Bazy danych. Laboratorium 1

1. Tworzenie tabel i typy danych w bazach MySQL.

Polecenia `CREATE, ALTER, DROP` są poleceniami DDL (Data Definition Language - język definicji danych), czyli takimi, które odpowiadają za tworzenie, zmianę i usuwanie struktur dla danych w bazie.

Aby stworzyć tabelę musimy podać jej nazwę oraz co najmniej jedną definicję kolumny.

Przykład:

```
CREATE TABLE osoba (id int);

ze zdefiniowaną domyślną wartością kolumny
CREATE TABLE osoba (id int, plec enum('K','M') default 'K');
```

Dokumentacja polecenia CREATE -> https://dev.mysql.com/doc/refman/8.0/en/create-table.html

Powyższe polecenie stworzy tabelę o nazwie `osoba` z jedną kolumną o nazwie `id`, która może przechowywać wartości liczbowe całkowite.

Tworząc projekt bazy danych pod konkretne rozwiązanie należy wybrać najbardziej optymalne typy danych dla dziedziny, którą chcemy zamodelować. Rozumie się przez to odpowiedni format oraz wielkość pamięci, która jest niezbędna dla przechowania jednej wartości danego typu. Np. dla przechowania imienia dowolnej osoby powinien nam wystarczyć typ `TINYTEXT`, który pozwala zachować do 255 znaków. Są też inne typy tekstowe, np. `VARCHAR`, który pozwala przechować do 65,535 znaków (bajtów). Jednak to jest maksymalna długość całego wiersza tabeli oraz wartość jest różna w zależności od wybranego kodowania znaków bazy. Typ `MEDIUMTEXT` pozwala na zapisanie 16,777,215 znaków i też się nada, ale niewłaściwe użycie spowoduje wykorzystanie znacznie większej ilości pamięci (RAM jak i dyskowej) niż faktycznie jest to potrzebne.

Poniżej tabela z popularnymi typami danych bazy MySQL.

Typ danych	Długość	Opis
TINYINT	1 bajt	Zakres od -128 do +127 (bez znaku: od 0 do 255)
SMALLINT	2 bajty	Zakres od -32 768 do +32 767 (bez znaku: od 0 do 65 535)
MEDIUMINT	3 bajty	Zakres od -8 388 608 do +8 388 607 (bez znaku: od 0 do 16 777 215)
INT,INTEGER	4 bajty	Zakres od -2 147 483 648 do +2 147 483 647 (bez znaku: od 0 do 4 294 967 295)
BIGINT	8 bajtów	Zakres od -9 223 372 036 854 775 808 do +9 223 372 036 854 775 807 (bez znaku: od 0 do 18 446 744 073 709 551 615)
BIT, BOOL	1 bajt	synonim dla TINYINT(1)

Typ danych	Długość	Opis
FLOAT		Zakres od -3.402823466E+38 do +3.402823466E+38
DOUBLE		Zakres od -1.7976931348623157E+308 do +1.7976931348623157E+308
DOUBLE PRECISION, REAL		synonim dla typu DOUBLE
DECIMAL(m,d), DEC(m,d), NUMERIC(m,d)		zakres ustawi parametry "m" oraz "d", maksymalny zakres jest taki sam jak dla typu DOUBLE
DATE	'0000-00-00'	data w formacie "rok-miesiąc-dzień" względnie "RRRR-MM-DD". Zakres od 1000-01-01 do 9999-12-31
DATETIME	'0000-00-00 00:00:00'	data oraz czas. Zakres od 1000-01-01 00:00:00 do 9999-12-31 23:59:59 (format = "RRRR-MM-DD HH:MM:SS")
TIMESTAMP(n)	'0000-00-00 00:00:00'	- data oraz czas w zakresie od 1970-01-01 00:00:00 do 2037-01-01 00:00:00 (zapisywanych jest zawsze wszystkich 14 liczb !) - format wyświetlenia (i dla zapytań) można ustawić przy pomocy parametru "m" o wartości 14 (lub bez wartości), 12, 10, 8, 6, 4, lub 2 - "RRRRMMDDHHMMSS", "RRMMDDHHMMSS", "RRMMDDHHMM", "RRRRMMDD", "RRMMDD", "YYMM", "YY" - jeżeli do pola bazy danych tego typu nie zostanie zapisana żadna wartość, wtedy MySQL sam uzupełni czas bieżący zmiany do danego wiersza
TIME	'00:00:00'	zakres czasu wynosi od "-838:59:59" do "838:59:59" (format "HH:MM:SS")
YEAR(n)	0000	YEAR(4) = zakres od 1901 do 2155, (format "RRRR"), YEAR(2) = zakres od 1970 do 2069
CHAR(m)		- długość łańcucha "m" może być w zakresie od 0 do 255 - jeżeli zapisywany łańcuch będzie krótszy niż jest ustawione, wtedy zostanie automatycznie uzupełniony o spacje (ma zatem "stałą" wielkość)
CHAR		(bez "m") jest uważane za CHAR(1)
VARCHAR(m)		- długość łańcucha "m" może być w zakresie od 0 do 65535 - jeżeli zapisywany łańcuch jest krótszy niż jest ustawione, wtedy brakujące znaki nie są uzupełniane (łańcuch ma zatem zmienną długość), lecz dodatkowo jest zapisywana informacja o jego długości
TINYBLOB, TINYTEXT		Długość łańcucha wynosi maks. 255 znaków
BLOB, TEXT		Długość łańcucha wynosi maks. 65 535 znaków

Typ danych	Długość	Opis
MEDIUMBLOB, MEDIUMTEXT		Długość łańcucha wynosi maks. 16 777 215 znaków
LONGBLOB, LONGTEXT		Długość łańcucha wynosi maks. 4 294 967 295 znaków
ENUM('item1','item2',...)		- tablica z góry przygotowanych łańcuchów (itemów) o maks. ilości 65 535 - w polu tabeli może się znaleźć tylko jeden z itemów, które zostały z góry przygotowane - zamiast nazwy 'item' można stosować również ich kolejność, a więc: 1 (zamiast 'item1'), 2 (zamiast 'item2')...
SET('item1','item2',...)		- tablica z góry definiowanych łańcuchów (itemów) o maks. ilości 64 - w polu tablicy może się znaleźć nawet kilka itemów, które są z góry zdefiniowane

Dokumentacja odnośnie typów danych dostępna jest pod adresem:
https://dev.mysql.com/doc/refman/8.0/en/data-types.html

Rozpatrzmy kolejny przykład:

```
CREATE TABLE osoba (id INT AUTO_INCREMENT PRIMARY KEY, imie TINYTEXT NOT NULL,
nazwisko TINYTEXT NOT NULL, wiek INT(3));
```

Kolumna id posiada dodatkowe atrybuty. `AUTO_INCREMENT` jest atrybutem, który uruchamia mechanizm sekwencji dla tej kolumny i domyślnie po dodaniu każdego kolejnego wiersza do tabeli zwiększa jej wartość o 1. Atrybut `PRIMARY KEY` ustawia tę kolumnę jako klucz główny tabeli co ma kilka skutków:

- wartości w kolumnie muszą być unikalne (mechanizm bazy będzie tego pilnował i uniemożliwiał wstawienie duplikatów)
- wartość w tej kolumnie nie może być `NULL`
- tabela może posiadać tylko jeden klucz główny, ale może on się składać z jednej lub wielu kolumn.

Klucze główne oraz obce (o czym później) można również dodać tak jak kolumnę tabeli. Takie podejście jest wymagane jeżeli klucz główny składa się z więcej niż jednej kolumny.

Przykład:

```
CREATE TABLE osoba (imie VARCHAR(50) NOT NULL, nazwisko VARCHAR(50) NOT NULL, wiek
INT(3), PRIMARY KEY (imie, nazwisko));
```

Typ `TINYTEXT` nie mógł być użyty dla klucza głównego ze względu na wymaganą długość, która musi być podana. Teraz kluczem głównym jest kombinacja imienia i nazwiska, która w tej tabeli musi być unikalna.

Atrybut `NOT NULL` powoduje, że wartość tej kolumny jest wymagana co oznacza, że wstawiając wiersz do tabeli osoba wartość imie i nazwisko nie mogą być `NULL`.

Do polecenia `CREATE TABLE` można jeszcze użyć opcjonalnej części `IF NOT EXISTS`, która sprawdzi czy taka tabela istnieje przed wykonaniem polecenia. Jeżeli tabela istnieje to różnica nie jest wielka bo polecenie bez `IF NOT EXISTS` zwróci Query OK i się nie wykona, a samo polecenie `CREATE` zwróci błąd (tabela już istnieje) i też się nie wykona.

Przykład:

```
CREATE TABLE IF NOT EXISTS osoba (imie VARCHAR(50) NOT NULL, nazwisko VARCHAR(50)
NOT NULL, wiek INT(3), PRIMARY KEY (imie, nazwisko));
```

Tabelę (i inne elementy bazy) usuwamy poleceniem `DROP`. **Przykład:**

```
DROP TABLE osoba;

bazę również możemy usunąć, ale ostrożnie z tym poleceniem
DROP DATABASE nazwa_bazy;
```

Listę tabel, które znajdują się w naszej bazie danych możemy sprawdzić poleceniem `show tables;`. Polecenie `show create table;` wyświetla natomiast SQL, którym można tę tabelę stworzyć. Warto sprawdzić kilka tabel w ten sposób, gdyż dodawane jest często formatowanie, które jest opcjonalne ale prawidłowe dla składni języka SQL.

Inne przydatne polecenie to `describe nazwa_tabeli;`, które wyświetla opis tabeli gdzie znajdziemy nazwy kolumn, typy danych i inne atrybuty, które zostały ustawione. Skrócona postać polecenia to `desc nazwa_tabeli;`.

2. Wstawianie i aktualizacja wartości w tabeli

Dane do istniejącej tabeli wstawiamy poleceniem `INSERT`.

Przykład:

```
INSERT INTO osoba VALUES(default, 'Jan', 'Kowalski', 35);
```

Do tabeli soba zostanie dodany nowy wiersz danych. W przypadku użycia powyższej konstrukcji musimy podać wartości dla wszystkich kolumn i w takiej kolejności jak zostały w tabeli zdefiniowane. Wartość `default` powoduje wstawienie wartości domyślnej dla danej kolumny, co w przypadku kolumny id (która posiada atrybut `auto_increment`) pobierze i wstawi kolejną wartość sekwencji. Format tekstowy (i inne nie numeryczne) wymagają ograniczenia wartości poprzez ' ' lub " ". Jeżeli część kolumn posiada zdefiniowane wartości domyślne to możemy ten mechanizm wykorzystać, ale wtedy postać polecenia `INSERT` wygląda nieco inaczej.

Przykład:

```
INSERT INTO osoba(imie, nazwisko, wiek) VALUES('Jan', 'Kowalski', 35);
lub
INSERT INTO osoba(wiek, nazwisko, imie) VALUES(35, 'Kowalski', 'Jan');
```

Możemy również dodac wiele wierszy jednocześnie. **Przykład:**

```
INSERT INTO osoba VALUES
    (default, 'Jan', 'Kowalski', 35),
    (default, 'Marian', 'Bąbel', 55),
    (default, 'Alina', 'Kamińska', 44);

lub

INSERT INTO osoba(wiek, nazwisko, imie) VALUES
(35, 'Kowalski', 'Jan'),
(55, 'Bąbel', 'Marian'),
(44, 'Kamińska', 'Alina');
```

Aktualizacja danych odbywa się poleceniem `UPDATE`.

Przykład:

```
UPDATE osoba SET wiek=56 WHERE nazwisko='Bąbel';
```

Zaktualizujemy wiek na 56 w każdym rekordzie, w którym nazwisko to Bąbel.

Możemy również zaktualizować wszystkie rekordy w tabeli.

Przykład:

```
UPDATE osoba SET wiek=99;
```

3. Zmiana struktury tabeli

Nierzadko istnieje konieczność zmiany struktury tabeli (lub innych elementów bazy danych). Służy do tego polecenie `ALTER`. Zacznijmy od przykładu, który dodaje nową kolumnę do tabeli `osoba`.

Przykład:

```
ALTER TABLE osoba ADD COLUMN data_urodzenia DATE AFTER wiek;
ALTER TABLE osoba ADD COLUMN data_urodzenia DATE FIRST wiek;
```

Zostanie utworzona kolumna `data_urodzenia`, która zostanie umieszczona za kolumną `wiek`, co i bez części `AFTER...` miałoby taki sam skutek, gdyż domyślnie kolumna dodawana jest jako ostatnia w tabeli. `FIRST` wstawia dodawaną kolumnę jako pierwszą.

Kolumnę możemy również usunąć.

Przykład:

```
ALTER TABLE osoba DROP data_urodzenia;
```

Słowo `COLUMN` zostało celowo pominięte, gdyż zarówno przy dodawaniu jak i usuwaniu kolumny jest ono opcjonalne.

Definicję kolumny można również zmieniać.

Przykład:

```
Zmiana typu kolumny
ALTER TABLE osoba MODIFY data_urodzenia int;

Ziana nazwy i typu kolumny
ALTER TABLE osoba CHANGE data_urodzenia data_ur date;

Zmiana samej nazwy kolumny (wersja 8.0 MySQL)
ALTER TABLE osoba RENAME COLUMN data_ur TO data_urodzenia;

Zmiana watości domyślnej w kolumnie
ALTER TABLE osoba ALTER wiek SET DEFAULT 99;

Usunięcie watości domyślnej w kolumnie
ALTER TABLE osoba ALTER wiek DROP DEFAULT;

Usunięcie klucza głównego (chociaż może nie pójść tak łatwo)
ALTER TABLE osoba DROP PRIMARY KEY;

To teraz dodanie klucza
ALTER TABLE osoba ADD PRIMARY KEY(id);

Zmiana wartości sekwencji
ALTER TABLE osoba AUTO_INCREMENT = 13;

Zmiana nazwy tabeli
RENAME TABLE `osoba` TO `osoba_old`;
```

W przykładzie ze zmianą nazwy tabeli po raz pierwszy pojawiły się znaki ``(backtick, słyszałem też nazwę 'psie uszy'), których używa się dla nazw elementów bazy - nazwy baz, tabel, kolumn, kluczy itp.

Zadania

Wykonaj zadania z pliku zadania_1.md a rozwiązania zapisuj w pliku tekstowym do późniejszego wykorzystania i przesłania prowadzącemu.

4. Klucze obce

W relacyjnych bazach danych do tworzenia relacji wykorzystywany jest klucz główny oraz klucz obcy. Klucz obcy można zdefiniować jako kopię (odwołanie) klucza głównego w innej tabeli. Pozwala to na rozbijanie większych zbiorów danych na mniejsze poprzez proces normalizacji bazy danych, tworzenie większej ilości tabel powiązanych ze sobą relacjami. Ten mechanizm pozwala również na zachowanie spójności danych. Rozważmy poniższy przykład.

Przykład:

```
CREATE TABLE uczelnia (id int auto_increment PRIMARY KEY, nazwa VARCHAR(400));

CREATE TABLE student(indeks VARCHAR(10) PRIMARY KEY, imie TINYTEXT, nazwisko
TINYTEXT, uczelnia int, FOREIGN KEY (uczelnia) REFERENCES uczelnia(id));
```

Fragment `FOREIGN KEY (uczelnia) REFERENCES uczelnia(id)` odpowiada za stworzenie klucza obcego i relacji kolumny `student.uczelnia -> uczelnia.id`. Ważne jest aby typ kolumn, które chcemy połączyć relacją był zgodny, a w przypadku typu numerycznego również identycznej długości. Wykonanie tego polecenia doda również unikalną nazwę klucza obcego, którą możemy podejrzeć poprzez polecenie `show create table student`.

Poniżej efekt komendy.

```
CREATE TABLE `student` (`indeks` varchar(10) NOT NULL,`imie` tinytext,`nazwisko`
tinytext,`uczelnia` int(11) DEFAULT NULL,PRIMARY KEY (`indeks`),KEY `uczelnia`
(`uczelnia`),CONSTRAINT `student_ibfk_1` FOREIGN KEY (`uczelnia`) REFERENCES
`uczelnia` (`id`)) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

Stworzony wcześniej klucz posiada nazwę `student_ibfk_1` i ta nazwa potrzebna jest aby taki klucz obcy usunąć.

Przykład:

```
ALTER TABLE student DROP FOREIGN KEY student_ibfk_1;
```

Utworzenie relacji na powyższych tabelach spowoduje, że do tabeli `student` w kolumnie `uczelnia` możemy umieścić tylko takie wartości, które występują w kolumnie, do której klucz obcy się odwołuje, czyli kolumnie `id` w tabeli `uczelnia`.

Co jednak gdy z tabeli `uczelnia` będziemy chcieli usunąć rekord, do którego odwołanie występuje w tabeli `student` ? Możemy wykorzystać mechanizm zachowania spójności (silnik InnoDB bazy MySQL) na kilka

sposobów za pomocą dodatkowej opcji klucza obcego `ON DELETE`, który może przyjąć jedną z wartości: `RESTRICT | CASCADE | SET NULL | NO ACTION | SET DEFAULT`

Wartość `RESTRICT` uniemożliwia usunięcie rekordu głównego póki występują rekordy potomne. Określenie tej wartości (lub wartości `NO ACTION`) lub pominięcie sekcji `ON DELETE` ma takie same działanie. `CASCADE` powoduje natomiast usunięcie rekordu głównego i wszystkich rekordów powiązanych. Ustawienie `SET NULL` powoduje usunięcie rekordu głównego i wstawienie wartości `NULL` w kolumnie powiązanego klucza obcego. Należy pamiętać, że kolumna klucza obcego nie może mieć w takim przypadku ustawionego atrybutu `NOT NULL`. `SET DEFAULT` mimo, że jest interpretowane przez parser MySQL nie jest dłużej obsługiwane.

Przykład:

```
CREATE TABLE student(indeks VARCHAR(10) PRIMARY KEY, imie TINYTEXT, nazwisko
TINYTEXT, uczelnia int, FOREIGN KEY (uczelnia) REFERENCES uczelnia(id) ON DELETE
CASCADE);
```

Klucze obce mogą być również dodawane do istniejących tabel poprzez polecenie `ALTER`.

Przykład:

```
CREATE TABLE student(indeks VARCHAR(10) PRIMARY KEY, imie TINYTEXT, nazwisko
TINYTEXT, uczelnia int);

ALTER TABLE student ADD FOREIGN KEY (uczelnia) REFERENCES uczelnia(id) ON DELETE
CASCADE;

Możemy też sami zdecydować o nazwie takiego klucza.
ALTER TABLE student ADD CONSTRAINT nazwa_klucza FOREIGN KEY (uczelnia) REFERENCES
uczelnia(id) ON DELETE CASCADE;
```

Zadania

Wykonaj zadania z pliku zadania_2.md a rozwiązania zapisuj w pliku tekstowym do późniejszego wykorzystania.