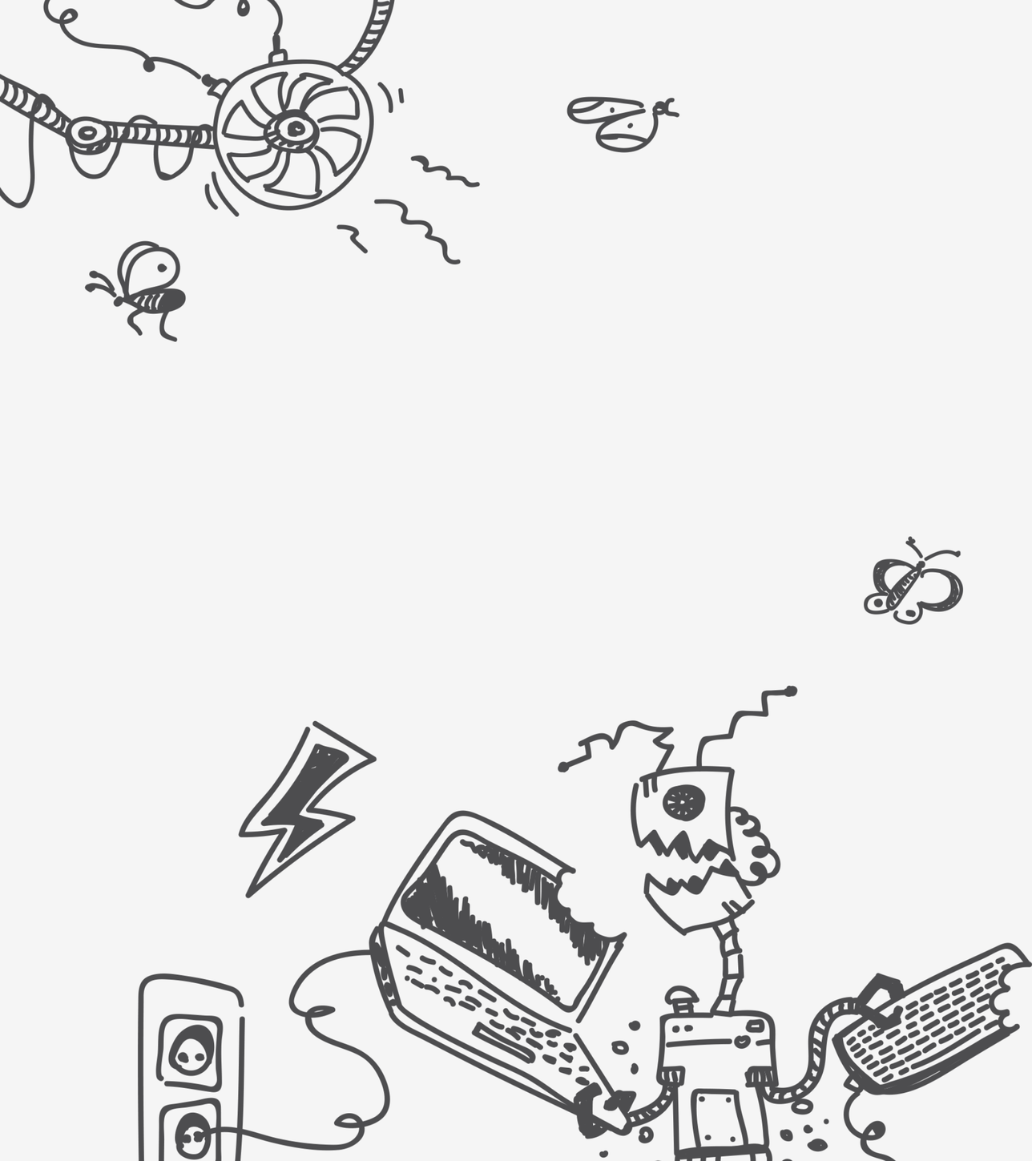

Podstawy SQL - dodawanie i wybieranie danych

Dodawanie elementów do tabeli

Dane do tabeli dodajemy za pomocą zapytania `INSERT INTO`:

```
INSERT INTO table_name(columnName1,
                       columnName2,
                       columnName3,
                       ...)
VALUES (value1,
        value2,
        value3,
        ...);
```

Jeżeli po nazwie tabeli nie podamy nazw kolumn, dane będą wkładane w kolejne kolumny tabeli (zgodnie z jej definicją).

Dodawanie elementów do tabeli

```
my_first_db=# INSERT INTO users VALUES (0, 'Jacek', 'jacek@gmail.com');
INSERT 0 1

my_first_db=# INSERT INTO users VALUES('Wojtek', 'wojtek@gmail.com');
ERROR:  invalid input syntax for integer: "Wojtek"
LINE 1: INSERT INTO users VALUES('Wojtek', 'wojtek@gmail.com');
                                 ^

my_first_db=# INSERT INTO users(user_name, user_email) VALUES('Wojtek', 'wojtek@gmail.com');
INSERT 0 1
```

Dodawanie elementów do tabeli

```
my_first_db=# INSERT INTO users VALUES (0, 'Jacek', 'jacek@gmail.com');
INSERT 0 1

my_first_db=# INSERT INTO users VALUES('Wojtek', 'wojtek@gmail.com');
ERROR:  invalid input syntax for integer: "Wojtek"
LINE 1: INSERT INTO users VALUES('Wojtek', 'wojtek@gmail.com');
                                 ^

my_first_db=# INSERT INTO users(user_name, user_email) VALUES('Wojtek', 'wojtek@gmail.com');
INSERT 0 1
```

→ Nie zbyt obszerny opis. psql mówi nam, że został dodany do bazy jeden rekord (`1`). `0` to identyfikator obiektu tabeli (OID). Domyślnie OID dla tabeli nie jest ustawiony stąd wartość 0. Nie będziemy się tym zajmować podczas kursu. Zainteresowanych odsyłam do dokumentacji: https://www.postgresql.org/docs/8.1/datatype-oid.html

Dodawanie elementów do tabeli

```
my_first_db=# INSERT INTO users VALUES (0, 'Jacek', 'jacek@gmail.com');
INSERT 0 1

my_first_db=# INSERT INTO users VALUES('Wojtek', 'wojtek@gmail.com');
ERROR:  invalid input syntax for integer: "Wojtek"
LINE 1: INSERT INTO users VALUES('Wojtek', 'wojtek@gmail.com');
                                 ^

my_first_db=# INSERT INTO users(user_name, user_email) VALUES('Wojtek', 'wojtek@gmail.com');
INSERT 0 1
```

→ Nie zbyt obszerny opis. psql mówi nam, że został dodany do bazy jeden rekord (`1`). `0` to identyfikator obiektu tabeli (OID). Domyślnie OID dla tabeli nie jest ustawiony stąd wartość 0. Nie będziemy się tym zajmować podczas kursu. Zainteresowanych odsyłam do dokumentacji: https://www.postgresql.org/docs/8.1/datatype-oid.html

Błąd spowodowany tym, że liczba kolumn nie jest równa liczbie przekazanych danych.

Czas zapytania SQL

Wspomnieliśmy, że odpowiedź psql, nie jest zbyt rozbudowana. Jeśli chcemy możemy włączyć logowanie czasu wykonania zapytania. Przydaje się to, kiedy np. optymalizujemy zapytania.

Do włączenia logowania czasu służy instrukcja sterująca `\timing`:

```
my_first_db=# \timing
Timing is on.
```

Przykład:

```
my_first_db=# INSERT INTO users VALUES (0, 'Jacek', 'jacek@gmail.com');
INSERT 0 1
Time: 9,125 ms
```

Mechanizm logowania czasu, wyłączamy za pomocą tej samej komendy.

Wczytywanie elementów z tabeli

Dane z tabeli wczytujemy za pomocą zapytania `SELECT`:

```
SELECT column_name, column_name
FROM table_name;
```

Żeby wybrać wszystkie kolumny możemy użyć *

```
SELECT * FROM table_name;
```

Wczytywanie elementów z tabeli

Elementy zwracane są w następującej postaci:

```
SELECT * FROM users;
```

```
 user_id | user_name |    user_email
---------+-----------+------------------
       0 | Jacek     | jacek@gmail.com
       1 | Wojtek    | wojtek@gmail.com
(2 rows)
```

Klauzula WHERE

Możemy zawężać wyniki wyszukiwania przez dodanie klauzuli `WHERE` do naszego zapytania `SELECT`.

```
SELECT column_name, column_name
FROM table_name
WHERE column_name = <szukana wartość>;
```

Np.:

```
SELECT * FROM users
WHERE user_name = 'Wojtek';
```

Klauzula WHERE

Elementy zwracane są w postaci tabelki asocjacyjnej, gdzie kluczem jest nazwa kolumny:

```
SELECT * FROM users
WHERE user_name LIKE 'W%';
```

```
 user_id | user_name
---------+-----------
       1 | Wojtek
       2 | Wojtek2
(2 rows)
```

Operacje porównania w PostgreSQL

`=`	Równe
`<> lub !=`	Nierówne
`>(>=)`	Większe niż (większe równe niż)
`<(<=)`	Mniejsze niż (mniejsze równe niż)
`BETWEEN a AND b`	Pomiędzy podanym zakresem (wliczając podany zakres)
`LIKE`	Szuka podanego wzorca (tylko napisy)
`IN(a,b,c)`	Znajduje się w zmiennych podanych w nawiasach
`NOT`	Może poprzedzać inne operacje
`OR / AND`	Operatory logiczne łączące poszczególne wyrażenia

Klauzula AS

Jeżeli z jakiegoś powodu w wynikach wyszukiwania mamy dwie kolumny o takiej samej nazwie, to w Pythonie będziemy mieli dostęp tylko do jednej z nich (jeżeli korzystamy z kursora zwracającego słownik).

Możemy zawsze nadać kolumnie nową nazwę (alias) na czas tego wyszukania.

Robimy to za pomocą klauzuli `AS`:

```
SELECT column_name AS column_alias
FROM table_name;
```

Np.:

```
SELECT user_id AS id FROM users;
```

Klauzula ORDER BY

Możemy sortować znalezione wyniki względem jednej kolumny (lub więcej). Służy do tego klauzula `ORDER BY`

```
SELECT column_name, column_name FROM
table_name
ORDER BY column_name ASC|DESC,
column_name ASC|DESC;
```

Przykład

```
SELECT * FROM users ORDER BY name;
```

```
SELECT * FROM users ORDER BY user_name;
```

```
 user_id | user_name
---------+-----------
       1 | Antek
       3 | Beata
       2 | Wojtek2
---------+-----------
(3 rows)
```

Klauzula ORDER BY

Możemy sortować znalezione wyniki względem jednej kolumny (lub więcej). Służy do tego klauzula `ORDER BY`

```
SELECT column_name, column_name FROM
table_name
ORDER BY column_name ASC|DESC,
column_name ASC|DESC;
```

Przykład

```
SELECT * FROM users ORDER BY name;
```

→ Wybieramy jedną z możliwości:
ASC – rosnąco (ascending),
DESC – malejąco (descending).

```
SELECT * FROM users ORDER BY user_name;
```

```
 user_id | user_name
---------+-----------
       1 | Antek
       3 | Beata
       2 | Wojtek2
---------+-----------
(3 rows)
```