Bazy danych MySQL w Java

dzień 2

v3.1

Plan

1. Zmiana i usuwanie danych w tabeli
2. Modyfikacja tabel
3. Relacje między tabelami
   - Łączenie tabel
4. Zaawansowany SQL

Zmiana i usuwanie danych w tabeli

Zmiana wartości danych

Dane w tabeli zmieniamy za pomocą zapytania **UPDATE**:

```
UPDATE table_name
SET
col_name_1=value1,
col_name_2=value2,
...
WHERE col_name_3=some_value;
```

UPDATE stosujemy zawsze z klauzulą **WHERE** (inaczej wszystkie dane z tabeli zostaną zmienione).

Zmiana wartości danych

```
UPDATE users SET user_name="Grzesiek" WHERE user_id=2;
SELECT * FROM users;
```

```
+---------+-----------+
| user_id | user_name |
+---------+-----------+
|       1 | Wojtek    |
|       2 | Grzesiek  |
|         ...         |
+---------+-----------+
4 rows in set (0.00 sec)
```

Zmiana wartości danych – preparedStatement

Schemat zapytania w Javie:

```
String sql = "UPDATE table_name SET col_name_1='value1' where id = ?";
PreparedStatement prepStm = conn.prepareStatement(sql);
prepStm.setInt(1, id);
prepStm.executeUpdate();
```

Przykład

```
int idUser = 3;
String sql = "UPDATE users SET user_name='coderslab' where user_id = ?";
PreparedStatement prepStm = conn.prepareStatement(sql);
prepStm.setInt(1, idUser);
prepStm.executeUpdate();
```

Usuwanie danych z tabeli

Dane z tabeli usuwamy za pomocą zapytania **DELETE**:

```
DELETE FROM table_name WHERE some_column=some_value;
```

DELETE stosujemy z klauzulą **WHERE** (inaczej wszystkie dane z tabeli zostaną usunięte).

Usuwanie danych z tabeli

Przykład w SQL

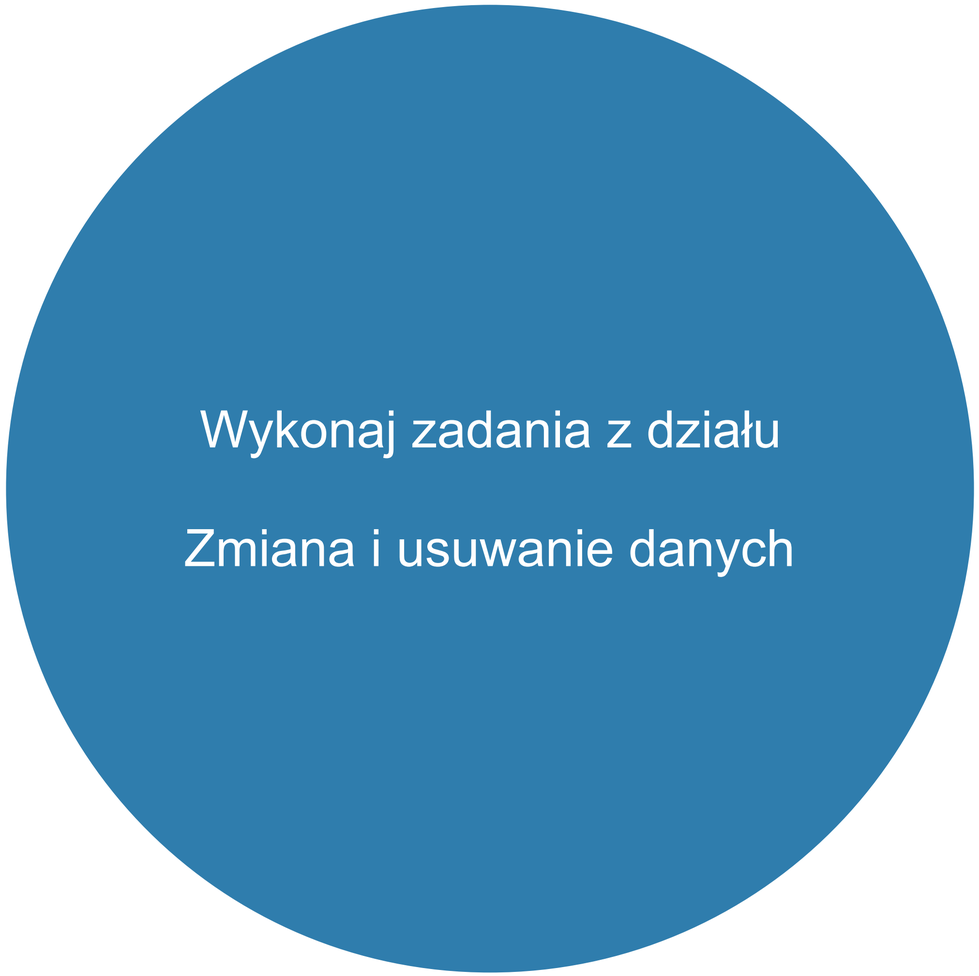
```
DELETE FROM users WHERE user_name = "Grzesiek";
SELECT * FROM users;
```

```
+---------+-----------+
| user_id | user_name |
+---------+-----------+
|       1 | Wojtek    |
|       3 | Paweł     |
+---------+-----------+
3 rows in set (0.00 sec)
```

Przykład w Java

```
int idUser = 2;
String sql = "DELETE FROM users WHERE user_id = ?";
PreparedStatement prepStm = conn.prepareStatement(sql);
prepStm.setInt(1, idUser);
prepStm.executeUpdate();
```

Zadania



Modyfikacja tabel

Modyfikacja tabeli

Wygląd tabeli (liczba kolumn, dane w nich przetrzymywane) możemy zmienić za pomocą polecenia **`ALTER TABLE`**:

- dodanie nowej kolumny:

```
ALTER TABLE table_name ADD column_name datatype;
```

- usunięcie kolumny:

```
ALTER TABLE table_name DROP COLUMN column_name;
```

- zmiana danych trzymanych w kolumnie:

```
ALTER TABLE table_name MODIFY COLUMN column_name new_datatype;
```

Modyfikacja tabeli

Tabela przed modyfikacjami wygląda następująco:

```
mysql> DESCRIBE users;
```

```
+---------+------------+-----+-----+---------+----------------+
|Field    |Type        |Null | Key | Default | Extra          |
+---------+------------+-----+-----+---------+----------------+
|user_id  |int(11)     | NO  | PRI | NULL    | auto_increment |
|user_name|varchar(255)| YES |     | NULL    |                |
+---------+------------+-----+-----+---------+----------------+
2 rows in set (0.00 sec)
```

Modyfikacja tabeli

```
mysql> ALTER TABLE users MODIFY COLUMN user_name varchar(30);
mysql> ALTER TABLE users ADD email varchar(30);

mysql> DESCRIBE users;
```

```
+---------+------------+-----+-----+---------+----------------+
|Field    |Type        | Null| Key | Default | Extra          |
+---------+------------+-----+-----+---------+----------------+
|user_id  |int(11)     | NO  | PRI | NULL    | auto_increment |
|user_name|varchar(30) | YES |     | NULL    |                |
|email    |varchar(30) | YES |     | NULL    |                |
+---------+------------+-----+-----+---------+----------------+
3 rows in set (0.00 sec)
```

Usunięcie tabeli

Tabelę możemy usunąć z naszej bazy danych przez użycie zapytania **DROP TABLE**:

```
DROP TABLE table_name;
```

Całą bazę danych możemy zniszczyć, używając zapytania **DROP DATABASE**:

```
DROP DATABASE db_name;
```

Zadania

Relacje między tabelami

Relacja jeden do jednego

Relacja, w której jeden element z danej tabeli może być połączony tylko z jednym elementem z innej tabeli.

Klient może mieć tylko jeden adres. Adres musi mieć jednego klienta.

Relację jeden do jednego tworzymy przez uzależnienie klucza głównego jednego obiektu od klucza głównego drugiego obiektu. Klucz główny nie może istnieć bez tej relacji.

W naszym przypadku będzie to przypisywanie klucza głównego klienta do klucza głównego adresu.

Relacja jeden do jednego

```
CREATE TABLE customers (
  customer_id int NOT NULL AUTO_INCREMENT,
  name varchar(255) NOT NULL,
  PRIMARY KEY(customer_id)
);

CREATE TABLE addresses (
  customer_id int NOT NULL,
  street varchar(255),
  PRIMARY KEY(customer_id),
  FOREIGN KEY(customer_id) REFERENCES customers(customer_id)
  ON DELETE CASCADE
);
```

Relacja jeden do jednego

```
CREATE TABLE customers (
  customer_id int NOT NULL AUTO_INCREMENT,
  name varchar(255) NOT NULL,
  PRIMARY KEY(customer_id)
);

CREATE TABLE addresses (
  customer_id int NOT NULL,
  street varchar(255),
  PRIMARY KEY(customer_id),
  FOREIGN KEY(customer_id) REFERENCES customers(customer_id)
  ON DELETE CASCADE
);
```

ON DELETE CASCADE jest opcjonalnym parametrem. Jego działanie jest opisane na kolejnych slajdach.

ON DELETE CASCADE

Podczas budowania relacji możemy dodać jeszcze opcję **`ON DELETE CASCADE`**.

Dzięki tej opcji zostaną usunięte nie tylko rekordy w tabeli głównej, na którą wskazują klucze obce, lecz także automatycznie znikną wszystkie rekordy w innych tabelach, które są połączone jakąś relacją.

Na przykład, jeżeli usuwamy użytkownika, to chcemy, aby wszystkie jego adresy w systemie zostały usunięte razem z nim.

Jeżeli nie dodamy tej opcji, to SQL nie pozwoli nam usunąć rekordu, dopóki są z nim powiązane jakiekolwiek wpisy w innych tabelach.

Na przykład SQL nie pozwoli nam usunąć użytkownika, dopóki w tabeli z adresami znajdują się rekordy przypisane do niego.

FOREIGN KEY

Atrybut **`FOREIGN KEY`** (klucz obcy), dopisany do jakiejś kolumny oznacza, że ta kolumna wskazuje na klucz główny innej tabeli.

Przyspiesza on pracę naszej bazy danych i powoduje zabezpieczenie przed wprowadzeniem niepoprawnych danych (np. nie pozwoli wpisać klucza, który nie występuje w drugiej tabeli).

FOREIGN KEY – dodawanie elementu

```
SELECT * FROM customers;
```

```
+-------------+-------+
| customer_id | name  |
+-------------+-------+
|           1 | Janusz|
|           3 | Wojtek|
+-------------+-------+
```

```
INSERT INTO addresses(customer_id, street) VALUES (5, "xxx");
```

Jeżeli w drugiej tabeli nie ma klucza głównego, do którego chcemy się odnieść przez klucz zewnętrzny, to SQL zwróci nam błąd:

```
ERROR 1452 (23000): Cannot add or update a child row:
a foreign key constraint fails ('db_name'.'addresses',
CONSTRAINT 'addresses_ibfk_1' FOREIGN KEY ('customer_id')
REFERENCES 'customers' ('customer_id'))
```

FOREIGN KEY – dodawanie elementu

```
SELECT * FROM customers;
```

```
+-------------+-------+
| customer_id | name  |
+-------------+-------+
|           1 | Janusz|
|           3 | Wojtek|
+-------------+-------+
```

```
INSERT INTO addresses(customer_id, street) VALUES (5, "xxx");
```

Jeżeli w drugiej tabeli nie ma klucza głównego, do którego chcemy się odnieść przez klucz zewnętrzny, to SQL zwróci nam błąd:

```
ERROR 1452 (23000): Cannot add or update a child row:
a foreign key constraint fails ('db_name'.'addresses',
CONSTRAINT 'addresses_ibfk_1' FOREIGN KEY ('customer_id')
REFERENCES 'customers' ('customer_id'))
```

W naszym wypadku w tabeli **customers** nie ma rekordu z **customer_id = 5**.

FOREIGN KEY – usuwanie elementu

```sql
SELECT * FROM addresses;
```

```
+----------+-----------+-------------+
|address_id|customer_id|street       |
+----------+-----------+-------------+
|         1|          1|Ulica Janusza|
|         2|          3|Ulica Wojtka |
+----------+-----------+-------------+
```

```sql
DELETE FROM customers WHERE customer_id = 3;
```

FOREIGN KEY – usuwanie elementu

```sql
SELECT * FROM addresses;
```

```
+----------+-----------+-------------+
|address_id|customer_id|street       |
+----------+-----------+-------------+
|         1|          1|Ulica Janusza|
|         2|          3|Ulica Wojtka |
+----------+-----------+-------------+
```

```sql
DELETE FROM customers WHERE customer_id = 3;
```

Wywołanie takiego zapytania spowoduje automatyczne usunięcie powiązanego rekordu (lub rekordów) z tabeli **addresses**. Rekordy są powiązane kluczem obcym oraz **ON DELETE CASCADE**.
W naszej tabeli zostanie usunięty użytkownik o id = 3 oraz jego adres z tabeli addresses.

Łączenie tabel

Łączenie tabel

Jeżeli chcemy pobrać wyniki jednocześnie z dwóch (lub większej liczby) tabel, możemy to uzyskać dzięki użyciu wyrażenia kluczowego **`JOIN... ON...`**

Oto trzy możliwości łączenia tabel:

- **`INNER`**
- **`LEFT`**
- **`RIGHT`**

Przykład

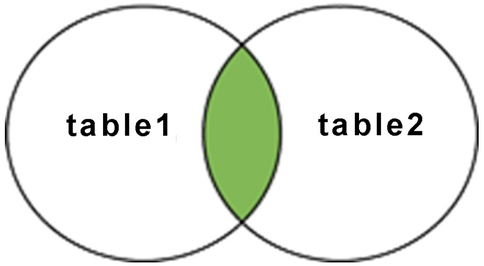
```
SELECT column_name(s)
FROM table1
JOIN table2
ON table1.column_name=table2.column_name;
```

INNER JOIN

`INNER JOIN` jest podstawowym typem łączenia tabel. Jeżeli wpiszemy tylko **`JOIN`**, to domyślnie wywołuje się właśnie **`INNER JOIN`**.

Jako wynik daje on połączone rekordy z obu tabel, które mają element wspólny wskazany w zapytaniu po komendzie **`ON`**.

Rekordy bez elementu wspólnego nie zostaną zwrócone.



INNER JOIN

Załóżmy, że mamy tabele z następującymi danymi:

```
SELECT * FROM customers;
```

```
+-------------+-------+
| customer_id | name  |
+-------------+-------+
|           1 | Jacek |
|           3 | Paweł |
|           4 | Kuba  |
+-------------+-------+
```

```
SELECT * FROM addresses;
```

```
+----------+-----------+-----------+
|address_id|customer_id|street     |
+----------+-----------+-----------+
|         1|          1|Adres Jacka|
|         2|          3|Adres Pawła|
|         3|         10|Adres Joli |
+----------+-----------+-----------+
```

INNER JOIN

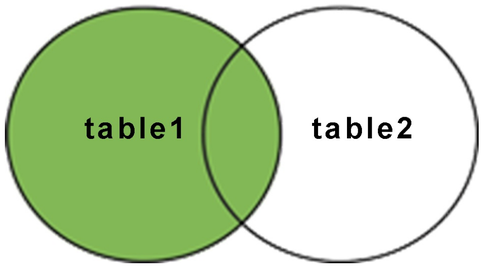
```
SELECT * FROM customers JOIN addresses ON
customers.customer_id=addresses.customer_id;
```

```
+-------------+-------+-------------+-------------+------------+
| customer_id | name  | address_id | customer_id |street      |
+-------------+-------+-------------+-------------+------------+
|           1 | Jacek |           1 |           1 |Adres Jacka |
|           3 | Paweł |           2 |           3 |Adres Pawła |
+-------------+-------+-------------+-------------+------------+
```

Zostały zwrócone tylko te rekordy, dla których obie tabele mają wspólną (o takiej samej wartości) kolumnę **`customer_id`**.

LEFT JOIN

`LEFT JOIN` zwraca jako wynik wszystkie wiersze z lewej tabeli, a z prawej tabeli zostaną dołączone tylko z rzędów spełniających warunek.



LEFT JOIN

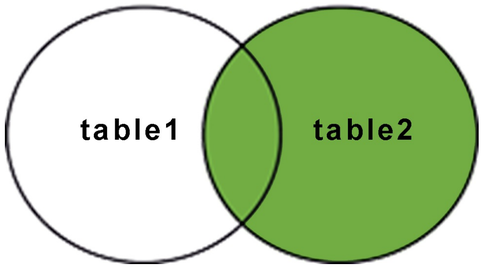
```
SELECT * FROM customers LEFT JOIN addresses ON
customers.customer_id=addresses.customer_id;
```

```
+-------------+-------+-------------+-------------+------------+
| customer_id | name  | address_id | customer_id |street      |
+-------------+-------+-------------+-------------+------------+
|           1 | Jacek |           1 |           1 |Adres Jacka |
|           3 | Pawel |           2 |           3 |Adres Pawła |
|           4 | Kuba  |        NULL |        NULL |NULL        |
+-------------+-------+-------------+-------------+------------+
```

Zwrócone zostały wszystkie rekordy z lewej tabeli (**`customers`**) oraz odpowiadające im rekordy z prawej tabeli, które – jeśli nie istnieją – zostają zwrócone jako **`NULL`**.

RIGHT JOIN

`RIGHT JOIN` zwraca jako wynik wszystkie wiersze z prawej tabeli, a z prawej tabeli zostaną dołączone tylko z rzędów spełniających warunek.



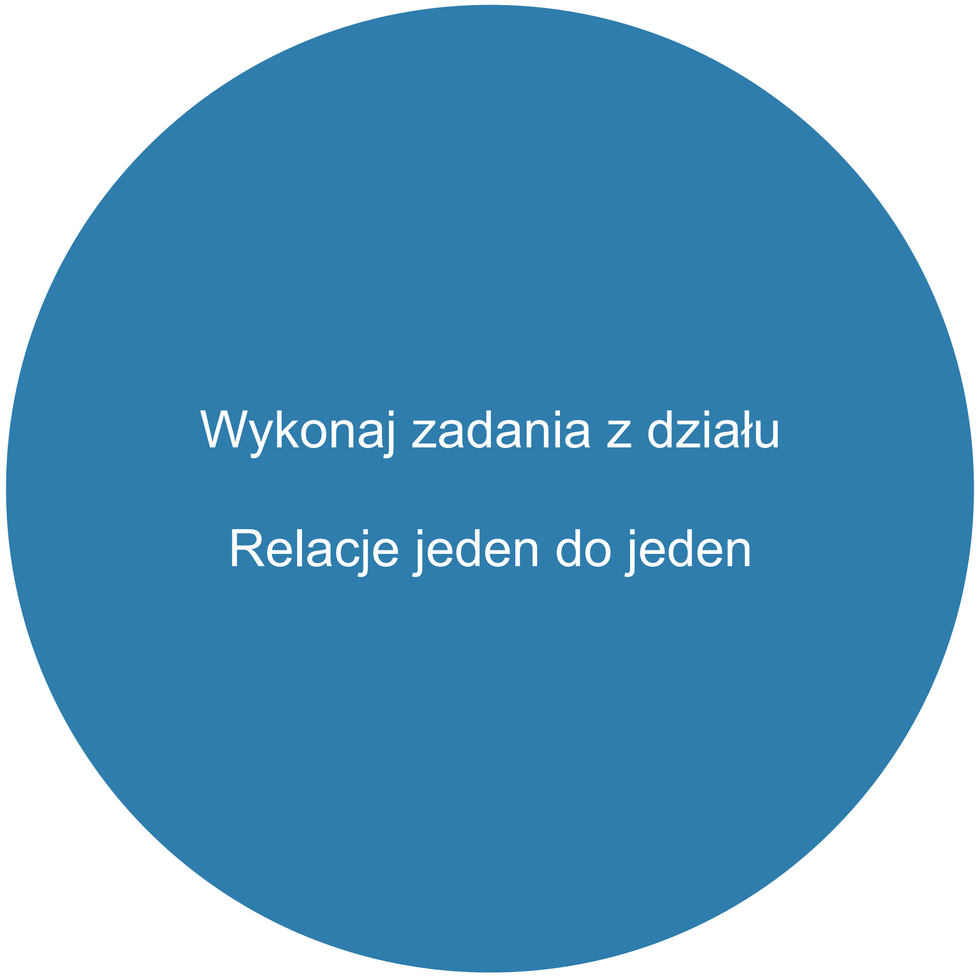
RIGHT JOIN

```
SELECT * FROM customers RIGHT JOIN addresses ON
customers.customer_id=addresses.customer_id;
```

```
+-------------+-------+------------+-------------+------------+
| customer_id | name  | address_id | customer_id |street      |
+-------------+-------+------------+-------------+------------+
|           1 | Jacek |          1 |           1 |Adres Jacka |
|           3 | Paweł |          2 |           3 |Adres Pawła |
|        NULL | NULL  |          3 |          10 |Adres Joli  |
+-------------+-------+------------+-------------+------------+
```

Zwrócone rekordy to wszystkie z prawej tabeli (**`addresses`**) oraz odpowiadające im rekordy z lewej tabeli, które – jeśli nie istnieją – zostają zwrócone jako NULL.

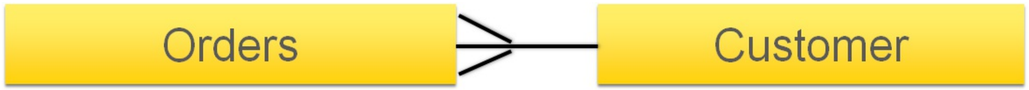
Zadania



Relacja jeden do wielu

Relacja, w której jeden element z danej tabeli, może być połączony z wieloma elementami z innej tabeli.

Klient może mieć wiele zamówień. Zamówienie musi mieć tylko jednego klienta.



Relacja jeden do wielu

Relację jeden do wielu tworzymy przez dodanie dodatkowej kolumny, w której trzymamy klucz główny obiektu z drugiej tabeli.

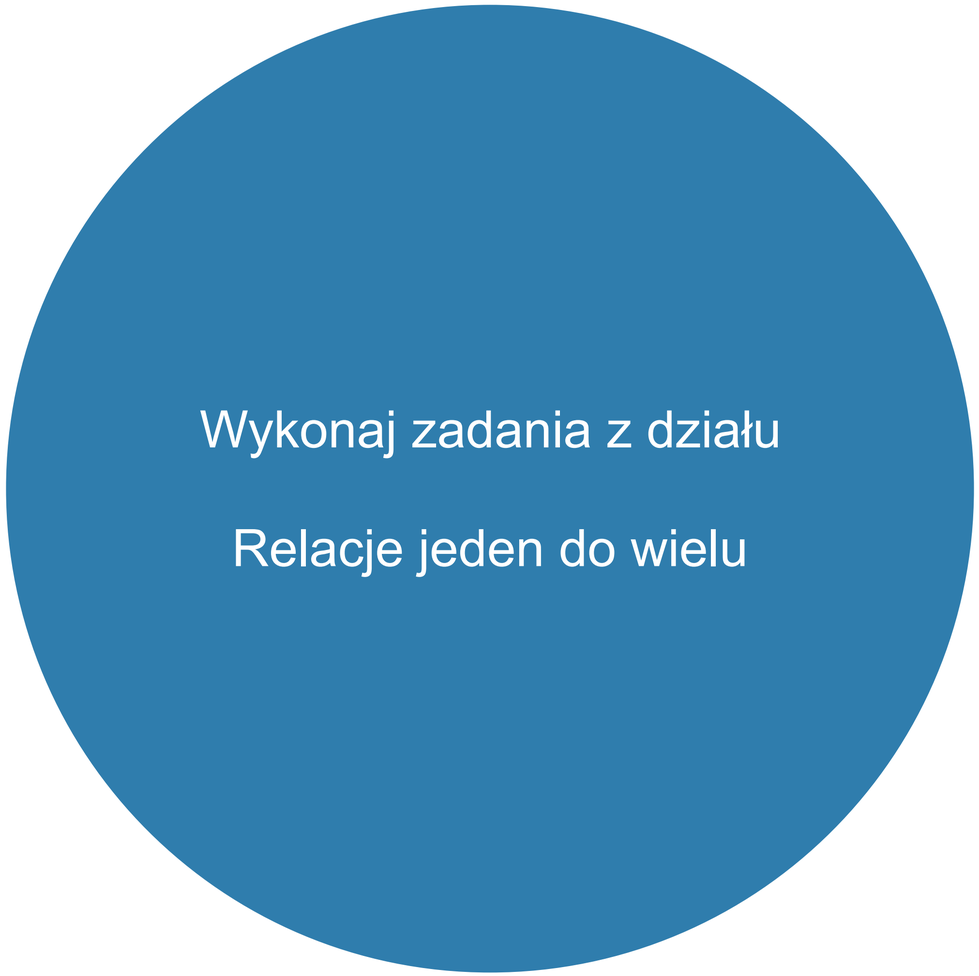
```
CREATE TABLE orders(
  order_id int NOT NULL AUTO_INCREMENT,
  customer_id int NOT NULL,
  order_details varchar(255),
  PRIMARY KEY(order_id),
  FOREIGN KEY(customer_id)
  REFERENCES customers(customer_id)
);
```

Relacja jeden do wielu

```sql
INSERT INTO orders(customer_id, order_details)
VALUES (3, "Order1"), (3, "Order2"), (1, "Order3");
SELECT * FROM customers JOIN orders
ON customers.customer_id=orders.customer_id
WHERE customers.customer_id=3;
```

```
+-------------+--------+----------+-------------+--------------+
| customer_id | name   | order_id | customer_id|order_details |
+-------------+--------+----------+-------------+--------------+
|           3 | Wojtek |        1 |           3 | Order1       |
|           3 | Wojtek |        2 |           3 | Order2       |
|           1 | Janusz |        3 |           1 | Order3       |
+-------------+--------+----------+-------------+--------------+
```

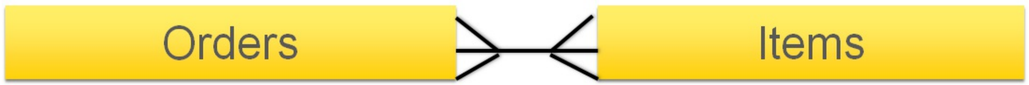
Zadania



Relacja wiele do wielu

Relacja, w której wiele elementów z jednej tabeli może być połączonych z wieloma elementami z innej tabeli.

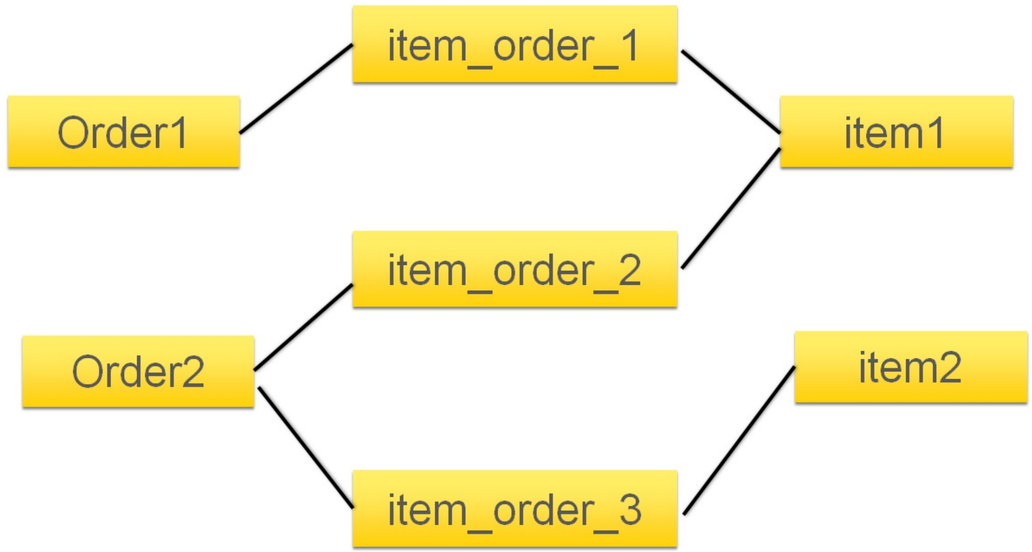
Na przykład zamówienie ma w sobie wiele przedmiotów, a przedmiot może być na wielu zamówieniach.



Relacja wiele do wielu

W SQL nie da się bezpośrednio powiązać tabel relacją wiele do wielu, należy stworzyć dodatkową tabelę, która opisuje nam taką relację. Tabela ta posiada dwie relacje jeden do wielu.

Może ona trzymać więcej informacji niż tylko klucze główne swoich relacji.



Relacja wiele do wielu

Tworzymy tabelę **`items`**:

```
CREATE TABLE items(
  item_id int NOT NULL AUTO_INCREMENT,
  name varchar(255),
  PRIMARY KEY(item_id)
);
INSERT INTO items(name) VALUES ("item1"), ("item2"), ("item3");
```

Tabelę **`orders`** utworzyliśmy podczas omawiania relacji jeden do wielu.

Relacja wiele do wielu

Tworzymy tabelę pomocniczą **`items_orders`**:

```
CREATE TABLE items_orders(
  id int AUTO_INCREMENT,
  item_id int NOT NULL,
  order_id int NOT NULL,
  PRIMARY KEY(id),
  FOREIGN KEY(order_id) REFERENCES orders(order_id),
  FOREIGN KEY(item_id) REFERENCES items(item_id)
);
```

Relacja wiele do wielu

Tworzymy tabelę pomocniczą **`items_orders`**:

```
CREATE TABLE items_orders(
  id int AUTO_INCREMENT,
  item_id int NOT NULL,
  order_id int NOT NULL,
  PRIMARY KEY(id),
  FOREIGN KEY(order_id) REFERENCES orders(order_id),
  FOREIGN KEY(item_id) REFERENCES items(item_id)
);
```

Nazwę tabeli pomocniczej dla relacji tworzymy zazwyczaj przez połączenie nazw dwóch tabel, które chcemy połączyć relacją **wiele do wielu**.

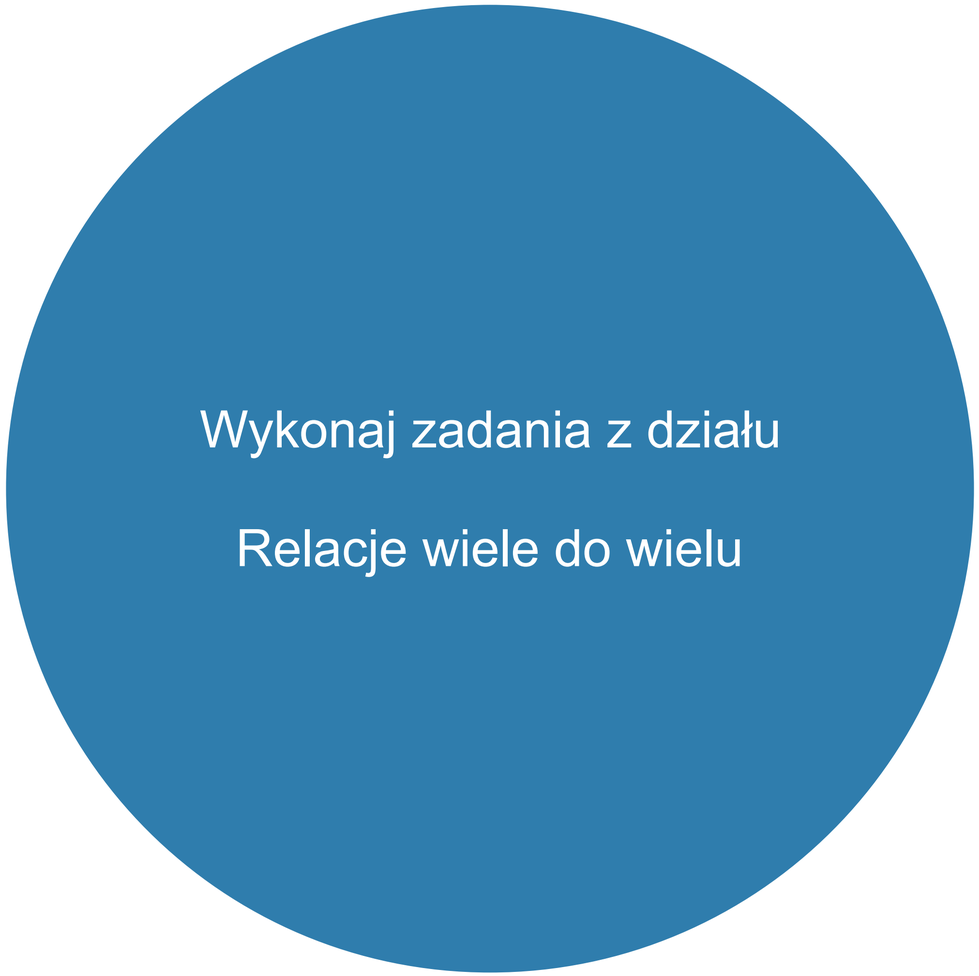
Relacja wiele do wielu

```
INSERT INTO items_orders(order_id, item_id) VALUES (1,1), (2,1), (2,2);

SELECT * FROM orders
JOIN items_orders ON orders.order_id=items_orders.order_id
JOIN items ON items.item_id=items_orders.item_id;
```

```
+--------+-----------+-------------+-------+--------+-------+--------+
|order_id|customer_id|order_details|item_id|order_id|item_id| name   |
+--------+-----------+-------------+-------+--------+-------+--------+
|      1 |         3 | Order1      |     1 |      1 |     1 | item1  |
|      2 |         3 | Order2      |     1 |      2 |     1 | item1  |
|      2 |         3 | Order2      |     2 |      2 |     2 | item2  |
+--------+-----------+-------------+-------+--------+-------+--------+
```

Zadania



Łączenie tabel

Łączenie tabel

Wyniki z dwóch (lub więcej) tabel naraz możemy uzyskać dzięki użyciu wyrażenia kluczowego **JOIN... ON...**

W MySQL wyróżniamy trzy podstawowe możliwości łączenia tabel:

- **INNER**
- **LEFT**
- **RIGHT**

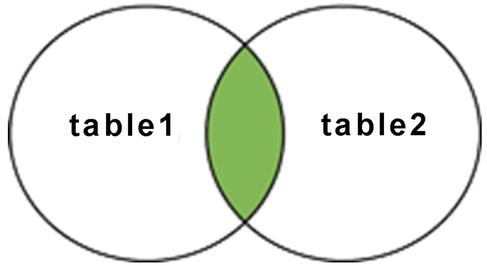
```
SELECT column_name(s)
FROM table1
JOIN table2
ON table1.column_name=table2.column_name;
```

INNER JOIN

`INNER JOIN` (lub zwykłe **`JOIN`**) jest podstawowym typem łączenia tabel.

Jako wynik daje on połączone rekordy z obu tabel, które mają element wspólny wskazany w zapytaniu po komendzie **`ON`**.

Rekordy bez elementu wspólnego nie zostaną zwrócone.



Join

Załóżmy, że mamy tabele z następującymi danymi:

```
SELECT * FROM customers;
```

```
+-------------+-------+
| customer_id | name  |
+-------------+-------+
|           1 | Jacek |
|           3 | Paweł |
|           4 | Kuba  |
+-------------+-------+
```

```
SELECT * FROM addresses;
```

```
+----------+-----------+------------+
|address_id|customer_id|street      |
+----------+-----------+------------+
|         1|          1|Adres Jacka|
|         2|          3|Adres Pawła|
|         3|         10|Zły adres  |
+----------+-----------+------------+
```

INNER JOIN

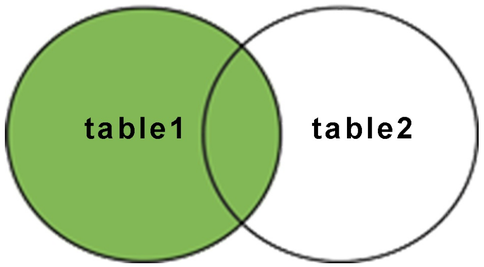
```
SELECT * FROM customers JOIN addresses ON
customers.customer_id=addresses.customer_id;
```

```
+-------------+-------+-------------+-------------+------------+
| customer_id | name  | address_id | customer_id |street      |
+-------------+-------+-------------+-------------+------------+
|           1 | Jacek |           1 |           1 |Adres Jacka |
|           3 | Paweł |           2 |           3 |Adres Pawła |
+-------------+-------+-------------+-------------+------------+
```

Zwrócone rekordy to te, dla których obie tabele mają wspólną (o takiej samej wartości) kolumnę **`customer_id`**.

LEFT JOIN

`LEFT JOIN` zwraca jako wynik wszystkie wiersze z lewej tabeli. Dane z prawej tabeli zostaną dołączone tylko w rzędach spełniających warunek.



LEFT JOIN

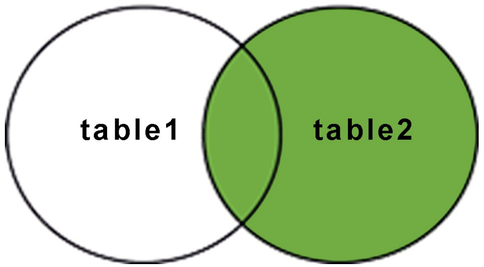
```
SELECT * FROM customers LEFT JOIN addresses ON
customers.customer_id=addresses.customer_id;
```

```
+-------------+-------+-------------+-------------+------------+
| customer_id | name  | address_id | customer_id |street      |
+-------------+-------+-------------+-------------+------------+
|           1 | Jacek |           1 |           1 |Adres Jacka |
|           3 | Pawel |           2 |           3 |Adres Pawła |
|           4 | Kuba  |        NULL |        NULL |NULL        |
+-------------+-------+-------------+-------------+------------+
```

Zwrócone rekordy to wszystkie z lewej tabeli (**`customers`**) oraz odpowiadające im rekordy z prawej tabeli, które jeśli nie istnieją, to zostają zwrócone jako **`NULL`**.

RIGHT JOIN

`RIGHT JOIN` zwraca jako wynik wszystkie wiersze z prawej tabeli. Dane z lewej zostaną dołączone tylko w rzędach spełniających warunek.



RIGHT JOIN

```
SELECT * FROM customers RIGHT JOIN addresses ON
customers.customer_id=addresses.customer_id;
```

```
+-------------+-------+------------+-------------+------------+
| customer_id | name  | address_id | customer_id |street      |
+-------------+-------+------------+-------------+------------+
|           1 | Jacek |          1 |           1 |Adres Jacka |
|           3 | Paweł |          2 |           3 |Adres Pawła |
|        NULL | NULL  |          3 |          10 |Adres błędny|
+-------------+-------+------------+-------------+------------+
```

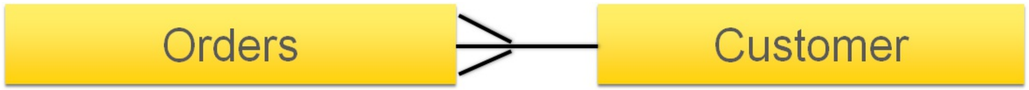
Zwrócone rekordy to wszystkie z prawej tabeli (**addresses**) oraz odpowiadające im rekordy z lewej tabeli, które jeśli nie istnieją, to zostają zwrócone jako NULL.

Zadania

Relacja jeden do wielu

Relacja, w której jeden element z danej tabeli, może być połączony z wieloma elementami z innej tabeli.

Klient może mieć wiele zamówień. Zamówienie musi mieć tylko jednego klienta.



Relacja jeden do wielu

Relację jeden do wielu tworzymy przez dodanie dodatkowej kolumny, w której trzymamy klucz główny obiektu z drugiej tabeli.

```
CREATE TABLE orders(
  order_id int NOT NULL AUTO_INCREMENT,
  customer_id int NOT NULL,
  order_details varchar(255),
  PRIMARY KEY(order_id),
  FOREIGN KEY(customer_id)
  REFERENCES customers(customer_id)
);
```

Relacja jeden do wielu

```
INSERT INTO orders(customer_id, order_details)
VALUES (3, "Zamówienie1"), (3, "Zamówienie2"), (1, "Zamówienie3");
SELECT * FROM customers JOIN orders
ON customers.customer_id=orders.customer_id
WHERE customers.customer_id=3;
```

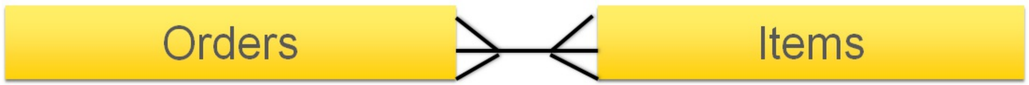
```
+-------------+--------+----------+-------------+--------------+
| customer_id | name   | order_id | customer_id|order_details |
+-------------+--------+----------+-------------+--------------+
|           3 | Wojtek |        1 |           3|Zamówienie1   |
|           3 | Wojtek |        2 |           3|Zamówienie2   |
+-------------+--------+----------+-------------+--------------+
```

Zadania

Relacja wiele do wielu

Relacja, w której wiele elementów z danej tabeli może być połączonych z wieloma elementami z innej tabeli.

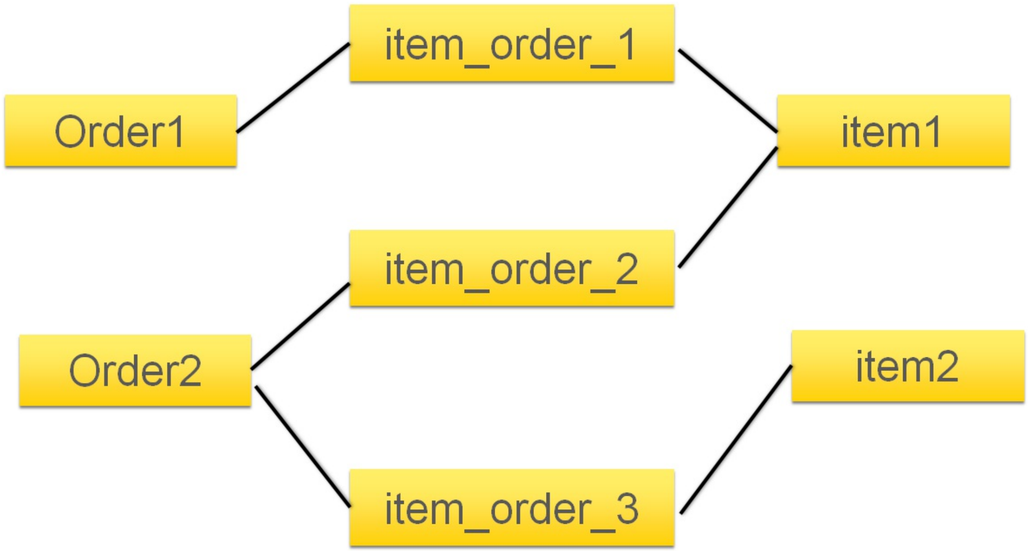
Na przykład zamówienie ma w sobie wiele przedmiotów, przedmiot może być w wielu zamówieniach.



Relacja wiele do wielu

W SQL nie ma relacji wiele do wielu. Jak sobie poradzić? Należy dodać dodatkową – opisuje nam taką relację – tabelę mającą dwie relacje jeden do wielu.

Tabela ta może trzymać więcej informacji niż tylko klucze główne swoich relacji.



Relacja wiele do wielu

```
CREATE TABLE items(
  item_id int NOT NULL AUTO_INCREMENT,
  description varchar(255),
  PRIMARY KEY(item_id)
);
INSERT INTO items(description) VALUES ("Item 1"), ("Item 2"), ("Items 3");
```

Relacja wiele do wielu

```
CREATE TABLE items_orders(
  id int AUTO_INCREMENT,
  item_id int NOT NULL,
  order_id int NOT NULL,
  PRIMARY KEY(id),
  FOREIGN KEY(order_id) REFERENCES orders(order_id),
  FOREIGN KEY(item_id) REFERENCES items(item_id)
);
```

Relacja wiele do wielu

```sql
CREATE TABLE items_orders(
  id int AUTO_INCREMENT,
  item_id int NOT NULL,
  order_id int NOT NULL,
  PRIMARY KEY(id),
  FOREIGN KEY(order_id) REFERENCES orders(order_id),
  FOREIGN KEY(item_id) REFERENCES items(item_id)
);
```

Nazwę tabeli pomocniczej dla relacji tworzymy zazwyczaj przez połączenie nazw dwóch tabel, które chcemy połączyć relacją **wiele do wielu**.

Relacja wiele do wielu

```
INSERT INTO items_orders(order_id, item_id) VALUES (1,1), (2,1), (2,2);
SELECT * FROM orders
JOIN items_orders ON orders.order_id=items_orders.order_id
JOIN items ON items.item_id=items_orders.item_id;
```

```
+--------+-----------+-------------+-------+--------+-------+-----------+
|order_id|customer_id|order_details|item_id|order_id|item_id|description|
+--------+-----------+-------------+-------+--------+-------+-----------+
|      1 |         3 |Zamówienie1  |     1 |      1 |     1 | Item 1    |
|      2 |         3 |Zamówienie2  |     1 |      2 |     1 | Item 1    |
|      2 |         3 |Zamówienie2  |     2 |      2 |     2 | Item 2    |
+--------+-----------+-------------+-------+--------+-------+-----------+
```

Zadania

Funkcje wbudowane w SQL

Język SQL implementuje również wiele funkcji ułatwiających pracę na napisach, liczbach i datach.

Pełną listę tych funkcji (wraz z opisami) znajdziesz tutaj:
https://www.w3schools.com/sql/sql_ref_mysql.asp

Indeksy

Indeksy są specjalnymi tabelami przeszukań przyspieszającymi przeszukiwanie tabeli względem jednej z kolumn.

Powinniśmy ich używać, gdy wiele klauzul **WHERE** zależy właśnie od tej kolumny.

Zbyt duża liczba indeksów może spowolnić działanie bazy danych – dlatego trzeba na nie uważać i dodawać je po dogłębnej analizie bazy danych.

Działanie indeksów najbardziej jest widoczne w prędkości wykonywania zapytań, jeśli indeks dodany jest do kolumny z typem liczbowym.

```
CREATE INDEX index_name
ON table_name (column_name);
```

Transakcje

SQL pozwala też na transakcje (**transactions**). Jest to grupa zapytań SQL wywoływana w całości na bazie danych. Jeżeli któreś z tych zapytań nie powiedzie się, baza wraca do stanu sprzed takiej transakcji.

Jest to zaawansowany mechanizm, który występuje np. w systemach bankowych.

Zapytanie w transakcjach oznaczają, iż wykonane muszą zostać prawidłowo wszystkie zapytania lub żadne.

Więcej o transakcjach: http://www.tutorialspoint.com/sql/sql-transactions.htm

Transakcje

Przykładem transakcji (w sensie SQL nie w sensie dosłownym) może być przelew środków z konta jednego użytkownika na konto innego. Obie operacje muszą się wykonać.

```
try {
    conn.setAutoCommit(false);
    Statement stat = conn.createStatement();
    stat.executeUpdate("UPDATE users SET balance=balance-1000 WHERE user_id=5;");
    stat.executeUpdate("UPDATE users SET balance=balance+1000 WHERE user_id=6;");
    conn.commit();
} catch (SQLException e) {
    conn.rollback();
    e.printStackTrace();
}
```

Transakcje

Przykładem transakcji (w sensie SQL nie w sensie dosłownym) może być przelew środków z konta jednego użytkownika na konto innego. Obie operacje muszą się wykonać.

```
try {
    conn.setAutoCommit(false);
    Statement stat = conn.createStatement();
    stat.executeUpdate("UPDATE users SET balance=balance-1000 WHERE user_id=5;");
    stat.executeUpdate("UPDATE users SET balance=balance+1000 WHERE user_id=6;");
    conn.commit();
} catch (SQLException e) {
    conn.rollback();
    e.printStackTrace();
}
```

Wyłącza tryb automatycznego zatwierdzania – rozpoczęcie transakcji.

Transakcje

Przykładem transakcji (w sensie SQL nie w sensie dosłownym) może być przelew środków z konta jednego użytkownika na konto innego. Obie operacje muszą się wykonać.

```
try {
    conn.setAutoCommit(false);
    Statement stat = conn.createStatement();
    stat.executeUpdate("UPDATE users SET balance=balance-1000 WHERE user_id=5;");
    stat.executeUpdate("UPDATE users SET balance=balance+1000 WHERE user_id=6;");
    conn.commit();
} catch (SQLException e) {
    conn.rollback();
    e.printStackTrace();
}
```

Zatwierdzenie transakcji.

Transakcje

Przykładem transakcji (w sensie SQL nie w sensie dosłownym) może być przelew środków z konta jednego użytkownika na konto innego. Obie operacje muszą się wykonać.

```
try {
    conn.setAutoCommit(false);
    Statement stat = conn.createStatement();
    stat.executeUpdate("UPDATE users SET balance=balance-1000 WHERE user_id=5;");
    stat.executeUpdate("UPDATE users SET balance=balance+1000 WHERE user_id=6;");
    conn.commit();
} catch (SQLException e) {
    conn.rollback();
    e.printStackTrace();
}
```

W przypadku jakiegokolwiek błędu cofamy transakcję.

Wyzwalacze (triggers)

W SQL jest możliwość stworzenia wyzwalaczy (**triggerów**).

Są to funkcje, które zostaną automatycznie uruchomione, jeżeli zajdzie określona przez nas sytuacja (zazwyczaj **`DELETE`**, **`INSERT`**, **`UPDATE`**).

Więcej o wyzwalaczach:

- http://www.tutorialspoint.com/plsql/plsql_triggers.htm
- http://www.sqlteam.com/article/an-introduction-to-triggers-part-i

Zaawansowany SQL

Funkcje wbudowane w SQL

Język SQL implementuje również wiele funkcji ułatwiających pracę na napisach, liczbach i datach.

Pełną listę tych funkcji (wraz z opisami) znajdziesz tutaj:

https://www.w3schools.com/sql/sql_ref_mysql.asp

Indeksy

Indeksy są specjalnymi tabelami przeszukań przyspieszającymi przeszukiwanie tabeli względem jednej z kolumn.

Powinniśmy ich używać, gdy wiele klauzul **WHERE** zależy właśnie od tej kolumny.

Zbyt duża liczba indeksów może spowolnić działanie bazy danych – dlatego trzeba na nie uważać i dodawać je po dogłębnej analizie bazy danych.

Działanie indeksów najbardziej jest widoczne w prędkości wykonywania zapytań, jeśli indeks dodany jest do kolumny z typem liczbowym.

```
CREATE INDEX index_name
ON table_name (column_name);
```

Transakcje

SQL pozwala też na stosowanie transakcji (**transactions**). Jest to zbiór zapytań SQL. Jeżeli któreś z tych zapytań nie powiedzie się, baza wraca do stanu sprzed takiej transakcji.

Jest to zaawansowany mechanizm, który występuje np. w systemach bankowych.

Zapytania umieszczone w transakcjach dają pewność, że albo wykonają się prawidłowo wszystkie zapytania albo żadne.

Więcej o transakcjach: http://www.tutorialspoint.com/sql/sql-transactions.htm

Transakcje

Przykładem transakcji (w sensie SQL, a nie w sensie dosłownym) może być przelew środków z konta jednego użytkownika na konto innego. Obie operacje muszą się wykonać.

```
try {
    conn.setAutoCommit(false);
    Statement stat = conn.createStatement();
    stat.executeUpdate("UPDATE users SET balance=balance-1000 WHERE user_id=5;");
    stat.executeUpdate("UPDATE users SET balance=balance+1000 WHERE user_id=6;");
    conn.commit();
} catch (SQLException e) {
    conn.rollback();
    e.printStackTrace();
}
```

Transakcje

Przykładem transakcji (w sensie SQL, a nie w sensie dosłownym) może być przelew środków z konta jednego użytkownika na konto innego. Obie operacje muszą się wykonać.

```
try {
    conn.setAutoCommit(false);
    Statement stat = conn.createStatement();
    stat.executeUpdate("UPDATE users SET balance=balance-1000 WHERE user_id=5;");
    stat.executeUpdate("UPDATE users SET balance=balance+1000 WHERE user_id=6;");
    conn.commit();
} catch (SQLException e) {
    conn.rollback();
    e.printStackTrace();
}
```

Wyłącza tryb automatycznego zatwierdzania – rozpoczęcie transakcji.

Transakcje

Przykładem transakcji (w sensie SQL, a nie w sensie dosłownym) może być przelew środków z konta jednego użytkownika na konto innego. Obie operacje muszą się wykonać.

```
try {
    conn.setAutoCommit(false);
    Statement stat = conn.createStatement();
    stat.executeUpdate("UPDATE users SET balance=balance-1000 WHERE user_id=5;");
    stat.executeUpdate("UPDATE users SET balance=balance+1000 WHERE user_id=6;");
    conn.commit();
} catch (SQLException e) {
    conn.rollback();
    e.printStackTrace();
}
```

Zatwierdzenie transakcji.

Transakcje

Przykładem transakcji (w sensie SQL, a nie w sensie dosłownym) może być przelew środków z konta jednego użytkownika na konto innego. Obie operacje muszą się wykonać.

```
try {
    conn.setAutoCommit(false);
    Statement stat = conn.createStatement();
    stat.executeUpdate("UPDATE users SET balance=balance-1000 WHERE user_id=5;");
    stat.executeUpdate("UPDATE users SET balance=balance+1000 WHERE user_id=6;");
    conn.commit();
} catch (SQLException e) {
    conn.rollback();
    e.printStackTrace();
}
```

W przypadku jakiegokolwiek błędu cofamy transakcję.

Wyzwalacze (triggers)

W SQL jest możliwość stworzenia wyzwalaczy (triggerów).

Są to funkcje (zazwyczaj **`DELETE`**, **`INSERT`**, **`UPDATE`**), które zostaną automatycznie uruchomione, jeżeli zostanie spełniony określony przez nas warunek.

Więcej o wyzwalaczach:

- http://www.tutorialspoint.com/plsql/plsql_triggers.htm
- http://www.sqlteam.com/article/an-introduction-to-triggers-part-i