Programowanie aplikacji w Java

Maciej Gowin

Zjazd 5 - dzień 2

Linki

Opis

https://maciejgowin.github.io/wsb-java/

Kod źródłowy przykładów oraz zadań

https://github.com/MaciejGowin/wsb-programowanie-aplikacji-java

MySQL w praktyce: łączenie table

W praktyce częstym przypadkiem jest konieczność łączenia informacji z kilku tabel. Dla przykładu - w przypadku rezerwacji dostępna jest tylko informacja o identyfikatorze klienta.

```
SELECT * FROM bookings;
```

Aby pobrać informacje dodatkowe o klientów, takie jak jest imię i nazwisko, powinniśmy połączyć rezerwacje razem z klientami za pomocą mechanizmu łączeń.

```
SELECT * FROM bookings JOIN customers ON bookings.customer_id = customers.id;
```

Aby uniknąć kolizji nazw oraz wybrać poszczególne wartości, możemy użyć aliasów oraz ograniczenia zwracanych danych.

```
SELECT bookings.id AS booking_id, bookings.flight_id, bookings.customer_id, bookings.price, customers.first_name, customers.last_name
    FROM bookings JOIN customers ON bookings.customer_id = customers.id;
```

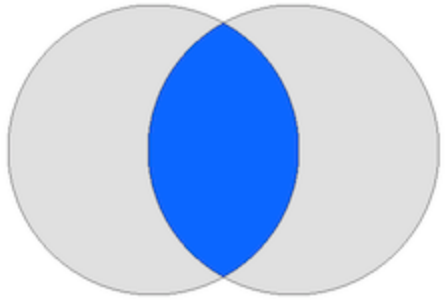
MySQL: opcje łączenia

Wyróżniamy kilka typów łączeń dostępnych różniących się sposobem wykonania łączenia:

- INNER JOIN
- OUTER JOIN
  - LEFT (OUTER) JOIN
  - RIGHT (OUTER) JOIN
- CROSS JOIN

MySQL: łączenie INNER JOIN

Łączenie `INNER JOIN` jest używane do pobrania tylko wspólnych pasujących wierszy i jest najczęściej używanym łączeniem.
W jego przypadku pobieramy tylko wiersze z tabeli A oraz tabeli B, które spełniają warunek łączenia.



MySQL w praktyce: łączenie INNER JOIN

Aby pobrać klientów z ich lotami, wykonamy:

```
SELECT * FROM customers INNER JOIN bookings ON customers.id = bookings.customer_id;
```

Jak widać, zwróceni zostali tylko klienci, dla których istnieje jakaś rezerwacja. Dodatkowo słowo kluczowe `INNER` jest opcjonalne. Poniższe zapytanie jest równoznaczne.

```
SELECT * FROM customers JOIN bookings ON customers.id = bookings.customer_id;
```

Co więcej, w przypadku tego łączenia warunek łączenia `ON`, może zostać zastąpiony warunkiem `WHERE`.

```
SELECT * FROM customers JOIN bookings WHERE customers.id = bookings.customer_id;
```

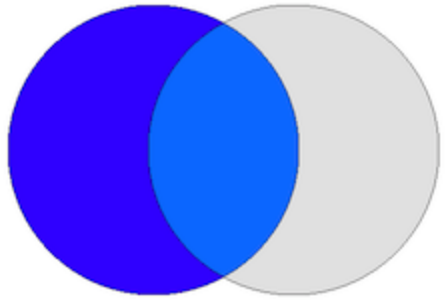
MySQL: łączenia OUTER JOIN

Łączenia `OUTER JOIN` w przeciwieństwie do `INNEJ JOIN` zwracają nie tylko pasujące wiersze, ale również te bez dopasowania.
W przypadku braku dopasowania wierszy w łączonej tabeli brakujące elementy przyjmują wartość NULL.

Istnieją dwa typy `OUTER JOIN`: `LEFT` oraz `RIGHT`.

MySQL: łączenie LEFT OUTER JOIN

Łączenie `LEFT OUTER JOIN` pobiera wszystkie wiersze z tabeli A razem z wierszami z tabeli B, dla których spełniony jest warunek łączenia.
Dla wierszy z tabeli A, które nie spełniają warunku łączenia, kolumny z tabeli B przyjmują wartość NULL.



MySQL w praktyce: łączenie LEFT OUTER JOIN

Aby rozszerzyć nasze zapytanie o wszystkich klientów z ich rezerwacjami, wykonamy:

```
SELECT * FROM customers LEFT OUTER JOIN bookings ON customers.id = bookings.customer_id;
```

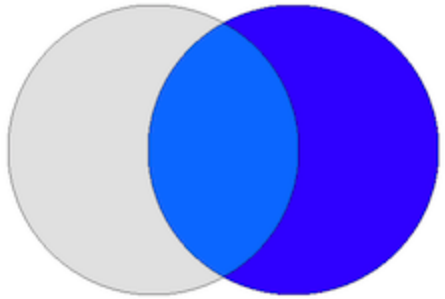
Jak widać, zwróceni zostali wszyscy klienci, nawet Ci, dla których nie istnieje żądna rezerwacja.

Słowo kluczowe `OUTER` jest opcjonalne. Poniższe zapytanie jest równoznaczne.

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

MySQL: łączenie RIGHT OUTER JOIN

Łączenie `RIGHT OUTER JOIN` pobiera wszystkie wiersze z tabeli B razem z wierszami z tabeli A, dla których spełniony jest warunek łączenia.
Dla wierszy z tabeli B, które nie spełniają warunku łączenia, kolumny z tabeli A przyjmują wartość NULL.



MySQL w praktyce: łączenie RIGHT OUTER JOIN

W tym przypadku zapytamy o rezerwacje wraz z wszystkimi klientami:

```
SELECT * FROM bookings RIGHT OUTER JOIN customers ON bookings.customer_id = customers.id;
```

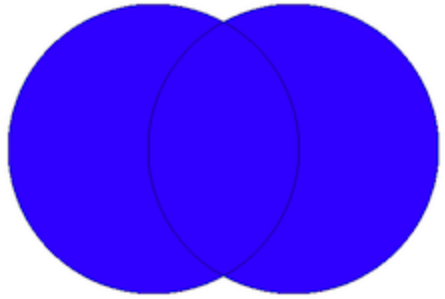
Jak widać, zwrócone zostały rezerwacje wraz z wszystkimi klientami, nawet tymi, dla których nie istnieje żadna rezerwacja.

Słowo kluczowe `OUTER` jest opcjonalne. Poniższe zapytanie jest równoznaczne.

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

MySQL: łączenie CROSS JOIN

Łączenie `CROSS JOIN` jest reprezentacją iloczynu kartezjańskiego, które zwraca wszystkie kombinacje wierszy z każdej z tabel.
W przypadku tego połączenia zwrócony jest zbiór powstały na podstawie połączenia każdego z wierszy tabeli A z każdym wierszem tabeli B.



MySQL w praktyce: łączenie CROSS JOIN

Wyobraźmy sobie, że chcielibyśmy poznać wszystkie możliwe opcje rezerwacji. Wynik ten powstałby przez złączenie wszystkich klientów z wszystkimi możliwymi lotami.

```
SELECT * FROM customers CROSS JOIN flights;
```

MySQL w praktyce: łączenie z aliasami

Do tej pory łączyliśmy dwie tabele. W przypadku lotów każdy z nich definiuje dwa lotniska: wylotu oraz przylotu. Aby pobrać każdy z lotów wraz z nazwą lotniska, musimy zatem połączyć tabelę lotów z tabelą lotnisk dwukrotnie. Aby możliwe było łączenie tej samej tabeli dwukrotnie, musimy użyć aliasów.

```
SELECT * FROM flights
    JOIN airports AS departure_airports ON flights.departure_airport_code = departure_airports.code
    JOIN airports AS arrival_airports ON flights.arrival_airport_code = arrival_airports.code;
```

Niestety pojawił się tylko jeden z lotów, ponieważ użyliśmy `INNER JOIN`. Jedno z lotnisk nie posiada informacji szczegółowych. Użycie `LEFT OUTER JOIN` powinno rozwiązać problem.

```
SELECT * FROM flights
    LEFT JOIN airports AS departure_airports ON flights.departure_airport_code = departure_airports.code
    LEFT JOIN airports AS arrival_airports ON flights.arrival_airport_code = arrival_airports.code;
```

MySQL: podzapytania

Podzapytania to zapytania zagnieżdżone w innych zapytaniach.

Istnieją dwa typy zapytań zagnieżdżonych:

- proste: wynik podzapytania jest użyty w porównaniu w zapytaniu głównym.
- skorelowane: kolumny z zapytania mogą być używane w podzapytaniu.

MySQL: podzapytania

Wraz z zapytaniami używane są operatory podzapytań.

Nazwa	Opis
`ANY`, `SOME`	zwraca prawdę, jeżeli porównanie będzie prawdą dla dowolnego wiersza z podzapytania
`IN ANY`	równoznaczne z `= ANY`
`ALL`	zwraca prawdę, jeżeli porównanie będzie prawdą dla wszystkich wierszy z podzapytania

MySQL w praktyce: podzapytania

Wyszukaj rezerwacji, których koszt jest większy od kosztu średniego:

```
SELECT * FROM bookings WHERE price > (SELECT AVG(price) FROM bookings);
```

Wyszukaj wszystkich klientów, którzy nie mają żadnej rezerwacji.

```
SELECT * FROM customers
    WHERE NOT EXISTS
        (SELECT * FROM bookings WHERE customers.id = bookings.customer_id);
```

Wyszukaj wszystkie loty, dla których istnieją zamówienia o wartości większej niż zadana.

```
SELECT * FROM flights
    WHERE id = ANY
        (SELECT flight_id FROM bookings WHERE price > 50);
```

MySQL: funkcje

MySQL dostarcza szereg wbudowanych funkcji pozwalających na operacje na danych. Do tej pory poznaliśmy już funkcje grupujące: `COUNT`, `SUM`, `AVG`, `MIN`, `MAX`.

Nie sposób przedstawić wszystkie z funkcji. Zapoznamy się z najczęściej używanymi na podstawie przykładów.

MySQL w praktyce: funkcje

`CONCAT` - łączenie ciągów znaków.

```
SELECT id, CONCAT(first_name, ' ', last_name) AS full_name FROM customers;
```

`UPPER`, `LOWER` - zmiana wielkości znaków.

```
SELECT LOWER(first_name), UPPER(last_name) FROM customers;
```

`LENGTH` - długość ciągu znaków.

```
SELECT last_name, LENGTH(last_name) AS last_name_length FROM customers;
```

MySQL w praktyce: funkcje

`TRIM` - usunięcie białych znaków z początki i końca ciągu znaków.

```
SELECT TRIM('   value   ') AS trimmed;
```

`SUBSTR`, `SUBSTRING` - tworzenie podciągu znaków.

```
SELECT id, CONCAT(SUBSTR(first_name, 1, 1), SUBSTR(last_name, 1, 1)) as initials FROM customers;
```

> Pierwszy indeks przyjmuje wartość 1 w przeciwieństwie do języka Java, gdzie pierwszy element znajduje się pod indeksem 0.

MySQL w praktyce: funkcje

`STRCMP` - porównanie ciągu znaków.

```
SELECT first_name, STRCMP(first_name, 'K') FROM customers;
```

`REPLACE` - zmiana danego ciągu znaku na inny ciąg znaków.

```
SELECT REPLACE('Never say never', 'say', 'sing') AS replaced;
```

MySQL w praktyce: funkcje

`FLOOR`, `CEIL`, `CEILING` - zaokrąglenie wartości do dołu, do góry.

```
SELECT price, FLOOR(price) AS price_from, CEIL(price) AS price_to FROM bookings;
```

`ROUND` - zaokrąglenie wartości do wskazanego miejsca po przecinku.

```
SELECT price, ROUND(price), ROUND(price, 1) FROM bookings;
```

`TRUNCATE` - pominięcie wartości do wskazanego miejsca po przecinku.

```
SELECT price, TRUNCATE(price, 1) FROM bookings;
```

MySQL w praktyce: funkcje

`NOW` - pobranie obecnej daty i czasu.

```
SELECT NOW();
```

`DATE`, `TIME` - pobranie daty i czasu.

```
SELECT id, DATE(departure_date_time), TIME(departure_date_time) as departure_date FROM flights;
```

`DATE_FORMAT` - formatowanie daty i czasu.

```
SELECT id, DATE_FORMAT(departure_date_time, "%Y-%m-%dT%H:%i:%s") as departure_date_time_iso FROM flights;
```

MySQL w praktyce: funkcje

`YEAR`, `MONTH`, `DAY`, `HOUR`, `MINUTE`, `SECOND` - pobranie danej wartości z daty i czasu.

```
SELECT id,
        YEAR(departure_date_time) AS year, MONTH(departure_date_time) as month,
        DAY(departure_date_time) as day, HOUR(departure_date_time) as hour,
        MINUTE(departure_date_time) as minute, SECOND(departure_date_time) as second
    FROM flights;
```

`MONTHNAME` - pobranie nazwy miesiąca.

```
SELECT id, MONTHNAME(departure_date_time) as month FROM flights;
```

MySQL w praktyce: funkcje

`DAYNAME` - pobranie nazwy dnia.

```
SELECT id, DAYNAME(departure_date_time) as day FROM flights;
```

`DAYOFWEEK` - pobranie numeru dnia tygodnia.

```
SELECT id, DAYOFWEEK(departure_date_time) as day FROM flights;
```

> Liczenie rozpoczynamy od niedzieli.

MySQL: indeksy

O indeksach wspomnieliśmy przy okazji klucza głównego. Każda kolumna będąca kluczem jest automatycznie indeksowalna.

Indeksy są używane to szybkiego wyszukiwanie wierszy na podstawie wartości danej kolumny. Wyszukiwanie z pominięciem indeksu odbywa się przez naiwne sprawdzanie warunku dla każdego z wierszy jeden po drugim. Jeżeli jednak wyszukiwanie odbywa się po kolumnie indeksowanej, silnik wyszukiwanie jest w stanie zoptymalizować proces bez konieczności sekwencyjnego sprawdzania danych.

MySQL: indeksy

Indeksy są przechowywane jako osobne dane, dlatego też wymagają dodatkowej przestrzeni dyskowej. Większość z indeksów jest przechowywana przy pomocy struktury B-drzewa.

Indeks może zostać zdefiniowany podczas definicji tabeli lub też dodany po jej utworzeniu. Indeks może składać się z jednej lub wielu kolumn.

MySQL w praktyce: indeksy

Aby usprawnić wyszukiwanie lotnisk po nazwie, moglibyśmy dodać indeks oparty o nazwę lotnika;

```
CREATE INDEX idx_name ON airports (name);
```

Lub też zdefiniować indeks dla klientów do wyszukiwania ich po imieniu o nazwisku.

```
CREATE INDEX idx_full_name ON customers (first_name, last_name);
```

Aby sprawdzić definicję indeksu:

```
SHOW INDEX FROM airports, customers;
```

W każdym momencie możemy też usunąć uprzednio utworzony indeks.

```
DROP INDEX idx_name ON airports;
DROP INDEX idx_full_name ON customers;
```

Relacyjne bazy dany: klucze obce

Klucz obcy jest kluczem łączącym dwie tabele. Opisuje on zależność pomiędzy dwoma tabelami.

Kluczem obcym nazywamy kolumnę lub zbiór kolumn, których wartości są takie same jak klucz główny innej tabeli.

Relacyjne bazy dany: klucze obce

Do tej pory używaliśmy niejawnie (implicite) kluczy obcych, łącząc tabele. Dla przykładu:

- `bookings.customer_id` może być zdefiniowany jako klucz obcy dla `customer.id`
- `bookings.flight_id` może być zdefiniowany jako klucz obcy dla `flight.id`
- `flight.departure_airport_code` może być zdefiniowany jako klucz obcy dla `airports.code`
- `flight.arrival_airport_code` może być zdefiniowany jako klucz obcy dla `airports.code`

Klucze obce, które są zdefiniowane explicite, pomagają nam w zapewnieniu spójności danych. Szczególnie przydatne jest to podczas ich dodawania i usuwania.

MySQL w praktyce: tworzenie klucza obcego

Aby utworzyć klucz obcy, aktualizujemy tabelę, definiując referencję do klucza głównego innej tabeli.

```
ALTER TABLE bookings
   ADD CONSTRAINT fk_customers_id FOREIGN KEY (customer_id) REFERENCES customers (id);
```

Aby usunąć klucz obcy:

```
ALTER TABLE bookings DROP FOREIGN KEY fk_customers_id;
```

MySQL w praktyce: ograniczenia klucza obcego

Moglibyśmy spróbować dodać klucz obcy dla kolejnej tabeli:

```
ALTER TABLE flights
   ADD CONSTRAINT fk_airports_departure_code FOREIGN KEY (departure_airport_code) REFERENCES airports (code);
```

Operacja ta jednak się nie powiedzie. Chcemy założyć ograniczenie klucza obcego, jednak w tabeli lotów użyliśmy kodu lotniska, który nie istnieje w tabeli lotnisk.

Po dodaniu lotniska `GDN` wszystko powinno zadziałać poprawnie.

```
INSERT INTO `airports` VALUES ('GDN', 'Gdańsk', 3.3, 33.33);
```

MySQL w praktyce: ograniczenia klucza obcego

Dzięki tak zdefiniowanym ograniczeniom:

- nie będziemy w stanie dodać wiersza, który narusza referencję klucza obcego.

```
INSERT INTO `flights` VALUES (4, 'BCN', 'DUB', '2021-01-03 08:30:00', '2021-01-03 10:45:00');
```

- nie będziemy w stanie usunąć wiersza, który narusza referencję klucza obcego.

```
DELETE FROM airports WHERE code = 'GDN';
```

MySQL w praktyce: usuwanie kaskadowe

MySQL wprowadza opcję kaskadowego usuwania, które może być przydatne dla usuwania wierszy z tabel ze zdefiniowanym kluczem obcym. Usuwanie kaskadowe spowoduje automatyczne usunięcie wierszy posiadających klucz obcy wskazujący na klucz główny usuwanego wiersza.

```
ALTER TABLE flights DROP FOREIGN KEY fk_airports_departure_code;
ALTER TABLE flights
   ADD CONSTRAINT fk_airports_departure_code
   FOREIGN KEY (departure_airport_code)
   REFERENCES airports (code)
   ON DELETE CASCADE;
DELETE FROM airports WHERE code = 'GDN';
```

> Opcja ta powinna byc używana z rozwagą, gdyż może spowodować usunięcie dużej ilości danych.

Relacyjne bazy dany: transakcje

Przez transakcję w systemach bazodanowych rozumiemy zbiór operacji, które wykonane muszą jako całość. Wynika z tego, że w kontekście transakcji powinny zostać wykonane wszystkie operacje lub żadna.

Jakikolwiek błąd w wykonywaniu operacji powinien skutkować w anulowaniu poprzednio wykonanych operacji.

Zasady ACID opisują warunki, jakie powinny spełniać transakcje.

Transakcje

Przykładem transakcji może być zbiór operacji służących do pobrania pieniędzy z bankomatu. Operacja powiedzie się jedynie, jeżeli pieniądze zostaną dostarczone klientowi oraz stan konta zostanie prawidłowo zredukowany.

Zasady ACID

ACID jest zbiorem własności transakcji bazodanowej mającej za zadanie zagwarantowanie spójności danych pomimo błędów.

Własność	
(A)tomicity	Niepodzielność
(C)onsistency	Spójność
(I)solation	Izolacja
(D)urability	Trwałość

Zasady ACID

Atomicity (Niepodzielność)

W kontekście danej transakcji muszą zostać wykonane wszystkie operacje lub żadna.

Consistency (Spójność)

Po wykonaniu transakcji system pozostaje spójny.

Zasady ACID

Isolation (Izolacja)

Podczas równoległego wykonywania się dwóch transakcji nie powinny one widzieć wzajemnie wprowadzanych zmian lub też zmiany są widoczne w zależności od zdefiniowanego poziomu izolacji.

Durability (Trwałość)

W przypadku jakiejkolwiek awarii system jest w stanie odtworzyć swój stan i zwrócić spójne oraz aktualne dane będące rezultatem wykonania transakcji.

Poziomy izolacji

Poziomy izolacji definiują sposób, w jaki oddziałują na siebie równolegle wykonujące się transakcje. Definiujemy 4 poziomy - od najbardziej do najmniej restrykcyjnego.

1. Serializable (najwyższy)
2. Repeatable read
3. Read committed
4. Read uncommitted (najniższy)

W przypadku mniej restrykcyjnych poziomów możliwe jest wystąpienie anomalii.

> Nie wszystkie poziomy izolacji są implementowane przez konkretne systemy relacyjnych baz danych.

Poziomy izolacji

Serializable

Transakcje wykonywane współbieżnie muszą dać taki sam rezultat jak w przypadku wykonaniu tych transakcji szeregowo.

Poziom ten wymaga blokad zapisu i odczytu aż do końca wykonania danej transakcji. Dodatkowo wprowadzone są blokady zakresu w przypadku odczytu warunkowego (z instrukcją WHERE).

Repeatable read

Transakcje nie mogą odczytywać ani zapisywać krotek, na których operuje inne transakcje.

Poziom ten nie wymaga blokad zakresu.

Anomalie: phantom reads

Poziomy izolacji

Read committed

Transakcje mogę odczytywać tylko zapisane krotki.

Poziom ten wprowadza blokady zapisu do końca transakcji, jednak blokady odczytu są zwalniane zaraz po wykonaniu instrukcji SELECT.

Anomalia: non-repeatable reads phenomenon

Read uncommitted

Transakcje mogą wzajemnie odczytywać krotki, na których pracują inne transakcje.

Poziom ten pozwala na odczytywanie zmian, które nie zostały jeszcze zatwierdzone.

Anomalie: dirty reads

MySQL w praktyce: tworzenie transakcji

Aby stworzyć transakcję oraz zaakceptować zmiany, jeżeli nie powstał żaden błąd:

```
START TRANSACTION;

SELECT @lastFlightId:=MAX(id) FROM flights;
SELECT @customerId:=id FROM customers WHERE last_name = 'Kowalski';
INSERT INTO bookings(customer_id, flight_id, price) VALUES (@customerId, @lastFlightId, 111.11);

COMMIT;
```

MySQL w praktyce: tworzenie transakcji

Aby stworzyć transakcję oraz odwrócić zmiany, jeżeli powstał błąd:

```
START TRANSACTION;

DELETE FROM customers WHERE id = 6;

ROLLBACK;
```

MySQL: procedury

W przypadku bardzo rozbudowanych zapytań lub zapytań, które chcielibyśmy sparametryzować, istnieje możliwość definiowania ich w postaci procedur.

Raz zdefiniowana procedura może zostać wywołana wielokrotnie. Rozwiązanie jest to bardzo podobne do funkcji znanych z języków programowania.

Z definicji procedura ma następujący format:

```
Create Procedure [Procedure Name] ([Parameter 1], [Parameter 2], [Parameter 3] )
Begin
SQL Queries..
End
```

A jej wywołanie odbywa się poprzez słowo kluczowe `CALL`:

```
CALL [Procedure Name] ([Parameters]..)
```

MySQL w praktyce: definicja procedury

Aby zdefiniować procedurę:

```
DELIMITER //
CREATE PROCEDURE sp_get_airport_codes()
BEGIN
    SELECT UPPER(code) as code FROM airports;
END //
DELIMITER ;
```

Aby wywołać procedurę;

```
CALL sp_get_airport_codes();
```

Możemy też procedurę usunąć lub usunąć, jeżeli istnieje:

```
DROP PROCEDURE sp_get_airport_codes;
DROP PROCEDURE IF EXISTS sp_get_airport_codes;
```

MySQL w praktyce: definicja procedury

Jak już wspomnieliśmy, procedura może przyjmować argumenty. Stwórzmy procedurę, która automatycznie zwiększa cenę rezerwacji.

```
DELIMITER //
CREATE PROCEDURE sp_increase_booking_prices(IN percent INT)
BEGIN
    UPDATE bookings SET price = price * (100 + percent) / 100;
END //
DELIMITER ;
```

Wywołajmy procedurę z parametrem procentowym:

```
CALL sp_increase_booking_prices(10);
```

Relacyjne bazy dany: normalizacja baz danych

Do normalizacji danych używamy postaci normalnych. Głównym celem normalizacji jest usunięcie duplikatów danych oraz zapewnienie ich spójności.

Postacie normalne definiowane są przez poziomy normalizacji:

- Pierwsza postać normalna (1NF)
- Druga postać normalna (2NF)
- Trzecia postać normalna (3NF)
- Postać normalna Boyce’a-Codda (BCNF lub 3.5NF)
- Czwarta postać normalna (4NF)

Każda z kolejnych postaci wprowadza kolejne restrykcje usuwające potencjalne duplikaty danych.