

Внесение изменений в существующую базу данных. Обратная совместимость

Если у вас есть рабочее приложение с базой данных, вы не можете просто так брать и вносить изменения в её структуру или данные. Ведь каждое изменение может повлиять на клиента. Что, например, если вы добавите изменение - сделаете поле "адрес" обязательным - но не заполните это поле у старых клиентов без адреса? Они не смогут войти в приложение так как запросы к базе данных для этих клиентов будут останавливаться с ошибкой "не заполнено обязательное поле".

То есть, мы рассматриваем случай, когда вы НЕ С НУЛЯ создаёте базу данных (в этом случае для разработчика хватит с вас физической модели или SQL DDL), а вам нужно написать целую постановку о том, КАК ПРАВИЛЬНО и КАКИЕ внести изменения в базу данных. В каждом конкретном случае нужно внимательно понять, какие существующие данные будет затрагивать изменения и как внести изменения чтобы попытаться сохранить обратную совместимость. Вот примеры изменений:

1. Добавление новой таблицы в существующую базу данных.
2. Наполнение таблицы данными (значениями).
3. Введение нового столбца в существующую таблицу.
4. Модификация существующих значений в таблице.
5. Установка ограничения "NOT NULL" для столбца таблицы.
6. Определение внешнего ключа (Foreign Key) для столбца.
7. Удаление или переименование таблиц и столбцов: Изменение структуры базы данных путём удаления существующих или переименования таблиц и полей.
8. Изменение типов данных в столбцах: Обновление типа данных для одного или нескольких столбцов, например, с изменением размера или формата.
9. Индексирование столбцов для оптимизации запросов: Добавление индексов к столбцам для улучшения производительности при выполнении запросов.
10. Разработка и изменение триггеров: Создание или модификация триггеров, которые автоматически выполняют определенные операции в ответ на изменения в базе данных.
11. Создание и изменение хранимых процедур и функций: Программирование на уровне базы данных для автоматизации сложных или повторяющихся задач.
12. Настройка ограничений целостности и правил каскадного удаления/обновления: Установка правил для поддержания целостности данных при удалении или обновлении связанных записей.

13. Нормализация и денормализация данных: Проектирование структуры базы данных для оптимизации хранения и запросов.

Обратная совместимость в контексте баз данных — это способность новой версии базы данных работать с существующими приложениями и системами, которые были разработаны для работы с предыдущими версиями этой базы данных.

1. Сохранение интерфейса: Обратная совместимость обычно подразумевает, что интерфейсы, такие как запросы SQL или схемы данных, остаются неизменными или изменяются таким образом, чтобы не нарушить существующую функциональность.
2. Поддержка старых данных: Новые версии должны уметь работать со старыми данными без необходимости их конвертации или изменения.
3. Минимизация изменений в приложениях: Цель состоит в том, чтобы требовать как можно меньше изменений в существующих приложениях, которые используют базу данных.

Общие вопросы, над которыми вам нужно подумать, чтобы понять по какому алгоритму (какие действия) нужно выполнить, чтобы аккуратно внести изменения в базу данных и попытаться сохранить обратную совместимость:

1. Анализ взаимодействия с существующими данными: Важно определить, где и как данные в настоящее время заполняются и используются. Необходимо учитывать, что в процессе обновления данных может возникнуть необходимость в двойном заполнении данных: в старом и в новом местах хранения.
2. Обработка параллельных операций во время обновления: Рассмотрите, что произойдет, если система будет обновляться без простоя, и пользователи продолжат использовать функциональность. Как это повлияет на процесс обновления и целостность данных?
3. Сценарии заполнения старых данных: Предусмотрите сценарий, что во время обновления новые механизмы заполнения еще не активированы, а старые данные уже заполняются. Это может привести к проблемам, особенно если новые поля являются обязательными (NOT NULL).
4. Стратегия отката при неудачных обновлениях: Разработайте план отката на случай, если обновление приведет к ошибкам. Необходимо иметь четкий алгоритм действий для восстановления предыдущего состояния базы данных.

5. Оценка объема обновляемых данных: Учитывайте объем данных, которые требуется обновить (несколько тысяч или миллионов записей). Оцените время, необходимое для обновления, учитывая количество атрибутов, алгоритмы заполнения (по умолчанию или вычисляемые значения) и общее количество существующих данных.
6. Очистка и управление данными: Решите, нужно ли удалять устаревшие данные или оставить их для поддержки старых отчетов и исторических данных. Это включает в себя решения об очистке "мусора" и управлении жизненным циклом данных.

Вы должны понять, что данные не существуют в вакууме. Смотрите на систему целиком - у вас есть какой-то frontend, есть backend, и есть база данных (в простом случае). Вы должны понимать кем, когда, и как используются ваши данные. Только тогда вы поймёте, как сможете внести изменения, чтобы ничего не сломать.

Данные

Кем, когда,
и как они
создаются?

Кем, когда,
и как они
читаются?

Кем, когда, и
как они
изменяются?

Кем, когда,
и как они
удаляются?

Пример сценария для безопасного внесения изменения - добавление нового обязательного (not null) поля

1. Реализация нового поля

- Осуществите добавление нового поля в соответствующую таблицу базы данных. Это первый шаг к расширению функциональности системы.

2. Обновление серверной и клиентской логики (backend and frontend)

- На этапе разработки внедрите логику проверки нового поля для всех операций создания и редактирования записей. Обеспечьте, чтобы в случае отсутствия данных возвращалась ошибка.
- На уровне пользовательского интерфейса (UI) также реализуйте требования к обязательному заполнению нового поля.

3. Адаптация функционала получения данных

- Обновите механизмы извлечения данных, чтобы они поддерживали отображение и обработку нового поля. Это включает в себя списки элементов и подробную информацию о сущностях, которая может формироваться из разных таблиц базы данных.

4. Инициализация старых записей

- Произведите заполнение нового поля для всех существующих записей. Это может быть реализовано либо путём установки значений по умолчанию, либо с использованием специфического алгоритма, соответствующего логике вашего приложения. Ведь если вы не заполните поле для старых записей - при установке ограничения NOT NULL на атрибут вы сломаете логику (СУБД будет отдавать ошибку по старым записям).

5. Проверка всей логики на тестовой среде

6. Установка ограничения NOT NULL

- После того, как все существующие записи будут обновлены, установите для нового поля ограничение NOT NULL, делая его обязательным для всех будущих операций.

7. Вывод на продакшн - на боевую среду

Вам, как специалисту проектирования, нужно описать каждую из пяти задач для разработчика. Что именно нужно сделать, с какими данными и так далее. Когда вы опишите задачи, не бойтесь обсудить с разработкой поэтапное выполнение задач, для того, чтобы двигаться постепенно и на каждом этапе проверять что вы всё спроектировали корректно, и в случае ошибки быстро откатиться "назад". Примеры разбиения задач на этапы:

- Этап 1: Задачи 1 и 2. Это позволит начать процесс адаптации системы под новый функционал и обеспечит корректное заполнение нового поля для всех новых и изменяемых записей.
- Этап 2: Задачи 3 и 4, 5. Реализация этих задач обеспечит полную интеграцию нового поля в систему и подготовку существующих данных к установке ограничения NOT NULL.
- Этап 3: Задачи 6 и 7. Финальный этап, который закрепит обязательность нового поля в структуре данных.

Дополнительные рекомендации:

- Мониторинг и откат: На каждом этапе важно внимательно мониторить систему на предмет ошибок или проблем в работе. Также стоит подготовить планы отката для каждого этапа на случай возникновения критических проблем.
- Коммуникация с заинтересованными сторонами и командами (в том числе внешними потребителями вашего API!): Убедитесь, что все ключевые участники проекта (разработчики, аналитики, конечные пользователи и т.д.) осведомлены о предстоящих изменениях и их влиянии на систему.
- Тестирование: Обширное тестирование на каждом этапе поможет выявить и устранить потенциальные проблемы до их внедрения в продакшн.

Следуя этому детализированному плану, можно максимально снизить риски и обеспечить бесперебойное внедрение новых изменений в систему.

Пример сценария для безопасного внесения изменения - разделение одного поля на два (например - поле ФАМИЛИЯ ИМЯ делим на два поля - поле ФАМИЛИЯ и поле ИМЯ)

Задачи:

1. Добавление новых атрибутов

- Внедрите два новых поля в соответствующую таблицу базы данных для хранения фамилии и имени отдельно.

2. Обновление функциональности сервера и UI (backend and frontend)

- Реализуйте поддержку новых полей в процессах создания и редактирования записей в серверных приложениях и пользовательском интерфейсе.
- Сохраните функциональность старого поля для обеспечения обратной совместимости и избежания потери данных в случае несвоевременного обновления API или пользовательского интерфейса (то есть не нужно удалять логику для старого поля!).

3. Копирование данных в новые поля

- Разработайте алгоритм для извлечения фамилии и имени из существующего поля и копирования этих данных в новые поля.

4. Изменение функциональности получения данных

- Переходите на использование новых полей для отображения списков и детальной информации о сущностях в пользовательском интерфейсе и серверных приложениях.

5. Проверка всей логики на тестовой среде

6. Переход на новые поля

- Постепенно перенаправляйте всю функциональность, связанную с заполнением и использованием старого поля, на новые поля.

7. Вывод на продакшн - на боевую среду

8. Удаление старого поля

- После полного обновления функциональности и удостоверения в том, что старое поле больше не используется, удалите его. Этот шаг может быть опущен, если такое изменение в таблице рискованно.

Примеры разбиения задач на этапы:

- Этап 1: Задачи 1 и 2. Начало процесса миграции с минимальным влиянием на текущую функциональность.
- Этап 2: Задачи 3 и 4, 5. Интеграция новых полей в основную логику системы.

- Этап 3: Задачи 6 и 7. Полный переход на новые поля.
- Этап 4: Задача 8. Окончательное удаление старого поля (если это безопасно для системы).

Дополнительные рекомендации такие же, как в примере выше - мониторинг и откат, коммуникация с заинтересованными сторонами, тестирование.

Общие рекомендации по сохранению обратной совместимости

Изменения, которые сохраняют обратную совместимость по умолчанию:

1. Добавление новых атрибутов (полей) в таблицы

- Добавление новых столбцов обычно безопасно, если они не являются обязательными или имеют значение по умолчанию.

2. Добавление новых таблиц

- Введение новых таблиц в схему базы данных не влияет на существующие таблицы и операции, поэтому это обычно безопасно.

3. Добавление новых представлений (Views) и хранимых процедур

- Эти изменения не влияют непосредственно на существующие структуры данных и могут обеспечивать дополнительную функциональность без нарушения существующих процессов.

4. Добавление индексов

- Создание новых индексов может улучшить производительность запросов и не влияет на логику существующих приложений.

Изменения, которые могут нарушить обратную совместимость

1. Установка обязательности (NOT NULL) для существующих атрибутов

- Преобразование существующих необязательных полей в обязательные может вызвать ошибки в приложениях, которые не ожидают такого требования.

2. Удаление атрибутов или таблиц

- Удаление существующих столбцов или таблиц может нарушить функциональность приложений, которые зависят от этих элементов.

3. Изменение типа данных или размера поля

- Изменение характеристик существующих полей может привести к проблемам с совместимостью, особенно если новый тип данных несовместим с ожиданиями приложения.

4. Изменение существующих ограничений целостности

- Изменения, влияющие на внешние ключи или другие ограничения, могут вызвать несоответствие данных или ошибки в приложениях, которые полагаются на предыдущие ограничения.

Финальные рекомендации:

- Перед внесением любых изменений тщательно оцените потенциальное воздействие на существующие системы.
- Тестирование важно для обнаружения проблем совместимости до применения изменений в продакшн-среде.
- Поддерживайте хорошо документированные версии схемы базы данных, чтобы облегчить откат в случае возникновения проблем.
- Обеспечьте эффективное общение с разработчиками и пользователями о предстоящих изменениях и их потенциальном влиянии на существующие системы.