Tajemnice bconsole czyli pierwsze kroki z tekstową konsolą Bacula cz.2

8 styczeń 2011 autor: **gani**

Druga część artykułu o konsoli bconsole w całości poświęcona jest tematowi wybierania danych z bazy danych przy pomocy zapytań SQL. Do lepszego zrozumienia treści pomocna może okazać się podstawowa wiedza z zakresu SQL.

Wstęp

Omówione w poprzedniej części artykułu komendy wybierania danych o wolumenach, zadaniach czy przetworzonych plikach są niczym innym jak pobieraniem informacji z bazy danych poprzez program bconsole. Niejednokrotnie komendy wybierające dane mogą okazać się dla czytelnika wystarczające. Co jednak, gdy zajdzie potrzeba uzyskania konkretnych informacji, które nie udostępnia żadna z komend konsoli Bacula, lub jeśli czytelnik zachce połączyć wyniki dwóch lub więcej operacji w jeden wynik? Tutaj przychodzą z pomocą zapytania SQL.

Gdzie wpisywać zapytania SQL

Istnieją co najmniej dwa miejsca, gdzie można by wybrać żądane dane z bazy danych. Pierwsze z nich to konsola udostępniona jako jeden z programów bazy danych. Drugim miejscem jest konsola bconsole. Ta ostatnia daje takie udogodnienie, że w codziennej pracy z bacula nie potrzeba przełączać się pomiędzy konsolą bconsole a konsolą bazy danych w celu wydania zapytania SQL. W niniejszym artykule preferowaną metodą wydawania zapytań do bazy danych jest właśnie konsola bconsole.

Do wpisania zapytania SQL w konsoli Bacula służy komenda o nazwie **sql**, po której wywołaniu ukaże się specjalny znak zachęty podobny do tego:

```
*sql
Automatically selected Catalog: MyCatalog
Using Catalog "MyCatalog"
Entering SQL query mode.
Terminate each query with a semicolon.
Terminate query mode with a blank line.
Enter SQL query:
```

Struktura tabel bazy danych

Aby móc wykonywać zapytania do bazy danych potrzebna jest wiedza o strukturze tabel bazy. Strukturę tą można łatwo uzyskać np. poprzez klienta (konsolę) bazy danych. Aby nie robić tego za każdym razem, gdy użytkownik zachce skonstruować nowe zapytanie SQL, można wykonać zrzut struktury tabel bazy danych przechowującej dane Bacula do pliku (tzw. dump). Łatwiejszym rozwiązaniem jest jednak podglądnięcie struktury tabel ze skryptu Bacula, który te tabele stworzył. W zależności od rodzaju bazy danych, jakiej się używa, skrypt może nazywać się:

- make_mysql_tables
- make_postresql_tables
- make_sqlite3_tables

a znaleźć go można:

- w przypadku własnej kompilacji - w źródłach Bacula (katalog src/cats/) lub w katalogu skryptów (przełącznik --with-scriptdir),

- w przypadku użycia pakietu binarnego – w różnej lokalizacji (w zależności od systemu operacyjnego i jego dystrybucji). Trzeba poszukać.

Poniżej przedstawiam wycinki skryptu tworzącego tabele bazy danych Bacula (tabele: Job, Media, Storage, Pool, Client):

```
TABLE Job (
  JobId INTEGER,
  Job VARCHAR(128) NOT NULL,
  Name VARCHAR(128) NOT NULL,
  Type CHAR(1) NOT NULL,
  Level CHAR(1) NOT NULL,
  ClientId INTEGER REFERENCES Client DEFAULT 0,
  JobStatus CHAR(1) NOT NULL,
  SchedTime DATETIME NOT NULL,
  StartTime DATETIME DEFAULT 0,
  EndTime DATETIME DEFAULT 0,
  RealEndTime DATETIME DEFAULT 0,
  JobTDate BIGINT UNSIGNED DEFAULT 0,
  VolSessionId INTEGER UNSIGNED DEFAULT 0,
  VolSessionTime INTEGER UNSIGNED DEFAULT 0,
  JobFiles INTEGER UNSIGNED DEFAULT 0,
  JobBytes BIGINT UNSIGNED DEFAULT 0,
  ReadBytes BIGINT UNSIGNED DEFAULT 0,
  JobErrors INTEGER UNSIGNED DEFAULT 0,
  JobMissingFiles INTEGER UNSIGNED DEFAULT 0,
  PoolId INTEGER UNSIGNED REFERENCES Pool DEFAULT 0,
  FileSetId INTEGER UNSIGNED REFERENCES FileSet DEFAULT 0,
  PriorJobId INTEGER UNSIGNED REFERENCES Job DEFAULT 0,
  PurgedFiles TINYINT DEFAULT 0,
  HasBase TINYINT DEFAULT 0,
  HasCache TINYINT DEFAULT 0,
  Reviewed TINYINT DEFAULT 0,
  Comment TEXT,
  PRIMARY KEY(JobId).
);

TABLE Media (
  MediaId INTEGER,
  VolumeName VARCHAR(128) NOT NULL,
  Slot INTEGER DEFAULT 0,
  PoolId INTEGER UNSIGNED REFERENCES Pool DEFAULT 0,
  MediaType VARCHAR(128) NOT NULL,
  MediaTypeId INTEGER UNSIGNED REFERENCES MediaType DEFAULT 0,
  LabelType TINYINT DEFAULT 0,
  FirstWritten DATETIME DEFAULT 0,
  LastWritten DATETIME DEFAULT 0,
  LabelDate DATETIME DEFAULT 0,
  VolJobs INTEGER UNSIGNED DEFAULT 0,
  VolFiles INTEGER UNSIGNED DEFAULT 0,
  VolBlocks INTEGER UNSIGNED DEFAULT 0,
  VolMounts INTEGER UNSIGNED DEFAULT 0,
  VolBytes BIGINT UNSIGNED DEFAULT 0,
  VolParts INTEGER UNSIGNED DEFAULT 0,
  VolErrors INTEGER UNSIGNED DEFAULT 0,
  VolWrites INTEGER UNSIGNED DEFAULT 0,
  VolCapacityBytes BIGINT UNSIGNED DEFAULT 0,
  VolStatus VARCHAR(20) NOT NULL,
  Enabled TINYINT DEFAULT 1,
  Recycle TINYINT DEFAULT 0,
  ActionOnPurge TINYINTDEFAULT 0,
  VolRetention BIGINT UNSIGNED DEFAULT 0,
```

```
  VolUseDuration BIGINT UNSIGNED DEFAULT 0,
  MaxVolJobs INTEGER UNSIGNED DEFAULT 0,
  MaxVolFiles INTEGER UNSIGNED DEFAULT 0,
  MaxVolBytes BIGINT UNSIGNED DEFAULT 0,
  InChanger TINYINT DEFAULT 0,
  StorageId INTEGER UNSIGNED REFERENCES Storage DEFAULT 0,
  DeviceId INTEGER UNSIGNED REFERENCES Device DEFAULT 0,
  MediaAddressing TINYINT DEFAULT 0,
  VolReadTime BIGINT UNSIGNED DEFAULT 0,
  VolWriteTime BIGINT UNSIGNED DEFAULT 0,
  EndFile INTEGER UNSIGNED DEFAULT 0,
  EndBlock INTEGER UNSIGNED DEFAULT 0,
  LocationId INTEGER UNSIGNED REFERENCES Location DEFAULT 0,
  RecycleCount INTEGER UNSIGNED DEFAULT 0,
  InitialWrite DATETIME DEFAULT 0,
  ScratchPoolId INTEGER UNSIGNED REFERENCES Pool DEFAULT 0,
  RecyclePoolId INTEGER UNSIGNED REFERENCES Pool DEFAULT 0,
  Comment TEXT,
  PRIMARY KEY(MediaId)
);

TABLE Storage (
  StorageId INTEGER,
  Name VARCHAR(128) NOT NULL,
  AutoChanger TINYINT DEFAULT 0,
  PRIMARY KEY(StorageId)
);

TABLE Pool (
  PoolId INTEGER,
  Name VARCHAR(128) NOT NULL,
  NumVols INTEGER UNSIGNED DEFAULT 0,
  MaxVols INTEGER UNSIGNED DEFAULT 0,
  UseOnce TINYINT DEFAULT 0,
  UseCatalog TINYINT DEFAULT 1,
  AcceptAnyVolume TINYINT DEFAULT 0,
  VolRetention BIGINT UNSIGNED DEFAULT 0,
  VolUseDuration BIGINT UNSIGNED DEFAULT 0,
  MaxVolJobs INTEGER UNSIGNED DEFAULT 0,
  MaxVolFiles INTEGER UNSIGNED DEFAULT 0,
  MaxVolBytes BIGINT UNSIGNED DEFAULT 0,
  AutoPrune TINYINT DEFAULT 0,
  Recycle TINYINT DEFAULT 0,
  ActionOnPurge TINYINTDEFAULT 0,
  PoolType VARCHAR(20) NOT NULL,
  LabelType TINYINT DEFAULT 0,
  LabelFormat VARCHAR(128) NOT NULL,
  Enabled TINYINT DEFAULT 1,
  ScratchPoolId INTEGER UNSIGNED REFERENCES Pool DEFAULT 0,
  RecyclePoolId INTEGER UNSIGNED REFERENCES Pool DEFAULT 0,
  NextPoolId INTEGER UNSIGNED REFERENCES Pool DEFAULT 0,
  MigrationHighBytes BIGINT UNSIGNED DEFAULT 0,
  MigrationLowBytes BIGINT UNSIGNED DEFAULT 0,
  MigrationTime BIGINT UNSIGNED DEFAULT 0,
  UNIQUE (Name),
  PRIMARY KEY (PoolId)
);

TABLE Client (
  ClientId INTEGER,
  Name VARCHAR(128) NOT NULL,
```

```
  Uname VARCHAR(255) NOT NULL,-- uname -a field
  AutoPrune TINYINT DEFAULT 0,
  FileRetention BIGINT UNSIGNED DEFAULT 0,
  JobRetention  BIGINT UNSIGNED DEFAULT 0,
  UNIQUE (Name),
  PRIMARY KEY(ClientId)
);
```

Listing 1: *wycinki skryptu tworzącego tabele bazy danych Bacula.*

W tym przypadku powyższe dane pochodzą ze skryptu tworzenia tabel bazy danych SQLite.

Zapytania SQL

Na początek spróbuję wybrać same nazwy zaetykietowanych wolumenów. Zgodnie z listingiem 1 tabelą przechowującą te informacje jest tabela **Media**, a kolumna z nazwami woluminów nazywa się **VolumeName**. Zapytanie może mieć postać:

```
SELECT VolumeName AS VolName FROM Media;
```

W konsoli bconsole wpisuję wspomnianą już komendę sql oraz powyższe zapytanie SQL:

```
 *sql
Automatically selected Catalog: MyCatalog
Using Catalog "MyCatalog"
Entering SQL query mode.
Terminate each query with a semicolon.
Terminate query mode with a blank line.
Enter SQL query: SELECT VolumeName AS VolName FROM media;
+-----------+
| VolName   |
+-----------+
| Full-0001 |
| DB-0002   |
| Inc-0003  |
+-----------+
```

(zapytanie SQL zostało zaznaczone wytłuszczonym tekstem)

Otrzymanym wynikiem jest lista z nazwami woluminów. Widać na niej, że mam trzy woluminy o nazwach: **Full-0001**, **DB-0002** oraz **Inc-0003**.

Powyższe zapytanie można rozbudować o wyświetlanie informacji o nazwach puli woluminów, z których woluminy pochodzą. Patrząc na listing 1 można z niego wyczytać, że tabela **Media** nie przechowuje informacji o nazwach puli woluminów, lecz jedynie identyfikatory (indeksy) puli woluminów. Aby przetłumaczyć identyfikatory puli woluminów potrzeba sięgnąć do tabeli o nazwie **Pool**, w której to tabeli znajdują się zarówno identyfikatory puli woluminów jak i ich nazwy. Zapytanie SQL wyświetlające nazwy woluminów jak i nazwy puli woluminów, do których woluminy należą, może wyglądać jak poniżej:

```
SELECT Media.VolumeName AS VolName, Pool.Name AS Pool FROM Media, Pool WHERE
Media.PoolId=Pool.PoolId;
```

a jego wykonanie daje wynik:

```
Enter SQL query: SELECT Media.VolumeName AS VolName, Pool.Name AS Pool FROM Media,
Pool WHERE Media.PoolId=Pool.PoolId;
+-----------+----------+
| VolName   | Pool     |
+-----------+----------+
| Full-0001 | Sys-Full |
| DB-0002   | DB-Full  |
| Inc-0003  | Sys-Inc  |
+-----------+----------+
```

Z powyższego listingu można wyczytać, że:

- wolumen o nazwie Full-0001 należy do puli wolumenów o nazwie Sys-Full,
- wolumen o nazwie DB-0002 należy do puli wolumenów o nazwie DB-Full,
- wolumen o nazwie Inc-0003 należy do puli wolumenów o nazwie Sys-Inc.

Rozbudowując dalej zapytanie SQL dodam do niego wybieranie również informacji o urządzeniu, które zaetykietowało woluminy. Podobnie jak w przypadku pobierania nazwy puli woluminów, tutaj również potrzeba odwołać się do innej tabeli przy użyciu identyfikatorów (indeksów).

Moje zapytanie SQL wygląda następująco:

```
SELECT Media.VolumeName AS VolName, Pool.Name AS Pool, Storage.Name AS
Storage FROM Media, Pool, Storage WHERE Media.PoolId=Pool.PoolId AND Media.
StorageId=Storage.StorageId;
```

a jego wynik wygląda jak poniżej:

```
Enter SQL query: SELECT Media.VolumeName AS VolName, Pool.Name AS Pool, Storage.
Name AS Storage FROM Media, Pool, Storage WHERE Media.PoolId=Pool.PoolId AND
Media.StorageId=Storage.StorageId;
+-----------+----------+---------+
| VolName   | Pool     | Storage |
+-----------+----------+---------+
| Full-0001 | Sys-Full | File    |
| DB-0002   | DB-Full  | Plikowe |
| Inc-0003  | Sys-Inc  | File    |
+-----------+----------+---------+
```

Widać na nim dwa urządzenia o nazwach **File** i **Plikowe**.

Rozszerzę zapytanie SQL o wybieranie dodatkowo bieżących statusów woluminów. Zapytanie ma postać:

```
SELECT Media.VolumeName AS VolName, Pool.Name AS Pool, Storage.Name AS Storage,
Media.VolStatus AS VolStatus FROM Media, Pool, Storage WHERE Media.PoolId=Pool.
PoolId AND Media.StorageId=Storage.StorageId;
```

a jego wynik jest następujący:

```
Enter SQL query: SELECT Media.VolumeName AS VolName, Pool.Name AS Pool, Storage.
Name AS Storage, Media.VolStatus AS VolStatus FROM Media, Pool, Storage WHERE
Media.PoolId=Pool.PoolId AND Media.StorageId=Storage.StorageId;
+-----------+----------+---------+-----------+
| VolName   | Pool     | Storage | VolStatus |
+-----------+----------+---------+-----------+
| Full-0001 | Sys-Full | File    | Append    |
| DB-0002   | DB-Full  | Plikowe | Append    |
| Inc-0003  | Sys-Inc  | File    | Append    |
+-----------+----------+---------+-----------+
```

Z rezultatu można wyczytać, że wszystkie wolumeny oznaczone są statusem **Append**.

W podobny sposób do wybierania informacji o wolumenach przy pomocy zapytań SQL można wybierać wszystkie informacje składowane w bazie danych. Np. poniższe zapytanie zwróci kolejno: identyfikator zadania, nazwę zadania, ilość przetworzonych danych w kilobajtach oraz nazwę klienta, który został użyty do zadania:

```
SELECT Job.JobId AS JobId, Job.Name AS JobName, Job.JobBytes/1024 AS JobKBytes,
Client.Name AS Client FROM Job, Client WHERE Job.ClientId=Client.ClientId LIMIT 5;
```

Oto wynik:

```
Enter SQL query: SELECT Job.JobId AS JobId, Job.Name AS JobName, Job.JobBytes/1024
AS JobKBytes, Client.Name AS Client FROM Job, Client WHERE Job.ClientId=Client.
ClientId LIMIT 5;
+-------+---------------+-----------+-----------------+
| JobId | JobName       | JobKBytes | Client          |
+-------+---------------+-----------+-----------------+
| 1     | BackupClient1 | 21844     | gani-desktop-fd |
| 2     | BackupCatalog | 42        | gani-desktop-fd |
| 3     | BackupClient1 | 1314      | gani-desktop-fd |
| 4     | BackupCatalog | 50        | gani-desktop-fd |
| 5     | BackupClient1 | 1325      | gani-desktop-fd |
+-------+---------------+-----------+-----------------+
```

Zapamiętywanie zapytań

Bacula udostępnia możliwość gromadzenia własnych zapytań SQL w pliku, a następnie użycie poszczególnych zapytań z pliku w linii komend bconsole. Dzięki temu nie trzeba wpisywać zapytań za każdym razem, lecz tylko raz.

Plik ma następującą składnię:

```
:Komentarz do zapytania1
Zapytanie1
:Komentarz do zapytania2
Zapytanie2
```

Z omówionych w artykule zapytań SQL tworzę własny plik o nazwie **query.sql**, którego zawartość to:

```
:Lista wolumenow i odpowiadajacych im puli oraz storage
SELECT Media.VolumeName AS VolName, Pool.Name AS Pool, Storage.Name AS Storage
FROM Media, Pool, Storage
WHERE Media.PoolId=Pool.PoolId
AND Media.StorageId=Storage.StorageId;

:Lista pierwszych 5 zadan, ilosci danych w KB oraz uzytego klienta
SELECT Job.JobId AS JobId, Job.Name AS JobName, Job.JobBytes/1024 AS JobKBytes,
Client.Name AS Client
FROM Job, Client
WHERE Job.ClientId=Client.ClientId LIMIT 5;
```

Aby móc używać powyższych komend, potrzeba podać poprawną lokalizację w/w pliku w pliku konfiguracyjnym zarządcy **bacula-dir.conf** w zasobie **Director** w dyrektywie **QueryFile**, np.:

```
Director {
    …
     QueryFile = "/etc/bacula/scripts/query.sql"
    …
}
```

(trzy kropki w powyższym listingu nie są elementem składni a sugerują tylko, że jest to wycinek pliku konfiguracyjnego)

UWAGA!

Plik **query.sql** jest ładowany przy każdym uruchomieniu konsoli bconsole.

Użycie zapytań zapisanych w pliku **query.sql** odbywa się w linii komend bconsole poprzez użycie komendy **query**, która wyświetli menu z listą zapytań SQL do wyboru.

```
*query
Automatically selected Catalog: MyCatalog
Using Catalog "MyCatalog"
Available queries:
     1: Lista wolumenow i odpowiadajacych im puli oraz storage
     2: Lista pierwszych 5 zadan, ilosci danych w KB oraz uzytego klienta
Choose a query (1-2): 1
+-----------+----------+---------+
| VolName   | Pool     | Storage |
+-----------+----------+---------+
| Full-0001 | Sys-Full | File    |
| DB-0002   | DB-Full  | Plikowe |
| Inc-0003  | Sys-Inc  | File    |
+-----------+----------+---------+
*query
Available queries:
     1: Lista wolumenow i odpowiadajacych im puli oraz storage
     2: Lista pierwszych 5 zadan, ilosci danych w KB oraz uzytego klienta
```

```
Choose a query (1-2): 2
+-------+--------------+-----------+-----------------+
| JobId | JobName      | JobKBytes | Client          |
+-------+--------------+-----------+-----------------+
| 1     | BackupClient1 | 21844     | gani-desktop-fd |
| 2     | BackupCatalog | 42        | gani-desktop-fd |
| 3     | BackupClient1 | 1314      | gani-desktop-fd |
| 4     | BackupCatalog | 50        | gani-desktop-fd |
| 5     | BackupClient1 | 1325      | gani-desktop-fd |
+-------+--------------+-----------+-----------------+
```

Podsumowanie

Dzięki obsłudze własnych zapytań SQL w Bacula oraz odrobinie wyobraźni można stworzyć interfejs z dostępem do najpotrzebniejszych i najczęściej używanych informacji używając do tego mniej lub bardziej skomplikowanych zapytań SQL.

Omówiony w artykule specjalny plik do gromadzenia zapytań SQL (tutaj **query.sql**) posiada również możliwość parametryzowania zapytań, dzięki czemu pobieranie danych staje się bardziej elastyczne. Po więcej informacji w tym temacie odsyłam do przykładowego pliku query.sql dostarczonego wraz ze źródłami Bacula.