результат кривых рук или осознанное архитектурное решение?



Итак, давайте обратимся к определению денормализации. **Денормализация** (англ. denormalization) — намеренное приведение структуры базы данных в состояние, не

соответствующее критериям нормализации, обычно проводимое с целью ускорения операций чтения из базы за счет добавления избыточных данных.

Вернемся к первой лекции, когда мы с вами обсуждали процесс нормализации данных. Устранение аномалий данных в соответствии с теорией реляционных баз данных требует, чтобы любая база данных была нормализована, то есть соответствовала требованиям нормальных форм. Соответствие требованиям нормализации минимизирует избыточность данных в базе данных и обеспечивает отсутствие многих видов логических ошибок обновления и выборки данных.

Получается, говоря о денормализации мы нарушаем базовые принципы теории баз данных? Да это так, конечно это должно быть вызвано определенными причинами и конечно изначально данные должны быть нормализованны.

При запросах большого количества данных операция соединения нормализованных отношений выполняется неприемлемо долго. Вследствие этого в ситуациях, когда производительность таких запросов невозможно повысить иными средствами, может проводиться денормализация — композиция нескольких отношений (таблиц) в одну, которая, как правило, находится во второй, но не в третьей нормальной форме. Новое отношение фактически является хранимым результатом операции соединения исходных отношений.

За счет такого перепроектирования операция соединения при выборке становится ненужной и запросы выборки, которые ранее требовали соединения, работают быстрее.

Следует помнить, что денормализация всегда выполняется за счет повышения риска нарушения целостности данных при операциях модификации. Поэтому денормализацию следует проводить в крайнем случае, если другие меры повышения производительности невозможны. Идеально, если денормализованная БД используется только на чтение.

Кроме того, следует учесть, что ускорение одних запросов на денормализованной БД может сопровождаться замедлением других запросов, которые ранее выполнялись отдельно на нормализованных отношениях.

Когда полезно использовать денормализацию

Прежде чем браться разнормализовывать то, что уже однажды было нормализовано, естественно, нужно четко понимать, зачем это нужно? Следует убедиться, что выгода от применения метода перевешивает возможные негативные

последствия. Вот несколько ситуаций, в которых определенно стоит задуматься о денормализации.

1. **Сохранение исторических данных.** Данные меняются с течением времени, но может быть нужно сохранять значения, которые были введены в момент создания записи. Например, могут измениться имя и фамилия клиента или другие данные о его месте жительства и роде занятий. Задача должна содержать значения полей, которые были актуальны на момент создания задачи. Если этого не обеспечить, то восстановить прошлые данные корректно не удастся. Решить проблему можно, добавив таблицу с историей изменений. В таком случае SELECT-запрос, который будет возвращать задачу и актуальное имя клиента будет более сложным. Возможно, дополнительная таблица — не лучший выход из положения.
2. **Повышение производительности запросов.** Некоторые запросы могут использовать множество таблиц для доступа к часто запрашиваемым данным. Пример — ситуация, когда необходимо объединить до 10 таблиц для получения имени клиента и наименования товаров, которые были ему проданы. Некоторые из них, в свою очередь, могут содержать большие объемы данных. При таком раскладе разумным будет добавить напрямую поле `client_id` в таблицу `products_sold`.
3. **Ускорение создания отчетов.** Бизнесу часто требуется выгружать определенную статистику. Создание отчетов по «живым» данным может требовать большого количества времени, да и производительность всей системы может в таком случае упасть. Например, требуется отслеживать клиентские продажи за определенный промежуток по заданной группе или по всем пользователям разом. Решающий эту задачу запрос в «боевой» базе перелопатит ее полностью, прежде чем подобный отчет будет сформирован. Нетрудно представить, насколько медленнее все будет работать, если такие отчеты будут нужны ежедневно.
4. **Предварительные вычисления часто запрашиваемых значений.** Всегда есть потребность держать наиболее часто запрашиваемые значения наготове для регулярных расчетов, а не создавать их заново, генерируя их каждый раз в реальном времени.

Вывод напрашивается сам собой: не следует обращаться к денормализации, если не стоит задач, связанных с производительностью приложения. Но если чувствуется, что система замедлилась или скоро замедлится, впору задуматься о применении данной техники. Однако, прежде чем обращаться к ней, стоит применить и другие возможности улучшения производительности: оптимизацию

запросов и правильную индексацию.

Не все так гладко

Очевидная цель денормализации — повышение производительности. Но всему есть своя цена. В данном случае она складывается из следующих пунктов:

- **Место на диске.** Ожидаемо, поскольку данные дублируются.
- **Аномалии данных.** Необходимо понимать, что с определенного момента данные могут быть изменены в нескольких местах одновременно. Соответственно, нужно корректно менять и их копии. Это же относится к отчетам и предварительно вычисляемым значениям. Решить проблему можно с помощью триггеров, транзакций и хранимых процедур для совмещения операций.
- **Документация.** Каждое применение денормализации следует подробно документировать. Если в будущем структура базы поменяется, то в ходе этого процесса нужно будет учесть все прошлые изменения — возможно, от них вообще можно будет к тому моменту отказаться за ненадобностью. (Пример: в клиентскую таблицу добавлен новый атрибут, что приводит к необходимости сохранения прошлых значений. Чтобы решить эту задачу, придется поменять настройки денормализации).
- **Замедление других операций.** Вполне возможно, что применение денормализации замедлит процессы вставки, модификации и удаления данных. Если подобные действия проводятся относительно редко, то это может быть оправдано. В этом случае мы разбиваем один медленный SELECT-запрос на серию более мелких запросов по вводу, обновлению и удалению данных. Если сложный запрос может серьезно замедлить всю систему, то замедление множества небольших операций не отразится на качестве работы приложения столь драматических образом.
- **Больше кода.** Пункты 2 и 3 потребуют добавления кода. В то же время они могут существенно упростить некоторые запросы. Если денормализации подвергается существующая база данных, то потребуется модифицировать эти запросы, чтобы оптимизировать работу всей системы. Также понадобится обновить существующие записи, заполнив значения добавленных атрибутов — это тоже потребует написания некоторого количества кода.

Давайте рассмотрим основные маркеры, которые указывают на то, что денормализация нам необходима.

Большое количество соединений таблиц.

В запросах к полностью нормализованной базе нередко приходится соединять до десятка, а то и больше, таблиц. А каждое соединение — операция весьма ресурсоемкая. Как следствие, такие запросы кушают ресурсы сервера и выполняются медленно.

В такой ситуации может помочь:

- денормализация путем сокращения количества таблиц. Лучше объединять в одну несколько таблиц, имеющих небольшой размер, содержащих редко изменяемую (как часто говорят, условно-постоянную, или нормативно-справочную) информацию, причем информацию, по смыслу тесно связанную между собой.
В общем случае, если в большом количестве запросов требуется объединять более пяти или шести таблиц, следует рассмотреть вариант денормализации базы данных.
- Денормализация путём ввода дополнительного поля в одну из таблиц. При этом появляется избыточность данных, требуются дополнительные действия для сохранения целостности БД.

Расчетные значения.

Зачастую медленно выполняются и потребляют много ресурсов запросы, в которых производятся какие-то сложные вычисления, особенно при использовании группировок и агрегатных функций (Sum, Max и т.п.). Иногда имеет смысл добавить в таблицу 1-2 дополнительных столбца, содержащих часто используемые (и сложно вычисляемые) расчетные данные.

Предположим, что необходимо определить общую стоимость каждого заказа. Для этого сначала следует определить стоимость каждого продукта (по формуле «количество единиц продукта» * «цена единицы продукта» – скидка). После этого необходимо сгруппировать стоимости по заказам.

Выполнение этого запроса является достаточно сложным и, если в базе данных хранятся сведения о большом количестве заказов, может занять много времени. Вместо выполнения такого запроса можно на этапе размещения заказа определить его стоимость и сохранить ее в отдельном столбце таблицы заказов. В этом случае для получения требуемого результата достаточно извлекать из данного столбца предварительно рассчитанные значения.

Создание столбца, содержащего предварительно рассчитываемые значения,

позволяет значительно сэкономить время при выполнении запроса, однако требует своевременного изменения данных в этом столбце.

Длинные поля.

Если у нас в базе данных есть большие таблицы, содержащие длинные поля (Blob, Long и т.п.), то серьезно ускорить выполнение запросов к такой таблице мы сможем, если вынесем длинные поля в отдельную таблицу. Хотим мы, скажем, создать в базе каталог фотографий, в том числе хранить в blob-полях и сами фотографии (профессионального качества, с высоким разрешением, и соответствующего размера). С точки зрения нормализации абсолютно правильной будет такая структура таблицы:

ID фотографии

ID автора

ID модели фотоаппарата

сама фотография (blob-поле).

А сейчас представим, сколько времени будет работать запрос, подсчитывающий количество фотографий, сделанных каким-либо автором...



Правильным решением (хотя и нарушающим принципы нормализации) в такой ситуации будет создать еще одну таблицу, состоящую всего из двух полей — ID фотографии и blob-поле с самой фотографией. Тогда выборки из основной таблицы (в которой огромного blob-поля сейчас уже нет) будут идти моментально, ну а когда захотим посмотреть саму фотографию — что ж, подождем...

А теперь поговорим о том какие существуют способы реализации денормализации данных. На самом деле существуют два основных способа применяемых при денормализации данных:

- Дублирование
- Предварительная подготовка

Дублирование данных

Допустим у нас есть две таблицы пользователи и города


 <i>Users</i>	 <i>Cities</i>
id	id
name	title
city_id	country

Чтобы выбрать название город или страну пользователя необходимо будет сделать либо два запроса, либо один join.

```
1 SELECT * FROM users u
2 JOIN cities c ON(c.id = u.city_id)
```

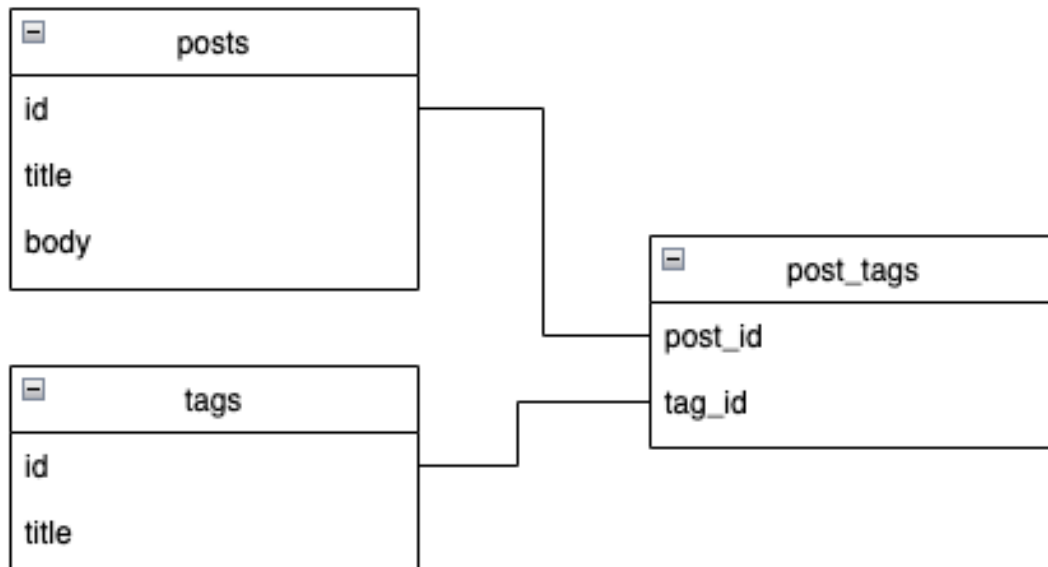
Чтобы этого не делать и использовать преимущество дублирования данных, нам надо добавить колонку city_title в таблицу пользователи и тогда мы сможем делать выборку одним запросом.

```
1 SELECT id, name, city_title FROM users
```

 <i>Users</i>
id
name
city_id
city_title

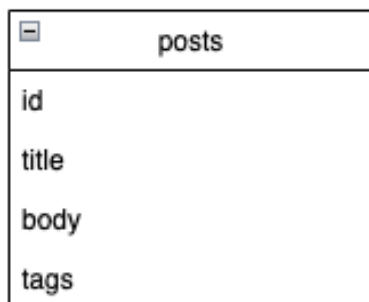
Связи один ко многим

Связи один ко многим также можно оптимизировать используя дублирование. Представим в качестве примера таблицы постов с метками в блоге:




```
1 SELECT * FROM tags t
2 JOIN post_tags pt ON (pt.tag_id = t.id)
3 WHERE pt.posts.id=1;
```


Чтобы выбрать метки нам понадобится сделать два отдельных запроса или один join. Чтобы этого избежать можно сохранить список меток в отдельном поле в таблице постов и тогда не будет необходимости делать join или дополнительные запросы к базе. Структура таблицы представлена на рисунке ниже.



Предварительная подготовка данных

Много времени занимают запросы, в которых происходят агрегация или вычисления. Поэтому, так же как мы с вами сохраняли дубликаты мы можем сохранять вычисленные значения. Например, у нас есть таблица студент и таблица группа.


 Students
id
name
group_id

 Groups
id
name

Нам нужно узнать какое количество студентов в группе.


```
1 SELECT count(*) FROM students
2 WHERE group_id = 133
```


Чтобы не производить каждый раз эти вычисления мы можем просто добавить поле количество студентов в таблицу групп.

 Groups
id
name
student_count


Теперь мы можем получать количество студентов в группе одним простым запросом, но мы должны помнить, что при добавление нового студента в группу мы должны обновлять поле student_count.


Я думаю, вы уже увидели, что такая организация данных совпадает с тем, что мы обсуждали на первой лекции, а именно таблица фактов и таблица измерений. У нас есть основная таблица группа и таблица измерений, содержащая количество студентов, количество студентов, получающих стипендию и количество студентов платников.

 Факты
group_id


 Измерения
student_count
scholarship_students
fee-paying_students


Давайте рассмотрим еще несколько примеров денормализации таблиц.

 <i>product</i>
id
product_name
unit
price_per_unit


 <i>product</i>
id
product_name
unit
price_per_unit
units_in_stock


Единственное нововведение в таблице `product` — строка `units_in_stock`. В нормализованной модели мы можем вычислить это значение следующим образом: заказанное наименование — проданное — (предложенное) — списанное (`units ordered — units sold — (units offered) — units written off`). Вычисление повторяется каждый раз, когда клиент запрашивает товар. Это довольно затратный по времени процесс. Вместо этого можно вычислять значение заранее так, чтобы к моменту поступления запроса от покупателя, все уже было наготове. С другой стороны, атрибут `units_in_stock` должен обновляться после каждой операции ввода, обновления или удаления.

 <i>task</i>
id
client_id
client_name
start_time
end_time
task_outcome_id


 <i>task</i>
id
client_id
client_name
start_time
end_time
user_assigned
user_first_last_name
task_outcome_id


В таблицу `task` добавлено два новых атрибута: `client_name` и `user_first_last_name`. Оба они хранят значения на момент создания задачи — это нужно, потому что каждое из них может поменяться с течением времени. Также нужно сохранить внешний ключ, который связывает их с исходным пользовательским и клиентским ID. Есть и другие значения, которые нужно хранить — например, адрес клиента или информация о включенных в стоимость налогах вроде НДС.

 <i>product_offered</i>
id
product_id
task_id
units

 <i>product_offered</i>
id
product_id
task_id
units
price_per_unit
price

Денормализованная таблица `product_offered` получила два новых атрибута: `price_per_unit` и `price`. Первый из них необходим для хранения актуальной цены на момент предложения товара. Нормализованная модель будет показывать лишь ее текущее состояние. Поэтому, как только цена изменится, изменится и «ценовая история». Нововведение не просто ускорит работу базы, оно улучшает функциональность. Строка `price` вычисляет значение $units_sold * price_per_unit$. Таким образом, не нужно делать расчет каждый раз, как понадобится взглянуть на список предложенных товаров. Это небольшая цена за увеличение производительности.

 <i>product_sold</i>
id
product_id
task_id
units

 <i>product_sold</i>
id
product_id
task_id
units
price_per_unit
price

Изменения в таблице `product_sold` сделаны по тем же соображениям. С той лишь разницей, что в данном случае речь идет о проданных наименованиях товара.

Как определить, когда денормализация оправдана?

Затраты и выгоды.

Один из способов определить, насколько оправданы те или иные шаги, — провести анализ в терминах затрат и возможных выгод. Во сколько обойдется денормализованной моделью данных?

Определить требования (чего хотим достичь) -> определить требования к данным (что нужно соблюдать) -> найти минимальный шаг, удовлетворяющий эти требования -> подсчитать затраты на реализацию -> реализовать.

Затраты включают в себя физические аспекты, такие как дисковое пространство, ресурсы, необходимые для управления этой структурой, и утраченные возможности из-за временных задержек, связанных с обслуживанием этого процесса. За

денормализацию нужно платить. В денормализованной базе данных повышается избыточность данных, что может повысить производительность, но потребует больше усилий для контроля за связанными данными. Усложнится процесс создания приложений, поскольку данные будут повторяться и их труднее будет отслеживать. Кроме того, осуществление ссылочной целостности оказывается не простым делом — связанные данные оказываются разделенными по разным таблицам.

К преимуществам относится более высокая производительность при выполнении запроса и возможность получить при этом более быстрый ответ. Кроме того, можно получить и другие преимущества, в том числе увеличение пропускной способности, уровня удовлетворенности клиентов и производительности, а также более эффективное использование инструментария внешних разработчиков.

Частота запросов и устойчивость производительности.

Например, 72% из тысячи запросов, ежедневно генерируемых предприятием, представляют собой запросы уровня сводных, а не детальных данных. При использовании таблицы сводных данных запросы выполняются примерно за 6 секунд вместо 4 минут, т.е. время обработки меньше на 3000 минут. Даже с поправкой на те 100 минут, которые необходимо еженедельно тратить на поддержку таблиц сводных данных, в итоге экономится 2500 минут в неделю, что полностью оправдывает создание таблицы сводных данных. Со временем может случиться так, что большая часть запросов будет обращена не к сводным данным, а к детальным данным. Чем меньше число запросов, использующих таблицу сводных данных, тем проще от нее отказаться, не затрагивая другие процессы.

Перечисленные выше критерии не единственные, которые следует учитывать, принимая решение о том, следует ли делать следующий шаг в оптимизации. Необходимо учитывать и другие факторы, в том числе приоритеты бизнеса и потребности конечных пользователей. Пользователи должны понимать, как с технической точки зрения на архитектуру системы влияет требование пользователей, желающих, чтобы все запросы выполнялись за несколько секунд. Проще всего добиться этого понимания — очертить затраты, связанные с созданием таких таблиц и их управлением.

Как грамотно реализовать денормализацию.

Сохранить детальные таблицы

Чтобы не ограничивать возможности базы данных, важные для бизнеса, необходимо придерживаться стратегии сосуществования, а не замены, т.е. сохранить детальные таблицы для глубинного анализа, добавив к ним

денормализованные структуры. Например, счетчик посещений. Для бизнеса необходимо знать количество посещений веб-станции. Но для анализа (по периодам, по странам ...) нам очень вероятно понадобятся детальные данные – таблица с информацией о каждом посещении.

Использование триггеров

Можно денормализовать структуру базы данных и при этом продолжать пользоваться преимуществами нормализации, если пользоваться триггерами баз данных для сохранения целостности (integrity) информации, идентичности дублирующихся данных.

К примеру, при добавлении вычисляемого поля на каждый из столбцов, от которых вычисляемое поле зависит, вешается триггер, вызывающий единую хранимую процедуру (это важно!), которая и записывает нужные данные в вычисляемое поле. Надо только не пропустить ни один из столбцов, от которых зависит вычисляемое поле.

Программная поддержка

если не использовать встроенные триггеры и хранимых процедуры, то заботиться об обеспечении непротиворечивости данных в денормализованной базе должны разработчики приложений. По аналогии с триггерами, должна быть одна функция, обновляющая все поля, зависящие от изменяемого поля.

Выводы

При денормализации важно сохранить баланс между повышением скорости работы базы и увеличением риска появления противоречивых данных, между облегчением жизни программистам, пишущим Select'ы, и усложнением задачи тех, кто обеспечивает наполнение базы и обновление данных. Поэтому проводить денормализацию базы надо очень аккуратно, очень выборочно, только там, где без этого никак не обойтись.

Если заранее нельзя подсчитать плюсы и минусы денормализации, то изначально необходимо реализовать модель с нормализованными таблицами, и лишь затем, для оптимизации проблемных запросов проводить денормализацию.

Денормализацию важно внедрять постепенно и только для тех случаев, когда присутствуют повторные выборки связанных данных с разных таблиц. Помните, при дублировании данных вырастит количество записей, но уменьшится количество чтений. Рассчитываемые данные также удобно сохранять в колонках, чтобы избежать ненужных агрегатных выборок.

Заключение

Сегодня мы рассмотрели, что же такое бизнес-аналитика и для чего она нужна. Поговорили о том какие этапы необходимо пройти для проведения качественной аналитики и какие цели должен ставить перед собой анализ данных. Так же мы рассмотрели таблицы фактов и измерений, узнали их особенности и различия.

Что можно почитать еще?

1. Для лучшего изучения темы можно почитать Бизнес-процессы. Языки моделирования, методы, инструменты. Авторы: Франк Шёнталер, Готфрид Фоссен, Андреас Обервайс, Томас Карле.
2. Путь аналитика. Практическое руководство IT-специалиста Авторы: Андрей Перерва, Вера Иванова.