Instytut Sterowania i Systemów Informatycznych
Uniwersytet Zielonogórski

Bazy Danych

Ćwiczenie 14: Tworzenie i wykorzystanie indeksów

opracował: dr hab. inż. Artur Gramacki (a.gramacki@issi.uz.zgora.pl)

1. Uwagi wstępne

Na niebiesko zaznaczono polecenia (tu oraz w dalszej części instrukcji) wpisywane przez studenta w konsoli tekstowej. Symbol `shell>` zawsze będzie oznaczać znak zachęty w konsoli tekstowej systemu Windows a symbol `mysql>` znak zachęty w konsoli tekstowej MySQL-a.

Niezwykle ważnym obszarem pracy administratora serwera bazodanowego (ale też w pewnym zakresie użytkownika/developera) jest jego optymalizacja i strojenie (ang. *optimization and tuning*). Zagadnienia z tym związane są złożone i trudne do praktycznego opanowania. Aby serwer bazodanowy działał wydajnie w środowisku produkcyjnym, zwykle nie można poprzestać na jego ustawieniach domyślnych i pewne elementy należy dopasować do aktualnych wymagań użytkowników oraz posiadanych zasobów sprzętowych. Mamy do dyspozycji kilkaset parametrów konfiguracyjnych, których wartości można (w większości) zmieniać. Po wydaniu polecenia:

```
mysql> show variables;
+--------------------------------------------------------+---------------------------------
| Variable_name                                          | Value
+--------------------------------------------------------+---------------------------------
| auto_increment_increment                               | 1
| auto_increment_offset                                  | 1
| autocommit                                             | ON
| automatic_sp_privileges                                | ON
| avoid_temporal_upgrade                                 | OFF
| back_log                                               | 80
| basedir                                                | /xampp_5.6.12_Portable/mysql
| big_tables                                             | OFF
| bind_address                                           | *
| binlog_cache_size                                      | 32768
| binlog_checksum                                        | CRC32

...

| updatable_views_with_limit                             | YES
| version                                                | 5.6.26
| version_comment                                        | MySQL Community Server (GPL)
| version_compile_machine                                | x86
| version_compile_os                                     | Win32
| wait_timeout                                           | 28800
| warning_count                                          | 1
+--------------------------------------------------------+---------------------------------
453 rows in set, 1 warning (0.01 sec)
```

można łatwo przekonać się, że praktyczne opanowanie tych parametrów nie jest na pewno rzeczą łatwą. Ustalając odpowiednie wartości właściwych parametrów można sporo osiągnąć ... ale też i posuć. Lepiej więc bezrefleksyjnie nie zmieniać parametrów „jak popadnie".

W niniejszym ćwiczeniu skupiamy się wyłącznie na jednym aspekcie związanym z optymalizacją oraz strojeniem, a mianowicie na tworzeniu i wykorzystywaniu indeksów. Na przykładzie kilku eksperymentów pokażemy dobroczynny wpływ używania indeksów.

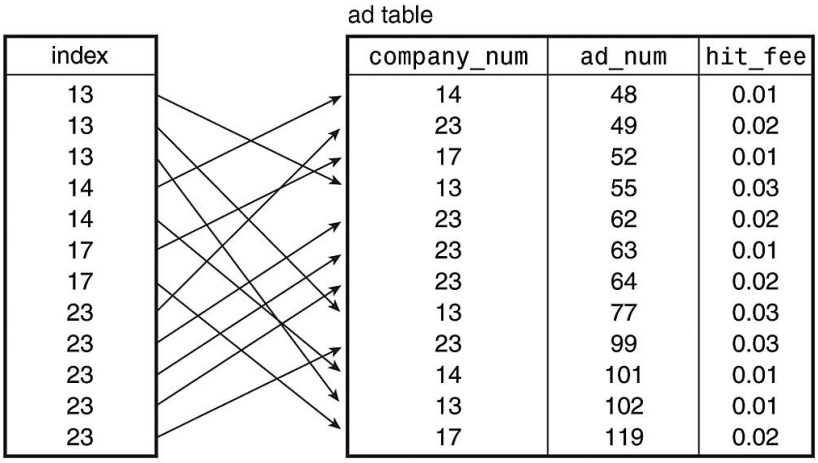
Trzeba też pamiętać, że nieroztropnie założone indeksy (np. ich zbyt duża liczba lub założone na niewłaściwych kolumnach) mogą negatywnie wpłynąć na działanie serwera, a w konsekwencji też na aplikacje. Po pierwsze dlatego, że obsługa indeksów (zwykle chodzi o ich uaktualnianie) zajmuje określony czas i dla dużej ilości wprowadzanych/modyfikowanych/kasowanych danych czas ten może być znaczący. Po drugie, indeksy zajmują określone zasoby dyskowe i może się zdarzyć, że gdy jest ich zbyt dużo zajmują one na dysku wielokrotnie więcej miejsca niż indeksowane dane! Trzeba więc starać się znaleźć złoty środek, co nie zawsze jest proste i oczywiste. Często właściwe rozwiązanie wymaga eksperymentowania.

W ćwiczeniu nie omawiamy wszystkich zagadnień związanych z indeksami. Pomijamy np. zagadnienia indeksów pełnotekstowych (FULLTEXT) oraz przestrzennych (SPATIAL).

2. O indeksach

Najlepszym sposobem zwiększenia wydajności operacji wyszukiwania w bazie danych (instrukcja `SELECT`) jest utworzenie indeksów na określonych kolumnach w tabelach. Indeksy można tworzyć na jednej lub kilku kolumnach, które zamierzamy przeszukiwać w zapytaniu.

Indeksy to struktury danych na dysku umożliwiające szybki dostęp do rekordów bez konieczności skanowania całej tabeli. Indeks można wyobrazić sobie jako **posortowaną kopię** indeksowanej kolumny, jak na rysunku poniżej:



Dzięki temu odnalezienie (używając klauzuli `WHERE` z różnymi operatorami, jak np: `=`, `>`, `≤`, `BETWEEN`, `IN` i inne) interesujących nas rekordów może odbyć się dużo szybciej, niż w przypadku sekwencyjnego przeszukiwania tabeli. Indeksy są najczęściej zorganizowane w postaci tzw. B-drzewa (*B-tree*) lub jakiegoś jego wariantu.

Najczęściej spotyka się **indeksy proste** zakładane na **pojedynczych kolumnach**. W uproszczeniu stworzenie indeksu powoduje przekopiowanie danych z indeksowanej kolumny, w sposób umożliwiający szybkie przeglądanie wierszy odpowiadających warunkowi podanemu w klauzuli `WHERE` i/lub `ORDER BY`. Współczesne silniki bazodanowe umożliwiają utworzenie dla pojedynczej tabeli wielu indeksów. Są jakieś limity maksymalne, ale zwykle są one na tyle duże, że niemal nigdy nawet się do nich nie zbliżamy.

MySQL (i wiele innych systemów baz danych) może tworzyć również **indeksy złożone** (tj. takie, które obejmują **kilka kolumn jednocześnie**). Jeśli podczas definiowania indeksu kolumny zostaną podane w odpowiedniej kolejności, pojedynczy indeks złożony może przyspieszać działanie kilku typów zapytań.

Załóżmy, że mamy tabelę *indeksy* składającą się z 3 kolumn *x*, *y*, *z*[1]:

```
CREATE TABLE indeksy (
  id  INT PRIMARY KEY AUTO_INCREMENT,
  x   INT,
  y   INT,
  z   INT
);
```

Zastanówmy się, jak MySQL pobiera dane, jeśli wybrana tabela **nie posiada indeksów** na polach, które uwzględnimy w zapytaniu (w klauzulach `WHERE`). Dla każdego zapytania silnik bazodanowy **skanuje od początku do końca wszystkie rekordy**, aż odnalezione zostaną te rekordy, które spełniające warunek podany w zapytaniu. Jeżeli przykładowo w tabeli znajduje się w sumie 1.000.000 rekordów, to każde zapytanie:

```
SELECT * FROM indeksy WHERE x = wart_1 AND y = wart_2 AND z = wart_3;
```

spowoduje przejrzenie wszystkich rekordów (1.000.000 odczytów) nawet w sytuacji, gdy tylko jeden wiersz zawiera interesującą nas wartość. Jeśli wiemy, ile rekordów może spełniać warunek zapytania, możemy próbować użyć klauzuli `LIMIT`, co może przyspieszyć wykonanie zapytania:

```
SELECT *
FROM indeksy WHERE x = wart_1 AND y = wart_2 AND z = wart_3 LIMIT 1;
```

Użycie tutaj klauzuli `LIMIT` spowoduje znalezienie pierwszego rekordu spełniającego warunek zapytania, bez konieczności sekwencyjnego przeszukiwania całej tabeli. Jeśli poszukiwany rekord jest na początku tabeli, przyrost wydajności może być zauważalny. Gdy poszukiwany rekord jest na końcu tabeli, przyrostu wydajności nie zauważymy wcale. Niestety, `LIMIT` to za mało, aby szybko pobierać dane. Prawdziwie wielkie przyśpieszenia możemy osiągnąć w zasadzie tylko za pomocą poprawnie zdefiniowanych indeksów.

Na naszej tabeli możemy przykładowo założyć następujące indeksy:

```
CREATE INDEX idx_x ON indeksy (x);
```

[1] W tabeli jest też klucz główny. Taka kolumna jest z definicji zaindeksowana.

```
CREATE INDEX idx_y ON indeksy (y);
CREATE INDEX idx_z ON indeksy (z);
CREATE INDEX idx_xy ON indeksy (x, y);
CREATE INDEX idx_xz ON indeksy (x, z);
CREATE INDEX idx_yz ON indeksy (y, z);
CREATE INDEX idx_xyz ON indeksy (x, y, z);
```

Usunąć można je za pomocą poleceń:

```
ALTER TABLE indeksy DROP INDEX idx_x;
ALTER TABLE indeksy DROP INDEX idx_y;
ALTER TABLE indeksy DROP INDEX idx_z;
ALTER TABLE indeksy DROP INDEX idx_xy;
ALTER TABLE indeksy DROP INDEX idx_xz;
ALTER TABLE indeksy DROP INDEX idx_yz;
ALTER TABLE indeksy DROP INDEX idx_xyz;
```

Jak MySQL wykorzystuje indeksy do wyszukiwania rekordów? W jednym zapytaniu, dla jednej tabeli użyty może być tylko jeden indeks. Jeśli baza może wybrać spośród kilku indeksów, postara się użyć ten o większej mocy, czyli taki, który szybciej odnajdzie poszukiwane przez nas dane (szczegóły techniczne, jak odbywa się wybieranie najbardziej odpowiedniego indeksu, pomijamy odsyłając zainteresowane osoby do literatury).

W przypadku indeksów złożonych, indeks może być użyty tylko wtedy, jeśli kolejność warunków w zapytaniu zgadza się z kolejnością pól w indeksie. Przykładowo, jeśli na kolumnach *x*, *y* i *z* założony jest indeks złożony, indeks ten będzie mógł być użyty w zapytaniach (między innymi):

```
SELECT * FROM indeksy WHERE x = war_1 AND y = war_2 AND z = war_3;
SELECT * FROM indeksy WHERE x = war_1 AND y = war_2;
SELECT * FROM indeksy WHERE x = wart_1;
```

ale w tych już nie:

```
SELECT * FROM indeksy WHERE y = wart_2;
SELECT * FROM indeksy WHERE z = wart_3;
SELECT * FROM indeksy WHERE y = wart_2 AND z = wart_3;
```

Generalnie, tworząc indeksy należy uwzględnić kolejność i sposób składania zapytań do danej tabeli i nie zawsze jest to oczywiste. Na szczęście, MySQL dostarcza narzędzie pozwalające sprawdzić jakie indeksy baza użyje (lub spróbuje użyć) w zapytaniu. Jest to polecenie `EXPLAIN`. Przykładowe użycie tego polecenia:

```
EXPLAIN SELECT * FROM indeksy WHERE x = wart_1 AND y = wart_2 AND z = wart_3;
```

3. Przykładowa sesja przy terminalu

Tworzymy tabelę testową i ładujemy do niej 1.000.000 przykładowych rekordów. Odpowiedni skrypt PHP (*dane.php*) generujący plik tworzący tabelę *indeksy* oraz dane w tej tabeli (*dane.sql*) umieszczono jako **załącznik do niniejszej instrukcji**. Na potrzeby testów użyjemy tabeli jak niżej:

```
CREATE TABLE indeksy (
  id  INT PRIMARY KEY AUTO_INCREMENT,
```

```
  x    INT,
  y    INT,
  z    INT
);
```

Tabela została wypełniona 1.000.000 rekordów w ten sposób, że dla każdej trójki wartości z przedziału 1-100 istnieje dokładnie jeden rekord. Pierwsze 10 oraz ostatnie 10 rekordów w poleceniu `INSERT` wygląda więc następująco:

```
INSERT INTO indeksy (x, y, z) VALUES
(1, 1, 1),
(1, 1, 2),
(1, 1, 3),
(1, 1, 4),
(1, 1, 5),
(1, 1, 6),
(1, 1, 7),
(1, 1, 8),
(1, 1, 9),
(1, 1, 10),
...
(100, 100, 91),
(100, 100, 92),
(100, 100, 93),
(100, 100, 94),
(100, 100, 95),
(100, 100, 96),
(100, 100, 97),
(100, 100, 98),
(100, 100, 99),
(100, 100, 100);
```

Procedura PHP *dane.php* utworzy skrypt (*dane.sql*), który trzeba wykonać.

```
mysql> source E:\xampp_5.6.12_Portable\htdocs\summit2\dane.sql
Query OK, 0 rows affected (0.31 sec)

Query OK, 0 rows affected (0.46 sec)

Query OK, 1000000 rows affected (1 min 41.25 sec)
Records: 1000000  Duplicates: 0  Warnings: 0

mysql> select count(*) from indeksy;
+----------+
| count(*) |
+----------+
|  1000000 |
+----------+
1 row in set (2.11 sec)
```

Za pomocą dyrektywy `set profiling = 1` włączamy funkcjonalność tzw. **profilowania** wydawanych poleceń.

```
mysql> set profiling = 1;
Query OK, 0 rows affected, 1 warning (0.00 sec)
```

Sprawdzamy, jakie mamy w tej chwili założone na tabeli indeksy. Jest tylko jeden indeks, pochodzący od klucza głównego (tabela wynikowa jest dość długa i poniżej pokazano tylko jej fragment).

```
mysql> SHOW INDEX FROM indeksy;
+---------+------------+----------+--------------+-------------+-----------+-------------+---
| Table   | Non_unique | Key_name | Seq_in_index | Column_name | Collation | Cardinality | Su
+---------+------------+----------+--------------+-------------+-----------+-------------+---
| indeksy |          0 | PRIMARY  |            1 | id          | A         |      998030 |
+---------+------------+----------+--------------+-------------+-----------+-------------+---
1 row in set (0.00 sec)
```

Wykonujemy przykładowe zapytanie. Czas wykonania jest dość spory (choć i tak przy takiej ilości danych całkiem dobry).

```
mysql> SELECT * FROM indeksy WHERE x=1 AND y=10 AND z=100;
+------+------+------+------+
| id   | x    | y    | z    |
+------+------+------+------+
| 1000 |    1 |   10 |  100 |
+------+------+------+------+
1 row in set (1.75 sec)
```

Patrzymy, jak wygląda tzw. plan wykonania zapytania SQL-owego. Widać, że żaden indeks nie został użyty (bo nie ma na razie żadnych indeksów, które mógłby pomóc w szybszym wykonaniu zapytania).

```
mysql> EXPLAIN SELECT * FROM indeksy WHERE x=1 AND y=10 AND z=100;
+----+-------------+---------+------+---------------+------+---------+------+--------+-------------+
| id | select_type | table   | type | possible_keys | key  | key_len | ref  | rows   | Extra       |
+----+-------------+---------+------+---------------+------+---------+------+--------+-------------+
|  1 | SIMPLE      | indeksy | ALL  | NULL          | NULL | NULL    | NULL | 998030 | Using where |
+----+-------------+---------+------+---------------+------+---------+------+--------+-------------+
1 row in set (0.00 sec)
```

Tworzymy więc trzy indeksy, po jednym na każdą kolumnę. Zauważmy, że indeksowanie tak dużej w sumie tabeli zajmuje całkiem sporo czasu. Na szczęście późniejsze uaktualnianie indeksu trwa już tylko ułamek tego czasu i jest często zupełnie niezauważalne przez użytkownika (*pytanie: kiedy użytkownik może odczuć spowolnienie systemy wynikające z aktualizowania jakiegoś indeksu?*).

```
mysql> CREATE INDEX idx_x ON indeksy (x);
Query OK, 0 rows affected (41.95 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> CREATE INDEX idx_y ON indeksy (y);
Query OK, 0 rows affected (51.50 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> CREATE INDEX idx_z ON indeksy (z);
Query OK, 0 rows affected (45.94 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> SHOW INDEX FROM indeksy;
+---------+------------+----------+--------------+-------------+-----------+-------------+---
```

```
| Table   | Non_unique | Key_name | Seq_in_index | Column_name | Collation | Cardinality | Su
+---------+------------+----------+--------------+-------------+-----------+-------------+---
| indeksy |          0 | PRIMARY  |            1 | id          | A         |      998030 |
| indeksy |          1 | idx_x    |            1 | x           | A         |         180 |
| indeksy |          1 | idx_y    |            1 | y           | A         |         170 |
| indeksy |          1 | idx_z    |            1 | z           | A         |         160 |
+---------+------------+----------+--------------+-------------+-----------+-------------+---
4 rows in set (0.00 sec)
```

Ponownie wykonujemy to samo zapytanie. Widać, że tym razem wykonało się ono dużo szybciej. Z pomocą polecenia `EXPLAIN` widać, że tym razem system użył trzech wcześniej zdefiniowanych indeksów.

```
mysql> SELECT * FROM indeksy WHERE x=1 AND y=10 AND z=100;
+------+------+------+------+
| id   | x    | y    | z    |
+------+------+------+------+
| 1000 |    1 |   10 |  100 |
+------+------+------+------+
1 row in set (0.05 sec)

mysql> EXPLAIN SELECT * FROM indeksy WHERE x=1 AND y=10 AND z=100;
+----+-------------+---------+-------------+------------------+------------------+---------+--
| id | select_type | table   | type        | possible_keys    | key              | key_len | r
+----+-------------+---------+-------------+------------------+------------------+---------+--
|  1 | SIMPLE      | indeksy | index_merge | idx_x,idx_y,idx_z | idx_x,idx_y,idx_z | 5,5,5   | N
+----+-------------+---------+-------------+------------------+------------------+---------+--
1 row in set (0.00 sec)
```

Tworzymy kolejny indeks, tym razem będzie to indeks złożony, zbudowany na trzech kolumnach.

```
mysql> CREATE INDEX idx_xyz ON indeksy (x, y, z);
Query OK, 0 rows affected (1 min 5.31 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> SHOW INDEX FROM indeksy;
+---------+------------+----------+--------------+-------------+-----------+-------------+---
| Table   | Non_unique | Key_name | Seq_in_index | Column_name | Collation | Cardinality | Su
+---------+------------+----------+--------------+-------------+-----------+-------------+---
| indeksy |          0 | PRIMARY  |            1 | id          | A         |      998030 |
| indeksy |          1 | idx_x    |            1 | x           | A         |         180 |
| indeksy |          1 | idx_y    |            1 | y           | A         |         170 |
| indeksy |          1 | idx_z    |            1 | z           | A         |         160 |
| indeksy |          1 | idx_xyz  |            1 | x           | A         |         190 |
| indeksy |          1 | idx_xyz  |            2 | y           | A         |       20792 |
| indeksy |          1 | idx_xyz  |            3 | z           | A         |      998030 |
+---------+------------+----------+--------------+-------------+-----------+-------------+---
7 rows in set (0.00 sec)
```

Z pomocą polecenia `EXPLAIN` widać, że tym razem system użył jednego potrójnego indeksu. Mając do wybory również 3 wcześniej zdefiniowane indeksy pojedyncze, nie użył ich.

```
mysql> EXPLAIN SELECT * FROM indeksy WHERE x=1 AND y=10 AND z=100;
+----+-------------+---------+------+--------------------------+---------+---------+--
| id | select_type | table   | type | possible_keys            | key     | key_len | r
+----+-------------+---------+------+--------------------------+---------+---------+--
|  1 | SIMPLE      | indeksy | ref  | idx_x,idx_y,idx_z,idx_xyz | idx_xyz | 15      | c
+----+-------------+---------+------+--------------------------+---------+---------+--
1 row in set (0.00 sec)
```

Polecenia `show profiles` oraz `show profile` pokazują zebrane dane na temat wykonywanych poleceń (w trakcie bieżącej sesji). To drugie pokazuje szczegóły ostatnio wykonanego polecenia.

```
mysql> show profiles;
+----------+------------+--------------------------------------------------------------------------+
| Query_ID | Duration   | Query                                                                    |
+----------+------------+--------------------------------------------------------------------------+
|        1 |  0.00169800 | SHOW INDEX FROM indeksy                                                 |
|        2 |  1.82039550 | SELECT * FROM indeksy WHERE x=1 AND y=10 AND z=100                      |
|        3 |  0.00062575 | EXPLAIN SELECT * FROM indeksy WHERE x=1 AND y=10 AND z=100              |
|        4 | 42.94518125 | CREATE INDEX idx_x ON indeksy (x)                                       |
|        5 | 43.41200525 | CREATE INDEX idx_y ON indeksy (y)                                       |
|        6 | 57.34573100 | CREATE INDEX idx_z ON indeksy (z)                                       |
|        7 |  0.00164775 | SHOW INDEX FROM indeksy                                                 |
|        8 |  0.05573650 | SELECT * FROM indeksy WHERE x=1 AND y=10 AND z=100                      |
|        9 |  0.00092175 | EXPLAIN SELECT * FROM indeksy WHERE x=1 AND y=10 AND z=100              |
|       10 | 71.21777575 | CREATE INDEX idx_xyz ON indeksy (x, y, z)                               |
|       11 |  0.28426475 | SHOW INDEX FROM indeksy                                                 |
|       12 |  0.22805450 | EXPLAIN SELECT * FROM indeksy WHERE x=1 AND y=10 AND z=100              |
+----------+------------+--------------------------------------------------------------------------+
12 rows in set, 1 warning (0.00 sec)

mysql> show profile;
+----------------------+----------+
| Status               | Duration |
+----------------------+----------+
| starting             | 0.000126 |
| checking permissions | 0.000013 |
| Opening tables       | 0.000042 |
| init                 | 0.000055 |
| System lock          | 0.000025 |
| optimizing           | 0.000025 |
| statistics           | 0.000371 |
| preparing            | 0.000034 |
| executing            | 0.000006 |
| Sending data         | 0.000095 |
| end                  | 0.000008 |
| query end            | 0.000016 |
| closing tables       | 0.000018 |
| freeing items        | 0.000147 |
| cleaning up          | 0.000034 |
+----------------------+----------+
15 rows in set, 1 warning (0.00 sec)
```

Za pomocą polecenia `SHOW CREATE TABLE` możemy również możemy zobaczyć, jakie mamy pozakładane na danej tabeli indeksy.

```
mysql> SHOW CREATE TABLE indeksy;
+---------+-------------------------------------------------------------------------------
| Table   | Create Table
+---------+-------------------------------------------------------------------------------
| indeksy | CREATE TABLE `indeksy` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `x` int(11) DEFAULT NULL,
  `y` int(11) DEFAULT NULL,
  `z` int(11) DEFAULT NULL,
  PRIMARY KEY (`id`),
  KEY `idx_x` (`x`),
  KEY `idx_y` (`y`),
  KEY `idx_z` (`z`),
```

```
  KEY `idx_xyz` (`x`,`y`,`z`)
) ENGINE=InnoDB AUTO_INCREMENT=1000001 DEFAULT CHARSET=utf8 COLLATE=utf8_bin |
+---------+-------------------------------------------------------------------------------
1 row in set (0.00 sec)
```

Wykasujemy teraz indeksy pojedyncze (kasowanie, w odróżnieniu od tworzenia indeksów, trwa już bardzo krótko):

```
mysql> ALTER TABLE indeksy DROP INDEX idx_x;
Query OK, 0 rows affected (0.45 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> ALTER TABLE indeksy DROP INDEX idx_y;
Query OK, 0 rows affected (0.58 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> ALTER TABLE indeksy DROP INDEX idx_z;
Query OK, 0 rows affected (0.39 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

Pozostał nam tylko indeks złożony (na trzech kolumnach: *x*, *y*, *z*) i indeks ten zostanie użyty tylko wówczas, gdy podana klauzula `WHERE` będzie miała odpowiednią postać (*pytanie: czy potrafisz podać warunki, jakie musi spełniać klauzula `WHERE`, aby indeks został użyty*):

```
mysql> EXPLAIN SELECT COUNT(*) FROM indeksy WHERE x=1 AND y=10 AND z=100;
+----+-------------+---------+------+---------------+---------+---------+--
| id | select_type | table   | type | possible_keys | key     | key_len | r
+----+-------------+---------+------+---------------+---------+---------+--
|  1 | SIMPLE      | indeksy | ref  | idx_xyz       | idx_xyz | 15      | c
+----+-------------+---------+------+---------------+---------+---------+--
1 row in set (0.00 sec)

mysql> EXPLAIN SELECT COUNT(*) FROM indeksy WHERE y=10 AND z=100;
+----+-------------+---------+-------+---------------+---------+---------+--
| id | select_type | table   | type  | possible_keys | key     | key_len | r
+----+-------------+---------+-------+---------------+---------+---------+--
|  1 | SIMPLE      | indeksy | index | NULL          | idx_xyz | 15      | N
+----+-------------+---------+-------+---------------+---------+---------+--
1 row in set (0.00 sec)

mysql> EXPLAIN SELECT COUNT(*) FROM indeksy WHERE z=100;
+----+-------------+---------+-------+---------------+---------+---------+--
| id | select_type | table   | type  | possible_keys | key     | key_len | r
+----+-------------+---------+-------+---------------+---------+---------+--
|  1 | SIMPLE      | indeksy | index | NULL          | idx_xyz | 15      | N
+----+-------------+---------+-------+---------------+---------+---------+--
1 row in set (0.00 sec)
```

Porównajmy więc czasy wykonywania się przykładowych poleceń. Wyraźnie widać, że indeks nie zawsze został użyty.

```
mysql> show profiles;
+----------+------------+-------------------------------------------------------------------+
| Query_ID | Duration   | Query                                                             |
+----------+------------+-------------------------------------------------------------------+
```

```
|        1 | 0.01440125 | SELECT COUNT(*) FROM indeksy WHERE x=1                    |
|        2 | 1.04310650 | SELECT COUNT(*) FROM indeksy WHERE y=10                   |
|        3 | 0.98092650 | SELECT COUNT(*) FROM indeksy WHERE z=100                  |
|        4 | 1.04094700 | SELECT COUNT(*) FROM indeksy WHERE z=100 AND y=10         |
|        5 | 0.00077825 | SELECT COUNT(*) FROM indeksy WHERE x=1 AND y=10 AND z=100 |
|        6 | 0.00078525 | SELECT COUNT(*) FROM indeksy WHERE z=100 AND y=10 AND x=1 |
+----------+------------+-----------------------------------------------------------+
6 rows in set, 1 warning (0.00 sec)
```

Na koniec zobaczmy jak zmienią się czasy wykonywania zapytania, gdy użyjemy klauzuli `LIMIT`. Wykasujemy więc najpierw istniejący indeks złożony a następnie zapytamy się o rekord, o którym wiemy (bo znamy polecenie `INSERT`, które utworzyło dane w tabeli, patrz skrypt PHP *dane.php*), że występuje na początku tabeli oraz o rekord występujący na końcu tabeli. Widać wyraźnie pozytywne działanie klauzuli `LIMIT`. Oczywiście w praktycznych zastosowaniach najczęściej nie znamy fizycznego położenia poszukiwanego rekordu lub rekordów i dlatego też używanie klauzuli `LIMIT` zwykle nie da tak dobrych wyników, jak, nieco sztuczny, zaprezentowany poniżej przykład.

```
mysql> ALTER TABLE indeksy DROP INDEX idx_xyz;
Query OK, 0 rows affected (0.39 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> SELECT * FROM indeksy WHERE x=1 AND y=1 AND z=1;
+----+------+------+------+
| id | x    | y    | z    |
+----+------+------+------+
|  1 |    1 |    1 |    1 |
+----+------+------+------+
1 row in set (1.85 sec)

mysql> SELECT * FROM indeksy WHERE x=100 AND y=100 AND z=100;
+---------+------+------+------+
| id      | x    | y    | z    |
+---------+------+------+------+
| 1000000 |  100 |  100 |  100 |
+---------+------+------+------+
1 row in set (1.85 sec)

mysql> SELECT * FROM indeksy WHERE x=1 AND y=1 AND z=1 LIMIT 1;
+----+------+------+------+
| id | x    | y    | z    |
+----+------+------+------+
|  1 |    1 |    1 |    1 |
+----+------+------+------+
1 row in set (0.00 sec)

mysql> SELECT * FROM indeksy WHERE x=100 AND y=100 AND z=100 LIMIT 1;
+---------+------+------+------+
| id      | x    | y    | z    |
+---------+------+------+------+
| 1000000 |  100 |  100 |  100 |
+---------+------+------+------+
1 row in set (1.88 sec)

mysql> show profiles;
+----------+------------+-----------------------------------------------------------+
| Query_ID | Duration   | Query                                                     |
```

```
+----------+------------+-------------------------------------------------------------+
|        1 | 1.84644750 | SELECT * FROM indeksy WHERE x=1 AND y=1 AND z=1              |
|        2 | 1.85420075 | SELECT * FROM indeksy WHERE x=100 AND y=100 AND z=100        |
|        3 | 0.00065300 | SELECT * FROM indeksy WHERE x=1 AND y=1 AND z=1 LIMIT 1      |
|        4 | 1.87563150 | SELECT * FROM indeksy WHERE x=100 AND y=100 AND z=100 LIMIT 1 |
+----------+------------+-------------------------------------------------------------+
4 rows in set, 1 warning (0.00 sec)
```

4. Zadanie do samodzielnego wykonania

Ponownie uruchamiamy skrypt *dane.php*, aby od nowa zbudować testową tabelę. Kasując tabelę, usuwane są oczywiście też wszystkie założone na niej indeksy. Zastanów się dlaczego w załączonym skrypcie nie zdecydowano się na wstawianie danych do tabeli *indeksy* za pomocą następującego kodu (jest on w skrypcie zakomentowany):

```
$ile = 100;
for ($i = 1; $i <= $ile; $i++) {
  for ($j = 1; $j <= $ile; $j++) {
    for ($k = 1; $k <= $ile; $k++) {
      $query = "INSERT INTO indeksy (x, y, z) VALUES ($i, $j, $k);";
      $wynik = mysqli_query ($link, $query);
    }
  }
}
```

Zamiast tego tworzony jest najpierw plik dyskowy i jego zawartość jest importowana do bazy danych z poziomu konsoli MySQL, korzystając z polecenia `source ścieżka\nazwa_pliku`.

Sama procedura testowa będzie polegała na kilkukrotnym wykonaniu zapytania:

```
SELECT * FROM indeksy WHERE x=1 AND y=10 AND z=100;
```

i odczytaniu czasu odpowiedzi. W kolejnych próbach zmieniamy wyłącznie organizację indeksów założonych na tabeli (jak niżej), bez zmiany wydawanego zapytania. Przykładowo pozycja numer 5 w tabeli poniżej („Indeksy proste na kolumnach x, y”) rozumiana jest tak, że w chwili wykonywania polecenia `SELECT` na tabeli testowej założone są tylko dwa indeksy proste, na kolumnach odpowiednio *x* oraz *y*. Aby przejść do pozycji 6, należy wykasować indeks na kolumnie *y* a założyć indeks na kolumnie *z*, itd.

	Aktualna konfiguracja indeksów
1	Brak indeksów
2	Indeks prosty na kolumnie x
3	Indeks prosty na kolumnie y
4	Indeks prosty na kolumnie z
5	Indeksy proste na kolumnach x, y
6	Indeksy proste na kolumnach x, z
7	Indeksy proste na kolumnach y, z

8	Indeksy proste na kolumnach x, y, z
9	Indeks złożony na kolumnach x, y
10	Indeks złożony na kolumnach x, z
11	Indeks złożony na kolumnach y, z
12	Indeks złożony na kolumnach x, y, z

Samodzielnie wykonaj odpowiednie eksperymenty. Zanotuj czasy wykonywania i zaprezentuj je prowadzącemu. Skomentuj wyniki.

5. Kolejne zadanie do samodzielnego wykonania

Przygotowano tabelę zawierającą 1.000.000 rekordów, gdzie:

- kolumna *imie* została zapełniona losowo imionami z listy 5. imion żeńskich i 5. imion męskich,
- kolumna *nazwisko* zawiera losowe frazy o długości pomiędzy 3 a 15. Pierwsza litera jest duża,
- w kolumnie *plec* jest wpisana losowo litera M lub K,
- w kolumnie *zarobki* wpisano losową liczbę z przedziału od 1 do 5000,
- w kolumnie *data_zatr* wpisano losową datę pomiędzy datą bieżącą (dzień, w którym generowano skrypt) a datą maksymalnie 5000 dni wcześniejszą.

Odpowiedni skrypt PHP (*dane.php*) generujący plik (*dane2.sql*) tworzący tabelę *indeksy2* oraz dane w tej tabeli umieszczono jako **załącznik do niniejszej instrukcji**.

Zwróć uwagę, jakich funkcji użyto do generowania losowych danych. Używając ich w odpowiedni sposób, łatwo jest szybko zbudować samodzielnie program do automatycznego generowania losowych danych na potrzeby testowania aplikacji bazodanowych. Takim automatem można wówczas szybko zapełnić tworzoną bazę danych wielką ilością testowych danych. Jest to bardzo ważny etap tworzenia i testowania aplikacji bazodanowych, często bowiem okazuje się, że aplikacja „przestaje działać choć jeszcze wczoraj działała", gdy przybywa w niej danych i zaczynają ujawniać się różne jej niedoróbki i błędy, których nie dostrzegamy, gdy w bazie jest minimalna ilość danych testowych (lub nie ma ich wcale).

```
DROP TABLE IF EXISTS indeksy2;

CREATE TABLE indeksy2 (
  id INT PRIMARY KEY AUTO_INCREMENT,
  imie VARCHAR(10),
  nazwisko VARCHAR(15),
  plec CHAR(1),
  zarobki INT,
  data_zatr DATE
);

INSERT INTO indeksy2 (imie, nazwisko, plec, zarobki, data_zatr) VALUES
('Maja', 'Gptwdnjbcfy', 'K', 2329, '2008-04-05'),
('Szymon', 'Msipkhtwq', 'K', 2792, '2012-05-26'),
('Jan', 'Rlkytfgurhmwpv', 'K', 3206, '2003-03-21'),
```

```
('Maja', 'Ruqivxowcngmbdyh', 'M', 916, '2013-11-03'),
('Szymon', 'Tckqun', 'K', 764, '2007-05-10'),
...
('Lena', 'Rxucpsngwbdahmol', 'M', 1502, '2012-01-31'),
('Szymon', 'Vnygotxfwrbc', 'M', 3225, '2011-08-26'),
('Jakub', 'Yjkhewzltbmdgcp', 'K', 4430, '2015-01-10'),
('Zuzanna', 'Hlvhonkriajsfwmu', 'K', 4962, '2009-09-26'),
('Jakub', 'Mtawezlnvdxs', 'M', 2163, '2009-10-29');
```

W ramach ćwiczenia należy samodzielnie zaproponować oraz przeprowadzić stosowne testy pokazujące działanie założonych wcześniej indeksów. Pamiętajmy, że mamy tym razem do czynienia z trzema różnymi typami danych (znakowe, numeryczne, typu data). Jakie to ma praktyczne konsekwencje?