ASTRAVERGE / PGDEVELOPER

Интеграция сообщений

AstraVerge PGDeveloper 0.7.0

Практическое руководство · 11 ноября 2026

Рабочая среда для PostgreSQL: от структуры базы до кода приложения.

Команды и имена файлов сохраняют техническое имя av-pgdeveloper. Параметры и границы применимости относятся к версии 0.7.0.

Содержание

Подключения 3

Публикация из Result Grid 3

Messaging SQL 4

CONSUME, timer и limit 4

- Kafka . . . 5
- RabbitMQ . . . 5
- Valkey / Redis . . . 5

JSON, INSERT и перенос в PostgreSQL 6

Проверки и границы v1 6

Этот документ описывает исходную v1.0. В 0.6.0 добавлены live JSON без Stop, pending inspection/replay и реальная TLS/SASL/нагрузочная приёмка: актуальное дополнение. Прежние ограничения ниже относятся к исходному выпуску там, где дополнение явно их изменяет.

Kafka, RabbitMQ, Valkey и Redis доступны из SQL Window и контекстного меню обычного Result Grid. Это операции публикации и получения сообщений, а не отдельная IDE администрирования брокеров. Дополнительного сервера, AV-Agent и собственного сетевого протокола нет.

Подключения

Session → Broker connections открывает брокерную часть управления подключениями. Создайте профиль, проверьте Test broker connection и нажмите Save broker profiles. Профили целиком, включая пароли и SASL/TLS properties, сохраняются в существующем защищённом WorkStore: Windows DPAPI либо разблокированное зашифрованное хранилище. В `connections.json` секреты брокеров не записываются. Имена профилей уникальны. Изменение профиля не переподключает активные операции.

Общие поля: host/port, TLS, username/password, connection timeout и read timeout в миллисекундах. Kafka host принимает также полный список bootstrap servers, например `host1:9092,host2:9092`. RabbitMQ имеет virtual host, Valkey/Redis - database и ACL username/password. Для Kafka username/password настраивают SASL PLAIN по умолчанию; `sasl.mechanism` допускает SCRAM-SHA-256/512. Для другого механизма задайте явную `sasl.jaas.config` в защищённом поле свойств.

Разрешённые Kafka properties: `security.protocol`, `sasl.mechanism`, `sasl.jaas.config`, `ssl.truststore.location/password/type`, `ssl.keystore.location/password/type`, `ssl.key.password`. Поле свойств - JSON-объект строк. Пример без секретов:

```
{"security.protocol":"SASL_SSL","sasl.mechanism":"SCRAM-SHA-512","ssl.truststore.location":"C:/certs/brok
er.p12","ssl.truststore.type":"PKCS12"}
```

TLS проверяет имя сервера. RabbitMQ/Redis используют доверенные сертификаты JVM; Kafka поддерживает свои truststore/keystore properties. Отключения проверки имени сервера в интерфейсе нет. Не помещайте пароли в SQL и имена профилей.

Публикация из Result Grid

Контекстное меню → Publish Result → тип брокера. Выберите connection, destination (topic/exchange/stream/channel), применимые routing/message key, JSON/JSONL/Text, выбранные либо загруженные строки и режим одной публикации на строку либо одной пакетной публикации. Preview показывает число сообщений, число строк, destination и ограниченный пример. Только Publish отправляет данные.

JSON сохраняет NULL, boolean и числовые значения без преобразования numeric в double; PostgreSQL JSON/JSONB остаётся JSON-структурой, UUID и даты/время представляются строками, bytea - base64. JSONL содержит один JSON-объект на строку. Text - значения колонок через TAB, NULL обозначен `\N`; это простой текст, а не формат с обратимым экранированием. Для обратимого переноса используйте JSON. Дублирующиеся имена колонок требуют явных SQL aliases перед публикацией.

Full result дочитывает только текущий курсор той же PostgreSQL-сессии, учитывая уже связанные параметры и незавершённую транзакцию. SELECT повторно не запускается. Если курсор закрыт, снимок частичный либо действуют пределы загрузки, публикация Full result отклоняется с объяснением. Можно явно выбрать Loaded rows. Сериализованный снимок публикации ограничен 64 МиБ. PostgreSQL COMMIT публикация не вызывает.

Результат показывает attempted, confirmed, errors и unknown. Ошибка отправки останавливает пакет; предыдущие подтверждения сохраняются. Сетевой сбой во время отправки означает неопределённый исход последнего сообщения. Автоматического повтора нет. Отмена также показывает частичный результат. Подтверждение брокера не означает обработку конечным получателем; Redis Pub/Sub может не иметь подписчиков, а сообщения RabbitMQ требуют подходящего binding.

Messaging SQL

Это синтаксис приложения, распознаваемый **до JDBC**. Выполняется выделение либо оператор под курсором через обычный Execute/F5. Execute Script с Messaging SQL отклоняется; смешанный скрипт не отправляется в PostgreSQL. SQL scanner не заменён.

`connection='profile-name'` явно выбирает профиль. Без него единственный совместимый профиль выбирается автоматически, несколько профилей вызывают диалог. Секреты в командах запрещены. Messaging SQL не записывается в SQL History; Payload и свойства подключения не включаются в диагностические ошибки.

```
publish kafka 'orders.events'
connection='events-kafka' key='ready-orders' format=json
from (
  select id, customer_id, amount
  from sales.orders where status='ready'
);

publish rabbitmq 'orders.exchange'
connection='events-rabbit' routing_key='order.created' format=jsonl
from (select * from sales.orders where status='ready');

publish valkey 'orders:events' connection='local-valkey' mode=stream
from (select * from sales.orders);

publish redis 'orders.notifications' connection='local-redis'
mode=pubsub format=json batch=true
from (select id, amount from sales.orders);
```

Все публикации, включая Messaging SQL, требуют просмотра и явного Publish. Запрос FROM выполняется в текущей PostgreSQL-сессии; параметры запрашиваются существующим диалогом. Сохранённый профиль брокера не меняет PostgreSQL connection. `format=json|jsonl|text`, `batch=true|false`; по умолчанию JSON, одна строка - одно сообщение. Неизвестные и неприменимые к брокеру опции отклоняются. Команд управления топиками, очередями и ключами в Messaging SQL нет.

CONSUME, timer и limit

```
consume kafka 'orders.events' connection='events-kafka'
group='pgdeveloper-diagnostics' start=earliest timer=10m limit=1000;

consume kafka 'orders.events' connection='events-kafka'
group='pgdeveloper-diagnostics' partition=0 offset=12345 timer=30s;

consume rabbitmq 'orders.queue' connection='events-rabbit'
ack=manual timer=10m limit=1000;

consume valkey 'orders:events' connection='local-valkey'
mode=stream start=earliest timer=10m limit=1000;

consume redis 'orders.notifications' connection='local-redis'
mode=pubsub timer=1h limit=1000;
```

timer - положительное целое с суффиксом `s`, `m`, `h`, максимум 7 суток. Отсчёт начинается при запуске операции и включает время подключения и паузы. limit - 1…1 000 000 сообщений. Операция заканчивается по первому достигнутому условию. Без timer работает до Stop либо ресурсного предела. Без limit используется Settings → Default CONSUME row limit (по умолчанию 10 000). Память на сессию настраивается в Settings (по умолчанию 64 МиБ). Внутренняя очередь до EDT ограничена 256 сообщениями; накопление не является неограниченным.

Каждое SQL Window/CONSUME имеет собственное соединение, worker, timer, счётчик, очередь и Result Grid. Kafka, RabbitMQ, Valkey/Redis и PostgreSQL в разных окнах работают одновременно. Pause/Resume/Stop относятся к конкретному результату. В панели видны состояние, источник, число сообщений, elapsed/remaining и ACK/NACK для ручного режима. После Stop строки сохраняются; закрытие результата/окна освобождает только его ресурсы. Переподключение и повторное получение - явные.

Kafka

group обязателен и задаётся явно. Диагностическое чтение использует назначение партиций без автоматического rebalance и **без commit offsets**. Сохранённые offsets группы не меняются. `start=earliest|latest`; `partition=N offset=M` задают точную стартовую позицию одной партиции. По умолчанию earliest. Новые партиции, созданные после запуска, автоматически не подхватываются. Result Grid содержит timestamp, topic, partition:offset, key, headers и payload. Headers сохраняют повторяющиеся имена и base64 значения. Невалидный UTF-8 payload/key показывается с префиксом `base64:`. Нет транзакционного Kafka producer и гарантии exactly-once между PostgreSQL и Kafka.

RabbitMQ

ack обязателен: `manual` либо `auto`. Перед стартом интерфейс показывает последствия. Auto подтверждает доставку при получении и может удалить сообщение до добавления в грид; это **не неразрушающий Browse**. В manual используются явные действия ACK selected deliveries и NACK / requeue selected. Неподтверждённые сообщения удерживаются активным channel и возвращаются в очередь при Stop/Close. После закрытия канала старые delivery tags больше не действуют. Result Grid сохраняет queue, exchange, routing key, delivery tag, headers, redelivered и payload. Publisher использует mandatory и publisher confirms; неподходящая маршрутизация считается ошибкой, автоматического повторения нет. Чтение диагностическое через basicGet; для высокой пропускной способности используйте специализированный потребитель приложения.

Valkey / Redis

`mode=stream` - XADD/XREAD, stream ID сохраняется. `start=earliest` читает с 0-0, `latest` фиксирует последний существующий ID при запуске и затем читает новые сообщения без повторного сброса позиции.

Для **уже существующей** consumer group:

```
consume valkey 'orders:events' connection='local-valkey'
mode=stream group='processing' consumer='diagnostics-1' timer=10m;
```

Используется XREADGROUP с >; группа не создаётся автоматически. ACK selected deliveries вызывает XACK. Stop/Close оставляет неподтверждённые entries в Pending Entries List группы. Нет автоматического XCLAIM, replay pending или изменения групповой позиции. `start` для group запрещён. NACK/requeue - только RabbitMQ.

`mode=pubsub` - PUBLISH/SUBSCRIBE. Pub/Sub не хранит сообщения отсутствующего подписчика. В Pause входящие сообщения **теряются**, скрытая очередь не растёт; при переполнении ограниченного буфера чтение заканчивается ошибкой. Восстановление пропущенного возможно только средствами хранящего сообщения источника, например Streams. Интерфейсы Redis v1 ориентированы на UTF-8 payload.

JSON, INSERT и перенос в PostgreSQL

Стандартный Result Grid поддерживает Export и Analysis. Inspect message payload открывает существующий ValueEditor с read-only JSON tree/pretty preview. После Stop действие Expand JSON to Columns принимает JSON Pointer пути, по одному на строку: `/id`, `/customer/name`, `/items/0/amount`. Исходные Payload и метаданные сохраняются, исходное сообщение не меняется. Missing/JSON null становятся NULL, объекты/массивы - JSON-текстом, большие числа сохраняют точность.

Generate INSERT / transfer to PostgreSQL предлагает connection/database/schema/ table через существующий SourceChooser, preview первых 20 строк и сопоставление колонок. INSERT создаётся существующим SqlInsertPolicy и открывается в редакторе без выполнения. Direct Transfer использует существующий SqlSession.importData / DataTransfer.importRows: типы целевой таблицы, NULL, JSONB, UUID, numeric, timestamps, batch, ограничения, savepoint, progress и Stop. Новые целевые окна открываются **с выключенным Autocommit**. Commit/Rollback выполняет пользователь. Ошибка импорта откатывает пакет к savepoint. Broker ACK/XACK никогда не вызывается из-за успешного PostgreSQL INSERT или COMMIT.

Проверки и границы v1

Unit-тесты: parser, исходный scanner/оператор под курсором, timer/limit, numeric/JSON/NULL, JSONL, expansion, mapping, ресурсные пределы, cancellation, Pause/Resume/Stop, независимые состояния и частичная отправка. Реальный стенд: четыре отдельные rootless-контейнера Kafka/RabbitMQ/Valkey/Redis, loopback-порты через проверяемый SSH-контроллер и отдельный PostgreSQL-кластер. Обычные профили и действующие базы в тестах не используются.

Задачи `messagingIntegration` и `messagingWindowSmoke` требуют явного marker `AVPG_MESSAGING_FIXTURE=avpg-msg-20261104a`, тестовых портов 39092/39672/39379/39380 и `AVPG_JAVA_TEST_PORT`/`AVPG_EVIDENCE_DIR`. Smoke проходит настоящий Swing GUI Preview/Publish, три независимых CONSUME и PostgreSQL SQL Window, JSON expansion, Generate INSERT, Direct Transfer и закрытие одного окна без остановки другого. `tools/messaging-brokers.sh` создаёт/убирает только собственные помеченные контейнеры; туннель - `tools/messaging-tunnel.py`. Тестовые топики/очереди/БД имеют уникальные имена.

Ограничения: нет distributed transaction PostgreSQL↔broker, exactly-once, schema registry/Avro/Protobuf, broker administration, stream pending replay/claim, автоматического reconnect или повторной отправки. JSON expansion выполняется после Stop. Ресурсный учёт памяти оценочный, с дополнительными bounded-буферами клиентов. При Stop/timer во время первоначального подключения завершение освобождения ресурсов может ждать настроенных connect/read timeout; сообщения после истечения timer не добавляются. Broker TLS/SASL использует возможности клиентов; интеграционный стенд v1 проверяет loopback без TLS/SASL, взаимный TLS отдельно проверен для backup. Пакет содержит Apache Kafka Clients 4.1.2, RabbitMQ Java Client 5.36.0, Jedis 6.2.0 и их зависимости; они не требуют установленного SDK у пользователя.

Семантика сверена с официальной документацией: KafkaConsumer, RabbitMQ Java API, Redis XREAD, Redis Pub/Sub с Jedis.