

1 SQL

SQL to język służący do budowania zapytań do baz danych. Istnieją różne warianty języka, np. MySQL, PostgreSQL, OracleSQL, MSSQL i wiele innych, które różnią się nieznacznie między sobą, ale główne zasady działania pozostają takie same.

Dane w bazach danych są zebrane w tabele. Tabele składają się z kolumn o predefiniowanych typach, a dane przechowywane są jako wiersze tabeli. Tabela powinna mieć też kolumnę będącą kluczem głównym (**UNIQUE_ID**), które nie może przyjąć dwa razy tej samej wartości (baza danych tego pilnuje). Zazwyczaj na takim polu ustawia się własność **AUTO_INCREMENT**, która automatycznie nadaje ID jako kolejne liczby naturalne. Niektóre z typów dostępnych w bazach to: **INT**, **FLOAT**, **DOUBLE**, **VARCHAR**, **TEXT**, **DATE**, **DATETIME** i inne. Istnieje również specjalny typ **REFERENCE**, który pozwala stworzyć odniesienie do wiersza innej tabeli. Przy próbie usunięcia wiersza, do którego istnieją odniesienia podniesiony zostanie wyjątek (przydatne w praktyce).

Podstawowe operacje w bazie danych to: **SELECT**, **CREATE**, **UPDATE**, **SET**, **DELETE**, **DROP**. Na maturze w części teoretycznej wystarczy operacja **SELECT**.

Poniżej przedstawiono dwie przykładowe tabele:

	id	name	position	sal	manager	dept
emp:	1367	"Marek"	"Operator serwera"	1200	1369	1
	1368	"Agata"	"Programista"	1400	1369	1
	1369	"Aneta"	"Manager zespołu"	2500	NULL	1
	1370	"Zbigniew"	"Dział Kadr"	1500	NULL	2

	id	str	city
dept:	0	"Warszawska 14"	"Poznań"
	1	"Startowa 9"	"Warszawa"
	2	"Zabłocie 15"	"Kraków"

Podstawowa składnia operacji **SELECT** to **SELECT ... FROM ... WHERE**. Przykładowo zapytanie

```
SELECT * FROM emp
```

zwróci całą tabelę **emp**.

Jeśli interesują nas tylko imiona pracowników zarabiających powyżej 1200, to możemy wykonać zapytanie:

```
SELECT id, name FROM emp WHERE sal > 1200
```

Uwaga: Wielkość liter w zapytaniach nie ma znaczenia, ale dla wygody warto pisać słowa kluczowe wielkimi literami, a pozostałe elementami małymi.

Wyniki możemy zapytań możemy sortować. Domyślne sortowanie jest rosnące i nie wymaga żadnych dodatkowych słów kluczowych. Sortowanie malejące wymaga dodatkowego słowa jak w przykładzie:

```
SELECT id, name FROM emp WHERE sal > 1200 ORDER BY name DESCENDING
```

Na typach dostępnych w bazie danych zdefiniowane są sensowne porządki (np. w tym wypadku leksykograficzny).

Najbardziej skomplikowaną operacją jest łączenie tabel przy pomocy klauzuli **JOIN**. Przykładowo, jeśli chcemy wypisać imiona pracowników wraz z miastem, w którym są zatrudnieni, to możemy to zrobić jak poniżej. Dodatkowo zastosowaliśmy dodatkowe słowo kluczowe **AS**, które pozwala nam aliasować tabelę i skrócić zapis.

```
SELECT a.id, a.name, b.city
FROM emp AS a
WHERE sal > 1200
JOIN dept AS b ON a.dept = b.id
```

Zauważmy, że jeżeli np. tabela po lewej stronie (**emp**) nie będzie miała odpowiednika w tabeli po prawej stronie (**dept**), to ten wiersz pierwszej tabeli nie pojawi się w wyniku – a może chcemy, żeby pojawił się z jakimiś **NULL**ami. Z tego powodu klauzula **JOIN** może występować w kilku wariantach:

- **LEFT JOIN** – wypisuje wszystkie elementy lewej tabeli i dopasowuje do nich elementy z prawej tabeli
- **RIGHT JOIN** – podobnie jak poprzednio, tylko w drugą stronę
- **FULL OUTER JOIN** – **LEFT JOIN** i **RIGHT JOIN** jednocześnie
- **NATURAL JOIN** – próbuje się domyślić po których kolumnach połączyć tabele – różnie to wychodzi, więc lepiej nie używać

Dostępne są też funkcje agregujące, np. **SUM**, **COUNT**, **MIN**, **MAX**, **AVERAGE** i inne. Korzystanie z nich wymaga dodania klauzuli **GROUP BY**, np. jeśli chcemy policzyć ilu jest pracowników o danym imieniu, to możemy zrobić to w następujący sposób:

```
SELECT a.name, COUNT(a.name) FROM emp AS a GROUP BY a.name
```

2 Zadanie

W bazie danych pewnego sklepu ze sprzętem komputerowym znajdują się tabele **klienci** i **zamowienia**, które przedstawiono poniżej. Dla tabel podano też kilka pierwszych wierszy.

imie	nazwisko	nr_telefonu
Weronika	Abacka	192 102 947
Jakub	Babacki	938 192 728
Szymon	Cabacki	722 719 382
...

nr_zamowienia	produkt	ilosc	nr_telefonu
1	"Kabel USB"	2	938 192 728
2	"Klawiatura Bluetooth"	1	722 719 382
3	"Baterie AAA"	12	938 192 728
...

Napisz zapytanie w języku SQL, które wygeneruje listę wszystkich klientów, którzy złożyli mniej niż 3, lub nie złożyli żadnego zamówienia. Posortuj listę rosnąco po liczbie zamówień. Możesz założyć, że klienci mają różne numery telefonów, a numer zamówienia jest kluczem głównym w swojej tabeli.

3 SQL – funkcje bardziej zaawansowane

W przykładach do tej części będziemy rozważać następującą tabelę o nazwie `emp`:

id	name	position	sal	manager	dept	login
1367	"Marek"	"Operator serwera"	1200	1369	1	"pracownik1"
1368	"Agata"	"Programista"	1400	1369	1	"pracownik10"
1369	"Aneta"	"Manager zespołu"	2500	NULL	1	"manager2"
1370	"Zbigniew"	"Dział Kadr"	1500	NULL	2	"pracownik3"
1371	"Weronika"	"CEO"	4000	NULL	2	"pracownik"
1372	"Arkadiusz"	"Programista"	1300	1369	1	"pracownik123"

3.1 Operator AS w nazwach kolumn

Do tej pory używaliśmy operatora `AS`, żeby nadać alias dla tabeli, ale możemy to także zrobić dla kolumn. Pomaga to nam na przykład, kiedy używamy funkcji agregujących:

- baza danych automatycznie nadaje nazwy kolumnom wynikowym – jeśli używamy funkcji agregujących, np. `COUNT`, to zostanie ona nazwana podobnie do `COUNT_OF_DEPT`, co nie wygląda zbyt ładnie, jeśli ma to być jakieś oficjalne zestawienie (może też być niewygodne, jeśli chcemy napisać sobie jakiś skrypt na bazie danych i potem np. zmieni się nazwa kolumny – będziemy musieli ją zmieniać w wielu miejscach)
- jeśli chcemy użyć agregacji w warunku `WHERE` i jednocześnie wypisać wartość tej agregacji w wynikowej tabeli i nie chcemy tego pisać dwa razy

Przykład:

```
SELECT a.dept, COUNT(a.dept) AS liczba_prac
FROM emp AS a
WHERE liczba_prac > 2
GROUP BY a.dept
```

3.2 Operator LIKE

Często w SQLu będziemy chcieli wyszukiwać stringi, które pasują do jakiegoś wzorca, dlatego w standardzie zawarty jest operator `LIKE`, który wspiera bardzo podstawowe funkcje regexa. Dwa wildcardy, z których będziemy korzystać, to `%` i `_`.

`%` – reprezentuje 0 lub więcej znaków; przykład zastosowania:

```
SELECT login
FROM emp
WHERE login LIKE "pracownik%"
```

Wówczas w odpowiedzi dostaniemy tabelę zawierającą loginy: `pracownik1`, `pracownik10`, `pracownik3`, `pracownik`, `pracownik123`.

_ – reprezentuje dokładnie jeden znak; przykład zastosowania:

```
SELECT login
FROM emp
WHERE login LIKE "pracownik_"
```

Wtedy odpowiedź będzie wyglądała następująco: `pracownik1`, `pracownik3`.

Możemy też połączyć oba te operatory, żeby np. wymusić przynajmniej `n` znaków, ale dopuścić więcej. W poniższym przykładzie szukamy pracowników, którzy w loginie mają co najmniej dwie cyfry:

```
SELECT login
FROM emp
WHERE login LIKE "pracownik_%"
```

W odpowiedzi dostajemy: `pracownik10`, `pracownik123`.

3.3 Operacje na czasie

Składnia dotycząca czasu, dostępne typy i funkcje różnią się (często znacząco) między różnymi implementacjami baz danych, więc dzisiaj skupimy się na funkcjonalności dostępnej w Microsoft Access (program, z którym będziemy pracowali).

- `Date()` – zwraca aktualną datę
- `Now()` – zwraca aktualny czas (czyli datę + godzinę)
- `Time()` – zwraca aktualny czas sformatowany jako string
- `DateAdd(interval, number, date)` – dodaje do daty odpowiedni czas zależny od `interval` (np. `s` – sekundy, `n` – minuty, `m` – miesiące, pełną listę możliwych jednostek czasu można znaleźć w internecie)
- `DatePart(interval, date, [firstdayofweek], [firstweekofyear])` – zwraca wartość podanej jednostki czasu; UWAGA! niedziela jest domyślnie pierwszym dniem tygodnia
- `Minute(date)`, `Day(date)`, `Month(date)`, `Year(date)`, `Weekday(date)` – podobnie jak `DatePart` i ta sama uwaga!
- `DateDiff(interval, date1, date2, [firstdayofweek], [firstweekofyear])` – zwraca różnicę między datami w podanej jednostce czasu

Na czasie można też wykonywać normalne operacje arytmetyczne `+/–`, ale nie zawsze będą się one dobrze zachowywały – jeśli to możliwe, lepiej użyć funkcji.

Pełna lista funkcji dostępna w [dokumentacji](#).

3.4 Zwracanie wyrażenia arytmetycznego

W wyrażeniu `SELECT` można oczywiście wykonywać standardowe operacje arytmetyczne, w których argumentami mogą być liczby lub kolumny. Kilka przykładów:

- wypisujemy dodatkową kolumnę, która jest liczbą:
`SELECT name, sal, 100 AS nazwa FROM emp`
- sprawdzamy, co by się stało, gdyby wszyscy dostali podwyżkę:
`SELECT name, sal, sal + 100 AS nowa_pensja FROM emp`
- podwyżka zależna od departamentu, w którym ktoś pracuje:
`SELECT name, sal, sal + dept * 100 AS nowa_pensja FROM emp`

3.5 Instrukcje warunkowe

Można w wyrażeniach `SELECT` używać instrukcji warunkowych `IF` i `CASE` – ich składnia znowu będzie się różniła między wariantami SQLa. Przykłady w miarę standardowe:

```
SELECT name, sal, IF (dept == 1) THEN sal + 100 ELSE sal END FROM emp
```

```
SELECT name, sal, CASE dept WHEN 1 THEN sal + 100 WHEN 2 THEN sal END FROM emp
```