Report

Ročníkový projekt

Dennis Vita

Názov projektu:
Analýza dát výsledkov dlhodobých meraní vybraných charakteristík Internetu

1 Popis a cieľ projektu

Cieľom bolo pre existujúcu SQL databázu obsahujúcu výsledky dlhodobých periodických meraní času odozvy vybraných IP adries v sieti zanalyzovať kvalitu dát. Do databázy boli pravidelne zadávané výsledky behu 'ping' a 'traceroute' pre rôzne IP adresy. Hlavným cieľom tohoto ročníkového projektu bolo určiť a vybrať z databázy množinu spoľahlivých IP adries, z ktorých bude neskôr možné určiť potrebné celkové štatistiky časov odozvy, trendy ich zmeny, atď.

2 Pripojenie sa k databázovému serveru

Na databázovom serveri beží databáza PostgreSQL (verzia 8.1.9). Server nie je priamo dostupný, port na ktorom je dostupná databáza bol pre účely tohoto projektu preto presmerovaný na server 'davinci.fmph.uniba.sk', pomocou ktorého sa bolo možné pripojiť k vzdialenému databázovému serveru. Prístup z lokálneho stroja bol realizovaný tak, že zvolený port na lokálnom stroji bol presmerovaný na 'davinci.fmph.uniba.sk' (na port, z ktorého bol prístupný databázový server). Na samotné pripojenie k databáze bol použitý klient 'psql' voľne dostupný na všetkých štandardných UNIX-ových systémoch. Pre prihlásenie sa bolo potrebné zadať prihlasovacie údaje používateľa vytvoreného pre účely tohoto projektu. Takto bolo možné poskytnúť read-only prístup iba k vybraným, potrebným tabuľkám.

3 Logická štruktúra dát

Dáta boli organizované v 3 hlavných tabuľkách:

- **hosts**: Tabuľka obsahuje súbor všetkých registrovaných IP adries a k nim doplňujúce údaje, ktoré boli použité pri jednotlivých meraniach 'ping' a 'traceroute'. Táto množina IP adries tvorí teda univerzum všetkých uvažovaných IP adries.
- **ping**: Tabuľka obsahuje výsledky jednotlivých meraní 'ping' pre cieľové IP adresy z tabuľky 'hosts'. Pre každú IP adresu boli pravidelne 1-krát za deň do tejto tabuľky doplnené zistené výsledky meraní 'ping'. Významný bude najmä stĺpec 'ping_ploss', ktorý zodpovedá počtu stratených packetov počas merania v percentách (alebo

chybovému kódu v prípade záporného čísla). Tento stĺpec bude neskôr použitý pri určovaní množiny spoľahlivých IP adries.

- **topology**: Tabuľka obsahuje výsledky jednotlivých meraní 'traceroute' pre cieľové IP adresy z tabuľky 'hosts'. Do tejto tabuľky boli, opäť pre každú IP adresu z tabuľky 'hosts', pravidelne 1-krát za týždeň doplnené zistené výsledky meraní 'traceroute'. Významný bude najmä stĺpec t_status', ktorý obsahuje exit status merania traceroute. Opäť, ako pri 'ping' bude použitý pri určovaní množiny spoľahlivých IP adries.

Podrobnejšie sú všetky analyzované tabuľky popísané v príslušných textových dokumentoch v repozitári projektu.

4 Analýza dát

4.1 Analýza IP adries

Všetky zdrojové kódy dotazov prislúchajúcich k nasledovnej časti sa nachádzajú v repozitári projektu.

Hlavným cieľom tohoto projektu bolo najmä pripraviť podklady pre ďalšiu analýzu dát. Našim hlavným zámerom bolo teda vybrať vyššie spomínanú množinu 'spoľahlivých' IP adries. Z tabuľky 'hosts' sa nám podarilo určiť takúto množinu na základe nasledujúcich kritérií:

1) Neplatné IP adresy: Niektoré IPv4 adresy sú principiálne neplatné, pretože sú vyhradené pre špecifické účely (napr. súkromné podsiete). Takéto adresy zjavne nemá zmysel uvažovať, keďže by sa vo verejnom internete nemali vyskytovať a pravdepodobne došlo pri danom meraní ku chybe. Takéto IP adresy boli teda z množiny všetkých adries odstránené a v ďalšej analýze nebudú uvažované. Ako zdroj takýchto vyhradených IP adries, bola použitá oficiálna stránka organizácie IANA (Internet Assigned Numbers Authority), ktorá okrem iného dohliada na globálnu alokáciu IP adries. Podrobný zoznam všetkých alokovaných a vyhradených rozsahov IP adries je dostupný na:
https://www.iana.org/assignments/ipv4-address-space/ipv4-address-space.xhtml

2) Výsledky meraní 'ping' a 'traceroute': Ďalším kritériom bolo, aby k danej adrese existoval aspoň jeden platný záznam v tabuľke 'ping' alebo 'topology'. Za platný záznam pre IP adresu v tabuľke 'ping' považujeme taký, počas ktorého merania sa aspoň 1 packet podarilo úspešne doručiť (teda vo vyššie spomínanom stĺpci 'ping_ploss' je hodnota nezáporná a ostro menšia ako 100%). Za platný záznam pre IP adresu v tabuľke 'topology' zase považujeme taký, ktorého hodnota exit statusu je rôzna od 'E' (error).

IP adresy, ktoré spĺňajú vyššie spomenuté 2 kritéria boli teda označené za relevantné a táto množina teda tvorí potenciálny súbor IP adries pre ďalšiu analýzu.

4.2 Analýza časov záznamov

Ďalej sme sa pri analýze dát pozreli na časy, v ktorých pribúdali záznamy. Každá z tabuliek 'hosts', 'ping' a 'topology' mala okrem dát pre každý záznam aj čas, kedy bol tento záznam pribudol do databázy. Cieľom tejto analýzy bolo zistiť dni, kedy nepribudol žiaden záznam. Princíp dotazu je nasledovný: Najskôr vygenerujeme všetky diskrétne časy (napr. dni), medzi minimálnym a maximálnym uvažovaným časom a následne z tejto množiny odfiltrujeme tie, ku ktorým prislúcha niektorý záznam z jednotlivých tabuliek (každú analyzujeme zvlášť). Takto nám zostanú diskrétne časy, v ktorých nepribudol žiaden záznam.

5 Technické problémy a riešenia

Počas formulácie dotazov sme narazili na niektoré problémy a komplikácie, ktoré bolo treba nejakým spôsobom vyriešiť, prípadne obísť. V tejto časti uvedieme ich zhrnutie.

5.1 Pomalý výkon databázového stroja

Pomerne komplikujúcou okolnosťou bol výkon stroja, na ktorom bežala databáza. Analyzované tabuľky boli obrovské (najmä 'ping' - rádovo 100 miliónov riadkov), preto zložitejšie dotazy najmä tie, ktoré počítali logický rozdiel množín (napr. 'NOT EXISTS'), na stroji nezbehli v únosnom čase. Jediným riešením pre takýto problém bolo zredukovať skúmanú množinu dát na rádovo menšiu (napr. vytvorením dočasnej tabuľky pomocou 'CREATE TEMPORARY TABLE') a klásť dotazy nad týmito menšími tabuľkami.

5.2 Staršia verzia PostgreSQL bežiaca na serveri

Na databázovom serveri beží staršia verzia PostgreSQL 8.1.9, ktorá spôsobila menšie komplikácie.

5.2.1 Nepodporovaná SQL klauzula 'WITH'

Štandardná konštrukcia zložitejších dotazov v jazyku SQL sa väčšinou nezaobíde bez použitia dočasných pomocných tabuliek. Na tento účel slúži najčastejšie práve spomínaná klauzula 'WITH', ktorá umožňuje vytvoriť pomocnú (abstraktnú) tabuľku s lokálnou platnosťou iba v aktuálnom dotaze. Táto konštrukcia je preto veľmi užitočná a preferovaná v prípade priameho použitia len v nasledovnom (aktuálnom) dotaze. Táto konštrukcia sa však dostala do štandardu SQL až neskôr a na mnohých starších implementáciach preto nie je podporovaná ako napríklad na databáze bežiacej na našom serveri. Možným riešením je použitie konštrukcie 'CREATE TEMPORARY TABLE', ktorá vytvorí dočasnú tabuľku platnú však po dobu celej transakcie. Pre účely tohoto projektu toto

nebol až taký problém, avšak pri zložitejších dotazov môže spôsobiť menšie problémy, napr. kolíziu identifikátorov tabuliek.

5.2.2 Funkcia generate_series()

Pri analýze časov záznamov sme potrebovali generovať všetky diskrétne časy z nejakého intervalu. Pre tento účel je v PostgreSQL štandardne implementovaná funkcia generate_series(), ktorá vie vygenerovať tabuľku s usporiadanými hodnotami z vybraného intervalu pre rôzne dátové typy (zároveň umožňuje zvoliť skok medzi vygenerovanými diskrétnymi hodnotami). Staršia verzia, ktorá beží na serveri však podporuje túto funkciu len pre dátový typ 'Integer'. Riešenie, ktoré sme použili je nasledovné: Pre celočíselnú reprezentáciu času vyhovuje napr. UNIX timestamp. Pomocou generate_series() sme teda vygenerovali všetky diskrétne časy ako UNIX-ové timestampy a tie sme následne skonvertovali do klasického ľudského času, ktorý sme potom použili pre spomenuté filtrovanie vybraných tabuliek.

6 Záver

Podarilo sa nám vytvoriť základný podklad pre ďalšiu analýzu nad vybranými dátami. Zároveň sa nám podarilo nájsť možné riešenia pre niektoré vzniknuté problémy, z ktorých možno čerpať, prípadne sa nimi inšpirovať aj pri ďalšej práce s databázou. Význam tohoto projektu bola teda najmä príprava na ďalšiu analýzu databázy. Predmetom ďalšej analýzy by mohla byť napríklad štatistika jednotlivých výsledkov meraní 'ping' a 'traceroute'. Ďalej je možné rozšíriť analýzu časov záznamov a to napríklad o podrobnú analýzu vývoja pribúdania záznamov do tabuliek. V neposlednom rade by mohlo byť užitočné nejakým spôsobom automatizovať pripájanie sa k serveru a spúšťanie dotazov (napr. Shell script a pod.).
Na záver ešte uvádzame odkaz na repozitár projektu:
https://github.com/wartyduke375626/DBanalysis

Príloha A: Dokumenty popisujúce analyzované tabuľky

```
Table hosts
===========
contains IP addresses that were used during respective measurements
- Primary key: ip_addr
- Number of rows: 77 225

Column ip_addr
--------------
the IP address of the host
- Type: cidr
- Nullable: NO
- Default value: not provided

Column rank_code
----------------
an integer value that references the type of the host specified in table h_types
- Type: integer
- Nullable: YES
- Default value: 0

Column enter_date
-----------------
the date when this entry was registered
- Type: timestamp without time zone
- Nullable: YES
- Default value: now()

Column source
-------------
the source this host was registered from
- Type: text
- Nullable: YES
- Default value: not provided

Column comment
--------------
optional comment on this entry
- Type: text
- Nullable: YES
- Default value: not provided


Output from psql \d command:
============================

                          Table "public.hosts"
   Column   |            Type             | Collation | Nullable | Default
------------+-----------------------------+-----------+----------+---------
 ip_addr    | cidr                        |           | not null |
 rank_code  | integer                     |           |          | 0
 enter_date | timestamp without time zone |           |          | now()
 source     | text                        |           |          |
 comment    | text                        |           |          |
Indexes:
    "hosts_pkey" PRIMARY KEY, btree (ip_addr)
Referenced by:
    TABLE "dns" CONSTRAINT "dns_dns_server1_fkey" FOREIGN KEY (dns_server1) REFERENCES
        hosts(ip_addr)
```

```
    TABLE "dns" CONSTRAINT "dns_ip_addr_fkey" FOREIGN KEY (ip_addr) REFERENCES hosts(
        ip_addr) ON DELETE CASCADE
    TABLE "ping" CONSTRAINT "ping_ip_addr_fkey" FOREIGN KEY (ip_addr) REFERENCES hosts(
        ip_addr) ON DELETE CASCADE
    TABLE "source" CONSTRAINT "source_ip_addr_fkey" FOREIGN KEY (ip_addr) REFERENCES
        hosts(ip_addr) ON DELETE CASCADE
    TABLE "topology" CONSTRAINT "topology_ip_addr_fkey" FOREIGN KEY (ip_addr) REFERENCES
         hosts(ip_addr) ON DELETE CASCADE


Sample (random 10 rows from table):
===================================

      ip_addr       | rank_code |         enter_date         |       source        |
          comment
--------------------+-----------+----------------------------+---------------------+---------

 82.208.151.245/32  |         0 | 2007-05-28 19:00:26.256155 | lepos.fiit.stuba.sk |
 128.138.238.174/32 |         0 | 2007-05-28 19:00:26.362139 | lepos.fiit.stuba.sk |
 222.83.251.105/32  |         0 | 2007-05-28 19:00:26.726358 | lepos.fiit.stuba.sk |
 81.170.68.105/32   |         0 | 2007-05-28 19:00:26.736267 | lepos.fiit.stuba.sk |
 81.170.68.108/32   |         0 | 2007-05-28 19:00:26.747066 | lepos.fiit.stuba.sk |
 81.170.68.115/32   |         0 | 2007-05-28 19:00:26.763333 | lepos.fiit.stuba.sk |
 81.170.68.118/32   |         0 | 2007-05-28 19:00:26.771586 | lepos.fiit.stuba.sk |
 81.170.68.123/32   |         0 | 2007-05-28 19:00:26.785091 | lepos.fiit.stuba.sk |
 81.170.68.129/32   |         0 | 2007-05-28 19:00:26.801024 | lepos.fiit.stuba.sk |
 81.170.68.161/32   |         0 | 2007-05-28 19:00:26.881526 | lepos.fiit.stuba.sk |
(10 rows)
```

```
Table h_types
=============
contains descriptions of defined host types
- Primary key: rank_code
- Number of rows: 4

Column rank_code
----------------
the code of the host type
- Type: integer
- Nullable: NO
- Default value: not provided

Column rank
-----------
name of the host type
- Type: text
- Nullable: YES
- Default value: not provided

Column rank_description
-----------------------
description of the host type
- Type: text
- Nullable: YES
- Default value: not provided


Output from psql \d command:
============================

                  Table "public.h_types"
      Column      |  Type   | Collation | Nullable | Default
------------------+---------+-----------+----------+---------
 rank_code        | integer |           | not null |
 rank             | text    |           |          |
 rank_description | text    |           |          |
Indexes:
    "h_types_pkey" PRIMARY KEY, btree (rank_code)


Content (complete):
===================

 rank_code |   rank   |                    rank_description
-----------+----------+-----------------------------------------------------------
         0 | HOST     | Ordinary host, details unspecified
         1 | SUSPECT  | Suspicious host, doing something strange
         2 | SCANNER  | Scanner host, sends various requests to us
         3 | ATTACKER | Attacker host, tries to find and exploit some weakness
(4 rows)
```

```
Table ping
==========
contains records from ping measurements
- Primary key: (ip_addr, ping_date)
- Foreign keys:
        (ip_addr) references hosts(ip_addr)
- Number of rows: 84 869 803

Column ip_addr
--------------
the IP address of the host that was pinged
- Type: cidr
- Nullable: NO
- Default value: not provided

Column ping_rttmin
------------------
minimum ping time
- Type: real
- Nullable: YES
- Default value: not provided

Column ping_rttmax
------------------
maximum ping time
- Type: real
- Nullable: YES
- Default value: not provided

Column ping_rttavg
------------------
average ping time
- Type: real
- Nullable: YES
- Default value: not provided

Column ping_rttmdev
-------------------
mean deviation of ping time
- Type: real
- Nullable: YES
- Default value: not provided

Column ping_ploss
-----------------
the percentage of lost ping packets or an error code if negative value
- Type: integer
- Nullable: YES
- Default value: not provided

Column ping_date
----------------
the date when this entry was registered
- Type: timestamp without time zone
- Nullable: NO
- Default value: now()


Output from psql \d command:
============================
```

```
                          Table "public.ping"
    Column    |            Type             | Collation | Nullable | Default
--------------+-----------------------------+-----------+----------+---------
 ip_addr      | cidr                        |           | not null |
 ping_rttmin  | real                        |           |          |
 ping_rttmax  | real                        |           |          |
 ping_rttavg  | real                        |           |          |
 ping_rttmdev | real                        |           |          |
 ping_ploss   | integer                     |           |          |
 ping_date    | timestamp without time zone |           | not null | now()
Indexes:
    "ping_pkey" PRIMARY KEY, btree (ip_addr, ping_date)
Foreign-key constraints:
    "ping_ip_addr_fkey" FOREIGN KEY (ip_addr) REFERENCES hosts(ip_addr) ON DELETE
        CASCADE


Sample (random 10 rows from table):
===================================

      ip_addr       | ping_rttmin | ping_rttmax | ping_rttavg | ping_rttmdev |
          ping_ploss |         ping_date
--------------------+-------------+-------------+-------------+--------------+------------+------------
 213.201.221.162/32 |      21.855 |      22.505 |      22.176 |        0.246 |
     0 | 2009-05-11 06:09:53.746052
 218.186.12.10/32   |             |             |             |              |
     100 | 2009-05-11 06:13:37.237716
 221.3.131.66/32    |     402.034 |     409.939 |     406.506 |        3.382 |
     0 | 2009-05-11 06:14:29.186345
 217.153.56.221/32  |             |             |             |              |
     100 | 2009-05-11 06:12:22.969227
 212.27.32.132/32   |      29.973 |      31.019 |      30.384 |        0.325 |
     0 | 2009-05-11 06:06:41.589475
 213.151.206.10/32  |             |             |             |              |
     -1 | 2009-05-11 06:08:44.01905
 213.205.36.90/32   |             |             |             |              |
     100 | 2009-05-11 06:10:11.440789
 217.67.18.105/32   |             |             |             |              |
     100 | 2009-05-11 06:11:28.109716
 217.112.87.148/32  |      28.857 |      32.442 |      29.353 |        1.044 |
     0 | 2009-05-11 06:11:31.710011
 218.56.61.114/32   |     408.528 |     416.276 |     412.281 |        3.379 |
     0 | 2009-05-11 06:12:40.986137
(10 rows)
```

```
Table topology
==============
contains records from traceroute measurements
- Primary key: (ip_addr, t_date)
- Foreign keys:
        (ip_addr) references hosts(ip_addr)
- Number of rows: 10 249 718

Column ip_addr
--------------
the IP address of the host that was traced
- Type: cidr
- Nullable: NO
- Default value: not provided

Column t_roundtrip
------------------
the latency for each hop from router to router
- Type: numeric[]
- Nullable: YES
- Default value: not provided

Column t_hops
-------------
the number of traceroute hops
- Type: integer
- Nullable: YES
- Default value: not provided

Column t_status
---------------
traceroute return status
- Type: character(1)
- Nullable: YES
- Default value: not provided

Column t_route
--------------
the IP addresses of routers the packet has gone through
- Type: cidr[]
- Nullable: YES
- Default value: not provided

Column t_flags
--------------
flags set in between every two routers of traceroute
- Type: text[]
- Nullable: YES
- Default value: not provided

Column t_comment
----------------
optional coment on this entry
- Type: text
- Nullable: YES
- Default value: not provided

Column t_date
-------------
the date when this entry was registered
- Type: timestamp without time zone
```

```
- Nullable: NO
- Default value: now()

Column t_yearweek
-----------------
???
- Type: integer
- Nullable: NO
- Default value: not provided


Output from psql \d command:
============================

                        Table "public.topology"
   Column    |            Type             | Collation | Nullable | Default
-------------+-----------------------------+-----------+----------+---------
 ip_addr     | cidr                        |           | not null |
 t_roundtrip | numeric[]                   |           |          |
 t_hops      | integer                     |           |          |
 t_status    | character(1)                |           |          |
 t_route     | cidr[]                      |           |          |
 t_flags     | text[]                      |           |          |
 t_comment   | text                        |           |          |
 t_date      | timestamp without time zone |           | not null | now()
 t_yearweek  | integer                     |           |          |
Indexes:
    "topology_pkey" PRIMARY KEY, btree (ip_addr, t_date)
Foreign-key constraints:
    "topology_ip_addr_fkey" FOREIGN KEY (ip_addr) REFERENCES hosts(ip_addr) ON DELETE
        CASCADE


Sample (random 10 rows from table):
===================================

      ip_addr      |     t_roundtrip     | t_hops | t_status |
         t_route                        |  t_flags   | t_comment |           t_date
                  | t_yearweek
-------------------+---------------------+--------+----------+-----------------------------------------

 0.0.161.69/32     |                     |     -1 | E        |
                                                    |           |           |
     2009-06-16 03:23:47.249628 |
 68.87.215.162/32  |                     |     -1 | E        |
                                                    |           |           |
     2012-01-08 13:05:40.106485 |
 67.198.201.2/32   |                     |     -1 | E        |
                                                    |           |           |
     2012-01-08 12:41:19.254521 |
 70.50.96.70/32    |                     |     -1 | E        |
                                                    |           |           |
     2012-01-08 13:39:56.51494  |
 221.215.97.162/32 | {0.406,0.655,0.608} |      3 | I        |
     {147.175.98.1/32,147.175.159.1/32,147.175.159.1/32} | {"","",!H} |           |
     2010-07-31 05:22:08.807007 |
 67.195.112.50/32  |                     |     -1 | E        |
                                                    |           |           |
     2012-01-08 12:39:53.716137 |
 70.174.1.136/32   |                     |     -1 | E        |
                                                    |           |           |
```

```
    2012-01-08 13:47:02.247543 |
 147.175.99.38/32  | {0.424,1.766}       |      2 | C        |
    {147.175.98.1/32,147.175.99.38/32}                  | {"","}    |           |
    2009-06-19 04:34:50.182701 |
 147.175.97.1/32   | {0.767}             |      1 | C        | {147.175.97.1/32}
                                       | {""}       |           | 2009-06-19
    04:34:21.349174 |
 67.228.177.191/32 |                     |     -1 | E        |
                                                   |            |           |
    2012-01-08 12:45:36.651379 |
(10 rows)
```

```
Table source
============
contains information about IP addresses that were used as hosts during respective
    measurements
- Primary key: (ip_addr, source_code)
- Foreign keys:
        (ip_addr) references hosts(ip_addr)
- Number of rows: 77 323

Column ip_addr
--------------
the IP address of the host
- Type: cidr
- Nullable: NO
- Default value: not provided

Column source_code
------------------
an integer value that references the type of the source specified in table source_types
- Type: integer
- Nullable: NO
- Default value: not provided

Column source_ref
-----------------
the number of entries registered by the source
- Type: integer
- Nullable: YES
- Default value: 1

Column source_date
------------------
the date when this entry was registered
- Type: timestamp without time zone
- Nullable: YES
- Default value: now()


Output from psql \d command:
============================

                         Table "public.source"
   Column    |            Type             | Collation | Nullable | Default
-------------+-----------------------------+-----------+----------+---------
 ip_addr     | cidr                        |           | not null |
 source_code | integer                     |           | not null |
 source_ref  | integer                     |           |          | 1
 source_date | timestamp without time zone |           |          | now()
Indexes:
    "source_pkey" PRIMARY KEY, btree (ip_addr, source_code)
Foreign-key constraints:
    "source_ip_addr_fkey" FOREIGN KEY (ip_addr) REFERENCES hosts(ip_addr) ON DELETE
        CASCADE


Sample (random 10 rows from table):
===================================

      ip_addr       | source_code | source_ref |        source_date
--------------------+-------------+------------+----------------------------
 69.59.137.140/32   |           0 |          2 | 2009-05-17 06:14:40.569369
```

```
 83.141.21.166/32   |           0 |           1 | 2009-05-11 06:15:56.150909
 74.6.17.189/32     |           0 |          18 | 2008-11-06 05:42:57.404312
 91.127.219.176/32  |           0 |           3 | 2011-10-23 07:28:43.473966
 217.175.237.230/32 |           0 |           1 | 2007-05-28 19:00:26.281396
 78.106.207.69/32   |           0 |          23 | 2008-04-09 04:15:35.34683
 78.98.69.203/32    |           0 |          22 | 2009-05-07 06:15:45.224571
 82.99.30.57/32     |           0 |          94 | 2008-11-11 05:52:42.516294
 147.175.195.36/32  |           0 |          12 | 2010-01-17 04:15:28.432273
 93.97.170.15/32    |           0 |          14 | 2009-04-28 06:20:20.136141
(10 rows)
```

```
Table source_types
==================
contains descriptions of defined source types
- Primary key: source_code
- Number of rows: 3

Column source_code
------------------
the code of the source type
- Type: integer
- Nullable: NO
- Default value: not provided

Column source_type
------------------
name of the source_type
- Type: text
- Nullable: YES
- Default value: not provided

Column source_description
-------------------------
description of the source type
- Type: text
- Nullable: YES
- Default value: not provided


Output from psql \d command:
============================

                 Table "public.source_types"
       Column       |  Type   | Collation | Nullable | Default
--------------------+---------+-----------+----------+---------
 source_code        | integer |           | not null |
 source_type        | text    |           |          |
 source_description | text    |           |          |
Indexes:
    "source_types_pkey" PRIMARY KEY, btree (source_code)


Content (complete):
===================

 source_code | source_type |            source_description
-------------+-------------+-----------------------------------------
           0 | LOG         | The host address apeared in our log files
           1 | DNS         | The host is DNS server
           2 | TRACEROUTE  | The host is a router
(3 rows)
```

Príloha B: SQL dotazy použité pri analýze databázy

```
/* This script contains queries used in analysis of table structure. */

/* Count number of rows: */
SELECT count(*) FROM hosts;
SELECT count(*) FROM h_types;
SELECT count(*) FROM source;
SELECT count(*) FROM source_types;
SELECT count(*) FROM ping;
SELECT count(*) FROM topology;

/* Random sample of 10 rows - randomly filters 1% of rows and selects the first 10: */
SELECT * FROM hosts WHERE random() <= 0.01 LIMIT 10;
SELECT * FROM source WHERE random() <= 0.01 LIMIT 10;
SELECT * FROM ping WHERE random() <= 0.01 LIMIT 10;
SELECT * FROM topology WHERE t_hops <= 3 AND random() <= 0.01 LIMIT 10;

/* Select all rows: */
SELECT * FROM h_types ORDER BY rank_code;
SELECT * FROM source_types ORDER BY source_code;
```

```
/* Select IP addresses of all hosts except those contained in reserved subnets according
     to: https://www.iana.org/assignments/ipv4-address-space/ipv4-address-space.xhtml */
SELECT h.ip_addr
FROM hosts h
WHERE NOT (
        h.ip_addr <<= '0.0.0.0/8' OR h.ip_addr <<= '0.0.0.0/32' OR h.ip_addr <<= '
            10.0.0.0/8'
        OR h.ip_addr <<= '100.64.0.0/10' OR h.ip_addr <<= '127.0.0.0/8' OR h.ip_addr <<=
             '169.254.0.0/16'
        OR h.ip_addr <<= '172.16.0.0/12' OR h.ip_addr <<= '192.0.2.0/24' OR h.ip_addr
            <<= '192.88.99.0/24'
        OR h.ip_addr <<= '192.88.99.2/32' OR h.ip_addr <<= '192.168.0.0/16' OR h.ip_addr
             <<= '192.0.0.0/24'
        OR h.ip_addr <<= '198.18.0.0/15' OR h.ip_addr <<= '198.51.100.0/24' OR h.ip_addr
             <<= '203.0.113.0/24'
        OR h.ip_addr <<= '255.255.255.255/32'
);

/* Complementary query: */
SELECT h.ip_addr
FROM hosts h
WHERE h.ip_addr <<= '0.0.0.0/8' OR h.ip_addr <<= '0.0.0.0/32' OR h.ip_addr <<= '
    10.0.0.0/8'
        OR h.ip_addr <<= '100.64.0.0/10' OR h.ip_addr <<= '127.0.0.0/8' OR h.ip_addr <<=
             '169.254.0.0/16'
        OR h.ip_addr <<= '172.16.0.0/12' OR h.ip_addr <<= '192.0.2.0/24' OR h.ip_addr
            <<= '192.88.99.0/24'
        OR h.ip_addr <<= '192.88.99.2/32' OR h.ip_addr <<= '192.168.0.0/16' OR h.ip_addr
             <<= '192.0.0.0/24'
        OR h.ip_addr <<= '198.18.0.0/15' OR h.ip_addr <<= '198.51.100.0/24' OR h.ip_addr
             <<= '203.0.113.0/24'
        OR h.ip_addr <<= '255.255.255.255/32';



/* Select IP addresses of all hosts with at least 1 successfull ping record in table
    ping or traceroute record in table topology: */
SELECT h.ip_addr
FROM hosts h
WHERE EXISTS (
        SELECT *
        FROM ping p
        WHERE h.ip_addr = p.ip_addr AND p.ping_ploss >= 0 AND p.ping_ploss < 100
) OR EXISTS (
        SELECT *
        FROM topology t
        WHERE h.ip_addr = t.ip_addr AND t.t_status <> 'E'
);

/* Complementary querry: */
SELECT h.ip_addr
FROM hosts h
WHERE NOT EXISTS (
        SELECT *
        FROM ping p
        WHERE h.ip_addr = p.ip_addr AND p.ping_ploss >= 0 AND p.ping_ploss < 100
) AND NOT EXISTS (
        SELECT *
        FROM topology t
        WHERE h.ip_addr = t.ip_addr AND t.t_status <> 'E'
);
```

```
/* Conjunctio of both conditions: */
SELECT h.ip_addr
FROM hosts h
WHERE NOT (
        h.ip_addr <<= '0.0.0.0/8' OR h.ip_addr <<= '0.0.0.0/32' OR h.ip_addr <<= '
            10.0.0.0/8'
        OR h.ip_addr <<= '100.64.0.0/10' OR h.ip_addr <<= '127.0.0.0/8' OR h.ip_addr <<=
             '169.254.0.0/16'
        OR h.ip_addr <<= '172.16.0.0/12' OR h.ip_addr <<= '192.0.2.0/24' OR h.ip_addr
            <<= '192.88.99.0/24'
        OR h.ip_addr <<= '192.88.99.2/32' OR h.ip_addr <<= '192.168.0.0/16' OR h.ip_addr
             <<= '192.0.0.0/24'
        OR h.ip_addr <<= '198.18.0.0/15' OR h.ip_addr <<= '198.51.100.0/24' OR h.ip_addr
             <<= '203.0.113.0/24'
        OR h.ip_addr <<= '255.255.255.255/32'
) AND (
        EXISTS (
                SELECT *
                FROM ping p
                WHERE h.ip_addr = p.ip_addr AND p.ping_ploss >= 0 AND p.ping_ploss < 100
        ) OR EXISTS (
                SELECT *
                FROM topology t
                WHERE h.ip_addr = t.ip_addr AND t.t_status <> 'E'
        )
);

/* Complementary query: */
SELECT h.ip_addr
FROM hosts h
WHERE h.ip_addr <<= '0.0.0.0/8' OR h.ip_addr <<= '0.0.0.0/32' OR h.ip_addr <<= '
    10.0.0.0/8'
        OR h.ip_addr <<= '100.64.0.0/10' OR h.ip_addr <<= '127.0.0.0/8' OR h.ip_addr <<=
             '169.254.0.0/16'
        OR h.ip_addr <<= '172.16.0.0/12' OR h.ip_addr <<= '192.0.2.0/24' OR h.ip_addr
            <<= '192.88.99.0/24'
        OR h.ip_addr <<= '192.88.99.2/32' OR h.ip_addr <<= '192.168.0.0/16' OR h.ip_addr
             <<= '192.0.0.0/24'
        OR h.ip_addr <<= '198.18.0.0/15' OR h.ip_addr <<= '198.51.100.0/24' OR h.ip_addr
             <<= '203.0.113.0/24'
        OR h.ip_addr <<= '255.255.255.255/32'
UNION
SELECT h.ip_addr
FROM hosts h
WHERE NOT EXISTS (
        SELECT *
        FROM ping p
        WHERE h.ip_addr = p.ip_addr AND p.ping_ploss >= 0 AND p.ping_ploss < 100
) AND NOT EXISTS (
        SELECT *
        FROM topology t
        WHERE h.ip_addr = t.ip_addr AND t.t_status <> 'E'
);



/* % of valid hosts: */
CREATE TEMPORARY TABLE valid_hosts AS (
```

```
        SELECT h.ip_addr
        FROM hosts h
        WHERE NOT (
                h.ip_addr <<= '0.0.0.0/8' OR h.ip_addr <<= '0.0.0.0/32' OR h.ip_addr <<=
                     '10.0.0.0/8'
                OR h.ip_addr <<= '100.64.0.0/10' OR h.ip_addr <<= '127.0.0.0/8' OR h.
                    ip_addr <<= '169.254.0.0/16'
                OR h.ip_addr <<= '172.16.0.0/12' OR h.ip_addr <<= '192.0.2.0/24' OR h.
                    ip_addr <<= '192.88.99.0/24'
                OR h.ip_addr <<= '192.88.99.2/32' OR h.ip_addr <<= '192.168.0.0/16' OR h
                    .ip_addr <<= '192.0.0.0/24'
                OR h.ip_addr <<= '198.18.0.0/15' OR h.ip_addr <<= '198.51.100.0/24' OR h
                    .ip_addr <<= '203.0.113.0/24'
                OR h.ip_addr <<= '255.255.255.255/32'
        ) AND (
                EXISTS (
                        SELECT *
                        FROM ping p
                        WHERE h.ip_addr = p.ip_addr AND p.ping_ploss >= 0 AND p.
                            ping_ploss < 100
                ) OR EXISTS (
                        SELECT *
                        FROM topology t
                        WHERE h.ip_addr = t.ip_addr AND t.t_status <> 'E'
                )
        )
);
SELECT (SELECT count(*) FROM valid_hosts)::float / (SELECT count(*) FROM hosts) * 100 AS
     p_of_valid_hosts;



/* IP addresses of hosts that responded to all ping requests: */
SELECT h.ip_addr
FROM hosts h
WHERE NOT EXISTS (
        SELECT *
        FROM ping p
        WHERE h.ip_addr = p.ip_addr AND (p.ping_ploss < 0 OR p.ping_ploss == 100)
);

/* Complementary query:  */
SELECT h.ip_addr
FROM hosts h
WHERE EXISTS (
        SELECT *
        FROM ping p
        WHERE h.ip_addr = p.ip_addr AND (p.ping_ploss < 0 OR p.ping_ploss == 100)
);
```

```
/* This script contains queries used to determine time intervals in which no entry was
    registered. Since th eserver runs PostgreSQL version 8.1.9 which does not support
    function generate_series(timestamp), generate_series(integer) is used instead (
    interpreted as UNIX timestamp) and then converted to timestamp. */

/* Dates of first entry for tables hosts, ping and topology: */
SELECT min(enter_date) FROM hosts; /* 2007-05-28 19:00:23 = 1180371623 (UNIX) */
SELECT min(ping_date) FROM ping; /* 2001-01-01 05:27:23 = 978323243 (UNIX) */
SELECT min(t_date) FROM topology; /* 2009-06-16 03:23:46 = 1245115426 (UNIX) */

/* Dates of last entry for tables hosts, ping and topology: */
SELECT max(enter_date) FROM hosts; /* 2013-07-01 03:57:53 = 1372651073 (UNIX) */
SELECT max(ping_date) FROM ping; /* 2013-07-01 04:51:03 = 1372647063 (UNIX) */
SELECT max(t_date) FROM topology; /* 2013-07-01 08:40:02 = 1372660802 (UNIX) */


/* Days when no host was recorded. */
CREATE TEMPORARY TABLE all_days_hosts AS (
    SELECT to_timestamp(generate_series)::date AS day
    FROM generate_series(1180371623, 1372651073, 86400) /* 86400 = day length in seconds
        */
);

/* Note: generate_series() generates timestamps inbetween min and max time found out in
    previous queries. The generation step is of 86400 seconds which is the exact day
    length in seconds.
  This means it will generate all days contained within the [min, max] interval,
      inclusive. This attribute is then cast to date type which will extract the day
      part of the timestamp. */

SELECT a.day
FROM all_days_hosts a
WHERE NOT EXISTS (
    SELECT *
    FROM hosts h
    WHERE h.enter_date::date = a.day
);

/* Complementary query. */
SELECT a.day
FROM all_days_hosts a
WHERE EXISTS (
    SELECT *
    FROM hosts h
    WHERE h.enter_date::date = a.day
);



/* Days when no ping entry was registered. */
CREATE TEMPORARY TABLE all_days_ping AS (
    SELECT to_timestamp(generate_series)::date AS day
    FROM generate_series(978323243, 1372647063, 86400) /* 86400 = day length in seconds
        */
);

SELECT a.day
FROM all_days_ping a
WHERE NOT EXISTS (
    SELECT *
    FROM ping p
```

```
    WHERE p.ping_date::date = a.day
);

/* Complementary query. */
SELECT a.day
FROM all_days_ping a
WHERE EXISTS (
    SELECT *
    FROM ping p
    WHERE p.ping_date::date = a.day
);


/* Days when no topology entry was registered. */
CREATE TEMPORARY TABLE all_days_t AS (
    SELECT to_timestamp(generate_series)::date AS day
    FROM generate_series(1245115426, 1372660802, 86400) /* 86400 = day length in seconds
        */
);

SELECT a.day
FROM all_days_t a
WHERE NOT EXISTS (
    SELECT *
    FROM topology t
    WHERE t.t_date::date = a.day
);

/* Complementary query. */
SELECT a.day
FROM all_days_t a
WHERE EXISTS (
    SELECT *
    FROM topology t
    WHERE t.t_date::date = a.day
);
```