Notatki - SQL - Relacje

Spis treści

Relacje między tabelami

Wspomniałem wcześniej, że mówimy o relacyjnych bazach danych. No dobrze, to gdzie te relacje?

Tabele mogą zawierać wzajemne odnośniki do siebie. Przedstawię to obrazowo w taki sposób. Wyobraźmy sobie, że każdy pracownik ma zapisany w bazie danych adres, ale z jakiegoś powodu jest to tylko miasto i ulica. Nie mamy zapisanego numeru bloku ani mieszkania. Moglibyśmy zapisać to w naszej tabelce w taki sposób:

ID	NAME	SURNAME	AGE	SALARY	DATE_OF_EMPLOYMENT	CITY	STREET
1	Aleksander	Wypłata	33	8791.12	2018-03-12	Warszawa	Marszałkowska
2	Roman	Pomidorowy	43	7612.12	2012-01-01	Gdańsk	Oliwska
3	Anna	Rosół	38	5728.90	2015-07-18	Warszawa	Marszałkowska
4	Urszula	Nowak	39	3817.21	2014-12-15	Gdańsk	Oliwska
5	Stefan	Romański	38	9201.23	2020-07-14	Warszawa	Marszałkowska
6	Jolanta	Kowalska	27	6521.22	2012-06-04	Szczecin	Biała

W przypadku gdy jakiś rodzaj danych zaczyna się bardzo często powtarzać, nie ma sensu zapisywać go "na płasko" - czyli wszystko do jednej tabeli. Druga kwestia to 'podział obowiązków', tabela powinna być odpowiedzialna za jedną i tylko jedną rzecz, a nie za przetrzymywanie wszystkich możliwych informacji o naszych pracownikach. W tym celu wykorzystuje się relacje. Sytuację wyżej można przedstawić w ten sposób:

Tabela EMPLOYEES

ID	NAME	SURNAME	AGE	SALARY	DATE_OF_EMPLOYMENT	ADDRESS_ID
1	Aleksander	Wypłata	33	8791.12	2018-03-12	1
2	Roman	Pomidorowy	43	7612.12	2012-01-01	2
3	Anna	Rosół	38	5728.90	2015-07-18	1
4	Urszula	Nowak	39	3817.21	2014-12-15	2
5	Stefan	Romański	38	9201.23	2020-07-14	1
6	Jolanta	Kowalska	27	6521.22	2012-06-04	3

Tabela ADDRESSES

ID	CITY	STREET
1	Warszawa	Marszałkowska
2	Gdańsk	Oliwska
3	Szczecin	Biała

W ten sposób możemy stworzyć relację między tabelami i między rekordami, wiedząc, że jeżeli `ADDRESS_ID` ma wartość `3`, to musimy szukać rekordu w tabeli `ADDRESS` pod `ID = 3`.

Stwórzmy teraz takie tabele w praktyce

```
CREATE TABLE ADDRESSES(
    ID     INT         NOT NULL,
    CITY   VARCHAR(32) NOT NULL,
    STREET VARCHAR(64) NOT NULL,
    PRIMARY KEY (ID)
);

CREATE TABLE EMPLOYEES(
    ID                 INT           NOT NULL,
    NAME               VARCHAR(20)   NOT NULL,
    SURNAME            VARCHAR(20)   NOT NULL,
    AGE                INT,
    SALARY             NUMERIC(7, 2) NOT NULL,
    DATE_OF_EMPLOYMENT DATE,
    ADDRESS_ID         INT           NOT NULL,
    PRIMARY KEY (ID),
    CONSTRAINT fk_address
        FOREIGN KEY (ADDRESS_ID)
            REFERENCES ADDRESSES (ID)
);
```

Dodając dopisek:

```
CONSTRAINT fk_address
    FOREIGN KEY (ADDRESS_ID)
        REFERENCES ADDRESSES (ID)
```

Dodajemy FOREIGN KEY (**klucz obcy**) w tabeli EMPLOYEES. Klucz ten mówi, że kolumna ADDRESS_ID ma wskazywać na PRIMARY KEY (**klucz główny**) z tabeli ADDRESSES. Klucz obcy jest oznaczeniem połączenia między tabelami, który oznacza, że dane z kolumny ADDRESS_ID z tabeli EMPLOYEES wskazują na dane z kolumny ID w tabeli ADDRESSES. Nazwa `fk_address` oznacza nazwę klucza obcego (**fk - foreign key**), który jest jednoznacznym identyfikatorem połączenia między tabelami EMPLOYEES i ADDRESSES. Oznacza to, że nie możemy w swojej bazie danych mieć innego klucza obcego o tej samej nazwie (fk_address).

Po stworzeniu tabel w sposób pokazany wyżej należy pamiętać o kilku kwestiach.

Uwaga 1

ADDRESS_ID w tabeli EMPLOYEES jest NOT NULL. Oznacza to, że nie dodamy rekordu do tabeli EMPLOYEES nie mając odnośnika do danych w tabeli ADDRESSES. Czyli najpierw musimy wstawić rekord do tabeli ADDRESSES, a dopiero mając ID rekordu w tabeli ADDRESSES możemy wstawiać dane do tabeli EMPLOYEES. Mówiąc kodem:

```
INSERT INTO ADDRESSES (ID, CITY, STREET) VALUES (1, 'Warszawa', 'Marszałkowska');
INSERT INTO ADDRESSES (ID, CITY, STREET) VALUES (2, 'Gdańsk', 'Oliwska');
INSERT INTO ADDRESSES (ID, CITY, STREET) VALUES (3, 'Szczecin', 'Biała');

INSERT INTO EMPLOYEES (ID, NAME, SURNAME, AGE, SALARY, DATE_OF_EMPLOYMENT, ADDRESS_ID)
    VALUES (1, 'Aleksander', 'Wypłata', 33, 8791.12, '2018-03-12', 1);
INSERT INTO EMPLOYEES (ID, NAME, SURNAME, AGE, SALARY, DATE_OF_EMPLOYMENT, ADDRESS_ID)
    VALUES (2, 'Roman', 'Pomidorowy', 43, 7612.12, '2012-01-01', 2);
INSERT INTO EMPLOYEES (ID, NAME, SURNAME, AGE, SALARY, DATE_OF_EMPLOYMENT, ADDRESS_ID)
    VALUES (3, 'Anna', 'Rosół', 38, 5728.90, '2015-07-18', 1);
INSERT INTO EMPLOYEES (ID, NAME, SURNAME, AGE, SALARY, DATE_OF_EMPLOYMENT, ADDRESS_ID)
    VALUES (4, 'Urszula', 'Nowak', 39, 3817.21, '2014-12-15', 2);
INSERT INTO EMPLOYEES (ID, NAME, SURNAME, AGE, SALARY, DATE_OF_EMPLOYMENT, ADDRESS_ID)
    VALUES (5, 'Stefan', 'Romański', 38, 9201.23, '2020-07-14', 1);
INSERT INTO EMPLOYEES (ID, NAME, SURNAME, AGE, SALARY, DATE_OF_EMPLOYMENT, ADDRESS_ID)
    VALUES (6, 'Jolanta', 'Kowalska', 27, 6521.22, '2012-06-04', 3);
```

Nie bylibyśmy w stanie dodać danych w odwrotnej kolejności ze względu na constraint NOT NULL na kolumnie ADDRESS_ID w tabeli EMPLOYEES.

Uwaga 2

Można w tym momencie spokojnie odpytywać o dane z tych tabel niezależnie, połączenie jest potrzebne wtedy gdy chcemy zobaczyć dane w 'sklejonym' widoku, czyli w takim. Poruszymy tę tematykę za moment

ID	NAME	SURNAME	AGE	SALARY	DATE_OF_EMPLOYMENT	CITY	STREET
1	Aleksander	Wypłata	33	8791.12	2018-03-12	Warszawa	Marszałkowska
2	Roman	Pomidorowy	43	7612.12	2012-01-01	Gdańsk	Oliwska
3	Anna	Rosół	38	5728.90	2015-07-18	Warszawa	Marszałkowska
4	Urszula	Nowak	39	3817.21	2014-12-15	Gdańsk	Oliwska
5	Stefan	Romański	38	9201.23	2020-07-14	Warszawa	Marszałkowska
6	Jolanta	Kowalska	27	6521.22	2012-06-04	Szczecin	Biała

Uwaga 3

`Klucz obcy` powoduje, że należy uważać z kolejnością usuwania danych z bazy danych. Jeżeli tabela `EMPLOYEES` ma dowiązanie do danych z tabeli `ADDRESSES`, to nie możemy najpierw usunąć danych z tabeli `ADDRESSES`, bo na te konkretnie informacje, które byśmy chcieli usunąć wskazuje klucz obcy z tabeli `EMPLOYEES`.

JOINy

Jak teraz wyświetlić połączone ze sobą tabele? Trzeba wykorzystać klauzulę `JOIN`. JOINy są używane do złączania ze sobą tabel przy wykorzystaniu kluczy obcych.

Da się wykonać połączenie między tabelami po kolumnach, które nie są kluczami obcymi, ale w praktyce zawsze robi się klucze obce. A przynajmniej tak być powinno ☺.

Teraz trzeba pamiętać o pewnej kwestii, gdyż poniższe zapytanie zwróci błąd.

```
SELECT ID, NAME, SURNAME, AGE, SALARY, DATE_OF_EMPLOYMENT, CITY, STREET
FROM EMPLOYEES AS EMP
    INNER JOIN ADDRESSES ADR ON EMP.ADDRESS_ID = ADR.ID;
```

Mamy 2 tabele, które mają teraz kolumnę `ID`, dlatego trzeba konkretnie określić o którą kolumnę nam chodzi. Przepiszmy zatem te zapytanie w ten sposób (różnica to `EMP.ID`):

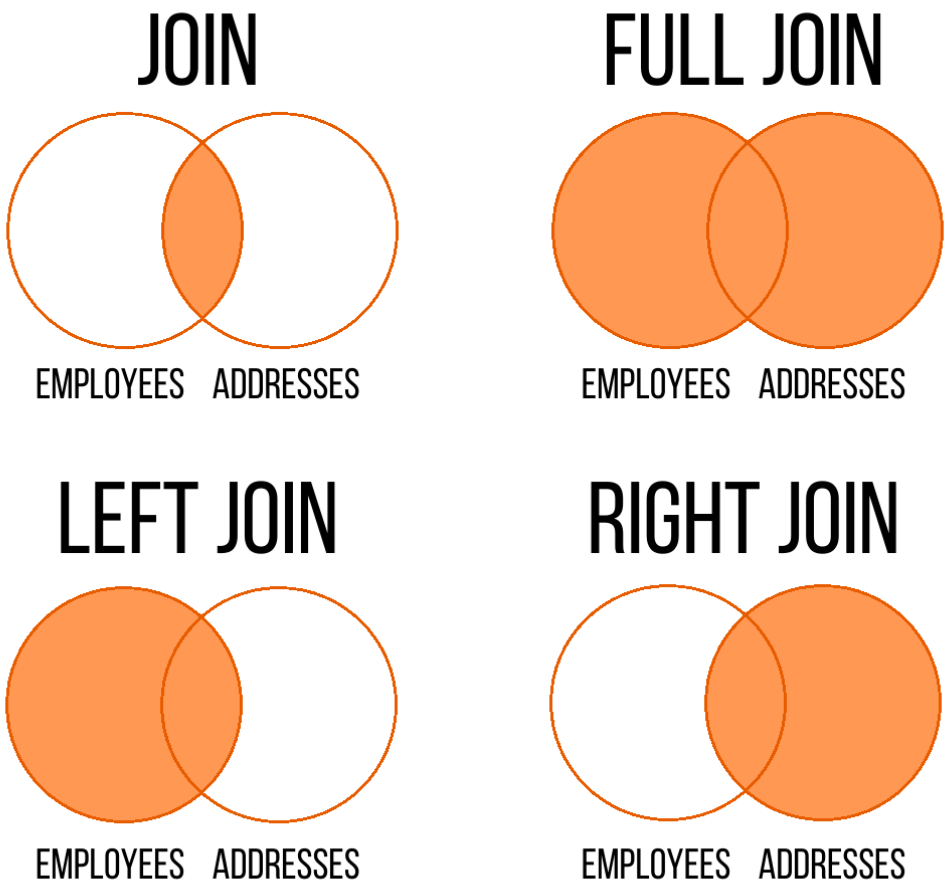
```
SELECT EMP.ID, NAME, SURNAME, AGE, SALARY, DATE_OF_EMPLOYMENT, CITY, STREET
FROM EMPLOYEES AS EMP
    INNER JOIN ADDRESSES ADR ON EMP.ADDRESS_ID = ADR.ID;
```

Wykorzystane tutaj zostały 2 aliasy `EMP` i `ADR` - jest to nazwa własna, która na potrzeby tego zapytania będzie używana jak zmienne. Używamy fragmentu `INNER JOIN`, który pozwala nam złączyć ze sobą 2 tabele na podstawie podanych kolumn `EMP.ADDRESS_ID = ADR.ID`.

W praktyce można napisać ten sam fragment w ten sposób (**zamiast `INNER JOIN` piszemy samo `JOIN`**). Oznacza to to samo, co `INNER JOIN`, ale napisanie `INNER JOIN` jest czytelniejsze, bo podajemy jawnie rodzaj JOINa.

```
SELECT EMP.ID, NAME, SURNAME, AGE, SALARY, DATE_OF_EMPLOYMENT, CITY, STREET
FROM EMPLOYEES AS EMP
    JOIN ADDRESSES ADR ON EMP.ADDRESS_ID = ADR.ID;
```

Czekaj, rodzaj JOINa? Tak, wyróżnia się 4 podstawowe rodzaje JOINów. Rozróżnienie to wynika z tego, że możemy mieć taką sytuację, że dane w 2 tabelach nie pasują do siebie 1 do 1 i możemy mieć rekordy w tabeli EMPLOYEES, dla których nie ma istniejących rekordów powiązanych w tabeli ADDRESSES i odwrotnie. Stąd poniższa grafika, gdzie zostało wyjaśnione działanie każdego JOINa wykorzystując analogię do zbiorów. Grafika powstała na przykładzie tabel EMPLOYEES i ADDRESSES.



Obraz 1. Rodzaje JOINów

Na grafice mamy wyróżnione 4 rodzaje joinów:

- **JOIN** (domyślnie JOIN oznacza INNER JOIN) - szukamy przecięcia 2 zbiorów, czyli wyświetlimy tylko te rekordy z tabeli EMPLOYEES, dla których znajdziemy dopasowanie w tabeli ADDRESSES i jednocześnie wyświetlimy tylko takie rekordy z tabeli ADDRESSES, dla których znajdziemy dopasowanie w tabeli EMPLOYEES. Stąd analogia do przecięcia zbiorów,
- **FULL JOIN** - zwrócimy wszystkie rekordy z tabeli EMPLOYEES, nawet te, dla których nie znajdziemy dopasowania w tabeli ADDRESSES i jednocześnie zwrócimy wszystkie rekordy z tabeli ADDRESSES, nawet te, dla których nie znajdziemy dopasowania w tabeli EMPLOYEES,
- **LEFT JOIN** - zwrócimy wszystkie rekordy z tabeli EMPLOYEES, nawet te, dla których nie znaleźliśmy

dopasowania w tabeli ADDRESSES i jednocześnie zwrócimy tylko te rekordy z tabeli ADDRESSES, dla których znaleźliśmy dopasowanie w tabeli EMPLOYEES,

- **RIGHT JOIN** - zwrócimy wszystkie rekordy z tabeli ADDRESSES, nawet te, dla których nie znaleźliśmy dopasowania w tabeli EMPLOYEES i jednocześnie zwrócimy tylko te rekordy z tabeli EMPLOYEES, dla których znaleźliśmy dopasowanie w tabeli ADDRESSES.

Przykład w praktyce? Musimy z definicji tabelki EMPLOYEES pozbyć się constrainta NOT NULL na kolumnie ADDRESS_ID.

```
ALTER TABLE EMPLOYEES
ALTER COLUMN ADDRESS_ID DROP NOT NULL;
```

Teraz możemy mieć taką sytuację, że dodamy rekord do tabeli EMPLOYEES, który nie będzie miał żadnego dowiązania do tabeli ADDRESSES. Możemy również dodać rekordy do tabeli ADDRESSES, do których nie będziemy w żaden sposób się łączyć z tabeli EMPLOYEES.

```
INSERT INTO ADDRESSES (ID, CITY, STREET) VALUES (1, 'Warszawa', 'Marszałkowska');
INSERT INTO ADDRESSES (ID, CITY, STREET) VALUES (2, 'Gdańsk', 'Oliwska');
INSERT INTO ADDRESSES (ID, CITY, STREET) VALUES (3, 'Szczecin', 'Biała');
INSERT INTO ADDRESSES (ID, CITY, STREET) VALUES (4, 'Szczecin', 'Niebieska');
INSERT INTO ADDRESSES (ID, CITY, STREET) VALUES (5, 'Zakopane', 'Wodna');
INSERT INTO ADDRESSES (ID, CITY, STREET) VALUES (6, 'Zakopane', 'Piaskowa');
INSERT INTO ADDRESSES (ID, CITY, STREET) VALUES (7, 'Kraków', 'Wawelska');

INSERT INTO EMPLOYEES (ID, NAME, SURNAME, AGE, SALARY, DATE_OF_EMPLOYMENT)
    VALUES (1, 'Aleksander', 'Wypłata', 33, 8791.12, '2018-03-12');
INSERT INTO EMPLOYEES (ID, NAME, SURNAME, AGE, SALARY, DATE_OF_EMPLOYMENT, ADDRESS_ID)
    VALUES (2, 'Roman' 'Pomidorowy' 43, 7612.12, '2012-01-01', 2);
INSERT INTO EMPLOYEES (ID, NAME, SURNAME, AGE, SALARY, DATE_OF_EMPLOYMENT)
    VALUES (3, 'Anna', 'Rosół', 38, 5728.90, '2015-07-18');
INSERT INTO EMPLOYEES (ID, NAME, SURNAME, AGE, SALARY, DATE_OF_EMPLOYMENT, ADDRESS_ID)
    VALUES (4, 'Urszula', 'Nowak', 39, 3817.21, '2014-12-15', 2);
INSERT INTO EMPLOYEES (ID, NAME, SURNAME, AGE, SALARY, DATE_OF_EMPLOYMENT, ADDRESS_ID)
    VALUES (5, 'Stefan', 'Romański', 38, 9201.23, '2020-07-14', 1);
INSERT INTO EMPLOYEES (ID, NAME, SURNAME, AGE, SALARY, DATE_OF_EMPLOYMENT)
    VALUES (6, 'Jolanta', 'Kowalska', 27, 6521.22, '2012-06-04');
```

Teraz mając te przykłady możemy pobawić się kolejnymi rodzajami joinów:

INNER JOIN

```
SELECT ID, NAME, SURNAME, AGE, SALARY, DATE_OF_EMPLOYMENT, CITY, STREET
FROM EMPLOYEES AS EMP
    INNER JOIN ADDRESSES ADR ON EMP.ADDRESS_ID = ADR.ID;
```

FULL JOIN

```
SELECT EMP.ID, NAME, SURNAME, AGE, SALARY, DATE_OF_EMPLOYMENT, CITY, STREET
FROM EMPLOYEES AS EMP
    FULL JOIN ADDRESSES ADR ON EMP.ADDRESS_ID = ADR.ID;
```

LEFT JOIN

```
SELECT EMP.ID, NAME, SURNAME, AGE, SALARY, DATE_OF_EMPLOYMENT, CITY, STREET
FROM EMPLOYEES AS EMP
    LEFT JOIN ADDRESSES ADR ON EMP.ADDRESS_ID = ADR.ID;
```

RIGHT JOIN

```
SELECT EMP.ID, NAME, SURNAME, AGE, SALARY, DATE_OF_EMPLOYMENT, CITY, STREET
FROM EMPLOYEES AS EMP
    RIGHT JOIN ADDRESSES ADR ON EMP.ADDRESS_ID = ADR.ID;
```

Przy czym zwróć uwagę, że w przypadku `LEFT JOIN` oraz `RIGHT JOIN`, lewy oraz prawy zbiór (odnoszę się do obrazka) są kolejno rozumiane jako tabela określona w sekcji `FROM` (lewy zbiór) oraz tabela dodawana w sekcji `JOIN` (prawy zbiór).