Nierelacyjne bazy danych

Spis treści:

Wykład 1

Wprowadzenie

Chociaż wykład ten dotyczy programu NoSQL, zaczynamy naszą historię od prawdziwego pattera. Może to być mylące, ale robimy to, ponieważ

- Pamiętaj, że termin NoSQL oznacza nie tylko SQL, ale NO SQL. W tym kontekście termin SQL refere nie jest definiuje SQL jako języka, a raczej cały ekosystem, w którym istniał SQL od dziesięcioleci i będzie istniał w przewidywalnej funkcji: relacyjnych baz danych. Nie mówimy, że nie potrzebujemy SQL (i relacyjnych baz danych). Zamiast tego mówimy *Ponieważ czasami model realny nie wystarczy, musimy go zmienić w pewnych aspektach*
- Jeśli istnieją sytuacje, w których relacyjne bazy danych nie są najlepszym rozwiązaniem dla naszego problemu biznesowego, powinniśmy wiedzieć, dlaczego ten wzór nie nadaje się dla nas i jakie rozwiązania (y) możemy wybrać zamiast tego.
- Model relacyjny jest dobrze znany i ma bardzo dobrą pomoc w zakresie oprogramowania, dokumentacji, specjalistów - zanim zrezygnujesz z tego opisu, powinieneś być pewien, co robisz.
- Aby poprawnie zrozumieć bazy danych NoSQL ich wszystkie zalety i wady, potrzebujemy trochę odniesienia - w tym przypadku porównamy je z jednym znanym nam trybem - modelem relacyjnym.

Model relacyjny

PRE-RELATIONSS ERA

Relacje z bazami danych od prawie 30 lat były tak dominujące na rynku baz danych, że nikt nie pamiętał o tym trudnym czasie urodzin. Trudno uwierzyć, że wielcy gracze na rynku komputerowym byli bardzo sceptyczni wobec tego modelu. Poszukując odpowiedzi, dlaczego tak się stało, musimy wrócić do końca lat pięćdziesiątych.

Bardzo ogólną definicją bazy danych, którą można znaleźć w Wikipedii jest: baza danych jest zorganizowaną kolekcją danych. Zbieranie danych wydaje się być istotnym elementem rozwoju ludzkiej cywilizacji i technologii. Pierwsze zbiory mają silnie egzekwowane przez strukturę ograniczeń fizycznych: wystarczy pomyśleć o książkach (zwłaszcza słownikach i encyklopediach) i bibliotekach. Rewolucja przemysłowa przyniosła nam zestaw nowych narzędzi, których możemy użyć. Możemy zobaczyć genezę dzisiejszej cyfrowej bazy danych przy przyjmowaniu dziurkowanych kart i innych mediów fizycznych, które mogłyby przechowywać informacje w formie, która może być przetwarzana mechanicznie. Z historii nauk komputerowych wiemy, że w XIX wieku karty krosowe były używane do programowania krosien tkaninowych w celu wytworzenia złożonych wzorów tkanin. Następnie, opierając się na koncepcji kart krosowych, maszyny tabulacyjne firmy Hollerith wykorzystywały dziurkowane karty do przetworzenia spisu powszechnego w USA w 1890 r. Wprowadzenie bazy danych terminów zbiegło się z dostępnością przechowywania danych bezpośrednich (dysków i bębnów), które zapewniają szybki dostęp do poszczególnych osób rejestry stały się możliwe.

Raczej szybko dowiedzieliśmy się o tym, że:

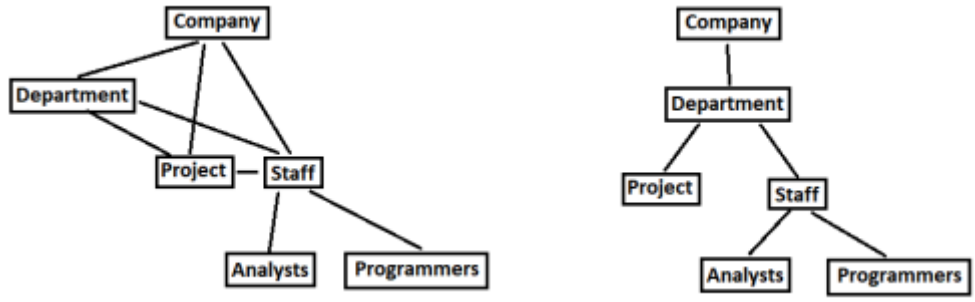
- jednoczesny dostęp lub zmiany danych wymagały wyrafinowanego kodu w celu uniknięcia logicznego lub fizycznego uszkodzenia danych;
- kod obsługi bazy danych został powtórzony z jednej aplikacji do drugiej;
- wszelkie błędy, które możemy wprowadzić w kodzie obsługi danych aplikacji, podczas gdy ten kod został przeniesiony z jednej aplikacji na drugą, prowadzonej nieuchronnie do uszkodzonych danych;
- nie wszystkie techniki optymalizacji i algorytmy mogły być łatwo przenoszone w każdej aplikacji.

Dlatego stało się zupełnie naturalne, aby zrobić to wszystko raz i wielokrotnie używać dla różnych baz danych. W ten sposób logika obsługi bazy danych została usunięta z aplikacji do warstwy pośredniej - systemu zarządzania bazą danych (DBMS).

Niestety, wczesne systemy bazodanowe, z racji wykonania i braku innych wcześniejszych doświadczeń i pomieszczeń, zostały wymuszone

- Schemat - definicja struktury danych w bazie danych. System zapewniający wymuszoną reprezentację danych zapewniający odpowiednią wydajność, dzięki czemu został precyzyjnie dostosowany do podstawowych mechanizmów dostępu.
- Ścieżka dostępu (otrzymujemy dane) za pomocą stałej sekwencji przejść z jednego rekordu do drugiego.

Systemy baz danych w tym czasie są nazwane nawigacyjnie. Dwa główne modele danych nawigacyjnych były modelem hierarchicznym (system IMS IBM) i modelem sieciowym (CODASYL).



Network and hierarchical database models

Jak widać, w modelu hierarchicznym, aby dostać się od firmy do projektu, musimy podążać ścieżką Company-Department-Project.

Można powiedzieć, że DBMS z dnia zostały zaprojektowane przez producentów, a nie użytkowników. Wszystkie modele, struktury danych, sprzęt i oprogramowanie były bardzo ściśle połączone w celu uzyskania pożądanej wydajności.

RELATIONALS ERA

Pod koniec lat sześćdziesiątych Edgar Codd pracował w laboratorium IBM w San Jose w Kalifornii. Był bardzo zaznajomiony z bazami danych dnia, a on zauważył wszystkich zalet i, co było ważne, wady ich projektu. W szczególności zauważył to

1. Istniejące bazy danych różnią się implementacjami logicznymi i fizycznymi. Reprezentacja danych w istniejących bazach danych odpowiadała formatowi fizycznej pamięci w bazie danych, a nie logicznej reprezentacji danych, które mogłyby zostać zrozumiane przez użytkownika nontechnical.

2. Istniejące bazy danych nie miały podstaw teoretycznych. Nie było wspólnych formalnych struktur i logicznych operacji, które moglibyśmy użyć.

3. Opierając się na arbitralnych przedstawieniach, które nie zapewniają logicznej spójności, nie można było poradzić sobie z problemami awarii i spójności.

4. W wyniku poprzednich, istniejących baz danych były zbyt trudne w użyciu. Bazy danych w ciągu dnia mogły być dostępne tylko przez osoby o specjalistycznych umiejętnościach technicznych i programowych, które były bardzo odległe od aplikacji bussines.

Codd opublikował wewnętrzny dokument IBM, w którym zaprezentował swoje pomysły na bardziej sformalizowany model systemów baz danych, który doprowadził do słynnego papieru z 1970 r. Model relacyjny danych dla dużych banków danych współdzielonych. (*A Relational Model of Data for Large Shared Data Banks)*

Nawet wielcy gracze IBM byli sceptyczni i kwestionowali możliwość stworzenia relacyjnych baz danych Codd, które mogłyby zapewnić odpowiednią wydajność. Jednak w 1974 roku IBM zainicjował program badawczy dotyczący opracowania prototypu relacyjnego systemu baz danych o nazwie System R. **System R** był pierwszą implementacją SQL. Był to także pierwszy system wykazujący, że relacyjny system zarządzania bazą danych może zapewnić dobrą wydajność przetwarzania transakcji. Pierwszym klientem systemu R był Pratt & Whitney w 1977 roku.

W tym samym czasie Mike Stonebraker w Berkeley zaczął pracować nad systemem baz danych, który ostatecznie nazywał się INGRES. Program INGRES był również relatywnie relatywny, ale używał języka zapytań innych niż SQL, zwanego QUEL.

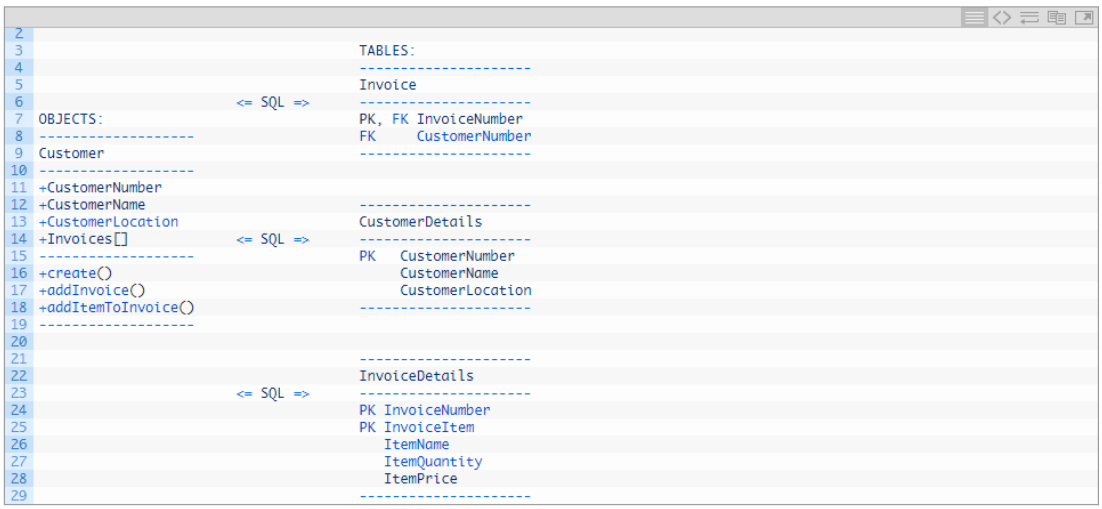
Inną osobą, o której warto wspomnieć, jest Larry Ellison, który był zaznajomiony zarówno z pracą Codd jak i System R, a co ważniejsze, uważał, że relacyjne bazy danych reprezentują przyszłość technologii baz danych. W 1977 roku Ellison założył firmę, która ostatecznie stanie się korporacją Oracle, która wyda pierwszy komercyjny system relacyjnych baz danych.

W następnych kilkudziesięciu latach wprowadzono wiele nowych systemów baz danych (system jako oprogramowanie nie poczęcie). Są to między innymi Sybase, Informix, DB2, Microsoft SQL Server i MySQL. Chociaż wszystkie z nich różnicują pod względem wydajności, dostępności, funkcjonalności lub gospodarki, wszystkie one mają trzy główne zasady: model relacyjny Codd, język SQL i model transakcji ACID.

OBJECT-ORIENTED ERA

Kolejny duży krok dokonany przez bazy danych spowodowany był programowaniem obiektowym. Ten paradygmat wiąże się z logiką. W tradycyjnych, proceduralnych, programowych językach, danych i logice były zasadniczo odrębne. Procedury ładowałyby i manipulowały danymi w ich logice, ale sama procedura nie zawierała danych w jakikolwiek sposób. Programowanie obiektowe scalało atrybuty i zachowania w pojedynczy obiekt. Wszystkie szczegóły dotyczące logicznej jednostki pracy będą przechowywane w jednej klasie lub bezpośrednio powiązane z tą klasą. Było to źródłem niezadowolenia wśród programistów, którzy zmuszeni byli współpracować z obiektowymi reprezentacjami ich danych w swoich programach i reprezentacją relacyjną w bazie danych. Trudno nie zgodzić się z następującym istotnym porównaniem: relacyjna baza danych jest jak garaż, który zmusza cię do odebrania samochodu i przechowywania elementów w małych szufladach.

Gdy obiekt został zapisany w bazie danych relacyjnych lub został pobrany z bazy danych relacyjnych, wiele operacji SQL byłoby wymaganych do konwersji z reprezentacji obiektowej do reprezentacji relacyjnej. Było to kłopotliwe dla programisty i mogło prowadzić do problemów z wydajnością lub niezawodnością.



Było jasne, że system zarządzania bazą danych *Object Oriented Database Management System* (OODBMS) był lepiej dostosowany do potrzeb współczesnych aplikacji. OODBMS przechowywałby obiekty programu bezpośrednio bez normalizacji i umożliwiłby aplikacjom łatwe ładowanie i przechowywanie obiektów. Przedstawienie zorientowane obiektowo wydaje się być naturalne, ale z natury nieracjonalne; rzeczywiście reprezentacja danych ściślej pasowała do sieciowych baz danych z poprzedniej ery, takich jak CODASYL. OODBMS oferował zalety dla twórcy aplikacji, ale zapomniał o tych, którzy chcieliby konsumować informacje w celach biznesowych. Może to być spowodowane tym, że systemy OODBMS całkowicie nie zdołały zdobyć udziału w rynku. Bazy danych nie istnieją, aby programiści mieli prostsze życie. Są to znaczące aktywa, które muszą być dostępne dla tych, którzy chcą kopiować informacje do podejmowania decyzji i analizy biznesowej. Poprzez wdrożenie modelu danych, który był niewidoczny i mógłby być używany przez programistę i ignorując użytkownika biznesowego interfejsu SQL, OODBMS nie uzyskał wsparcia poza programistą.

Wzorzec relacyjny

ROW-STORE DESIGN PATTERN

Wzorzec relacyjny jest przykładem wzornika projektowania wierszy. Technicznie rzecz biorąc, tego typu bazy danych zapisują swoje dane w jednolitym obiekcie zwanym wierszem. Wiersze składają się z określonej liczby pól danych związanych z nazwą kolumny i pojedynczym typem danych. Wiersze są dodawane, zmieniane i usuwane jako jednostki atomowe (jednostka pracy, która jest niezależna od innych dodawanych, zmieniających lub usuwających podczas tej operacji). Ten typ baz danych jest powszechnie znany pod nazwą bazy danych RDBMS lub SQL.

Najbardziej opisowymi cechami tego schematu są:

- Wstawiane wiersze zawierają tabele w RDBMS.
- Tabele mogą być powiązane z innymi tabelami i relacje danych są również przechowywane w tabelach.
- Całą tabelę, wraz ze wszystkimi definicjami kolumn i ich typami danych, należy utworzyć przed pierwszym wierszem wstawionym do tabeli.
- Kolumny muszą mieć unikalne nazwy w tabeli i pojedynczy typ danych, który jest tworzony przy pierwszym zdefiniowaniu tabeli.
- Aby zwiększyć szybkość dostępu, indeksy są tworzone na tablicach zawierających wiele wierszy.
- Otrzymujemy dane w postaci użytecznej, wybierając wszystkie powiązane wiersze z instrukcjami JOIN.

TEORIA RELACYJNA PODSTAWOWE KONCEPCJE

Kluczowe pojęcia modelu relacyjnego to:

- **Krotka [Tuple]**. W matematyce krotka jest skończoną listą porządkową (sekwencją) elementów. N-tuple jest skończoną listą (sekwencją) elementów n, gdzie n jest liczbą całkowitą nieujemną. Jest tylko jedna wartość 0-krotki, pusta sekwencja. N-krotkę definiuje się indukcyjnie przy użyciu konstrukcji pary zamówionej. W informatyce krotki są w wielu

formach. W systemie relacyjnych baz danych krotka odpowiada wierszowi, a atrybut (element krotki) do wartości kolumny.

Przykłady Tuple z języka programowania Swift

```
// Tuple of type (Int, String)
let warning = (123, "This is a critical warning")
var (currentMessageCode, currentMessageText) = warning
print ("Message text " + currentMessageText)
print ("Message text \(currentMessageText)")
```

Jeśli nie obchodzi nas pierwszy element krotki, możemy go napisać jak `(_, currentMessageText) = warning`

Możemy również używać numerów indeksowych zaczynających się od zera, aby uzyskać element krotki `print ("Message text " + warning.1)`

Aby uczynić nasz kod bardziej czytelnym, możemy wymienić poszczególne elementy w krotce, gdy krotka jest zdefiniowana

```
let alert = (messageCode: 456, messageText: "This is an alert")
let info = (messageCode: 456, messageText: "This is an information you can simply ignore")

print ("Message text " + alert.messageText)
```

- **Relacje [Relation(s)].** Relacja jest zbiorem odrębnych krotek i odpowiada tabeli w systemie relacyjnych baz danych.
- **Ograniczenia [Constraints].** Ograniczenia wymuszają spójność bazy danych. Podstawowe ograniczenia są używane do identyfikacji krotek i relacji między krotkami.
- **Operacje [Operations].** Działania to działania, które możemy wykonać w relacjach: łączenia, projekcje, związki itd. Operacje zawsze zwracają relacje, co oznacza, że zapytanie w tabeli zwraca dane w formacie tabelarycznym.

NORMAL FORMS

Co ciekawe, relacyjne modele definiują kilka poziomów zgodności, które są opisane pod nazwami zwykłych nazw. Teraz omówimy trzy pierwsze z nich, aby uzyskać pomysł, na co są. Tak, w tym samouczku będziemy rozmawiać o NoSQL, ale chcę mieć dobre zrozumienie relacji międzyludzkich, aby nie ślepo śledzić trend NoSQL.

W praktycznych zastosowaniach zobaczymy 1NF, 2NF i 3NF. Inne formy: BCNF lub 3.5NF, 4NF i 5NF nie są tak często widziane, więc nie omówimy ich, ponieważ trzy pierwsze wystarczą, aby uzyskać to co chcemy.

- *Pierwsza postać normalna (1NF)* jest właściwością relacji w relacyjnej bazie danych. Relacja jest w pierwszej postaci normalnej, gdy tylko domena każdego atrybutu zawiera tylko wartości atomowe (niepodzielne), a wartość każdego atrybutu zawiera tylko jedną wartość z tej domeny.

Pierwsza postać normalna (1NF) wyznacza bardzo podstawowe zasady dla zorganizowanej bazy danych i wymusza następujące kryteria:

- Wyeliminuj powtarzające się grupy w poszczególnych tabelach.

- Utwórz osobną tabelę dla każdego zestawu powiązanych danych.

- Identyfikuj każdy zestaw powiązanych danych za pomocą klucza podstawowego

- *Druga postać normalna (2NF)* Oprócz klucza podstawowego relacja może zawierać inne kandydujące klucze (klucz kandydata: minimalny zbiór atrybutów związku, jednoznacznie identyfikujący każdą krotkę tego związku).

  Kluczem kandydującym do relacji jest zestaw atrybutów takich, że
  - ✓ relacja nie ma dwóch oddzielnych krotek (tj. wierszy lub rekordów we wspólnym języku bazy danych) z tymi samymi wartościami dla tych atrybutów (co oznacza, że zestaw atrybutów to superkey)
  - ✓ nie ma odpowiedniego podzbioru tych atrybutów, dla których (1) utrzymuje się (co oznacza, że zestaw jest minimalny).

  Składowe atrybuty nazywane są atrybutami podstawowymi. Odwrotnie, atrybut, który nie występuje w dowolnym Kluczu kandydującym, nazywany jest atrybutem non-prime.

Aby mieć relacje w 2NF, konieczne jest ustalenie, że żadne atrybuty no-prime mają zależne od części klucza dowolne kandydujące klucze. Każda funkcjonalna zależność od jakiejkolwiek kandydującego klucza stanowi naruszenie 2NF.
Musimy pamiętać, że normy są skumulowane. Aby baza danych była w formacie 2NF, musi najpierw spełniać wszystkie kryteria bazy danych 1NF; być w 3NF, to mus w 2NF itp.
Innymi słowy, druga postać normalna (2NF) odnosi się do pojęcia usuwania duplikatów danych

1. Spełnia wszystkie wymagania pierwszej postaci normalnej.
2. Usuwa podzestawy danych odnoszące się do wielu wierszy tabeli i umieszcza je w oddzielnych tabelach.
3. Tworzy relacje między tymi tabelami a ich poprzednikami za pomocą kluczy obcych.

- *Trzecia postać normalna (3NF)* Trzecia postać normalna to zwykła forma stosowana w normalizowaniu projektu bazy danych w celu zmniejszenia duplikacji danych i zapewnienia integralności referencyjnej, zapewniając, że wszystkie atrybuty w tabeli są określone tylko przez kandydujące klucze tego związku, a nie przez wszelkie inne niż pierwsze atrybuty.

Innymi słowy, trzecia postać normalna (3NF) idzie o krok dalej w koncepcji usuwania duplikatów: usuń kolumny, które nie są zależne od klucza podstawowego

1. Spełnia wszystkie wymagania drugiej postaci normalnej.
2. Usuwa kolumny, które nie są zależne od klucza podstawowego (każda kolumna musi zależeć bezpośrednio od klucza podstawowego i nic innego niż klucz podstawowy).

Dobrze znana fraza związana z 3NF to [Every] non-key [atrybut] musi dostarczyć informacji na temat klucza, całego klucza i nic poza kluczem, więc pomoże mi Codd.

- Wymaganie istnienia klucza zapewnia stół w 1NF.
- Wymaganie, że atrybuty nie-kluczowe są zależne od całego klucza zapewnia 2NF.
- Wymaganie, że atrybuty nie-kluczowe będą zależne od niczego, ale klucz zapewnia 3NF.

Chociaż to wyrażenie jest użytecznym mnemonikiem, fakt, że wspomina tylko o jednym kluczu, definiuje pewne konieczne, ale nie wystarczające warunki, aby zaspokoić drugie i trzecie formy normalne. Zarówno 2NF, jak i 3NF dotyczą wszystkich kluczy kandydata w tabeli, a nie tylko jednego klucza.

Zilustrujmy to przykładem. Będziemy posługiwać się bardzo popularnym przypadkiem związanym z fakturą (porównaj trzy samouczki z bazą danych o bardziej skomplikowanym przykładzie).

- Initial case

```
Table: invoice
Invoice   Customer   Customer          Customer          Item
number    number     name              location          (name, quantity, price)
1         10         Dart Vader        Star Destroyer    {lightsaber, 1, 100}, {black cloak, 2, 50}, {air filter, 10, 2}
2         20         Luke Skywalker    Naboo             {lightsaber, 5, 75}, {belt, 1, 5}
3         30         C3PO              Tatooine          {battery, 1, 25}
4         30         C3PO              Tatooine          {wires, 1, 10}
```

- 1NF

1NF jest naruszona, ponieważ

- ✓ nie mamy wartości atomowych ({lightsaber, 1, 100})
- ✓ a wartość każdego atrybutu zawiera nie tylko pojedynczą wartość (wiele elementów {...}).

Możemy to naprawić dzięki następującej reorganizacji danych

```
Table: invoice
Invoice
number
=======
1
2
3

Table: invoice_details
Invoice   Invoice   Customer   Customer          Customer          Item          Item       Item
number    item      number     name              location          name          quantity   price
=============================================================================================
1         1         10         Dart Vader        Star Destroyer    lightsaber    1          100
1         2         10         Dart Vader        Star Destroyer    black cloak   2          50
1         3         10         Dart Vader        Star Destroyer    air filter    10         2
2         1         20         Luke Skywalker    Naboo             lightsaber    5          75
2         2         20         Luke Skywalker    Naboo             belt          1          5
3         1         30         C3PO              Tatooine          battery       1          25
4         1         30         C3PO              Tatooine          wires         1          10
```

- 2NF

2NF jest naruszona, ponieważ

- ✓ mamy podzestaw danych, które mają zastosowanie do wielu wierszy tabeli.

W tym przypadku mamy następujący klucz kandydata: {Numer faktury, faktura} `{Invoice number, Invoice item}`. Informacje o klientach (numer, nazwa i lokalizacja) (number, name and location) zależą od właściwego podzbioru: Numer faktury.

Problem polega na tym, że informacje o klientach (liczba, imię i lokalizacja) idą z każdą linią na fakturze. Powoduje to nadmiarowe dane, z nieodłącznym kosztem i trudnymi modyfikacjami. Normalizacja drugiej formy wymaga umieszczenia informacji o klientach w oddzielnej tabeli (faktura).

Możemy to naprawić dzięki następującej reorganizacji danych

```
Table: invoice
Invoice  Customer  Customer         Customer
number   number    name             location
=========================================
1        10        Dart Vader       Star Destroyer
2        20        Luke Skywalker   Naboo
3        30        C3PO             Tatooine
4        30        C3PO             Tatooine

Table: invoice_details
Invoice   Invoice  Item         Item       Item
number    item     name         quantity   price
=================================================
1         1        lightsaber   1          100
1         2        black cloak  2          50
1         3        air filter   10         2
2         1        lightsaber   5          75
2         2        belt         1          5
3         1        battery      1          25
4         1        wires        1          10
```

- 3NF

3NF jest naruszona, ponieważ

- ✓ w tabeli faktur znajdują się kolumny, które nie są zależne od klucza podstawowego.

Problem polega na tym, że informacje o klientach (liczba, nazwa i lokalizacja), które znajdują się w tabeli faktur, powodują redundancję danych, jeśli kilka faktur jest dla tego samego klienta (patrz C3PO). Mogłoby to również powodować problemy, gdy klient zmieni niektóre z nich (na przykład: lokalizacja). Normalizacja w trzecim formularzu wymaga, aby informacje o klientach trafiły do osobnej tabeli z własnym kluczem (numer klienta), z tylko numerem klienta w tabeli faktur.

Możemy to naprawić dzięki następującej reorganizacji danych

```
Invoice  Customer
number   number
================
1        10
2        20
3        30
4        30

Table: customer_details
Customer  Customer         Customer
number    name             location
================================
10        Dart Vader       Star Destroyer
20        Luke Skywalker   Naboo
30        C3PO             Tatooine

Table: invoice_details
Invoice   Invoice  Item         Item       Item
number    item     name         quantity   price
=================================================
1         1        lightsaber   1          100
1         2        black cloak  2          50
1         3        air filter   10         2
2         1        lightsaber   5          75
2         2        belt         1          5
3         1        battery      1          25
4         1        wires        1          10
```

TRANSACTION MODEL AND ACID

Model relacyjny nie definiuje w jaki sposób baza danych obsługuje równoczesne żądania zmiany danych o nazwanych transakcjach. W celu zapewnienia spójności i integralności danych model transakcji ACID jest używany i stał się de facto standardem dla wszystkich ważnych implementacji relacyjnych baz danych. Transakcja ACID powinna być:

- **Atomic.** Atomowa. Transakcja nie może być podzielona - wszystkie instrukcje w bazie danych są stosowane do bazy danych lub żadne nie są.
- **Consistent.** Zgodna. Baza danych pozostaje w stałym stanie przed i po wykonaniu transakcji.
- **Isolated.** Odosobniona. Mimo że wiele transakcji może być wykonanych przez jednego lub więcej użytkowników jednocześnie, jedna transakcja nie powinna widzieć efektów innych transakcji będących w toku.
- **Durable.** Trwała. Gdy transakcja zostanie zapisana (zobowiązana) do bazy danych, jej zmiany będą utrzymywane nawet wtedy, gdy wystąpi awaria systemu operacyjnego lub sprzętu.

Z jednej strony ACID wraz z relacjami jest źródłem mocy relacyjnych baz danych. Z drugiej strony jest to powód do poważnych i bardzo trudnych do przezwyciężenia problemów.

REGUŁY POKEROWE COOD

Codd utworzył **trzynastą regułę**, aby określić, co jest wymagane w systemie zarządzania bazą danych, aby można je było uznać za relacyjne. Zrobił to jako część osobistej kampanii, aby zapobiec rozminowywaniu pierwotnej relacyjnej bazy danych, ponieważ dostawcy baz danych zaczęli mieszkać na początku lat osiemdziesiątych w celu przepychania istniejących produktów jako relacyjne.

- Codd’s Twelve Rules
- Codd's 12 rules
- E.F. Codd’s 12 Rules for RDBMS

PROCESY I ZARZĄDZENIA RDBMS

Model relacyjny niewątpliwie charakteryzuje się następującym zestawem pozytywnych cech:

- Transakcje ACID na poziomie bazy danych ułatwiają rozwój i użycie.
- Większość kodu SQL jest przenośna dla innych baz danych SQL.
- Wpisane kolumny i ograniczenia pomagają walidować dane przed ich dodaniem do bazy danych, co zwiększa spójność danych przechowywanych w bazie danych.
- Zbudowany mechanizm, np. Widoki lub role, uniemożliwia zmianę lub wyświetlenie danych przez osoby nieuprawnione.

Szczerze mówiąc nie można zapomnieć o negatywnej stronie modelu relacyjnego:

- Transakcje ACID mogą blokować system przez krótki czas, co może być niedopuszczalne.
- Mapowanie obiektowo-relacyjne jest możliwe, ale może być złożone i dodać kolejną warstwę pośrednią.
- RDBMS nie są skalowane. Serwowanie wielu serwerów może być wykonane, ale wymaga nowego lub dostrojonego kodu aplikacji i będzie działać nieskutecznie.
- Trudno jest przechowywać dane o dużej zmienności w tabelach.
- Trudno jest przechowywać dane w czasie rzeczywistym i przetwarzać w czasie rzeczywistym.

Model relacyjny jest dobry dla wielu zadań, ale nie wystarczy do wszystkich dzisiejszych problemów. Wady tego modelu zmusiły ludzi do poszukiwania rozwiązań alternatywnych - tak powstał NoSQL.