УДК 004.624

АНАЛИЗ ПОТЕРЬ ДАННЫХ В ЦЕПОЧКЕ МЕЖСИСТЕМНОГО ОБМЕНА ФОРМАТАМИ

С. В. ЛУКАШЕВИЧ, В. Л. ШАБАЛОВ, В. В. ЛУКАШЕВИЧ
(Представлено: А. А. СКУКОВСКАЯ)

В статье рассматривается проблема потери и искажения данных при передаче информации между различными программными системами. Анализируется типичная цепочка обмена: Excel → CSV → Python → JSON → веб-форма → база данных. Оцениваются этапы, на которых наиболее вероятны ошибки, и предлагаются рекомендации по их минимизации.

Ключевые слова: *обмен данными, совместимость форматов, CSV, JSON, Excel, Python, база данных.*

Введение. В условиях цифровой трансформации всё чаще возникает необходимость передачи данных между разнородными системами: от офисных приложений до веб-сервисов и баз данных. Однако, несмотря на стандартизацию форматов, на практике при конвертации данных между системами часто происходят потери информации или её искажение. Особенно уязвимыми оказываются текстовые данные, даты, числа с плавающей точкой и специальные символы. Цель данной работы – выявить критические точки в цепочке обмена данными и проанализировать причины возникающих ошибок.

Основная часть. Типичная цепочка обмена данными может выглядеть следующим образом:

1. Исходные данные создаются в Microsoft Excel.
2. Файл экспортируется в формат CSV.
3. Данные считываются скриптом на языке Python.
4. Преобразуются в формат JSON.
5. Передаются через веб-форму (например, REST API).
6. Сохраняются в реляционной базе данных (например, PostgreSQL).

На каждом из этих этапов возможны потери или искажения.

Для проверки этого взяты проверочные данные, представленные на рисунке 1.

	1	2	3	4
1	ФИО	ДАТА РОЖДЕНИЯ	СРЕДНИЙ БАЛЛ	КОММЕНТАРИЙ
2	Иванов И.И.	01.03.2004	8,75	Студент, "отличник", №123

Рисунок 1. – Входные данные

В самом начале цепочки обмена данными – уже на этапе экспорта из Microsoft Excel в формат CSV – происходит первая и зачастую самая критичная трансформация. Excel, будучи мощным инструментом для анализа и визуализации, сохраняет в своих файлах не только значения ячеек, но и богатую метаинформацию: формулы, условное форматирование, комментарии, типы данных, а также локальные настройки пользователя. Однако при экспорте в CSV вся эта семантическая нагрузка теряется безвозвратно – остаются лишь «голые» строки. Более того, в русскоязычных версиях Excel по умолчанию применяется кодировка Windows-1251, а в качестве разделителя полей используется точка с запятой (;), что обусловлено региональными настройками операционной системы. Это принципиально расходится с международным стандартом RFC 4180, который предписывает использовать запятую в качестве разделителя и рекомендует кодировку UTF-8. В результате при попытке открыть такой CSV-файл в системах, ориентированных на глобальные стандарты (например, Linux-серверах, облачных сервисах или скриптах на Python), кириллические символы отображаются как набор иероглифов или знаков вопроса, а структура таблицы нарушается из-за неправильного определения колонок. Даже если файл открывается корректно в другом экземпляре Excel, это создаёт ложное ощущение совместимости, которое рушится при передаче данных в сторонние системы.

Следующим этапом является чтение CSV-файла с помощью скрипта на языке Python, чаще всего с использованием библиотеки pandas. Здесь возникает целый спектр потенциальных ошибок, связанных с неявными предположениями. Если разработчик не указывает явно параметры encoding и sep, библиотека пытается автоматически определить их, что часто приводит к неверным выводам. Например, при чтении файла в кодировке cp1251 как UTF-8 текст превращается в «кракозябры», а при неверном выборе разделителя вся строка может быть интерпретирована как одно поле. Особенно уязвимыми оказываются текстовые поля, содержащие запятые, кавычки, символы новой строки или другие специальные знаки.

Согласно стандарту CSV, такие поля должны быть заключены в двойные кавычки, а внутренние кавычки – удвоены. Однако Excel не всегда следует этому правилу строго, особенно при наличии сложных символов. В результате парсер может разбить одну логическую ячейку на несколько колонок, что вызывает каскадное смещение данных по всей строке. Ещё более коварной проблемой является обработка дат: Excel хранит их как целые числа – количество дней, прошедших с 1 января 1900 года (с известной ошибкой, учитывающей несуществующий 29 февраля 1900 года). Если ячейка с датой не была явно отформатирована как текст перед экспортом, в CSV попадает именно это число. Python, в свою очередь, прочитает его как обычное целое и не сможет восстановить исходную дату без дополнительной логики, что приводит к необратимой потере временной информации.

Далее данные преобразуются в формат JSON для передачи через веб-интерфейс. На этом этапе возникают новые сложности, обусловленные фундаментальными различиями в типовых системах Python и JSON. Формат JSON поддерживает лишь ограниченный набор типов: строки, числа, логические значения, массивы, объекты и null. Всё остальное – включая объекты datetime, десятичные числа Decimal, множества или пользовательские классы – должно быть явно сериализовано. Без этого преобразование завершится ошибкой. Даже при корректной сериализации возможны потери: например, числа с плавающей точкой могут изменить своё значение из-за особенностей представления в стандарте IEEE 754, особенно при многократных преобразованиях. Кроме того, JSON требует строгого экранирования специальных символов: двойные кавычки, обратные слэши, управляющие символы (включая переводы строк) должны быть заменены на соответствующие escape-последовательности (\", \\, \n и т.д.). Если это не сделано, полученный документ перестаёт быть валидным JSON, и любой стандартный парсер откажется его обрабатывать, что приведёт к полному сбою передачи данных.

После формирования JSON данные поступают в веб-форму – например, через REST API или HTML-интерфейс. На этом этапе они подвергаются строгой валидации со стороны серверного приложения. Современные веб-фреймворки (Django, Flask, Express.js и др.) часто используют схемы данных (например, Pydantic, JSON Schema), которые задают точные требования к каждому полю: тип, формат, длина, допустимые символы. Если дата передана в привычном пользователю формате DD.MM.YYYY, но сервер ожидает ISO 8601 (YYYY-MM-DD), запрос будет отклонён или автоматически преобразован, что может привести к искажению смысла (например, 01.03.2024 может быть ошибочно интерпретирована как 1 марта или 3 января в зависимости от локали). Текстовые поля также могут подвергаться санитизации: для защиты от XSS-атак фреймворки по умолчанию удаляют или экранируют HTML-теги и потенциально опасные символы. В результате фраза вроде Студент "отличник" №123 может превратиться в Студент отличник №123, где кавычки исчезли, а символ номера заменён на HTML-сущность. Такие изменения часто происходят незаметно для пользователя и разработчика, создавая иллюзию корректной работы.

Завершающим этапом является сохранение данных в реляционной базе данных, такой как PostgreSQL, MySQL или SQLite. Здесь возможны ошибки, связанные с несоответствием типов и ограничений схемы. Например, если в колонку с типом NUMERIC или FLOAT попытаться вставить строку "8,75" (с запятой как десятичным разделителем), СУБД вернёт ошибку, так как запятая не распознаётся как часть числа. Аналогично, если текстовое поле определено как VARCHAR(50), а входная строка содержит 60 символов, она будет автоматически усечена – без предупреждения, если настройки базы не предусматривают строгую проверку. Особенно сложной остаётся работа с датами и временем: если значение передаётся без указания временного пояса, СУБД может интерпретировать его в своей локальной зоне, что приведёт к расхождению при последующем извлечении в другом часовом поясе. Кроме того, некоторые базы данных (например, MySQL) менее строги к формату дат и могут «угадывать» значение, что иногда приводит к неожиданным результатам вроде автоматической подстановки 0000-00-00 вместо невалидной даты.

Таким образом, на каждом этапе цепочки обмена данными – от экспорта в CSV до записи в базу – существует реальный и измеримый риск потери, искажения или полного отклонения информации. Особенно уязвимыми оказываются этапы экспорта из Excel и передачи через веб-интерфейс, где отсутствие единых соглашений о формате, кодировке, типах данных и методах валидации приводит к систематическим, но трудноуловимым ошибкам. Только при строгом соблюдении открытых стандартов (UTF-8, ISO 8601, RFC 4180), явном контроле типов и кодировок на всех этапах, а также внедрении сквозной валидации и логирования можно обеспечить надёжную, предсказуемую и безопасную передачу данных между разнородными системами. Данные которые дошли до базы данных представлены на рисунке 2.

FULL_NAME	BIRTH_DATE	AVG_SCORE	COMMENT
Иванов И.И.	NULL	NULL	Студент, отличник, №123

Рисунок 2. – Конечный результат

Заключение. Цепочка обмена данными между системами – это последовательность трансформаций, каждая из которых несёт риск потери или искажения информации. Наиболее уязвимыми этапами являются экспорт из Excel в CSV и передача через веб-интерфейс. Для минимизации ошибок рекомендуется:

– Использовать UTF-8 с BOM при экспорте CSV;
– Явно указывать разделители и кодировки при чтении;
– Применять стандартизированные форматы (например, ISO 8601 для дат);
– Валидировать данные на каждом этапе передачи.
– Только системный подход к совместимости форматов позволяет обеспечить целостность и достоверность данных в межсистемном взаимодействии.

ЛИТЕРАТУРА

1. McKinney, W. Python for Data Analysis. – O'Reilly Media, 2022.
2. Bray, T. (Ed.). The JavaScript Object Notation (JSON) Data Interchange Format. – RFC 8259, 2017.
3. Microsoft. Excel Specifications and Limits. – Документация Microsoft, 2023.
4. PostgreSQL Global Development Group. PostgreSQL Documentation. – https://www.postgresql.org/docs/