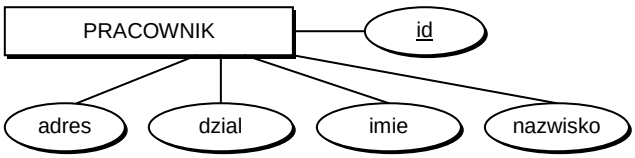
Normalizacja bazy danych, czyli o dobrym projektowaniu

Załóżmy, że mamy przygotować bazę danych pracowników pewnej firmy. Oczywiście pierwszym krokiem jest rozmowa z jej przedstawicielami (aby uzyskać wiedzę o modelowanej rzeczywistości). Kiedy już wiemy, co chcemy umieścić w bazie danych, rysujemy diagram ERD.
Spójrzmy teraz na niego krytycznie. Czy jest dobry? Może pewne rzeczy można było inaczej rozwiązać? Jak ocenić jakość naszego pomysłu? To są typowe pytania, które powinien zadawać sobie projektant bazy. Każdy praktyk wie, że najmniejszy błąd popełniony teraz zemści się – jeżeli wykryjemy go na etapie implementacji, albo – co gorsza – wdrażania bazy danych.

Załóżmy, że w naszej firmowej bazie danych informacje o pracownikach wyglądają tak:



Wyobraźmy sobie, że jesteśmy z takiego diagramu zadowoleni i tworzymy tabelę:

```
CREATE TABLE pracownik (id int auto_increment primary key, adres char(100),
dzial char(100), imie char(30), nazwisko char(50))
```

Zaimplementowaliśmy bazę danych, wypełniliśmy tabele. Popatrzmy na przykładowe dane:

1	Gdańsk, Kozia 7, 80-110 Polska	Wydz. A	Jan	Kowalska
2	Oslo, Hagewatt 8 Norwegia	WA	Tadeusz	Malinowska
3	Ul. Gdańska 8, Oslowo 45-234 Polska	Wydz a	Karolina	Kozłowska
4	Oslo, Anderssen Str. 89 Norway	Wydział B	Anna	Pietrewicz

Użytkownik bazy zwraca się do nas z prośbą o przygotowanie raportu zawierającego wszystkich pracowników, którzy mieszkają w Gdańsku. I nagle okazuje się, że jest problem: jak to zrobić? Nie mamy kolumny miasto – informacja o nim jest częścią adresu. Jedynym rozwiązaniem jest więc próba stosowania jakiejś formy wyszukiwania tekstowego. Na przykład:

SELECT * FROM pracownik WHERE adres like ‘%gdańsk%’

Niestety – takie podejście zwróci również pracownika o ID=3, który mieszka przy ulicy Gdańskiej. Podobny problem będzie z próbą odszukania pracowników mieszkających w Oslo – tym razem możemy trafić na mieszkańca Oslowa. Warto zauważyć – że nasze kłopoty spowodowane są dwoma przyczynami:

* niedokładnym zanalizowaniu problemu (nie przewidzieliśmy, że w firmie będzie konieczność wyszukiwania pracowników według miast)
* nienajlepszym diagramie ERD – który okazał się po prostu za mało elastyczny (przy czym nie można powiedzieć, że jest błędny!)

Analizując powyższy przypadek, dochodzimy do wniosku, że jakość naszego diagramu nie jest najwyższa. Czy więc istnieją jakieś metody, które pozwalają ocenić, czy dany ERD jest lepszy czy gorszy? Odpowiedź brzmi: tak.

Twórcą systematycznego podejścia w dziedzinie „ulepszania” modeli danych jest słynny matematyk i teoretyk baz danych, Edgar Codd. Wprowadził on pojęcie **postaci normalnych** (jest ich kilka), czyli zbioru własności, które powinny mieć relacje. W tym miejscu konieczna jest pewna uwaga. Codd – jako matematyk – posługiwał się pojęciem relacji, czyli specyficznego obiektu matematycznego (relacja jest pojęciem teorii mnogości). Ponieważ jednak nasz kurs jest poświęcony bazom danych, a nie formalizmom matematycznym, przyjmijmy, że postaci normalnie odnoszą się do tabel baz danych (większość praktyków zgadza się z takim uproszczeniem).

Jak zaznaczyliśmy, istnieje kilka różnych postaci normalnych – każde z nich oznacza, że tabele ma pewien zestaw własności. Zasadniczo: im „wyższa" postać normalna, tym związane z nią własności są bardziej rygorystyczne. Jak widać, przypomina to trochę systematykę w naukach biologicznych. Wyobraźmy sobie klasyfikację: „zwierzę" (pierwsza postać normalna świata biologii) i „ssak" (druga postać normalna). Istota jest zwierzęciem jeśli posiada określone cechy (własności): musi być wielokomórkowe, mieć możliwość wzrostu i rozmnażania, być cudzożywne. Aby być ssakiem, istota żyjąca musi być zwierzęciem, a jego gatunek musi ssać mleko jako młode etc.

Zapamiętajmy więc:

Każda **postać normalna** to zbiór własności, którymi muszą się charakteryzować dane, aby mogły być uznane za znormalizowane do danej postaci. Mówimy: te dane są (lub nie) w pierwszej (drugiej, trzeciej...) postaci normalnej.

Pierwsza postać normalna

Postać ta jest oznaczana jako 1NF, co jest skrótem od 1st Normal Form. Istnieje pewien teoretyczny spór co do tego, jakie powinny być ścisłe (formalne) warunki zaliczenia danych (tabel) do 1NF. Generalnie, wygodne kryterium podaje np. Christopher Date, jeden z najbardziej znanych teoretyków baz danych (pracował m.in. na Cambridge i w IBM). Uznaje on, że aby tabela była w 1NF, musi ona zapewniać, że:

1. nie ma powtarzających się krotek
2. że każdy atrybut jest jednowartościowy (czyli, że w danym polu można zapisać tylko jedną wartość z dopuszczalnego zbioru)
3. Nie ma wartości pustych (NULL)

Formalna wersja tych warunków wygląda tak: „zbiór danych jest w pierwszej postaci normalnych wtedy i tylko wtedy, gdy istnieje taka relacja, z którą zbiór ten jest izomorficzny".

Uwaga: Postać 1NF opisywana przez Date'a zawiera także wymóg nieistotności kolejności rekordów (w bazie – bo wynik zapytania oczywiście można sortować) i kolejności kolumn, ale w przypadku typowych silników baz danych jest to dość oczywiste.

Przyjrzyjmy się po kolei postulatom Date'a.

Postulat pierwszy mówi o tym, że krotki nie mogą się powtarzać. Jest on dość jasny, bo inaczej nie można by było rozróżnić dwóch rekordów o tych samych danych. Pamiętajmy, że warunek ten odnosi się do całych rekordów. Oznacza to, że tabela po lewej stronie jest w postaci 1NF, a po prawej – nie:

Jan	Kowalski
Jan	Nowak
Piotr	Kowalski

Jan	Kowalski
Jan	Kowalski
Piotr	Nowak

Nawiasem mówiąc – najprostszym sposobem zapewnienia tego warunku jest nadanie jakiejś kolumnie warunku UNIQUE (bo skoro wartości w kolumnie nie powtarzają się – to nie powtarzają się również rekordy). Praktyk zauważy oczywiście, że sprawę rozwiązuje dodanie kolumny id z własnością primary key i auto_increment.

Postulat drugi mówi o jednowartościowości atrybutów. Zastanówmy się chwilę nad naszą bazą danych pracowników – załóżmy, że chcemy przechowywać numery telefonów komórkowych. Wyobraźmy sobie tabelę:

1	Jan	Kozioł	603456783
2	Maria	Stefaniszyn	502345384
3	Piotr	Pomianowski	609348509

Co będzie, jeżeli Maria dostanie drugi telefon? Najprostszym wyjściem byłoby oczywiście dołożenie drugiego numeru do czwartej kolumny:

1	Jan	Kozioł	603456783
2	Maria	Stefaniszyn	502345384 502387699
3	Piotr	Pomianowski	609348509

Oczywiście takiej możliwości nie udostępniają nam bazy danych (jedno pole = jedna wartość). Dzieje się tak dlatego, że byłoby to złamanie drugiego postulatu Date'a. Możemy więc spróbować inaczej:

1	Jan	Kozioł	603456783
2	Maria	Stefaniszyn	502345384, 502387699
3	Piotr	Pomianowski	609348509

Teraz pole zawiera JEDNĄ wartość – jest nim po prostu dłuższy łańcuch. Tu pojawia się pewien niuans formalny – teoretycznie można uznać taką tabelę za 1NF. Jeśli jednak zastanowimy się głębiej, zauważymy, że czwarta kolumna powinna zawierać jedynie numery telefonów komórkowych. Numery to ciągi 9 cyfr – nie można ich wydłużać w nieskończoność, ani dodawać przecinków. Oznacza to, że praktycznie również następuje tu złamanie drugiego postulatu, ponieważ wartość atrybutu jest co prawda pojedyncza, ale nie należy do przestrzeni dozwolonych wartości!

Pojawia się więc trzecie rozwiązanie – bardzo niezgrabne, ale teoretycznie możliwe (jest ono zazwyczaj wskazywane przez początkujących bazodanowców). Dołóżmy jeszcze jedną kolumnę, w której możemy przechowywać telefony.

1	Jan	Kozioł	603456783	NULL
2	Maria	Stefaniszyn	502345384	502387699
3	Piotr	Pomianowski	609348509	NULL

Takie rozwiązanie jest bardzo toporne – bo nakłada ograniczenie, że żaden z pracowników nie może mieć więcej niż 2 telefony. Teoretycznie jednak mogłoby zostać uznane za dopuszczalne przez 1NF. Aby jednak nie promować takich dziwacznych obejść, Date wprowadził postulat trzeci - dość dziwnie wyglądający zakaz wartości NULL.

Z kolei Codd proponował, aby wprowadzić wymóg atomowości danych. Atomowość (ang. atomicity) jest pojęciem wywodzącym się z matematyki - w uproszczeniu możemy je rozumieć jako nierozkładalność na mniejsze czynniki (w podobnym znaczeniu używał tego pojęcia Demokryt). Zauważmy, że formalnie bazy danych nie zapewniają atomowości – na przykład typ date składa się w istocie z 3 liczb typu int. Date zauważył więc, że propozycja Codda nie jest formalnie poprawna, ale w sumie jest dość zrozumiała i oczywista.

Ten spór – w istocie czysto teoretyczny – doprowadził do tego, że uznano, że postulat atomowości powinien być rozumiany jako połączenie idei Codda i Date'a. Oznacza to, że atomowość rozumiemy jako:

1. silną jednowartościowość atrybutu,
2. wymóg niepowtarzalności przestrzeni wartości.

Silna jednowartościowość atrybutu oznacza, że nie dopuszczamy wartości takich jak „502345384, 502387699" – bo w jednej wartości „ukryte" są rzeczywiste dwa numery.

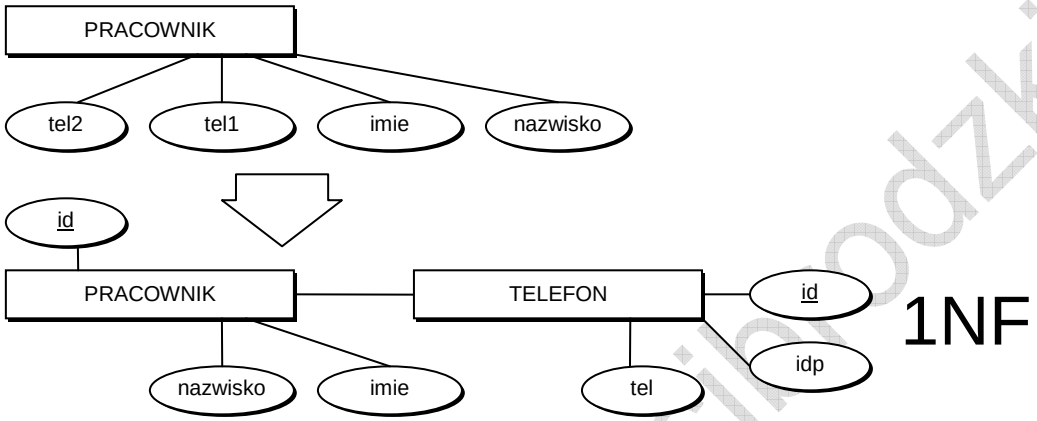
Niepowtarzalność przestrzeni wartości oznacza, że w danej tabeli nie może być dwóch kolumn, które odwołują się do tej samej przestrzeni wartości i odnoszą się do tego samego obiektu ze świata rzeczywistego. Czyli nie mogę w tabeli zastosować kolumn telefon1 i telefon2, bo mają one te same wartości dozwolone (numer telefonu, czyli 9 cyfr) i odnoszą się do tej samej rzeczy: numer telefonu pracownika.

Tu pojawia się drobny niuans formalny. Załóżmy, że chcemy pamiętać datę urodzin i datę zatrudnienia pracownika. Takie dwie kolumny mają wspólną przestrzeń wartości – jest nią data. Ktoś bardzo przywiązany do formalizmów, mógłby uznać, że łamie to postulat atomowości (podobnie jak kolumny telefon1 i telefon2) – dlatego dodaliśmy powyżej zastrzeżenie, że atomowość nie jest złamana, jeśli modelujemy dwa różne obiekty.

Ostatecznie, możemy podać ostateczną postać warunków 1NF:

1. rekordy (krotki) są rozróżnialne
2. atrybuty są atomowe

Najczęstszą konsekwencją normalizacji do 1NF (czyli doprowadzania do postaci 1NF) jest stworzenie dodatkowych tabel. Jest to tak naturalne, że praktycy robią to automatycznie, praktycznie nie zastanawiając się nad tym.



Druga postać normalna

Istotna drugiej postaci normalnej sprowadza się do redundancji, czyli zwykle szkodliwego powtarzania danych przechowywanych w tabeli. Największy problem z redundancją jest związany z faktem, że jeśli ta sama informacja jest przechowywana w wielu miejscach, to ktoś może zmodyfikować tylko jeden zapis. Baza straci spójność – bo nie wiadomo która z informacji jest poprawna.

Wyobraźmy sobie następującą tabelę. Jej projektant chciał przechowywać informację o pracownikach, miastach w których mieszkają i językach, którymi władają.

Jan	Kowalski	Polski	Gdańsk
Jan	Kowalski	Niemiecki	Gdańsk
Piotr	Nowak	Polski	Szczecin
Stefan	Kozłowski	Polski	Warszawa
Piotr	Nowak	Angielski	Szczecin

Czy ta tabela jest w pierwszej postaci normalnej? Oczywiście: jest. Zauważamy jednak łatwo, że nie jest ona bynajmniej dobra. Informacja o miejscu zamieszkania Jana Kowalskiego znajduje się w 2 rekordach. Jest powtórzona – występuje redundancja. Co będzie, jeśli po edycji pierwszego rekordu tabela będzie wyglądała tak:

Jan	Kowalski	Polski	Gdańsk
Jan	Kowalski	Niemiecki	Poznań
Piotr	Nowak	Polski	Szczecin
Stefan	Kozłowski	Polski	Warszawa
Piotr	Nowak	Angielski	Szczecin

Nie wiadomo, gdzie mieszka Jan Kowalski – i nic nie wskazuje na to, który rekord zawiera poprawną informację.

Do zdefiniowania 2NF potrzebna będzie nam znajomość dwóch terminów: klucza kandydującego oraz zależności funkcyjnej.

Przypomnijmy rozważania na temat identyfikacji rekordów. Nie zawsze przecież wprowadzamy „sztuczny" identyfikator id – tak jest w przypadku naszej bazy danych. Wprowadzono więc pojęcie klucza kandydującego. Jest to każdy atrybut (lub najmniejsza z możliwych grupa atrybutów), których wartość jest unikalna w danej tabeli (innymi słowy – rekordy tabeli możemy rozróżnić po tym zestawie wartości). Od projektanta bazy danych zależy, który klucz kandydujący uzna za klucz główny.

Popatrzmy na naszą tabelę – jak widać mamy tylko jeden klucz kandydujący. Jest on złożony z 3 kolumn: imię, nazwisko i język (bo w całej tabeli nie ma 2 takich samych trójek imię-nazwisko-język). Zauważmy też, że kluczem kandydującym nie są wszystkie 4 kolumny – bo nie jest to minimalny zestaw – wprawdzie nie istnieją dwie takie same czwórki imię-nazwisko-język-miasto, ale możemy usunąć z takiej czwórki miasto nie tracąc unikalności.

Przy okazji możemy zastanowić się jak wyglądałaby tabela z dwoma kluczami kandydującymi:

Jan	Kowalski	Polski	73020103456	Gdańsk
Jan	Kowalski	Niemiecki	73020103456	Poznań
Piotr	Nowak	Polski	43121102934	Szczecin
Stefan	Kozłowski	Polski	98101045777	Warszawa
Piotr	Nowak	Angielski	43121102934	Szczecin

Jak widać, klucze kandydujące to: imię-nazwisko-język oraz pesel-język. Wróćmy jednak do naszej pierwotnej tabeli.

Zależność funkcyjna pomiędzy dwoma atrybutami (kolumnami tabeli) A i B oznacza, że dla każdej wartości atrybutu A występuje zawsze jedna wartość B (pojęcie to pochodzi od matematycznego pojęcia funkcji – jednej wartości x może odpowiadać tylko jeden y). Mówimy wtedy, że B jest funkcyjnie zależy od A (albo, że B jest funkcją A).

Rozważmy to na przykładzie:
Atrybut B1 jest zależny funkcyjnie od A1, bowiem dla określonej wartości w kolumnie A1 występuje zawsze ta sama wartość B1. Jeśli A1=1, to B1=A, jeśli A1=2, to B1=B, a jeśli A1=3 to B1=A (jak widzimy, tym różnym wartościom A1 mogą odpowiadać te same wartości B1 – nie ma tu złamania zależności funkcyjnej).
W drugiej tabeli atrybut B1 nie jest zależny funkcyjnie od B2, bo tym samym wartościom A2 odpowiadają różne wartości B2. Jak widać, jeśli A2=3 to B2=A albo B2=B.

A1	B1
1	A
1	A
2	B
2	B
3	A
3	A

A2	B2
1	A
1	A
2	B
2	B
3	A
3	B

Sformułujmy teraz warunki drugiej postaci normalnej.

Tabela jest w 2NF wtedy i tylko wtedy gdy:

1. jest w 1NF
2. żaden z atrybutów, które nie wchodzą w skład klucza kandydującego nie jest funkcjonalnie zależny od części tego klucza (inaczej: żaden z atrybutów nie jest w częściowej zależności funkcyjnej od klucza głównego)

Przyjrzyjmy się temu ostatniemu zapisowi. Mówi on mniej więcej tyle: jeżeli weźmiemy dowolny klucz kandydujący i dowolny atrybut nie będący jego częścią to atrybut ten nie może być funkcją części klucza kandydującego.

Popatrzmy na naszą tabelę:

Jan	Kowalski	Polski	Gdańsk
Jan	Kowalski	Niemiecki	Gdańsk
Piotr	Nowak	Polski	Szczecin
Stefan	Kozłowski	Polski	Warszawa
Piotr	Nowak	Angielski	Szczecin

Kluczem kandydującym (jedynym) jest Imię-Nazwisko-Język. Atrybut nie wchodzący w skład tego klucza to oczywiście Miasto. Czy jest **pełna** zależność funkcyjna pomiędzy CAŁYM kluczem a Miastem? Oczywiście TAK (MUSI tak być, bo w kluczu WSZYSTKIE wartości są różne, a więc nie można mówić o „tych samych wartościach klucza"!). Popatrzmy jednak na **częściową** zależność funkcyjną. Wybierzmy z klucza kandydującego atrybuty: Imię i Nazwisko i porównajmy je z wartościami atrybutu Miasto:

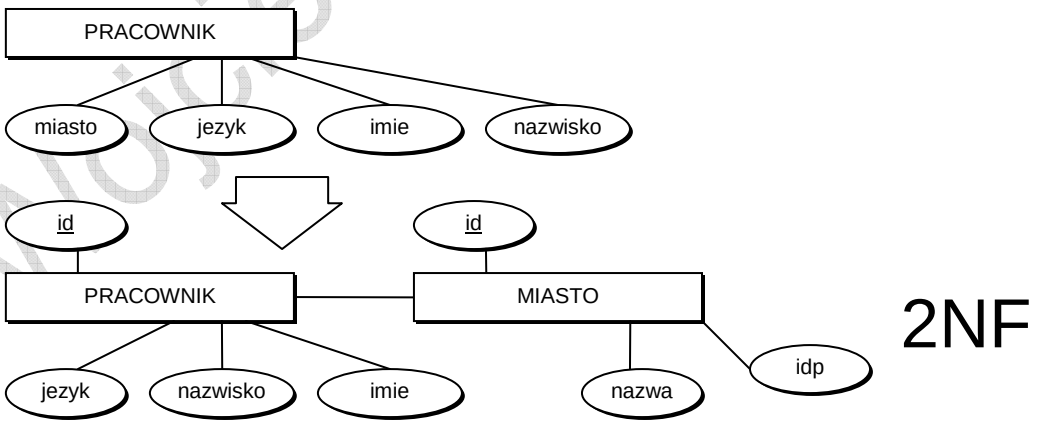
Jan	Kowalski	Gdańsk
Jan	Kowalski	Gdańsk
Piotr	Nowak	Szczecin
Stefan	Kozłowski	Warszawa
Piotr	Nowak	Szczecin

Widzimy, że zależność funkcyjna w takiej tabeli również istnieje! Dzieje się tak dlatego, że parze Jan-Kowalski ZAWSZE odpowiada jedno miasto (Gdańsk), parze Piotr-Nowak ZAWSZE odpowiada wartość Szczecin, a parze Stefan-Kozłowski odpowiada Warszawa.

Skoro więc istnieje częściowa zależność funkcyjna Miasta od klucza głównego (czyli Miasto zależy od CZĘŚCI klucza głównego) to tabela nie spełnia wymogów 2NF!

Podkreślamy: zawsze istnieje zależność funkcyjna od CAŁOŚCI klucza głównego. Nawiasem mówiąc, oznacza to, że jeśli wszystkie klucze główne w bazie danych składają się z jednego atrybutu to wszystkie tabele są w 2NF (oczywiście jeżeli spełniają warunek pierwszy – czyli są w 1NF).

Efektem normalizacji do 2NF najczęściej jest stworzenie nowych tabel (a czasem – dodanie jednoatrybutowego klucza głównego).



Trzecia postać normalna

Trzecia postać normalna również ma na celu usunięcie redundancji w danych. Jest ona jeszcze silniej wiążąca niż 2NF, ponieważ zakłada wykluczenie tzw. przechodnich zależności funkcyjnych.

W ramach ciekawostki: pośród praktyków funkcjonuje zabawna postać 3NF: „Każdy atrybut nie należący do klucza dostarcza wiedzy o rekordach identyfikowanych przez klucz, cały klucz i tylko klucz – tak mi dopomóż Codd".

Wyobraźmy sobie następującą tabelę, zawierającą zwycięzców rajdów Formuły 1.

Silverstone	2000	Kubica	Polska
Monza	2001	Hamilton	Anglia
Hungaroring	2001	Kubica	Polska
Silverstone	2001	Hamilton	Anglia

Widzimy, że mamy jeden klucz kandydujący: Tor-Rok i dwa atrybuty nie stanowiące jego części: Nazwisko i Kraj. Ułóżmy tabelki odpowiadające tym atrybutom. Najpierw weźmy „na warsztat" atrybut Nazwisko.

Silverstone	2000	Kubica
Monza	2001	Hamilton
Hungaroring	2001	Kubica
Silverstone	2001	Hamilton

Zbadajmy częściowe zależności: Nazwisko = f(Tor) i Nazwisko = f(Rok). Pierwsza jest złamana – bo wartości Silverstone odpowiadają „Kubica" i „Hamilton". Druga, bo wartości 2001 odpowiadają „Hamilton" i „Kubica"

Przyjrzyjmy się drugiemu atrybutowi nie stanowiącemu części klucza kandydującego, czyli Kraj.

Silverstone	2000	Polska
Monza	2001	Anglia
Hungaroring	2001	Polska
Silverstone	2001	Anglia

Łatwo zauważyć, że występują podobne złamania funkcyjności – czyli nasza tabela jest 2NF.

Sęk w tym, że atrybut Kraj jest przejściowo funkcyjnie zależny od atrybutu Nazwisko. Popatrzmy:

Kubica	Polska
Hamilton	Anglia
Kubica	Polska
Hamilton	Anglia

Wartości Kubica odpowiada ZAWSZE Polska, zaś wartości Hamilton odpowiada ZAWSZE Anglia. Podkreślamy, że nie jest to złamanie 2NF, ponieważ zależność funkcyjna występuje pomiędzy dwoma atrybutami NIE STANOWIĄCYMI części klucza głównego.

Problem wynikający z niezgodności z 3NF najłatwiej zrozumieć, zauważając, że do bazy danych można wprowadzić niespójność poprzez wprowadzenie następującego błędu:

Silverstone	2000	Kubica	Polska
Monza	2001	Hamilton	Anglia
Hungaroring	2001	Kubica	Polska
Silverstone	2001	Hamilton	Czechy