И в заключении будет сказано об аналитике состоянии сети в целом, о возможности прогнозирования ситуаций и нагрузок на сеть благодаря наличию большого количества данных, применении современных алгоритмов из обработки и машинного обучения. И какую пользу все это может принести как владельцу сервиса, так и пользователям.

Список использованных источников:
1. Маккинни У., Python и анализ данных, 2015. – 482с
2. Cisco ASR 5x00 System Administration Guide, StarOS Release 2x.x, 2018. – 682s
3. Вандер Плас Дж., Python для сложных задач. Наука о данных и машинное обучение, 2018. – 336с
4. Daniel Lee, Visualize Your Data With Grafana, 2017. – 57s

СПОСОБЫ МИГРАЦИИ СХЕМ ДАННЫХ С СОБЛЮДЕНИЕМ ПРИНЦИПОВ ИДЕМПОТЕНТНОСТИ И КОНВЕРГЕНТНОСТИ

Данильчик В.В.

Белорусский государственный университет информатики и радиоэлектроники
г. Минск, Республика Беларусь

Прытков В.А. – проректор по уч. работе, к.т.н., доцент

Основным направлением изучения способов миграции схем данных из одной структуры в другую является определение оптимальных принципов модификации структур и алгоритма перестроения объектов с учетом существующих зависимостей.

Актуальность анализа способов миграции схем данных обусловлена необходимостью постоянной поддержки структуры базы данных и ее последующей оптимизации при выходе новой версии программного продукта. В настоящее время процессу миграции данных часто сопутствуют проблемы их дублирования и возникновения ошибок, связанных с изменением структуры данных.

При разработке программного продукта и, соответственно, написании кода разработчики пользуются системами контроля версий, такими как Git или Subversion. Данные системы позволяют вести совместный процесс разработки несколькими сотрудниками, хранить только разницу между версиями кода, отслеживать и, при необходимости, возвращаться к конкретной версии исходного кода. В то же время применение систем контроля версий при работе со структурами схем данных довольно ограничено.

В результате в процессе развития программного продукта отследить изменения структуры данных весьма проблематично ввиду того, что DDL, язык определения схем данных, не предусматривает документирование изменения структуры на каждом шаге [1].

Одним из методов решения данной задачи является создание в каталоге проекта пронумерованных скриптов модификации структуры базы данных при этом, не прибегая к использованию дополнительных инструментов. В таком случае все изменения будут накапливаться в проекте при условии соблюдения заранее определенных правил участниками проекта.

Одним из наиболее современных и практичных способов контроля структуры схем данных является использование дополнительного инструмента Liquibase. Основной задачей системы является контроль номера последнего выполненного скрипта изменения структуры данных и обеспечение единовременного линейного выполнения скриптов по изменению структуры данных. Однако, использование инструментов, в основе которых лежит принцип накопления change-скриптов, не позволяет решить задачу так же легко, как написать программный код приложения. Главным минусом данного подхода является необходимость хранения огромного количества неактуальных change-скриптов, что затрудняет получение представления о фактически используемой структуре схемы данных.

Для решения поставленных задач исследованы такие принципы, как конвергентность и идемпотентность. Соблюдение приведенных принципов в разработанном модуле миграции позволяет наиболее эффективно контролировать состояние схемы данных и не допускать повторного выполнения скрипта миграции данных при его успешном запуске ранее.

Наиболее оптимальным решением задачи миграции схем данных и является следование данным принципам: идемпотентности и конвергентности [2]. Отличительной особенностью такого подхода является то, что скрипт, который удовлетворяет условиям принципов, описывает состояние, к которому объект должен быть приведен, а не действия, которые требуется произвести над ним.

Идемпотентность обеспечивает отсутствие изменений объекта при выполнении скрипта повторно в случае, если скрипт был выполнен успешно при первом его запуске. Соблюдение данного принципа позволяет избежать дублирования и искажения данных. Однако, в случае невыполнения скрипта или неудачного завершения его выполнения, система, при повторном выполнении данного

скрипта, будет пытаться прийти к тому состоянию, которое было задано в качестве желаемого. Такое поведение системы обусловлено принципом конвергентности, который направлен на приведение системы в нужное состояние при каждом следующем запуске одного и того же скрипта без выполнения операций, успешно выполненных ранее. Т.е. изменения, не выполненные при одной попытке, система будет пытаться выполнить еще раз при повторном запуске.

Список использованных источников:

1. Клеппман. М. Высоконагруженные приложения. Программирование, масштабирование, поддержка/М. Клеппман // СПб: Питер, 2018.
2. Database Migration: What It Is and How to Do It [Электронный ресурс] - Режим доступа: https://rollout.io/blog/database-migration - Дата доступа: 22.03.2019

ПРОГРАММНАЯ СИСТЕМА СИМУЛЯЦИИ РАЗРУШАЕМОСТИ НА ОСНОВЕ UNREAL ENGINE 4

Дмитриев А.С.

Белорусский государственный университет информатики и радиоэлектроники
г. Минск, Республика Беларусь

Золоторевич Л.А., доцент

В данной статье рассматривается одна из возможных реализаций системы симуляции разрушаемости в Unreal Engine 4. Для реализации отдельных компонентов используется внешний плагин Apex Destruction для Unreal Engine и отдельная программа PhysX Lab.

В основе разрабатываемой системы симуляции применяется Unreal Engine 4 как один из наиболее стабильных и производительных движков. При проектировании системы в первую очередь материалы объектов разрушения были разбиты на четыре группы:

1) породы – различные каменные образования и структуры;
2) металл – тугоплавкие материалы и металлоконструкции;
3) дерево – древесина и изделия деревообрабатывающей промышленности;
4) стекло – изделия стекольной промышленности и бьющиеся материалы.

Для моделирования разрушаемости объектов первой группы используется алгоритм Воронова, основанный на одноименной диаграмме или так называемом разбиении Дирихле. В качестве его программной реализации используется внешний плагин Apex Destruction для Unreal Engine 4, который заранее делит объекты на сцене на определенное количество элементов, называемых чанками. Все чанки являются параллелепипедами различных размеров, содержащими один из будущих обломков разрушаемого объекта. Так как разделение объекта запекается заранее, разрушаемость является довольно предсказуемой, но с другой стороны позволяет обеспечить высокую скорость моделирования в режиме реального времени (рисунок 1).

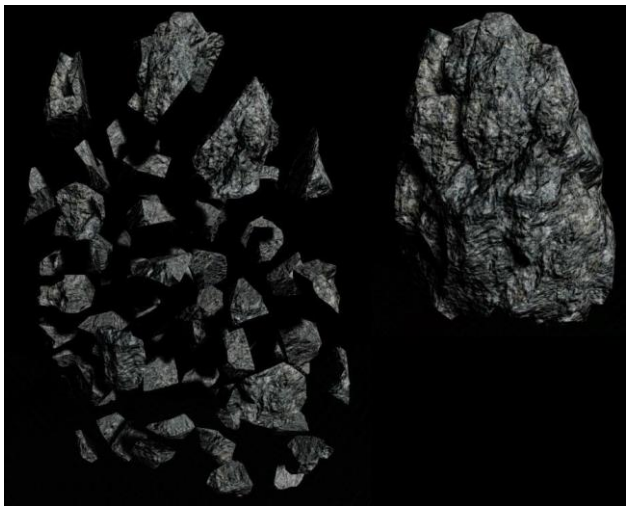

Рисунок 1 – Пример деления объектов в Apex Destruction

Вторая группа основана на трёхмерной разметке модели и её компонентов или так называемом скелетал меше. Каждая такая точка в разметке называется нодом. Положение каждого нода может