## РЕЗЕРВНОЕ КОПИРОВАНИЕ

### Основные понятия. Типы резервного копирования

Резервное копирование баз данных является важной частью любой комплексной стратегии обеспечение безопасности базы дан- ных с точки зрения резервирования и восстановления данных. Ре- зервное копирования БД имеет свои особенности и в определенных ключевых аспектах отличается от резервного копирования других типов файлов. Нельзя рассчитывать на то, что простое копирование файла базы всегда позволит восстановить ее при необходимости из копии, не принимая во внимание особенности, связанные с правами доступа к данным, состоянием базы данных и т.д.

В качестве основных причин необходимости резервного копи- рования можно выделить следующие:

* аварийное восстановление данных, необходимое из-за выхода из строя аппаратного обеспечения, физического уничтожения сервера, например, в случае пожара или стихийного бедствия, по- тери данных из-за несанкционированного доступа и т.д.;
* ошибки пользователей, например, удаление таблицы или данных. В частности, пользователь может забыть указать усло- вия удаления строк в предложении WHERE оператора DELETE, что приведет к полному удалению данных из таблицы;
* аудит: иногда появляется необходимость получения данных или схемы базы данных в некоторый момент времени в про- шлом;
* тестирование: периодическое обновление данных на тесто- вом сервере с использованием последних рабочих данных поз- волит проводить тестирование на реальных данных. При нали- чии резервной копии можно легко восстановить копию данных на тестовом сервере.

### Классификация резервного копирования по резервируемым данным

**Полное резервное копирование** баз данных – это тип резерв- ного копирования, который создает копию всей базы данных. Это самый очевидный и самый надежный тип резервного копирования базы данных. Однако он требует больше всего времени и ресурсов. Поэтому, как правило, полное резервное копирование баз данных выполняется реже всего.

При **дифференциальном (или разностном) копировании** вы- полняется резервное копирование только тех данных, которые из- менились в базе данных с момента последнего **полного** резервного копирования. Так, например, если полное резервное копирование выполняется по понедельникам, а во все остальные дни недели – разностное, то во вторник в резервную копию будут помещены из- менения, которые произошли с понедельника по вторник, в среду – с понедельника по среду и т.д. Этот способ гораздо быстрее, чем полное резервное копирование (конечно если запланировано регу- лярное полное резервное копирование и размер дифференциальных резервных копий не приближается к размеру полных резервных ко- пий), но сопряжен с дополнительным риском того, что произойдет что-то непредвиденное, что приведет к невозможности восстанов- ления базы данных.

**Инкрементное резервное копирование** аналогично диффе- ренциальному, за исключением того, что в качестве момента вре- мени, к которому относятся измененные данные, будет использо- ваться дата последней резервной копии, инкрементной или полной. Таким образом, при восстановлении из инкрементной резервной ко- пии может потребоваться восстановить последнюю полную резерв- ную копию, а также одну или несколько инкрементных резервных копий, чтобы добраться до текущего момента.

Для таких СУБД, которые поддерживают журналы транзакций, можно создавать инкрементные резервные копии этих журналов и использовать их для восстановления базы данных. В журнале транзакций фиксируются все транзакции и производимые ими в базе изменения, и его можно использовать для восстановления базы данных на определенный момент времени. При выполнении резервной копии журнала транзакций происходит его усечение, что является необходимым для всех БД с активированным журналом транзакций, в противном случае, за счет журнала транзакций размер базы данных будет расти и довольно быстро, вплоть до завершения места на диске. В журнале транзакций, если он активирован, фик- сируются все транзакции, с момента его последнего усечения.

Так, например, если полное резервное копирование выполняется по понедельникам, а во все остальные дни недели – резервное копиро- вание журнала транзакций, то во вторник в резервную копию будут записаны те изменения, которые произошли с понедельника по втор- ник, в среду будут записаны изменения со вторника по среду и т.д. Такое резервное копирование будет выполняться быстрее, чем диффе- ренциальное, но придется пожертвовать скоростью восстановления.

Если сбой произойдет, например, в четверг, то при использова- нии дифференциального резервного копирования потребуются две резервные копии: за понедельник (полная) и за четверг (разност- ная). В той же ситуации при использовании инкрементного резерв- ного копирования журналов транзакций потребуются полная копия за понедельник и копии журналов транзакций за вторник, среду и четверг.

### Классификация резервного копирования по способу создания

По способу создания резервные копии данных можно разде- лить на две большие группы: физические и логические.

**Физическая резервная копия** – это копия исходных двоичных данных, часто создаваемая на уровне операционной системы. Лю- бой метод резервного копирования, включающий копирование дан- ных без использования построчного доступа к БД, считается мето- дом физического резервного копирования. Такой тип резервного копирования подходит для любых баз данных, но рекомендуем для критически важных баз, которые необходимо быстро восстанавли- вать после сбоев.

**Логическая резервная копия** – это набор команд SQL, содер- жащий логическую структуру базы данных (запросы DDL) и ее со- держимое (запросы INSERT). Резервное копирование выполняется посредством сканирования таблицы с прохождением каждой строки данных. Такой тип резервного копирования подходит для небольших баз данных, не требовательных к скорости восстановле- ния.

#### Достоинства и недостатки физического и логического копирования данных

* + - * 1. Физическое резервное копирование

Время создания физических резервных копий ниже, чем логи- ческих, т.к. в процессе создания резервной копии происходит только запись данных содержащихся в файлах БД, в то время как при создании логической копии происходит преобразование физи- ческих данных к их логической структуре и запись соответствую- щих команд SQL в файл.

Время восстановления физических резервных копий также ниже, чем логических, т.к. фактически происходит перезапись фай- лов базы данных данными из копии, в то время как при восстанов- лении логической копии происходит последовательное исполнение SQL-скрипта для создания объектов БД и вставки данных.

В то же время, для выполнения физической резервной копии сервер как правило должен быть остановлен. Существуют подход к резервному копированию файловой системы, который заключается в создании «целостного снимка» каталога с данными, если это под- держивает файловая система. Типичная процедура включает созда- ние «замороженного снимка» тома, содержащего базу данных, за- тем копирование всего каталога с этого снимка на устройство резервного копирования, и наконец освобождение замороженного снимка. При этом сервер базы данных может не прекращать свою работу. Однако резервная копия, созданная таким способом, содер- жит файлы базы данных в таком состоянии, как если бы сервер баз данных не был остановлен штатным образом.

Объем физических и логических копий сильно зависит от структуры базы данных. С одной стороны, логическая копия пред- ставляет собой SQL скрипт в некомпактном текстовом виде, где каждой записи в каждой таблице соответствует оператор INSERT, что, конечно, больше, чем размер физических файлов. С другой сто- роны, логические копии могут не содержать индексов, т.к. доста- точно команд их пересоздания.

Физические резервные копии обладают низкой степенью пере- носимости, как правило базы данных из этих копий могут быть вос- становлены только на сервере с аналогичной архитектурой, под управлением той же ОС и в той же версии СУБД, в то время как логические копии позволяют восстановить базу данных на сервере с иной архитектурой, под управлением другой ОС и в другой версии СУБД, при наличии совместимости на уровне команд. Поскольку СУБД, как правило, не имеют обратной совместимости по форма- там резервных копий, иногда использование логической резервной копии остается единственным вариантом переноса базы данных на сервер с более ранней версией СУБД.

* + - * 1. Логическое резервное копирование

Логическое резервное копирование не требует остановки сер- вера. Логические копии могут быть относительно легко перенесены на компьютер/БД с другой архитектурой системы.

Основные недостатки использования логического резервного копирования заключаются во времени работы. Время создания ло- гической резервной копии существенно выше, чем физической. Время восстановления из логической резервной копии также выше, так как требуется загружать и интерпретировать команды добавле- ния данных и перестраивать индексы.

Важно, что и физические резервные копии, и логические могут восстанавливаться с ошибками. Однако физическая копия либо вос- становится, либо нет, а логическая копия может восстановиться не вполне корректно. Ошибки могут возникать в части скрипта, опи- сывающей вставку данных, и обычно причиной тому служат раз- личные проблемы в строковых данных (например, проблемы с ко- дировкой). Соответственно, строки данных, при вставке которых произошла ошибка, в восстанавливаемую базу данных не попадут, что в свою очередь может привести к нарушению ссылочной це- лостности. Поэтому при восстановлении из логических копий же- лательно вести лог ошибок и анализировать его.

Логическое резервное копирование обычно больше подходит для решения задач клонирования и переноса баз данных, для реше- ния задачи восстановления баз данных после сбоев желательно ис- пользовать физические резервные копии, если используемая СУБД предоставляет такие возможности.

### Классификация резервного копирования по доступности сервера

**Автономным резервным копированием** называют процесс создания резервной копии на остановленном сервере, т.е. на момент

резервного копирования работа с БД не ведется, файлы БД не изме- няются, и все данные находятся в согласованном состоянии. Благо- даря этому можно быстро скопировать файлы, не беспокоясь о со- хранении состояния на текущий момент, пока другие процессы читают и записывают данные. Такие виды резервного копирования проще в реализации, но требуют остановки сервера, что часто не- возможно.

**Оперативным (горячим) резервным копированием** назы- вают процесс создания резервной копии, не требующий остановки сервера, т.е. на момент резервного копирования пользователи про- должают работу с БД. Кроме того, при оперативном резервном ко- пировании возрастает нагрузка на сервер. Такой вид резервного ко- пирования сложнее, т.к. после копирования файлов данных требуется еще и анализ журнала транзакций, но является единствен- ным доступным видом резервирования для критических систем.

### Факторы планирования резервного копирования

### Показатели RTO и RPO

Расписание для каждого типа резервного копирования будет зависеть от показателей **RTO** и **RPO** информационной системы. С этими показателями желательно определиться до этапа планиро- вания стратегии резервного копирования и восстановления баз дан- ных. Без понимания этих показателей есть вероятность непра- вильно оценить риски, сопряженные с отсутствием доступа к данным, и спланировать неподходящую стратегию резервирова- ния и восстановления.

RPO (Recovery Point Objective) – это максимальный период вре- мени, за который могут быть потеряны данные в результате инци- дента. Допустим, имеется информационная система и показатель RPO для нее определен в один час. Это значит, что при аварии мы

готовы к тому, что после восстановления системы допускается по- теря данных не более, чем за один последний час. В определенных ситуациях количество потерянных данных может быть и меньше, но ни при каких условиях не более одного часа. Этот показатель говорит о том, как часто необходимо выполнять резервирование данных, и какие технологии применять, чтобы выполнять этот показатель. Тео- ретически этот показатель может быть равен 0, но на практике орга- низовать это достаточно сложно. Это можно организовать, только, если запись идет как минимум в два разных хранилища.

RTO (Recovery Time Objective) – это промежуток времени, в те- чение которого система может оставаться недоступной в случае аварии. Допустим, в центре обработки данных произошел пожар, но необходимо, чтобы система была снова доступна для работы через два часа – это и есть показатель RTO. Необходимо планировать резервирование и восстановление данных так, чтобы за указанный промежуток восстановить работоспособность информационной системы на резервном оборудовании или площадке. Это можно реализовать с помощью различных технологий, однако в любом случае необходимо обеспечить этот показатель при инциденте.

### Возможность тестирования резервных копий

Также на планирование резервного копирования данных ока- зывает влияние то, какие имеются возможности тестирования ре- зервных копий, чтобы проверить, что данные действительно могут быть восстановлены. При минимальной возможности тестирования резервных копий в первую очередь следует рассматривать страте- гию полного резервного копирования БД по причине его макси- мальной надежности. Однако, если имеется возможность деталь- ного тестирования дифференциальных резервных копий или резервных копий журнала транзакций, можно запланировать пре- имущественное выполнение таких типов резервирования, а полное

резервное копирование выполнять нечасто, например, раз в неделю или в месяц.

### Характеристики базы данных

Объем хранимых данных и режим их использования также вли- яют на планирование стратегии резервного копирования. Если дан- ных меняются достаточно редко, то можно обойтись только полным резервным копированием, которое выполняется достаточно часто. Очевидно, что чем больше объем данных, тем медленнее будет про- исходить их полное резервное копирование, и, если режим доступа к данным подразумевает внепиковый период (например, ночью), рекомендуется запланировать полное резервное копирование базы данных именно на это время. При высокой интенсивности измене- ния данных желательно подключать периодическое разностное ко- пирование или копирование журнала транзакций.

### Ограничения на ресурсы

Ограничения на ресурсы, такие как оборудование, персонал, объем хранилища резервных копий и его физическая безопасность также влияют на стратегии резервного копирования. Если с аппа- ратными ограничениями все достаточно прозрачно, то ограничения на персонал, например, могут выражаться в том, что нет возможно- сти держать специалиста, ответственного за тестирование резерв- ных копий, и это в некотором смысле определяет возможную стра- тегию резервирования данных.

### Стратегии резервного копирования

Стратегия резервного копирования определяется рассмотрен- ными выше факторами, и для высоконагруженных многопользова- тельских систем с высокой стоимостью времени простоя такие стратегии индивидуальны, вплоть до аппаратной части.

Единственной стратегии резервного копирования, которая под- ходила бы всем пользователям, не существует. Стратегия, хорошо работающая в системе с одним пользователем, может оказаться не- пригодной для системы, обслуживающей сто пользователей. Ана- логично, стратегия, разработанная для системы, в которой каждый день меняется множество файлов, окажется неэффективной в си- стеме, в которой данные меняются редко.

Однако существуют некоторые базовые стратегии, примени- мые в большинстве случаев для массовых информационных систем:

* «простая» стратегия. Это стратегия ежедневного полного резервного копирования, обычно ночью. При этом обычно хра- нятся все резервные копии за последнюю неделю или две, а также резервные копии на начало месяца (чтобы можно было при необходимости восстановить базу данных и использовать ее для формирования отчетности). Такая стратегия макси- мально проста в реализации, не требует тестовых серверов для проверки восстановления и имеет хороший показатель RTO. Что касается показателя RPO, то нетрудно понять, что он мо- жет достигать восьми часов (при учете использование БД только в рабочее время). Фактически, в худшем случае допус- кается потеря данных за весь рабочий день. Несмотря на про- стоту и надежность, такая стратегия может быть рекомендо- вана только для БД с низкой интенсивностью изменения данных;
* «сбалансированная» стратегия. Этот вариант стратегии ис- пользует все три вида резервного копирования. Стратегия за- ключается в том, что полное резервное копирование выполня- ется раз в неделю (например, в воскресенье), разностное резервное копирование выполняется каждую ночь, и несколько раз в день выполняется резервное копирование журналов тран- закций. В отличии от «простой» стратегии, реализация этой

стратегии требует тестирования резервных копий, т.к. исполь- зует потенциально ненадёжные типы резервного копирования. Также «сбалансированная» стратегия несколько хуже по пока- зателю RTO, но может быть значительно лучше по показателю RPO, который меньше и зависит от частоты резервных копий журнала транзакций в течение рабочего дня. Такая стратегия может быть рекомендована как для больших баз данных (де- сятки и сотни гигабайт), так и для тех баз данных, которые ак- тивно изменяются.

Но, конечно, в зависимости от факторов, влияющих на плани- рования резервирования данных, в конкретных случаях могут при- меняться и другие стратегии, реализующие некие промежуточные варианты.

### Общие рекомендации по резервированию данных

Резервные копии не следует хранить на том же физическом хранилище, что и файлы исходной базы данных. В таком случае, если возникнет какая-либо проблема с диском, будут потеряны и база данных, и ее резервная копия. Следует помнить, что физиче- ские устройства, на которых хранятся резервные копии, также под- вержены износу. Поэтому рекомендуется хранить несколько ре- зервных копий на разных физических дисках или в хранилищах с зеркалированием данных. При организации хранения резервных ко- пий также стоит помнить о том, что резервная копия – это абсо- лютно та же база данных, для которой необходимо учитывать во- просы конфиденциальности данных и уделить надлежащее внимание безопасности хранилища.

Автоматизацию процессов резервного копирования необхо- димо настроить по заданному расписанию. В этом случае исключа-

ется человеческий фактор, и можно быть уверенным, что все ре- зервные копии, необходимые для восстановления в случае инци- дента, сделаны и актуальны. Также необходимо проверять, что за- планированные задачи по созданию резервных копий успешно выполняются.

Рекомендуется использовать тестовый сервер для проверки процедуры восстановления резервных копий. Даже если процесс резервного копирования завершился успешно, есть шанс, что вос- становление данных из этой резервной копии невозможно. Един- ственный способ убедиться в корректности резервной копии – это выполнить процесс восстановления в реальном сценарии, на тесто- вом сервере. Это минимизирует вероятность ошибки восстановле- ния в случае инцидента. Кроме того, проведение восстановления на тестовом сервере позволяет оценить показатель RTO и, соответ- ственно, адекватность стратегии резервирования данных в целом.

Следующая рекомендация – резервирование системных БД. Как правило, в любой СУБД критические данные хранятся в си- стемных базах данных (в MySQL это БД mysql, в PostgreSQL – БД postgres). В системных базах данных хранятся параметры кон- фигурации сервера, параметры конфигурации отдельных баз дан- ных, права доступа к объектам и т.д. В связи с этим рекомендуется резервировать не только пользовательские БД, но и служебные, так как при возникновении инцидента, если повреждены системные базы данных, невозможно будет восстановить доступ к данным в разумное время, не имея их резервных копий.

Не всегда риск прекращения нормального функционирования базы данных и клиентского программного обеспечения сопряжен с какими-либо инцидентами. Действия по изменению структуры базы данных, массовое изменение данных, операции массового им- порта данных из сторонних источников, ввод новой функциональ-

ности в клиентское ПО и т.д. – всё это может привести к несогласо- ванности данных и, соответственно, к невозможности штатного функционирования БД и клиентских приложений. В связи с чем ре- комендуется выполнять внеплановое полное резервное копирова- ние данных в указанных случаях.

В добавок, рекомендуется использовать все предоставляемые СУБД доступные параметры верификации при выполнении резерв- ного копирования. Это позволит быть уверенным в том, что все со- здаваемые резервные копии будут созданы корректно и последова- тельны с точки зрения транзакций.

### Резервное копирование в MySQL

Развитая система физического резервного копирования MySQL Enterprise Backup в настоящий момент предлагается только в составе коммерческой версии MySQL Enterprise Edition. Эта си- стема поддерживает высокопроизводительное резервирование дан- ных в оперативном или горячем режимах, инкрементное резервное копирование, частичное и выборочное резервное копирование, шифрование данных алгоритмом AES-256, компрессию резервных копий и другие функции, свойственные подсистемам резервного копирования современных СУБД. Физическое резервное копирова- ние выполняется значительно быстрее, чем логическое копирова- ние с использование команды mysqldump.

Создание логических резервных копий доступно с использо- ванием инструмента mysqldump, который позволяет выполнить оперативное логическое резервное копирование, не блокируя таблицы.

Свободно распространяемая версия СУБД MySQL предлагает только один способ полного физического резервного копирования:

копирование файлов базы данных средствами ОС. Файловая орга- низация базы данных MySQL (для движка InnoDB) подразумевает следующий набор файлов:

* файлы данных (ibdata\* и. ibd);
* лог-файлы (ib\_logfile\*);
* файл конфигурации (my.cnf).

Для копирования указанных файлов требуется остановить сер- вер. Такой способ не всегда приемлем, так как резервирование про- изводится в автономном режиме, однако альтернативой является логическая резервная копия, восстановление из которой происхо- дит чрезвычайно медленно.

Другим способом получения копии данных является формиро- вание текстовых файлов с данными. Для создания текстового файла, содержащего данные таблицы, можно использовать ко- манду:

**SELECT** \* **INTO OUTFILE** 'file\_name' **FROM** table\_name;

Файл создается на узле сервера MySQL, а не на узле клиента. Для создания структуры таблиц необходимо дополнительно созда- вать файлы, которые содержат команды CREATE TABLE.

Следующим способом создания текстовых файлов с данными

является использование утилиты mysqldump с параметром -- tab.

В СУБД MySQL поддерживается инкрементное резервное ко- пирование, которое основано на копировании бинарных лог-фай- лов. Такой лог файл является журналом транзакций, который содер- жит «события», описывающие изменения базы данных, такие как операции создания таблицы или изменения данных таблицы. Ко- манды из журнала транзакции могут использоваться для изменения состояния базы данных.

Также одним из способов является создание резервных копий с использованием реплик. Если возникают проблемы с производи- тельностью исходного сервера при создании резервных копий, одна из стратегий, которая может помочь – настроить репликацию. Ре- пликация – это механизм синхронизации содержимого нескольких копий объекта. После получения точной копии базы данных на дру- гом сервере можно выполнять резервное копирование на реплике, а не на источнике.

### Логическое резервное копирование в MySQL

Для создания логической резервной копии базы данных в ком- плекте с MySQL Server поставляется утилита mysqldump, которая формирует последовательность операторов SQL, при выполнении которых восстанавливаются исходные определения объектов базы данных и данных таблиц.

Утилита поддерживает два типа выходных данных:

* без параметра --tab операторы SQL записываются в стан- дартный выходной поток. Выходные данные состоят из операто- ров CREATE для создания объектов (баз данных, таблиц, храни- мых процедур и т.д.) и операторов INSERT для загрузки данных в таблицы. Вывод можно сохранить в файле и загрузить с помо- щью mysql для воссоздания выгруженных объектов;
* с параметром --tab mysqldump создает два выходных файла для каждой выгруженной таблицы. Сервер записывает один файл в виде текста с разделителями табуляции, по одной строке на строку таблицы. Этот файл называется tbl\_name.txt в выходном каталоге. Оператор CREATE TABLE записывается в файл с именем tbl\_name.sql в выходном каталоге.

По умолчанию mysqldump записывает результат в стандарт- ный выходной поток. Результат можно сохранить в файл следую- щим образом:

$> mysqldump [arguments] > file\_name

Утилита может быть использована для создания резервной ко- пии всех БД сервера или заданных БД:

$> mysqldump --all-databases > dump.sql

$> mysqldump --databases db1 db2 db3 > dump.sql

В случае создания резервной копии одной БД можно опустить опцию

--databases:

$> mysqldump test > dump.sql

В этом случае дамп не содержит инструкций CREATE DATABASE или USE:

* при загрузке копии необходимо указать имя базы данных по умолчанию, чтобы сервер знал, какую базу данных необходимо восстанавливать;
* при загрузке копии можно указать имя базы данных, отличное от исходного имени, что позволит повторно загрузить данные в другую БД;
* БД для загрузки данных должна быть предварительно создана. Утилита mysqldump поддерживает более 100 параметров, ос-

новные из них представлены в таблице 5.1.

Таблица 5.1. Основные параметры утилиты mysqldump

|  |  |
| --- | --- |
| **Параметры** | **Описание** |
| --host | Адрес MySQL сервера |
| --port | Порт MySQL сервера |
| --user | Имя пользователя MySQL сервера |
| --password | Пароль пользователя MySQL сервера |
| --routines | Включать в резервную копию хранимые проце-  дуры и функции |

*Окончание табл. 5.1*

|  |  |
| --- | --- |
| **Параметры** | **Описание** |
| --single- transaction | Создать резервную копии в рамках одной транзак- ции |
| --source-data | Включить в резервную копию позицию и имя би- нарного лог-файла сервера на момент создания ко-  пии |
| --flush-logs | Создать новый бинарный лог-файл перед созданием  резервной копии |

Параметры --source-data и --flush-logs необходимы для настройки инкрементного резервного копирования с использо- ванием бинарных логов.

Для восстановления БД из резервной копии необходимо ис- пользовать файл с дампом в качестве входных данных для клиента mysql. Если файл дампа был создан с опцией --all-databases или --databases, то он содержит операторы CREATE DATABASE и USE, и нет необходимости указывать базу данных по умолчанию, в которую следует загружать данные:

$> mysql --user root --password < dump.sql

Из mysql то же действие может быть выполнено с использо- ванием команды source:

mysql> source dump.sql

Если файл представляет собой дамп одной базы данных, не со- держащий операторов CREATE DATABASE и USE, сначала необхо- димо создать базу данных, затем указать ее при импорте данных:

$> mysql db1 < dump.sql

Другой способ – подключиться к mysql, создать БД командой CREATE DATABASE, выбрать ее как используемую и выполнить импорт командой source.

### Инкрементное резервное копирование в MySQL

В MySQL поддерживается инкрементное резервное копирова- ние, которое основано на копировании двоичных журналов. Двоич- ный журнал содержит события, которые описывают изменения в БД (например, создание таблиц, изменения в данных таблицы и т.д.) и представляет собой упорядоченную последовательность одноимен- ных файлов (обычно это server\_name-bin) c возрастающим чис- ловым расширением (SERV-bin.000001, SERV-bin.000002, SERV- bin.000003 и т.д.). Запись событий ведется в текущий журнал – последний файл в последовательности. Новый файл двоичного жур- нала создаются сервером автоматически при каждом перезапуске сервера, по достижении текущим журналом предельного размера или при выполнении сброса журнала с помощью команды FLUSH LOGS. Создание инкрементной резервной копии состоит из следую- щих шагов. Каждый раз, когда необходимо сделать инкрементную резервную копию (содержащую все изменения, произошедшие со времени последнего полного или инкрементного резервного копи- рования), необходимо сбросить двоичный журнал с помощью ко- манды FLUSH LOGS и скопировать все двоичные журналы, начи- ная с момента последнего полного или инкрементного резервного копирования в хранилище резервных копий. Этот набор двоичных журналов и является инкрементной резервной копией. Соответ- ственно, каждый раз при создании полной резервной копии также необходимо сбрасывать журнал, используя команду FLUSH LOGS

или утилиту mysqldump с параметром --flush-logs.

Для восстановления из инкрементной резервной копии сначала необходимо восстановить полную резервную копию, а затем после- довательно восстановить изменения из файлов журнала.

Для восстановления инкрементной резервной копии использу- ется утилита mysqlbinlog, которая используется для обработки журналов:

$> mysqlbinlog [arguments] log\_file

Для просмотра журнала в консоли или записи содержимого в текстовый файл можно выполнить следующие команды:

$> mysqlbinlog AGAFONOV-bin.000024

$> mysqlbinlog AGAFONOV-bin.000024 > incremen- tal.log

Сохраненный файл можно использовать для восстановления данных:

$> mysql --user root --password < incremental.log

Можно выполнить восстановление без использования проме- жуточного файла. Если необходимо восстановить из нескольких файлов журнала, лучше это делать одной командой:

$> mysqlbinlog AGAFONOV-bin.000024 AGAFONOV- bin.000025 | mysql -u root -p

Можно также выполнить восстановление на конкретный мо- мент времени:

$> mysqlbinlog --stop-datetime="2022-11-01 9:59:59" AGAFONOV-bin.000024 | mysql -u root –p

### Резервное копирование в PostgreSQL

PostgreSQL поддерживает три разных подхода к резервному копированию данных:

* логическое копирование – выгрузка в SQL;
* физическое копирование на уровне файлов;
* непрерывное архивирование, являющееся по сути инкре- ментным резервным копированием.

### Логическое резервное копирование в PostgreSQL

Логическое резервное копирование в СУБД PostgreSQL выпол- няется с помощью утилиты pg\_dump. Эта программа генерирует файл с командами SQL, которые при выполнении на сервере пере- создадут объекты базы данных в том же самом состоянии, в кото- ром они были на момент выгрузки. Результат работы программы записывается в стандартный поток вывода. Резервное копирование выполняется в оперативном режиме.

Синтаксис команды имеет следующий вид:

pg\_dump [***параметр подключения***...] [***пара- метр***...] [***имя\_бд***]

Список основных параметров представлен в таблице 5.2.

Таблица 5.2. Параметры pg\_dump

|  |  |
| --- | --- |
| **Параметр** | **Описание** |
| --host, -h | Адрес сервера |
| --port, -p | Порт сервера |
| --username, -  U | Имя пользователя, под которым производится под-  ключение |
| --password, -  W | Принудительно запрашивать пароль перед подклю-  чением |
| --format, -F | Формат вывода копии (plain, custom, directory, tar) |

Утилита поддерживает большое число параметров, среди кото- рых можно особо отметить формат вывода резервной копии. По умолчанию используется формат plain – формируется текстовый SQL-скрипт. Формат custom позволяет выгрузить данные в специ- альном архивном формате, пригодном для дальнейшего использо- вания утилитой pg\_restore. Наряду с форматом directory явля- ется наиболее гибким форматом, позволяющим вручную выбирать и сортировать восстанавливаемые объекты. Вывод в этом формате

по умолчанию сжимается. Directory выгружает данные в виде ката- лога, tar выгружает данные в формате tar, без сжатия.

Простейший вариант вызова утилиты выглядит следующим об- разом:

$> pg\_dump имя\_базы > файл\_дампа

Пример создания резервной копии БД pagila с подключение от пользователя postgres:

$> pg\_dump -U postgres pagila > pagila.sql

Текстовые файлы, созданные pg\_dump, используются для по- следующего чтения программой psql. Общий вид команды для восстановления дампа:

$> psql имя\_базы < файл\_дампа,

где файл\_дампа — это файл, содержащий вывод команды pg\_dump. База данных, заданная параметром имя\_базы, не будет создана данной командой, поэтому ее необходимо создать самосто- ятельно перед запуском psql. Программа psql принимает пара- метры, указывающие сервер, к которому осуществляется подклю- чение, и имя пользователя, подобно pg\_dump.

Дампы, выгруженные не в текстовом формате, восстанавлива- ются утилитой pg\_restore:

$> pg\_restore -d имя\_базы файл\_дампа

По умолчанию, если происходит ошибка SQL, программа psql продолжает выполнение. Если же запустить psql с установ- ленной переменной ON\_ERROR\_STOP, это поведение поменяется, и psql завершится с кодом 3 в случае возникновения ошибки SQL:

$> psql --set ON\_ERROR\_STOP=on *имя\_базы* <

*файл\_дампа*

В любом случае возможно только частичное восстановление базы данных. В качестве альтернативы можно указать, что весь дамп должен быть восстановлен в одной транзакции, так что восстановле- ние либо полностью выполнится, либо полностью отменится. Вклю- чить данный режим можно, передав psql аргумент -1 или

--single-transaction. Выбирая этот режим, стоит учитывать, что даже незначительная ошибка может привести к откату восстанов- ления, которое могло продолжаться несколько часов. Однако это всё же может быть предпочтительней, чем вручную вычищать сложную базу данных после частично восстановленного дампа.

Благодаря способности pg\_dump и psql писать и читать ка- налы ввода/вывода, можно скопировать базу данных непосред- ственно с одного сервера на другой:

pg\_dump -h host1 имя\_базы | psql -h host2 имя\_базы

Программа pg\_dump выгружает только одну базу данных в один момент времени и не включает в дамп информацию о ролях и табличных пространствах (так как это информация уровня кла- стера, а не самой базы данных). Для удобства создания дампа всего содержимого кластера баз данных предоставляется программа pg\_dumpall, которая делает резервную копию всех баз данных кластера, а также сохраняет данные уровня кластера. Базовое ис- пользование этой команды имеет вид:

pg\_dumpall > файл\_дампа

Полученную копию можно восстановить с помощью psql:

psql -f файл\_дампа postgres

Для резервного копирования больших баз данных можно использовать сжатые дампы, разбивать выводимые данные на не- большие файлы средствами Unix или использовать специальный формат дампа pg\_dump.

### Резервное копирование на уровне файлов в PostgreSQL

Альтернативной стратегией резервного копирования является непосредственное копирование файлов, в которых PostgreSQL хра- нит содержимое базы данных

Однако существуют два ограничения, которые делают этот ме- тод непрактичным или как минимум менее предпочтительным по сравнению с pg\_dump:

* чтобы полученная резервная копия была работоспособной,

сервер баз данных должен быть остановлен. Запрещение всех подключений к серверу не даст необходимого результата из-за того, что tar и подобные средства не получают мгновенный снимок состояния файловой системы, кроме того, в сервере есть внутренние буферы;

* копирование и восстановление отдельных таблиц или баз данных в соответствующих файлах или каталогах также невоз- можно, потому что информацию, содержащуюся в этих фай- лах, нельзя использовать без файлов журналов транзакций, ко- торые содержат состояние всех транзакций. Без этих данных файлы таблиц непригодны к использованию. Таким образом, копирование на уровне файловой системы будет работать, только если выполняется полное копирование и восстановле- ние всего кластера баз данных.

Ещё один подход к резервному копированию файловой си- стемы заключается в создании «целостного снимка» каталога с дан- ными, если это поддерживает файловая система. Типичная проце- дура включает создание «замороженного снимка» тома, содержащего базу данных, затем копирование всего каталога с дан- ными (а не его отдельных частей) из этого снимка на устройство резервного копирования, и наконец освобождение замороженного снимка.

### Непрерывное архивирование в PostgreSQL

Другое возможное решение для создания физической резерв- ной копии – использование механизма непрерывного архивирова- ния и восстановление на момент времени (Point-in-Time Recovery, PITR). Такие резервные копии не могут пострадать от изменений файловой системы в процессе резервного копирования. Для этого требуется включить непрерывное архивирование только на время резервного копирования, для восстановления применяется проце- дура восстановления из непрерывного архива.

В процессе работы PostgreSQL ведётся журнал предзаписи (Write-Ahead Log, WAL), который расположен в подкаталоге pg\_wal/ каталога с данными кластера баз данных. В этот журнал записываются все изменения, вносимые в файлы данных. Прежде всего, журнал необходим для безопасного восстановления после аварийной остановки сервера. В этом случае целостность СУБД мо- жет быть восстановлена в результате «воспроизведения» записей, зафиксированных после последней контрольной точки.

Однако наличие журнала делает возможным использование инкрементной стратегии копирования баз данных. Можно сочетать резервное копирование на уровне файловой системы с копирова- нием файлов WAL.

Если потребуется восстановить данные, можно восстановить ко- пию файлов, а затем воспроизвести журнал из скопированных файлов WAL, и, таким образом, привести систему в нужное состояние.

По сути, этот подход аналогичен использованию бинарных журналов в MySQL.

#### Непрерывное архивирование WAL-файлов

Описанный подход с использованием непрерывного архивиро- вания WAL-файлов более сложен для администрирования, чем лю- бой из описанных раннее способов резервного копирования, но он имеет значительные преимущества:

* в качестве начальной точки для восстановления необяза- тельно иметь полностью согласованную копию на уровне фай- лов. Внутренняя несогласованность копии будет исправлена при воспроизведении журнала;
* поскольку при воспроизведении можно обрабатывать не- ограниченную последовательность файлов WAL, непрерыв- ную резервную копию можно получить, просто продолжая ар- хивировать файлы WAL. Это особенно ценно для больших баз данных, полные резервные копии которых делать как минимум неудобно;
* воспроизводить все записи WAL до самого конца нет необ- ходимости. Воспроизведение можно остановить в любой точке и получить целостный снимок базы данных на выбранный мо- мент времени. Таким образом, данная технология поддержи- вает восстановление на момент времени: можно восстановить состояние базы данных на любое время с момента создания ре- зервной копии;
* если непрерывно передавать последовательность файлов WAL другому серверу, получившему данные из базовой копии того же кластера, получается система «тёплого» резерва. В лю- бой момент можно запустить второй сервер, и он будет иметь практически текущую копию баз данных.

Также следует отметить, что программы pg\_dump и pg\_dumpall не создают копии на уровне файловой системы и не могут применяться как часть решения по непрерывной архивации. Создаваемые ими копии являются логическими и не содержат ин- формации, необходимой для воспроизведения WAL.

Как и обычное резервное копирование файловой системы, этот метод позволяет восстанавливать только весь кластер баз данных целиком, но не его части. Кроме того, для архивов требуется боль- шое хранилище: базовая резервная копия может быть объёмной,

а нагруженные системы будут генерировать многие мегабайты тра- фика WAL, который необходимо архивировать. Тем не менее этот метод резервного копирования является предпочтительным во мно- гих ситуациях, где необходима высокая надёжность.

Для успешного восстановления с применением непрерывного архивирования необходима непрерывная последовательность заар- хивированных файлов WAL, начинающаяся не позже, чем с мо- мента начала копирования. Так что для начала необходимо настро- ить процедуру архивирования файлов WAL до того, как необходимо получить первую базовую копию.

В абстрактном смысле, запущенная СУБД PostgreSQL произ- водит неограниченно длинную последовательность записей WAL. СУБД физически делит эту последовательность на файлы сегмен- тов WAL, которые обычно имеют размер 16 МБ (хотя размер сег- мента может быть изменён при инициализации). Файлы сегментов получают цифровые имена, которые отражают их позицию в аб- страктной последовательности WAL. Когда архивирование WAL не применяется, система обычно создаёт только несколько файлов сег- ментов и затем перезаписывает их. Предполагается, что файлы сег- ментов, содержимое которых предшествует последней контроль- ной точке, уже не представляют интереса и могут быть переработаны.

При архивировании данных WAL необходимо считывать со- держимое каждого файла-сегмента, как только он заполняется, и со- хранять эти данные, прежде чем файл-сегмент будет переработан и использован повторно.

В зависимости от применения и доступного аппаратного обес- печения, возможны разные способы сохранения данных: можно скопировать файлы-сегменты в смонтированный каталог на другом сервере, записать их на диск и т.д. PostgreSQL позволяет админи-

стратору указать команду оболочки или библиотеку архивирова- ния, которая будет запускаться для копирования файла завершён- ного сегмента в нужное место. Её действие может заключаться как в выполнении простых команд оболочки, включая copy, так и в вы- зове функции на языке C.

Чтобы включить архивирование WAL, необходимо установить в параметре конфигурации wal\_level уровень replica или выше, в archive\_mode – значение on и задать команду архиви- рования в параметре archive\_command. На практике эти пара- метры всегда задаются в файле postgresql.conf.

В archive\_command символы %p заменяются полным путём к файлу, подлежащему архивации, а %f заменяются только именем файла. Простейшая команда для архивации имеет следующий вид:

archive\_command = 'test ! -f /mnt/server/ar- chivedir/%f

&& cp %p /mnt/server/archivedir/%f' # Unix archive\_command = 'copy "%p"

"C:\\server\\archivedir\\%f"' # Windows

Для Windows команда будет копировать архивируемые сег- менты WAL в каталог C:\server\\archivedir\ .

Команда или функция архивирования вызывается, только ко-

гда сегмент WAL заполнен до конца. Таким образом, если сервер постоянно генерирует небольшой трафик WAL (или есть продол- жительные периоды, когда это происходит), между завершением транзакций и их безопасным сохранением в архиве может образо- ваться большая задержка. Чтобы ограничить время жизни неархи- вированных данных, можно установить archive\_timeout, чтобы сервер переключался на новый файл сегмента WAL как ми- нимум с заданной частотой.

#### Непрерывное архивирование. Базовая копия

Проще всего получить базовую резервную копию, используя программу pg\_basebackup. Эта программа выполняет оператив- ное копирование и сохраняет базовую копию в виде обычных фай- лов или в архиве tar. Для запуска утилиты указываются параметры подключения к БД и параметры создания резервной копии.

pg\_basebackup [***параметр-подключения***...] [***па- раметр***...]

Простейший вариант вызова выглядит следующим образом:

$> pg\_basebackup -D каталог

$> pg\_basebackup -U postgres -D "E:\\backup"

После выполнения этой команды в каталоге E\backup будет создана физическая резервная копия.

PostgreSQL не поддерживает встроенные средства для шифро- вания резервных копий, поэтому при необходимости следует защи- щать архивные данные внешними средствами шифрования.

Процесс создания базовой резервной копии записывает файл истории резервного копирования, который сохраняется в каталоге архивации WAL файлов. Данный файл получает имя по имени файла первого сегмента WAL, который потребуется для восстанов- ления скопированных файлов. Файл истории резервного копирова- ния – это текстовый файл, в который записывается метка, которая была передана pg\_basebackup, а также время и текущие сег- менты WAL в момент начала и завершения создания резервной ко- пии.

#### Непрерывное архивирование. Восстановление данных

Для восстановления данных из полной резервной копии (полу- ченной с помощью утилиты pg\_basebackup) и инкрементных ко- пий (файлы журналов WAL) следует выполнить следующее:

1. остановить сервер баз данных, если он запущен;
2. скопировать весь текущий каталог кластера баз данных и все табличные пространства во временный каталог на слу- чай, если они вам понадобятся. Как минимум, сохранить со- держимое подкаталога pg\_wal каталога кластера, так как он может содержать журналы, не попавшие в архив перед остановкой системы;
3. удалить все содержимое директории сервера, на котором будет выполняться восстановление данных;
4. скопировать содержимое папки, в которую была выполнена полная резервная копия, в директорию данных сервера;
5. удалить все файлы из pg\_wal/; они восстановились из ре- зервной копии файлов и поэтому, скорее всего, будут старее текущих;
6. если на шаге 2 были сохранены незаархивированные файлы с сегментами WAL, необходимо скопировать их в pg\_wal/. Сегменты WAL, которые не найдутся в архиве, система будет искать в pg\_wal/; благодаря этому можно использовать последние незаархивированные сегменты. Однако файлы в pg\_wal/ будут менее предпочтитель- ными, если такие сегменты окажутся в архиве;
7. для восстановления данных из инкрементных копий (архив- ных WAL-файлов) в конфигурационном файле сервера написать команду восстановления, например (для Windows):

restore\_command = 'copy "PATH\\%f" "%p"'

где PATH – это путь к каталогу с архивными файлами, кото- рый был настроен в переменной archive\_command. Для восстановления данных до заданного момента времени можно использовать параметр recovery\_target\_time;

1. создать пустой файл в директории сервера с названием

recovery.signal;

1. запустить сервер. Сервер запустится в режиме восстановле- ния и начнёт считывать необходимые ему архивные файлы WAL. По завершении процесса восстановления сервер уда- лит файл recovery.signal (чтобы предотвратить по- вторный запуск режима восстановления), а затем перейдёт к обычной работе с базой данных.