Uniwersytet Wrocławski

Bazy danych 2022

Wykład 2

SQL: proste zapytania, podstawy normalizacji, złączenia

Bartosz Brzostowski

Wydział Fizyki i Astronomii UWr
semestr letni r. akad. 2021/22

10 marca 2022

SQL: **S**tructured **Q**uery **L**anguage

- Język do komunikacji z relacyjną bazą danych
- Bardzo stary (1974), niewiele młodszy od czystego C (1972)
- Kilka standardów, wiele implementacji (i wersji. . .)
- Zgodnie z nazwą — język zapytań (DQL)
- Także język definicji (DDL) oraz modyfikacji danych (DML)
- Deklaratywny — mówimy, *co* chcemy uzyskać, nie *jak*[1]
- Z założenia podobny do naturalnego języka angielskiego

```
SELECT login FROM users;
```

- Trochę odbiega od „czystego” modelu relacyjnego (np. dozwolone duplikaty wierszy), ale nie będziemy się tym przejmować

DQL → Data Query Language

DDL → Data Definition Language

DML → Data Manipulation Language

[1] Większość powszechnie używanych i znanych języków to języki imperatywne. Do języków deklaratywnych należą m.in. języki funkcyjne, np. Lisp, Haskell, Erlang, F♯

CRUD: **c**reate, **r**ead, **u**pdate, **d**elete

- Podstawowe cztery operacje na złożonym zasobie danych, tu — BD
- Stwórz, odczytaj, zaktualizuj, usuń (rekord)
- Odpowiadające wyrażenia SQL: `INSERT`, `SELECT`, `UPDATE`, `DELETE`
- `SELECT` należy do DQL, pozostałe — do DML
- (Gdzie DDL? Nie tutaj — dotyczy schematu BD, nie stanu)
- Rozbudowana aplikacja bazodanowa również powinna implementować wszystkie cztery

Podstawy składni SQL

- Komentarze inline: od `--␣` (spacja istotna!) lub `#`
- Komentarze blokowe: `/* ... */`
- Złamania wierszy, wcięcia — nieistotne (ale ładnie je stosować)
- Wielkość liter nieistotna (oczywiście poza stałymi napisowymi) — wersaliki to tylko konwencja
- Delimitery:
  - stałych napisowych: ’, " (na standardowej klawiaturze koło Enter)
  - identyfikatorów (gdy potrzebne): ‘ (w lewym górnym rogu, koło 1)
- Klient MySQL sam określa, gdzie kończy się wyrażenie, ale… najczęściej nie może ze względu na składnię wyrażeń — ręcznie za pomocą ;

Dość często zapominają Państwo o średniku!

Proste zapytania `SELECT`

- Już je widzieliśmy: `SELECT PI() + SQRT(2);`
- Aż za proste... i gdzie tu bazy danych?
- `SELECT * FROM` *`[tabela]`*`;` zwraca całą tabelę (wszystkie kolumny)
- `SELECT DISTINCT ...` usuwa duplikaty
- `SELECT [DISTINCT]` *`[kolumna1]`*`, ` *`[kolumna2]`*`, ...` — wybrane kolumny
- `SELECT DISTINCT` *`[kol1]`*`, DISTINCT` *`[kol2]`*`, ...` — błędne (bo `DISTINCT` tylko na niektórych z wybranych kolumn byłoby źle zdefiniowane)

Przykładowy SELECT

```
mysql> SELECT * FROM klienci;
+-----+--------------+---------+-----------------------+-----------
| idk | nazwa        | miasto  | adres                 | telefon
+-----+--------------+---------+-----------------------+-----------
|   1 | Astro        | Wrocław | Cybulskiego 12/2      | 0713229563
|   2 | BCA          | Wrocław | ul. Pomorska 321/12   | 0719563372
|   3 | XYZ          | Wrocław | Pl. Borna 5/1         | 0713753372
|   4 | ERE          | Warszawa| Marszałkowska 1 /2    | 0221122563
|   5 | OCY          | Łódź    | ul. Piotrkowska 111/1 | 0427213372
|   6 | ATest        | Wrocław | Nozownicza 1          | 606753717
|   7 | JAFO         | Toruń   | Wirtualna             | 765092123
|   8 | Cesoft       | Wrocław | Rynek 0               | 456789765
|  12 | INNOV        | Warszawa| Marszałkowska 1       | 0223456712
|  24 | Jan Kowalczyk| Wrocław | Opolska 119c          | 666777888
|  27 | SP201        | Wrocław | Borna 4               | 567321555
+-----+--------------+---------+-----------------------+-----------
11 rows in set (0,00 sec)
```

Kiedy kolumny nie mieszczą się w oknie

```
mysql> SELECT * FROM klienci \G
*************************** 1. row ***************************
    idk: 1
  nazwa: Astro
 miasto: Wrocław
  adres: Cybulskiego 12/2
telefon: 0713229563
*************************** 2. row ***************************
    idk: 2
  nazwa: BCA
 miasto: Wrocław
  adres: ul. Pomorska 321/12
telefon: 0719563372
*************************** 3. row ***************************
    idk: 3
  nazwa: XYZ
 miasto: Wrocław
  adres: Pl. Borna 5/1
...
```

SELECT a SELECT DISTINCT

```
mysql> SELECT miasto
    -> FROM klienci;
+----------+
| miasto   |
+----------+
| Wrocław  |
| Wrocław  |
| Wrocław  |
| Warszawa |
| Łódź     |
| Wrocław  |
| Toruń    |
| Wrocław  |
| Warszawa |
| Wrocław  |
| Wrocław  |
+----------+
11 rows in set (0,00 sec)
```

```
mysql> SELECT DISTINCT miasto
    -> FROM klienci;
+----------+
| miasto   |
+----------+
| Wrocław  |
| Warszawa |
| Łódź     |
| Toruń    |
+----------+
4 rows in set (0,00 sec)

mysql> SELECT DISTINCT miasto,
    -> DISTINCT nazwa FROM klienci;
ERROR 1064 (42000): You have an
error in your SQL syntax; check the
manual that corresponds to your ...
```

Jak skomplikować klauzulę SELECT

- SELECT *[kolumna1]* + *[funkcja]*(*[kolumna2]*) ...
  — wyrażenia kolumnowe
- Kolumna nazywa się jak wyrażenie

```
mysql> SELECT nazwa, IF(miasto = "Wrocław", "TAK", "NIE")
    -> FROM klienci;
+--------------+--------------------------------------+
| nazwa        | IF(miasto = "Wrocław", "TAK", "NIE")  |
+--------------+--------------------------------------+
| Astro        | TAK                                   |
...
```

- Aliasy! SELECT *[wyr_kolumnowe]* AS *[alias]* ...

```
mysql> SELECT nazwa, IF(miasto = "Wrocław", "TAK", "NIE")
    -> AS lokalny FROM klienci;
+--------------+---------+
| nazwa        | lokalny |
+--------------+---------+
| Astro        | TAK     |
...
```

Filtrowanie wyników — klauzula `WHERE`

- `SELECT ... FROM ... WHERE` *[wyr_logiczne]*

```
mysql> SELECT * FROM produkty WHERE cena > 3000;
+-----+----------------+---------+-------+
| idp | nazwa          | cena    | ilosc |
+-----+----------------+---------+-------+
|  33 | Apple Iphone 8 | 3152.00 |     2 |
+-----+----------------+---------+-------+
1 row in set (0,00 sec)
```

- Operatory binarne: =, != lub <>, <, <=, >, >=, <=>
- Operatory unarne: *[wyr]* `IS [NOT] NULL` (postfiksowe!)
- Dla napisów: `[NOT] LIKE`, `[NOT] REGEXP`

```
mysql> SELECT "Raz Dwa Trzy" = "R_z % Trzy";
0
mysql> SELECT "Raz Dwa Trzy" LIKE "R_z % Trzy";
1
```

% reprezentuje zero lub więcej znaków

_ reprezentuje pojedynczy znak

- Spójniki logiczne: ! lub `NOT`, && lub `AND`, || lub `OR`

Funkcje biblioteczne

- Jest ich... sporo, niestety zależnych od implementacji
- A po co nam to? Wszystko zrobimy po stronie aplikacji!
- Filtrowanie wyników powinno odbywać się jak najwcześniej
- Funkcje kontrolne:
  - `IFNULL(`*[w1]*`,` *[w2]*`)` — zwraca *[w1]* jeśli nie jest on NULLem, w przeciwnym przypadku *[w2]*
  - Uogólnienie: `COALESCE(...)` zwraca pierwszy nieNULLowy argument
  - `IF` — jak `?` `:` w C (przypomnienie: prawda to wartość, a więc nie NULL, różna od 0)
  - `CASE` *[w]* `WHEN` *[w1]* `THEN` *[r1]* `[WHEN` *[w2]* `THEN` *[r2]* `...] [ELSE` *[r]*`] END` — przyrównuje *[w]* kolejno do *[w1]*, *[w2]*... i zwraca odpowiednie *[rN]*; jeśli żadna równość nie zaszła, zwraca *[r]* lub NULL jeśli brak `ELSE`
  - `CASE WHEN` *[wyr1]* `THEN` *[r1]* `[WHEN` *[wyr2]* `THEN` *[r2]* `...] [ELSE` *[r]*`] END` — analogicznie, tylko sprawdza się prawdziwość *[wyrN]*

Funkcje daty i czasu

- Typy danych: `DATE`, `TIME[(`*[prec]*`)]`, `DATETIME[(`*[prec]*`)]` — opcjonalna precyzja sekund od 0 (domyślna) do 6
- `TIMESTAMP[(`*[prec]*`)]` głównie do datowania zmian w wierszach; zakres taki, jak czasu UNIX
- Funkcje:
  - `CURDATE()`, `CURTIME([`*[prec]*`])`, `NOW([`*[prec]*`])`
  - Wyciąganie składowych z daty-i-czasu: `DATE(...)`, `TIME(...)`
  - Wyciąganie składowych z daty: `YEAR(...)`, `MONTH(...)`, `MONTHNAME(...)`, `DAY(...)`
  - Wyciąganie składowych z czasu: `HOUR(...)`, `MINUTE(...)`, `SECOND(...)`, `MICROSECOND(...)`
  - Ogólniej: `EXTRACT(`*[jednostka]* `FROM ...)`
  - Formatowanie: `DATE_FORMAT(..., `*[spec_formatu]*`)`
  - `DAYOFYEAR(...)`
  - `DAYNAME(...)` (nazwa); `DAYOFWEEK(...)`, `WEEKDAY(...)`
  - `WEEKOFYEAR(...)`, `WEEK(...)` (opc. tryb); `YEARWEEK(...)`

Proszę zwrócić uwagę na różnice.
UWAGA! Oba niezgodne z normą ISO

Arytmetyka daty i czasu

- Liczby możemy dodawać i odejmować od siebie, otrzymując liczby
- Różnica dat nie jest datą, tylko interwałem; do daty możemy dodać interwał, a nie datę
- Jest typ `INTERVAL...` w PostgreSQLu
- Funkcje:
  - `ADDDATE(..., INTERVAL ... [jedn])` lub `ADDDATE(..., [liczba_dni])`; `SUBDATE()` analogicznie
  - `ADDTIME(..., [czas])`, `SUBTIME(..., [czas])`
  - `DATEDIFF(..., ...)` zwraca liczbę dni
  - `TIMEDIFF(..., ...)` zwraca `TIME`

Inne ważne funkcje

- Napisowe:
  - `LOWER`, `UPPER`, `REVERSE` (przekształcenia)
  - `CHAR_LENGTH` a `LENGTH` (długość w znakach lub bajtach)
  - `LEFT`, `RIGHT`, `TRIM`, `SUBSTR`, `SUBSTRING_INDEX` (różne podnapisy)
  - `LOCATE` (pozycja wzorca w tekście)
  - `CONCAT`, `CONCAT_WS` (sklejanie z separatorem lub bez)
- Arytmetyczne:
  - `POW`, `SQRT`, `EXP`, `...` (działania matematyczne)
  - Operatory: `MOD` (także funkcja) lub `%` (reszta z dzielenia), `DIV` (dzielenie całkowitoliczbowe)
  - Dzielenie przez 0 ma naturalny wynik — jaki?
  - `FLOOR`, `CEIL`, `ROUND`, `TRUNCATE` (zaokrąglanie)
  - `PI`; `RAND`
- JSON: `JSON_ARRAY`, `JSON_OBJECT`, `...`
- Konwersja typów: `CAST` (standard SQL) lub `CONVERT`

Czy warto wszystko mieć w jednej tabeli?

ID	nazwisko	...	rola
1	Kowalski		student
2	Nowak		nauczyciel
3	Wiśniewska		nauczyciel
4	Wójcik		prac. admin.
	...		

- Zalety: prosta baza
- Redundancja:
  - powtórzenia marnują pamięć (co może najmniej istotne)
  - podatne na niespójność (np. rola osoby wprowadzona z literówką)
- Anomalie:
  - aktualizacja zawsze we wszystkich wystąpieniach
  - jeśli rola nie ma reprezentanta, to nie istnieje
  - nie da się wstawić nowej roli
  - po usunięciu reprezentantów rola znika

Postać znormalizowana

PRACOWNIK

ID	nazwisko	...	idRoli
1	Kowalski		30
2	Nowak		10
3	Wiśniewska		10
4	Wójcik		20
	...		

FOREIGN KEY → PRIMARY KEY

ROLA

idRoli	nazwa roli
10	nauczyciel
20	prac. admin.
30	student
99	współpr. zewn.
	...

- Zalety: brak omówionych wad
- Jak zapewnić spójność danych? PRACOWNIK.idRoli ⊆ ROLA.idRoli
  - Warunki i tak prostsze niż poprzednio: wartości z kolumn tabeli potomnej muszą występować w tabeli macierzystej
  - Więzy klucza obcego
  - O resztę zadba DBMS
- Jak wyciągać dane z kilku tabel naraz?

Złączenia

- Nie zawsze złączane tabele są związane kluczem obcym
- W SQLu: wyrażenia tabelowe w klauzuli `FROM`
- Najprostsze „złączenie”: iloczyn kartezjański

ID	nazwisko	...	idRoli	idRoli	nazwaRoli
1	Kowalski		30	10	nauczyciel
1	Kowalski		30	20	prac. admin.
			...		
2	Nowak		10	10	nauczyciel
2	Nowak		10	20	prac. admin.
			...		

- `SELECT ...`
  `FROM` *`[tab1]`* `[CROSS | INNER] JOIN` *`[tab2]`*
  (albo od przecinka)

Złączenia (prawdziwe)

- Jak łączyć tylko wiersze wiążące się ze sobą?

ID	nazwisko	...	idRoli	idRoli	nazwaRoli
1	Kowalski		30	30	student
2	Nowak		10	10	nauczyciel
3	Wiśniewska		10	10	nauczyciel
4	Wójcik		20	20	prac. admin.
			...		

- `SELECT ...`
  `FROM` *`[tab1]`* `[CROSS | INNER] JOIN` *`[tab2]`*
  `ON` *`[kol1]`* `=` *`[kol2]`*
- Ujednoznacznianie kolumn: *`[tab1].[kol]`*, *`[tab2].[kol]`*
- Według standardu `CROSS JOIN` nie powinien dopuszczać warunku `ON`, a `INNER JOIN` powinien go wymagać

Złączenia (nieco inaczej)

- Powtórzona kolumna, względem której łączyliśmy
- Można tego uniknąć, jeśli kolumny mają tę samą nazwę:

ID	nazwisko	...	idRoli	nazwaRoli
1	Kowalski		30	student
2	Nowak		10	nauczyciel
3	Wiśniewska		10	nauczyciel
4	Wójcik		20	prac. admin.
		...		

- `SELECT ...`
  `FROM` *`[tab1]`* `[CROSS | INNER] JOIN` *`[tab2]`*
  `USING (`*`[kolumna]`*`)`
- Jeszcze zachłanniej:
  `SELECT ... FROM` *`[tab1]`* `NATURAL JOIN` *`[tab2]`*
  — łączy wszystkie pary kolumn o tych samych nazwach

Złączenia (ogólniej)

- W konstrukcji `[tab1] JOIN [tab2] ON [warunek]` dozwolony dowolny `[warunek]` logiczny, nie tylko koniunkcja równości par kolumn
- Zapytanie
  `SELECT ... FROM [tab1] JOIN [tab2] ON [warunek]`
  jest równoważne
  `SELECT ... FROM [tab1], [tab2] WHERE [warunek]`
- Jak lepiej? Czytelność wyrażeń (koncepcyjna różnica między *warunkiem złączenia* a *warunkiem filtrowania*)
- Nie przerzucać wszystkiego do klauzuli `WHERE`: im mniejsze pośrednie wyniki obliczeń (zwłaszcza przy wielokrotnych złączeniach), tym lepiej
- Złączenia wielu tabel (czytelniejsze z nawiasami):
  `SELECT ... FROM`
  `[tab1] JOIN [tab2] ON [w1] JOIN [tab3] ON [w2]`

Klauzula `SELECT` przy złączeniach

- Też czasem konieczne ujednoznacznienie, np.:
  `produkty.nazwa`, `klienci.nazwa`
- Można wybrać wszystkie kolumny z jednej tabeli:
  `SELECT` *`[t1].*, [coś_z_t2]`* `FROM` *`[t1]`* `JOIN` *`[t2]`* `...`
- Przy wyborze wyłącznie kolumn z jednej tabeli (niekoniecznie wszystkich), efekt podobny do filtrowania: zostają tylko te wiersze tej tabeli, które zdołały się z czymś połączyć… potencjalnie z powtórzeniami:

```
mysql> SELECT idk, nazwa
    -> FROM klienci
    -> WHERE nazwa LIKE "%t";
+-----+--------+
| idk | nazwa  |
+-----+--------+
|   6 | ATest  |
|   8 | Cesoft |
+-----+--------+
```

```
mysql> SELECT idk, nazwa FROM klienci
    -> JOIN zamow ON idk = k_id
    -> WHERE nazwa LIKE "%t";
+-----+--------+
| idk | nazwa  |
+-----+--------+
|   8 | Cesoft |
|   8 | Cesoft |
+-----+--------+
```

Złączenia bez „filtrowania”

- Niektóre wiersze nie „przeżywają” złączeń
- Jak można je „ocalić”? Uzupełniając NULLami
- Złączenie zewnętrzne *prawostronne*: zawiera wszystko to, co złączenie wewnętrzne, plus uzupełnione NULLami wiersze z tabeli po *prawej*, które nie mają się z czym połączyć (po jednym razie)
- Wszystkie role wraz z ich przedstawicielami, o ile istnieją:

ID	nazwisko	...	idRoli	idRoli	nazwaRoli
1	Kowalski		30	30	student
2	Nowak		10	10	nauczyciel
3	Wiśniewska		10	10	nauczyciel
4	Wójcik		20	20	prac. admin.
NULL	NULL	NULLe	NULL	99	współpr. zewn.
			...		

Złączenia zewnętrzne cd.

- Złączenie lewostronne — analogiczne:

ID	nazwisko	...	idRoli	idRoli	nazwaRoli
1	Kowalski		30	30	student
2	Nowak		10	10	nauczyciel
3	Wiśniewska		10	10	nauczyciel
4	Wójcik		20	20	prac. admin.
66	Bezrolny		NULL	NULL	NULL
			...		

- Składnia:
  - `[t1] {LEFT | RIGHT} [OUTER] JOIN [t2] {ON ... | USING (...)}`
  - `[t1] NATURAL {LEFT | RIGHT} [OUTER] JOIN [t2]`
- Złączenie zewnętrzne *musi* być z warunkami albo naturalne

Co już wiemy o SELECTach

- `SELECT` *[wyrażenia_kolumnowe]*
  `FROM` *[wyrażenie_tabelowe]*
  `WHERE` *[wyrażenie_filtrujące]*
- Sortowanie: `ORDER BY` *[wyr1]*, *[wyr2]* ...
- Najczęściej względem którychś `SELECT`owanych wyrażeń
- Można używać aliasów z klauzuli `SELECT` i pozycji kolumn
- Dozwolone dowolne wyrażenie kolumnowe
- `ORDER BY` *[wyr]* `DESC` odwraca kolejność