Предлагаемая структура таблицы-очереди

**Исходящие сообщения:**

|  |  |  |
| --- | --- | --- |
| Queue\_out\_head | Таблица очереди исходящих сообщений | |
| Поле | Тип | Описание |
| Guid | String | Guid записи – ХЗ зачем, то типа так современнее) |
| Status | Byte | Статус позиции:  0 - ожидает обработку  1 – обработана  2 – ошибка |
| Try\_count | Byte | Количество попыток |
| Datetime\_in | Datetime | Дата-время создания записи очереди |
| Datetime\_out | Datetime | Дата-время последней обработки записи очереди (не важно успешная или нет) |
| Object\_nrec | Comp | Ссылка на экземпляр объекта |
| cQueue\_template | Comp | Ссылка на шаблон выгрузки для обмена - Шаблон однозначно определяет тип объекта, структуру выгрузки и адрес получателя |

|  |  |  |
| --- | --- | --- |
| Queue\_out\_log | Таблица логов очереди исходящих сообщений | |
| Поле | Тип | Описание |
| Queue\_guid | string | GUID записи очереди. Ссылка на запись очереди. (можно и по nrec - пофиг) |
| Status | Byte | Статус попытки:  1 – обработана  2 – ошибка |
| Datetime\_try | Datetime | Дата-время создания записи лога/попытки |
| Log\_text | String | Текст попытки или ошибки |
| Log\_text\_memo | Memo | Полный текст попытки или ошибки – по сути полный ответ, например, от внешней API или текст возникшего исключения |

Логика работы системы.

В каком то смысле мы реализуем микросервис.

Точка расширения/альтер/алгоритм/сборщик изменений таблиц заполняет таблицу Queue\_out\_head с указанием ссылки на объект и на шаблон выгрузки:

* когда шаблон не указан и тогда применяется шаблон по умолчанию
* когда не указана ссылка на объект, то выгружается целиком все (актуально для первичной выгрузки или принудительной синхронизации всех объектов)

Все.

Далее обработчик очереди начинает свою работу – фоновый процесс/запуск по расписанию/вручную/

Идет по всем записям со статусом Queue\_out\_head.Status = 0 и пытается выполнить действие по шаблону (не важно что это будет, создание файла в папке, запрос во внешнюю API, выполнение интерфейса внутри и т.д.)

Каждая попытка генерирует запись лога в Queue\_out\_log, увеличивает счетчик Queue\_out\_head.Try\_count на 1 и записывает в Queue\_out\_log.status статус выполнения (1 или 2), а в Queue\_out\_log.Log\_text (Queue\_out\_log.Log\_text\_memo) – текст ошибки или то, что реализовал программист.

Если количество Queue\_out\_head.Try\_count превысило порог (берется из шаблона и задается в настройках системы – используется первый найденный), то Queue\_out\_head.Status приравнивается к Queue\_out\_log.Status – запись больше не обрабатывается. Этим флагом можно играться – вручную поменять его и тогда записи снова пойдут (а вот реальное количество попыток будет видно в Queue\_out\_log)

**Входящие сообщения – надо или нет ХЗ, по идее это все сервисной шиной снаружи реализуется:**

|  |  |  |
| --- | --- | --- |
| Queue\_in\_head | Таблица очереди входящих сообщений | |
| Поле | Тип | Описание |
| Guid | String | Guid записи – вот тут он нужен – возвращается сразу внешней API и по нему будет опрашиваться система для получения статуса выполнения |
| Status | Byte | Статус позиции:  0 – ожидает обработку  1 – в работе  2 – ошибка |
| Try\_count | Byte | Количество попыток – надо или нет вопрос.. будем ли пытаться повторять при неудаче? |
| Datetime\_in | Datetime | Дата-время создания записи очереди |
| Datetime\_out | Datetime | Дата-время последней обработки записи очереди (не важно успешная или нет) |
| Method | String | Какой метод будет вызываться в системе |
| Text\_query | Memo | Полное тело запроса снаружи |
| Result\_query | Memo | Полный текст ответа |

|  |  |  |
| --- | --- | --- |
| Queue\_in\_log | Таблица логов очереди исходящих сообщений | |
| Поле | Тип | Описание |
| Queue\_guid | string | GUID записи очереди. Ссылка на запись очереди. (можно и по nrec - пофиг) |
| Status | Byte | Статус попытки:  1 – обработан  2 – ошибка |
| Datetime\_try | Datetime | Дата-время создания записи лога/попытки |
| Log\_text | String | Текст попытки или ошибки |
| Log\_text\_memo | Memo | Полный текст попытки или ошибки – по сути полный ответ, например, от интерфейса Галактики или текст возникшего исключения |

Логика работы системы.

В каком то смысле мы реализуем микросервис.

Реализуем одну точку входа, например, /api/gal-queue-put

Структура запроса body:

{

“Method”: “methodname”,

“Text\_query”: “all necessary structure and parameters for method ”

}

Разбираем запрос и кладем в Queue\_out\_head

Queue\_out\_head.method = “methodname”

Queue\_out\_head. Text\_query = “all necessary structure and parameters for method ”

API возвращает GUID очереди для получения статуса следующим запросом.

Реализуем одну точку входа, например, /api/gal-queue-status

Структура запроса body:

{

“queue\_id”: “guid”,

}

В ответе будут значения поля Queue\_in\_head.Status и Queue\_in\_head.Result\_query

Все.

Далее обработчик очереди начинает свою работу – фоновый процесс/запуск по расписанию/вручную/

Идет по всем записям со статусом Queue\_in\_head.Status = 0 и пытается выполнить необходимые действия и заполняет поля Queue\_in\_head.Status и Queue\_in\_head.Result\_query.

Каждая попытка генерирует запись лога в Queue\_out\_log, увеличивает счетчик Queue\_in\_head.Try\_count на 1 и записывает в Queue\_out\_log.status статус выполнения (1 или 2), а в Queue\_in\_log.Log\_text (Queue\_in\_log.Log\_text\_memo) – текст ошибки или то, что реализовал программист.

Если количество Queue\_in\_head.Try\_count превысило порог (берется из шаблона и задается в настройках системы – используется первый найденный), то Queue\_in\_head.Status приравнивается к Queue\_in\_log.Status – запись больше не обрабатывается. Этим флагом можно играться – вручную поменять его и тогда записи снова пойдут (а вот реальное количество попыток будет видно в Queue\_in\_log)