Systemy Baz Danych

Wprowadzenie

Bartosz Zieliński

WYDZIAŁ FIZYKI
i INFORMATYKI STOSOWANEJ
Uniwersytet Łódzki

Bazy Danych

Baza danych to zbiór danych zapisanych zgodnie z określonymi regułami [Wikipedia]

Do zarządzania bazami danych służą
systemy zarządzania bazami danych
(ang. *Database Management Systems, DBMS*)

Przykład Zbioru Danych

Dane często prezentujemy w formie zbioru rekordów (tablicy), np:

Author's Name	Birth	Title	Year	Price	Genre
C.S. Lewis	1898	*Prince Caspian*	1951	10$	Fantasy
C.S. Lewis	1898	*The Last Battle*	1956	10$	Fantasy
C.S. Lewis	1898	*Perelandra*	1943	12$	Science Fiction
C.S. Lewis	1898	*The Screwtape Letters*	1942	8$	Satire, Fantasy
W.H. Hodgson	1877	*The House on the Borderland*	1908	5$	Horror
W.H. Hodgson	1877	*The Night Land*	1912	8$	Horror, Fantasy
H.P. Lovecraft	1890	*At the Mountains of Madness*	1931	5$	Horror
H.P. Lovecraft	1890	*The Shadow over Innsmouth*	1931	5$	Horror
H.P. Lovecraft	1890	*The Call of Cthulhu*	1928	2$	Horror
H.P. Lovecraft	1890	*Herbert West–Reanimator*	1922	5$	Horror, Comedy
...	...	...	...	...	...

Problemy Baz Danych

- Jakich typów są dane?
- Jak reprezentować na dysku dane różnych typów?
- Jaką nadać strukturę danym?
- Jak zapewnić wielu użytkownikom dostęp do tych samych danych tak by nie przeszkadzali sobie nawzajem?
- Jak kontrolować dostęp do danych?
- Jak zapewnić możliwość wykonywania złożonych operacji na danych (w tym zapytań)?
- Jak zapewnić trwałość i spójność danych?
- ...

Rolą systemów zarządzania bazami danych jest rozwiązywanie (lub ułatwianie rozwiązywania) powyższych problemów.

Jakich Typów są Dane

DBMS dostarcza standardowego zestawu typów. Rolą użytkownika jest przypisanie typów do poszczególnych elementów danych.

- Imię autora to niewątpliwie napis (łańcuch znaków).
- A rok urodzenia? Liczba całkowita, napis czy coś innego?
- Numer telefonu to liczba czy napis? A PESEL?
- A cena? Liczba stało- czy zmienno-przecinkowa?

Problem Reprezentacji Danych Różnych Typów

Dane numeryczne

- Reprezentacja maszynowa — zwarta i ułatwia wykonywanie operacji, ale jest nieprzenośna.
- Reprezentacja tekstowa — zajmuje więcej miejsca, wymaga konwersji w celu wykonania operacji ale jest przenośna.

Dane tekstowe

- Reprezentacja o stałej długości, np. `C.S._Lewis________` (marnuje miejsce, ułatwia skok do następnego pola rekordu).
- Reprezentacja o zmiennej długości, np. `C.S._Lewis\0`.

Problem Możliwej Redundancji Danych

Te same informacje o autorach powtarzają się w wielu rekordach:

Author's Name	Birth	Title	Year	Price	Genre
C.S. Lewis	1898	*Prince Caspian*	1951	10$	Fantasy
C.S. Lewis	1898	*The Last Battle*	1956	10$	Fantasy
C.S. Lewis	1898	*Perelandra*	1943	12$	Science Fiction
C.S. Lewis	1898	*The Screwtape Letters*	1942	8$	Satire, Fantasy
W.H. Hodgson	1877	*The House on the Borderland*	1908	5$	Horror
W.H. Hodgson	1877	*The Night Land*	1912	8$	Horror, Fantasy
H.P. Lovecraft	1890	*At the Mountains of Madness*	1931	5$	Horror
H.P. Lovecraft	1890	*The Shadow over Innsmouth*	1931	5$	Horror
H.P. Lovecraft	1890	*The Call of Cthulhu*	1928	2$	Horror
H.P. Lovecraft	1890	*Herbert West–Reanimator*	1922	5$	Horror, Comedy
...	...	...	...	...	...

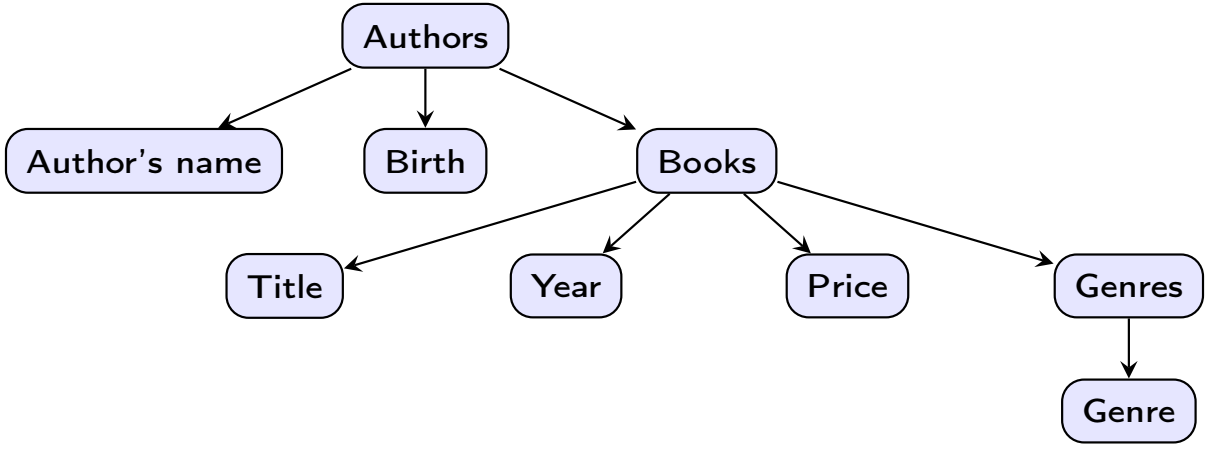
Marnowane jest miejsce na dysku. A co jeśli chcemy zmodyfikować datę urodzenia jednego z autorów? Jak utrzymać spójność danych?

Przechowywanie Danych w Postaci Hierarchicznej

W tym wypadku można uniknąć redundancji przechowując dane w postaci hierarchicznej:

		Books			
					Genres
Author's Name	**Birth**	**Title**	**Year**	**Price**	**Genre**
C.S. Lewis	1898	*Prince Caspian*	1951	10$	Fantasy
		The Last Battle	1956	10$	Fantasy
		Perelandra	1943	12$	Science Fiction
		The Screwtape Letters	1942	8$	Satire Fantasy
W.H. Hodgson	1877	*The House on the Borderland*	1908	5$	Horror
		The Night Land	1912	8$	Horror Fantasy
H.P. Lovecraft	1890	*At the Mountains of Madness*	1931	5$	Horror
		The Shadow over Innsmouth	1931	5$	Horror
		The Call of Cthulhu	1928	2$	Horror
		Herbert West Reanimator	1922	5$	Horror Comedy
...	...	...	...	...	...

Struktura Hierarchiczna Danych

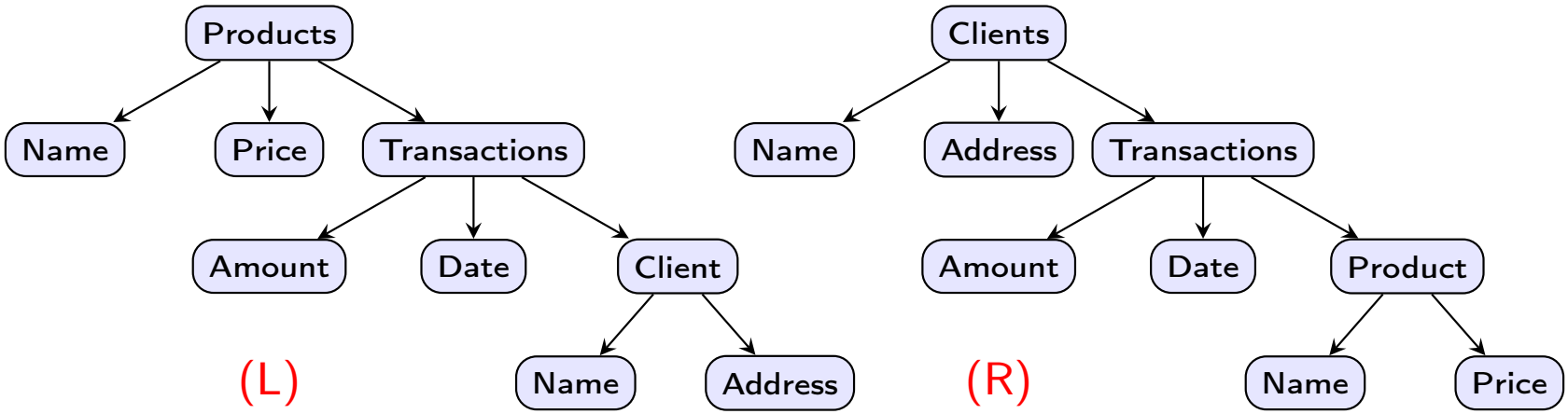


Hierarchiczny Model Danych

- Ciągle używany (rejestr Windows, usługi katalogowe)
- Działa dobrze gdy dla danych istnieje pojedyńcza, naturalna hierarchia. Co jednak zrobić gdy możliwych naturalnych hierarchii jest kilka (niekompatybilnych) albo żadna?
  - Czy fotografie układamy najpierw latami a potem tematami czy na odwrót?

Przykład: Baza Hierarchiczna „Produkty-Transakcje-Klienci”

Możemy albo uznać klientów za część składową produktów które kupili (L) albo uznać produkty za część składową klientów (R):



	(L)	(R)
Zapisz dane o produktach których jeszcze nikt nie kupił	Można	Nie można
Zapisz dane klientów którzy jeszcze nic nie kupili	Nie można	Można
Wyszukaj produkty kupione przez danego klienta	Trudno	Łatwo
Wyszukaj klientów którzy kupili dany produkt	Łatwo	Trudno
Redundantne kopie danych	Klientów	Produktów

Zobacz model sieciowy dla tego przykładu.

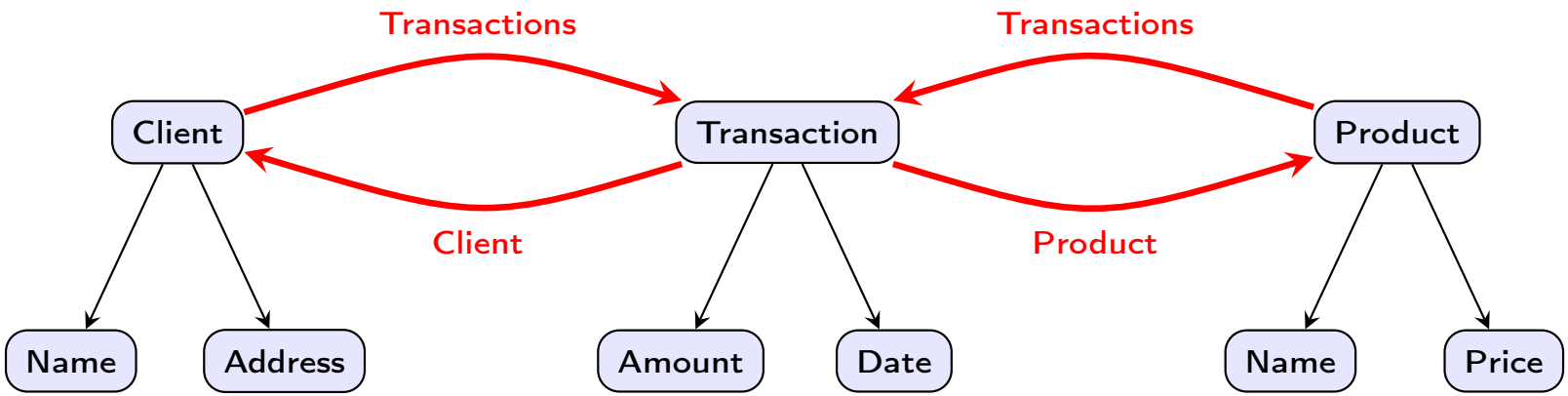
Model Sieciowy Danych

Sieciowy model danych pozwala na nawigację pomiędzy elementami danych po zdefiniowanych przez projektanta bazy ścieżkach które, inaczej niż w modelu hierarchicznym, nie muszą tworzyć drzewa.

- Model sieciowy danych (patrz: CODASYL) ma wiele wspólnego ze współczesnymi modelami grafowym i obiektowym.
- Co zrobić jeśli chcemy skojarzyć elementy danych po ścieżkach których nie przewidział projektant?

Przykład: Baza Sieciowa „Produkty-Transakcje-Klienci”

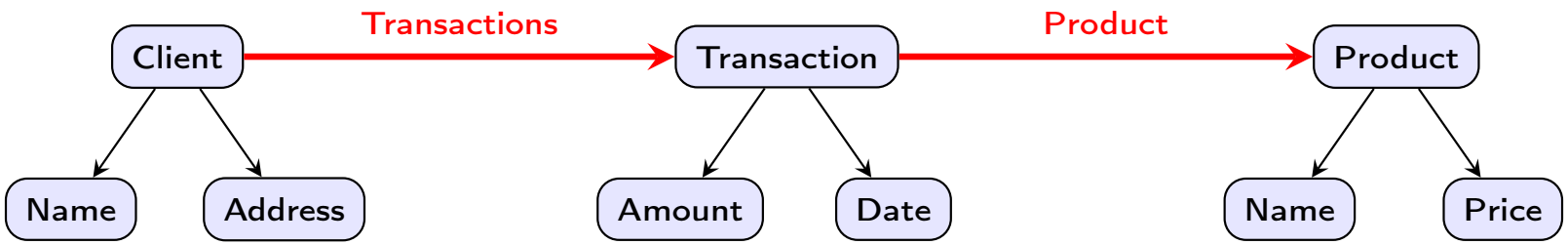
Mamy trzy „klasy obiektów” połączonych ze sobą „referencjami”:



- Każda ze strzałek z etykietą “Transactions” oznacza nie pojedyńczą referencję ale zbiór referencji (od, odpowiednio, klienta i produktu do każdej transakcji w której ten klient i produkt brali udział).
- Powyższy schemat nie ma wad schematów hierarchicznych. Można jedynie zauważyć redundancję referencji z uwagi na ich jednokierunkowość.

Za Mało Referencji

W modelu sieciowym danych już na etapie projektowania bazy należy znać wszystkie możliwe zapytania jakie będą z tą bazą używane ponieważ możliwość zadania pewnych zapytań zależy od obecności w bazie odpowiednich referencji.



Do bazy danych o strukturze przedstawionej powyżej możemy zadać zapytanie o produkty kupione przez danego klienta ale nie o klientów którzy kupili dany produkt.

Model Relacyjny

- **Model relacyjny** to najbardziej rozpowszechniony model danych używany współcześnie w DBMS-ach.
- Całkowicie wyparł w zastosowaniach ogólnobiznesowych modele hierarchiczny i sieciowy, dominujące w latach 70-tych i na początku 80-tych (ponieważ nie miał ich wad).
- Omówienie modelu relacyjnego i relacyjnych baz danych zajmie większą część wykładu i prawie całość ćwiczeń.

Inne Modele

Istnieje wiele innych modeli danych. Są one jednak używane głównie w zastosowaniach specjalnych i żaden z nich nie dorównuje w popularności ani powszechności zastosowań modelowi relacyjnemu:

- Bazy obiektowe
- Bazy grafowe (np. w analizie sieci społecznościowych)
- Bazy XML
- Bazy hierarchiczne (Windows register, DNS, usługi katalogowe)
- Bazy par klucz wartość (Riak, Redis, Memcached)
- Bazy wielowymiarowe (w analizie danych).

Kontrola Współbieżności

Przypuśćmy że dwóch użytkowników modyfikuje jednocześnie pewien rekord. Jeden z nich zapisuje **“Horror, Fantasy”**, drugi **“Fantasy, Horror”**. Bez kontroli współbieżności ostatecznie w rekordzie mogło zostać zapisane **“Fontory Forrasy”**

- Wynik jednoczesnego wykonania złożonych operacji może zależeć od ich przeplecenia.
- Zwykle akceptowalne są wyniki wykonania złożonych operacji jedna po drugiej (bez nietrywialnego przeplotu).
- **Serializowalność** złożonych operacji jest wymuszana przez DBMS-y przez zakładanie odpowiednich blokad na czytane i modyfikowane elementy danych.
- Użytkownik wymusza traktowanie grupy operacji jako pojedyńczej złożonej operacji wykonując te operacje w ramach jednej **transakcji**.

Inny Przykład Kłopotów ze Współbieżnością

Następujące instrukcje wykonywane są jednocześnie przez użytkowników A i B (n jest zmienną prywatną dla każdego z użytkowników):

```
n:=read(f);
n++;
write(n,f);
```

Dwa różne przeplecenia poleceń A i B:

```
//f zawiera 0
n:=read(f);  //A
n++;         //A
write(n,f);  //A
n:=read(f);  //B
n++;         //B
write(n,f);  //B
//f zawiera 2
```

```
//f zawiera 0
n:=read(f);  //A
n:=read(f);  //B
n++;         //A
n++;         //B
write(n,f);  //A
write(n,f);  //B
//f zawiera 1
```

Transakcje

Transakcje spełniają także inną rolę poza kontrolą współbieżności.

Rozważmy przelew bankowy 100$ z konta A do konta B. Składa się on z dwóch operacji: **dodania 100 do konta** B i **odjęcia 100 od konta** A. Przypuśćmy że po **dodaniu 100 do** B doszło do awarii.

- Grupa poleceń umieszczonych wewnątrz transakcji wykonuje się jako jedna całość albo wcale.
- Nieukończoną transakcję można wycofać (cofnąć skutki wykonanych w ramach tej transakcji poleceń).
- Skutki nieukończonej transakcji przerwanej przez awarię zostają automatycznie wycofane przez DBMS podczas jego ponownego uruchomienia.

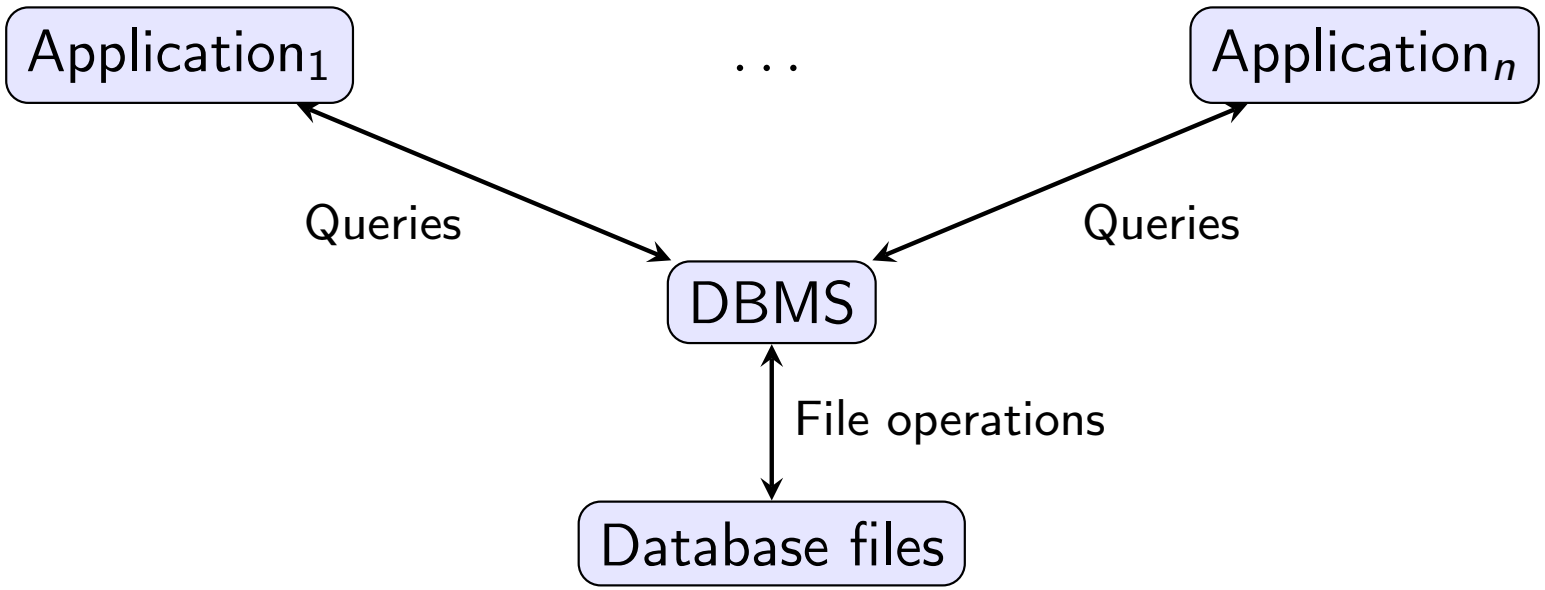
Kontrola Dostępu

Gdy z danych korzysta wielu użytkowników potrzebne jest często ograniczenie uprawnień użytkowników do oglądania lub modyfikowania danych:

- Pracownik może oglądać dane o swoich zarobkach ale nie o zarobkach innych pracowników.
- Pracownik nie może modyfikować swojej pensji.

- Uprawnienia można kontrolować na poziomie aplikacji
- Lepszym rozwiązaniem jest kontrola dostępu na poziomie DBMS. Pozwala to np. na wspólną politykę dla aplikacji dzielących dane.

Architektura Aplikacji Bazodanowych



Aplikacje wykonują operacje na danych wysyłając odpowiednie polecenia do DBMS, nigdy nie wykonują bezpośrednio operacji na plikach danych (do których i tak zwykle nie mają dostępu).

Języki Zapytań

DBMS korzystają ze specjalnych języków zapytań. W językach tych można opisywać wymagane modyfikacje danych, dane, oraz jakie informacje chcemy wyciągnąć z bazy.

Najbardziej rozpowszechnionym językiem zapytań w relacyjnych bazach danych jest **SQL** (*Structured Query Language*).

Praktycznie wszystkie relacyjne bazy danych rozpoznają (jakiś dialekt) SQL

Składowe SQL

- **DML** (*Data Manipulation Language*) — instrukcje służące do wybierania, modyfikowania, usuwania i wstawiania danych do bazy.
- **DDL** (*Data Definition Language*) — instrukcje służące do tworzenia struktur danych takich jak tabele.
- **DCL** (*Data Control Language*) — instrukcje służące do opisu kontroli dostępu do danych.

Przykłady Poleceń SQL

```
SELECT avg(salary) AS "Average Salary", department_id
FROM employees GROUP BY department_id ORDER BY "Average Salary";
```

```
UPDATE employees SET salary = salary + 500
WHERE department_id = 10;
```

```
CREATE TABLE employees(
   employee_id INTEGER PRIMARY KEY,
   name VARCHAR(60) NOT NULL,
   salary NUMBER(6,2) NOT NULL,
   department_id INTEGER);
```

Przykład Programu Java Korzystającego z JDBC

```
Connection conn
     = DriverManager.getConnection(URL,USR,PASS);
Statement stmt = conn.createStatement();
String sql = "SELECT emp_id, name FROM Employees";
ResultSet rs = stmt.executeQuery(sql);
while(rs.next()){
   int id  = rs.getInt("emp_id");
   String name = rs.getString("name");
   System.out.println("emp_id: " + id);
   System.out.println("name: " + name);
}
rs.close();
stmt.close();
conn.close();
```

Systemy Baz Danych

Model Związków Encji

Bartosz Zieliński

WYDZIAŁ FIZYKI
i INFORMATYKI STOSOWANEJ
Uniwersytet Łódzki

Modelowanie Fizyczne, Logiczne i Pojęciowe

- **Model pojęciowy** — o jakiego rodzaju obiektach chcemy przechowywać informacje w bazie, jakie są ich cechy, jakie są powiązania pomiędzy obiektami różnych typów.
  - Tworząc model pojęciowy staramy się zrozumieć strukturę opisywanego w bazie fragmentu rzeczywistości.
- **Model logiczny** — staramy się tu odwzorować model pojęciowy na konkretny model danych stosowany przez DBMS — np. na model relacyjny.
  - Model logiczny zawiera strukturę zmiennych relacyjnych (włącznie z więzami), także strukturę zmiennych modelujących powiązania wiele-do-wielu pomiędzy zmiennymi, widoki, itp.
- **Model fizyczny** — zawiera szczegóły implementacyjne i poprawiające efektywność wykonywania zapytań takie jak partycjonowanie danych, indeksy, rozmieszczenie plików z danymi po różnych dyskach dla zrównoleglenia dostępu i wiele innych.

Rozdzielenie Modelu Fizycznego od Logicznego w Relacyjnych DBMS

Jedną z największych zalet relacyjnych DBMS jest to że kod aplikacji korzystającej z bazy danych zależy (za pośrednictwem poleceń DML i zapytań) jedynie od modelu logicznego a nie fizycznego danych.

Przykład

Indeksy są specjalnymi strukturami które można założyć na kolumnach tabel w celu przyśpieszenia wyszukiwania po tych kolumnach. Np. index na kolumnie **JobID** przyśpieszy wykonanie zapytań typu

```
SELECT * FROM Employees WHERE JobId=10
```

nie jest jednak wymagany aby takie zapytania były wykonalne, ani jego obecność nie uwidacznia się w żaden sposób w kodzie zapytania

Model Związków Encji

Do modelowania pojęciowego najczęściej korzysta się z **modelu związków encji** (*entity/relationship model*, inaczej modelu **ER**) a przede wszystkim ze związanym z tym modelem graficznym językiem diagramów **ER**.

W modelu związków encji wyróżniamy:

- **Encje** (*Entities*) — modelują typy (klasy) rzeczy, np. **osoba, pracownik, wydział, produkt**, itp.
- **Atrybuty** — cechy encji, np. *kolor* dla **produktu**, *imię, nazwisko* i *PESEL* dla **osoby** itp.
- **Związki** — modelują powiązania pomiędzy encjami, np. **pracownik** zatrudniony na **wydziale**.
  - Związki są powiązaniami pomiędzy instancjami encji, tzn. np. nie pracownik jako typ jest zatrudniony na typie wydziału, ale konkretny pracownik na konkretnym wydziale.

Bibliografia

Wykład o diagramach **ER** opiera się przede wszystkim na

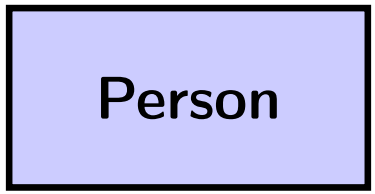
- H. Garcia-Molina, J.D. Ullman, J. Widom „*Systemy Baz Danych. Pełny Wykład*", **Wydawnictwa Naukowo-Techniczne**, Warszawa 2006
- Pável Calado „*The Tikz-er2 Package for Drawing Entity-Relationship Diagrams*" (2010) [Stąd wziąłem część przykładów. Pakiet został też wykorzystany do rysowania diagramów]

Istnieje wiele różnych wariantów notacji diagramów **ER**. Korzystam z wariantu przedstawionego w książce H. Garcia-Moliny.

Interesującą krytykę **ER** można znaleźć w książce

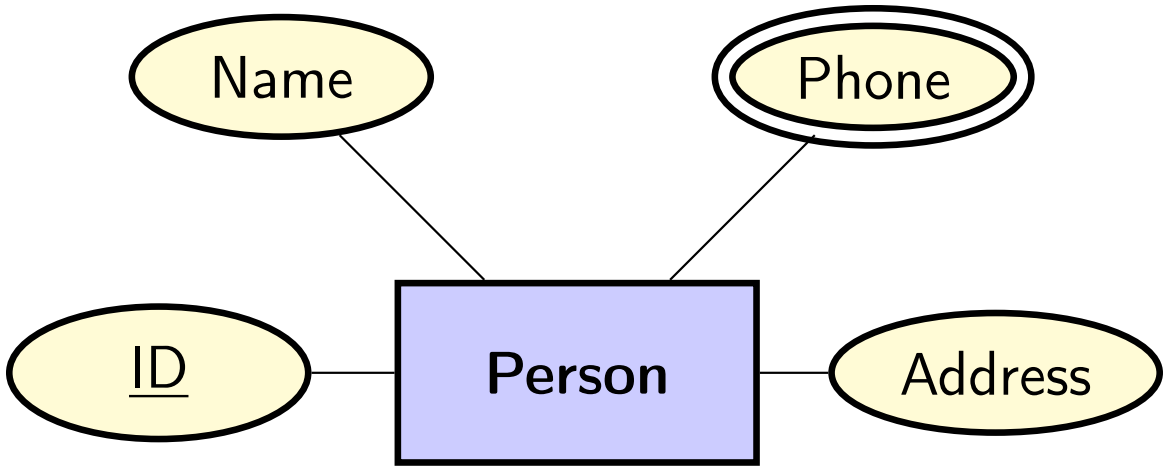
- C.J. Date „*An Introduction to Database Systems*", **Addison Wesley**, 2004

Encje



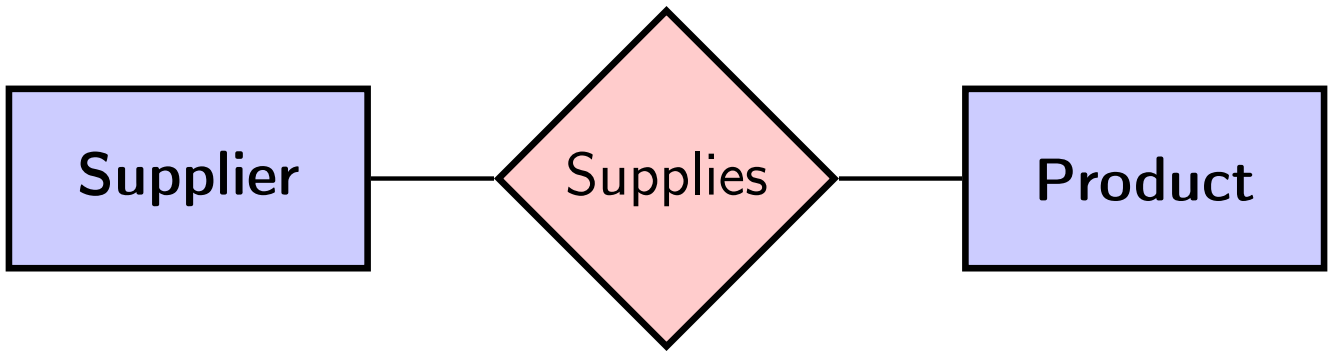
- **Encje** (*Entities*) — modelują typy (klasy) rzeczy, np. **osoba, pracownik, wydział, produkt**, itp.
- Na diagramach są zaznaczane przy pomocy prostokąta z nazwą encji w środku (powyżej: encja **Person**).
- Encje można zinterpretować jako zbiory swoich możliwych instancji — faktycznie istniejących obiektów danego typu.
  - Np. encja **Person** z diagramu powyżej to zbiór wszystkich prawdziwych osób o których informacje chcemy przechowywać w bazie danych.

Atrybuty



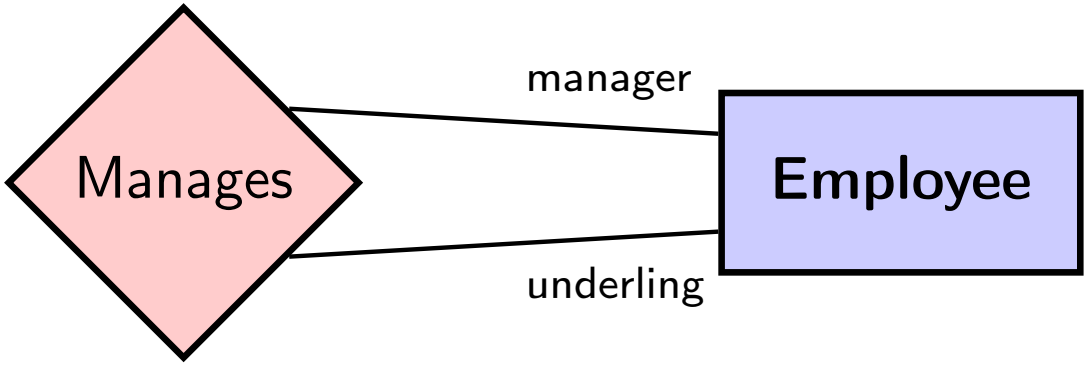
- **Atrybuty** opisują cechy instancji encji, np. *kolor* dla **produktu**, *imię, nazwisko* i *PESEL* dla **osoby** itp.
- **Podkreślamy** (jak ID powyżej) atrybuty jednoznacznie identyfikujące instancję danej encji.
- **Z podwójnym brzegiem** rysujemy te atrybuty które są wielowartościowe. Np. osoba może mieć wiele telefonów.

Związki



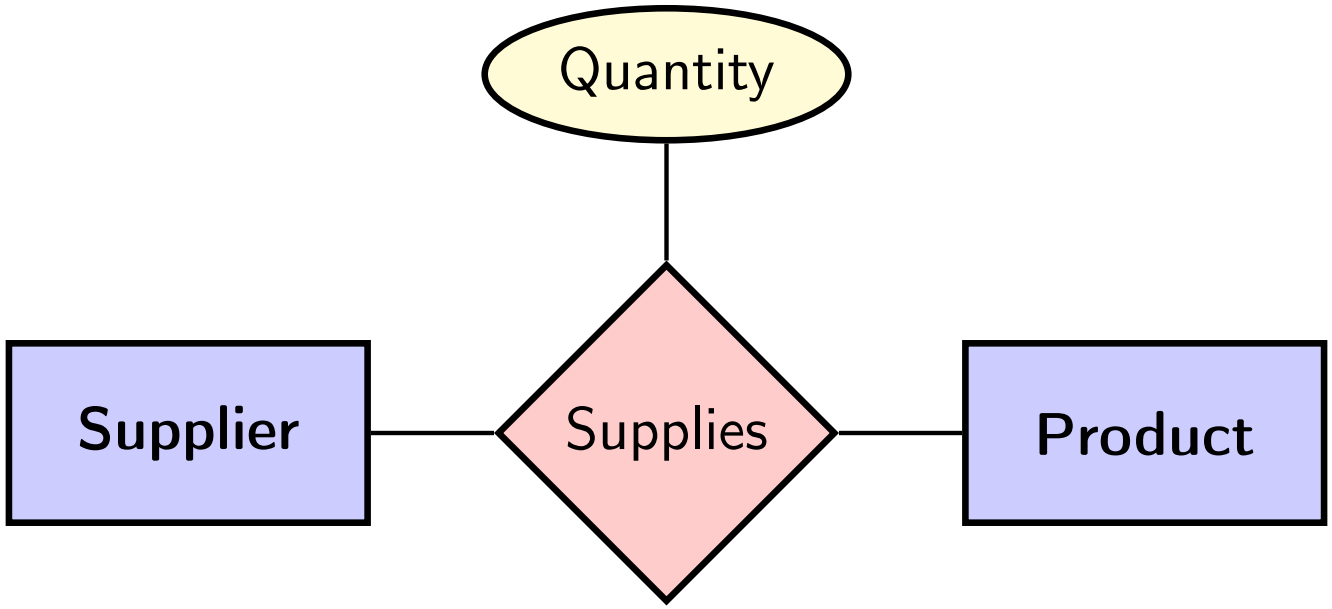
- **Związki** — wyrażają powiązania pomiędzy instancjami encji.
- Powyżej: związek **Supplies** wyrażający fakt że (dany) dostawca **dostarcza** (danego) produktu.
- W większości przypadków **związki** są binarne (łączą dwie, niekoniecznie różne, encje). Model **ER** dopuszcza jednak **związki** łączące więcej niż dwie encje.

Role Uczestników Związku



- Ta sama encja może w **związku** występować wielokrotnie, w różnych rolach (najczęściej oznacza to że związek wyraża powiązanie różnych instancji tej samej encji).
- Na diagramie można nazwać role w jakich występuje w danym związku encja (także gdy występuje jednokrotnie).
- Np. powyżej mamy związek **manages** łączący pracownika będącego podwładnym (**underling**) z jego szefem (**manager**) który oczywiście jest także pracownikiem

Atrybuty Związków



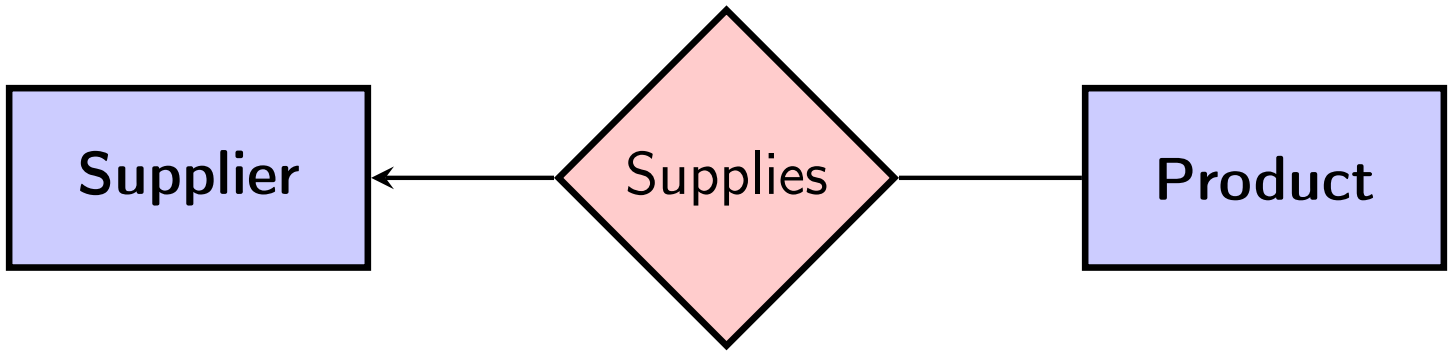
- **Związki** mogą mieć własne **atrybuty** opisujące cechy danego związku.
- Np. w przykładzie powyżej atrybut **Quantity** opisuje ilość danego towaru **dostarczanego** przez danego dostawcę.

Krotność Związków

- Na diagramach **ER** można zaznaczać **krotność związków**.
- Brzeg po stronie „**jeden**” oznaczamy strzałką, brzeg po stronie „**wiele**” pozostawiamy bez oznaczenia.
- Krotność związku najłatwiej zrozumieć dla **związków binarnych** gdzie wyróżniamy:
  - Związek **jeden do wielu** (i **wiele do jeden**)
  - Związek **wiele do wielu**
  - Związek **jeden do jeden**

Krotność Związków

Związek Jeden do Wielu



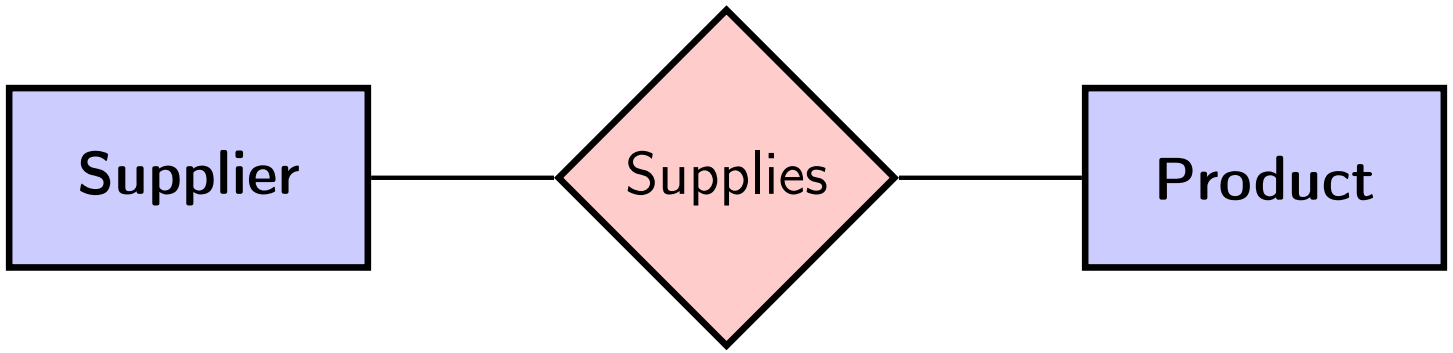
Dany **produkt** dostarczany jest przez **co najwyżej** jednego **dostawcę** (ale dany dostawca może dostarczać dowolną ilość produktów).

Związek jest jeden do wielu (lub wiele do jeden)

gdy instancja encji stojącej po stronie „**wiele**" może być w tym związku z **co najwyżej** jedną instancją encji stojącej po stronie „**jeden**" (oznaczonej strzałką). Instancja po stronie „**jeden**" może być w związku z dowolną ilością instancji (także żadną) po stronie „**wiele**".

Krotność Związków

Związek Wiele do Wielu



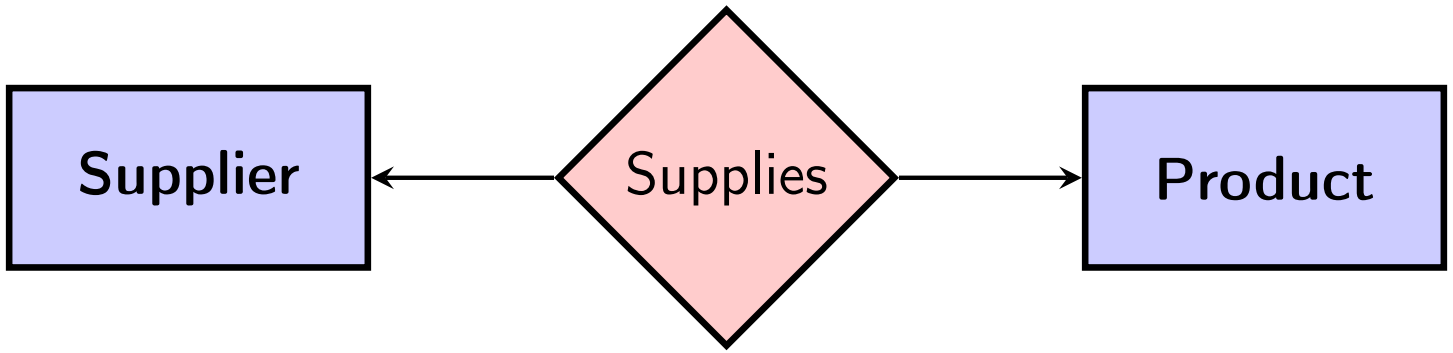
Dany **dostawca** może **dostarczać** dowolną ilość **produktów**, a dany **produkt** może być **dostarczany** przez dowolną ilość **dostawców**.

Związek jest jeden do wielu

kiedy instancje jednej z encji mogą być w związku z dowolną ilością instancji drugiej encji i odwrotnie.

Krotność Związków

Związek Jeden do Jeden



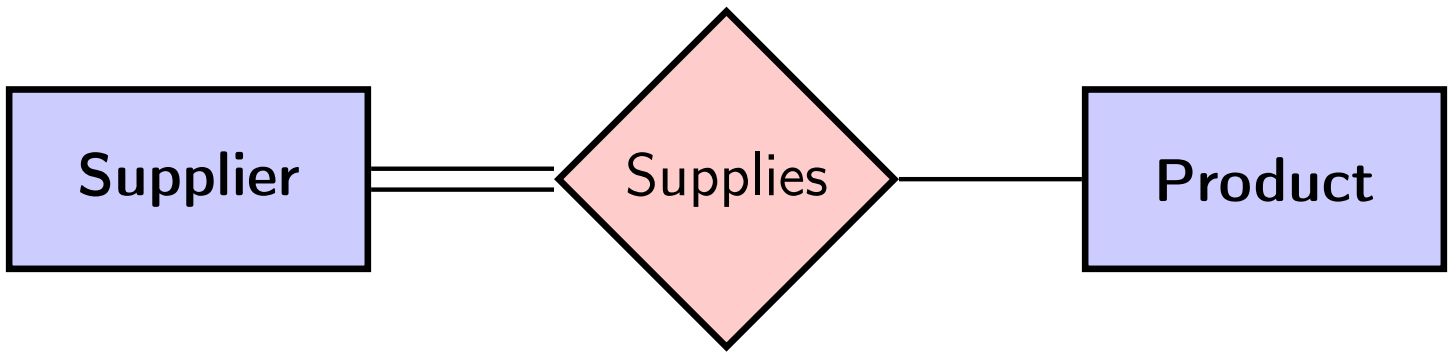
Dany **produkt** może być **dostarczany** przez co najwyżej jednego **dostawcę** a każdy **dostawca dostarcza** co najwyżej jeden **produkt**

Związek jest Jeden do Jeden

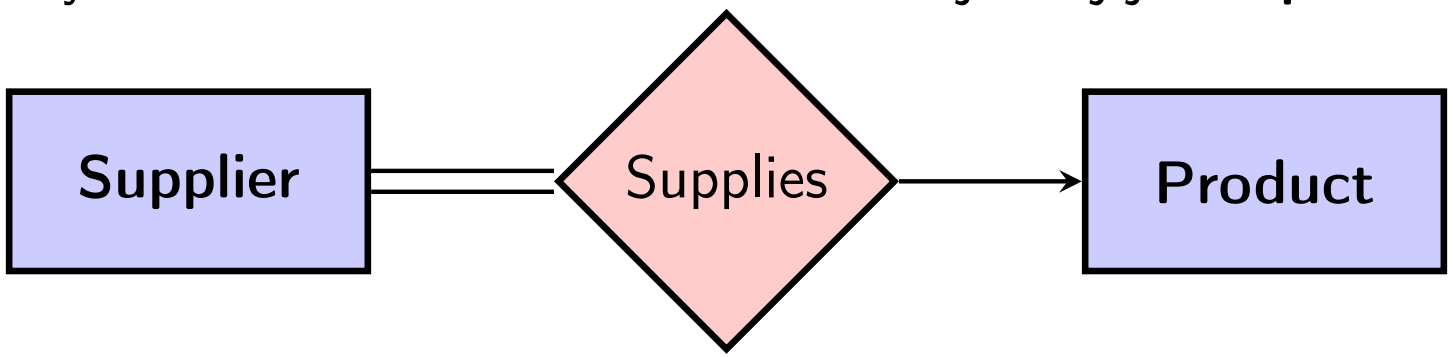
kiedy instancje jednej z encji mogą być w związku z co najwyżej jedną instancją drugiej encji i odwrotnie.

Całkowite Uczestnictwo w Związku

Mówimy że encja **uczestniczy całkowicie w związku** gdy każda instancja encji musi uczestniczyć w danym związku z co najmniej jedną instancją drugiej encji uczestniczącej w związku.

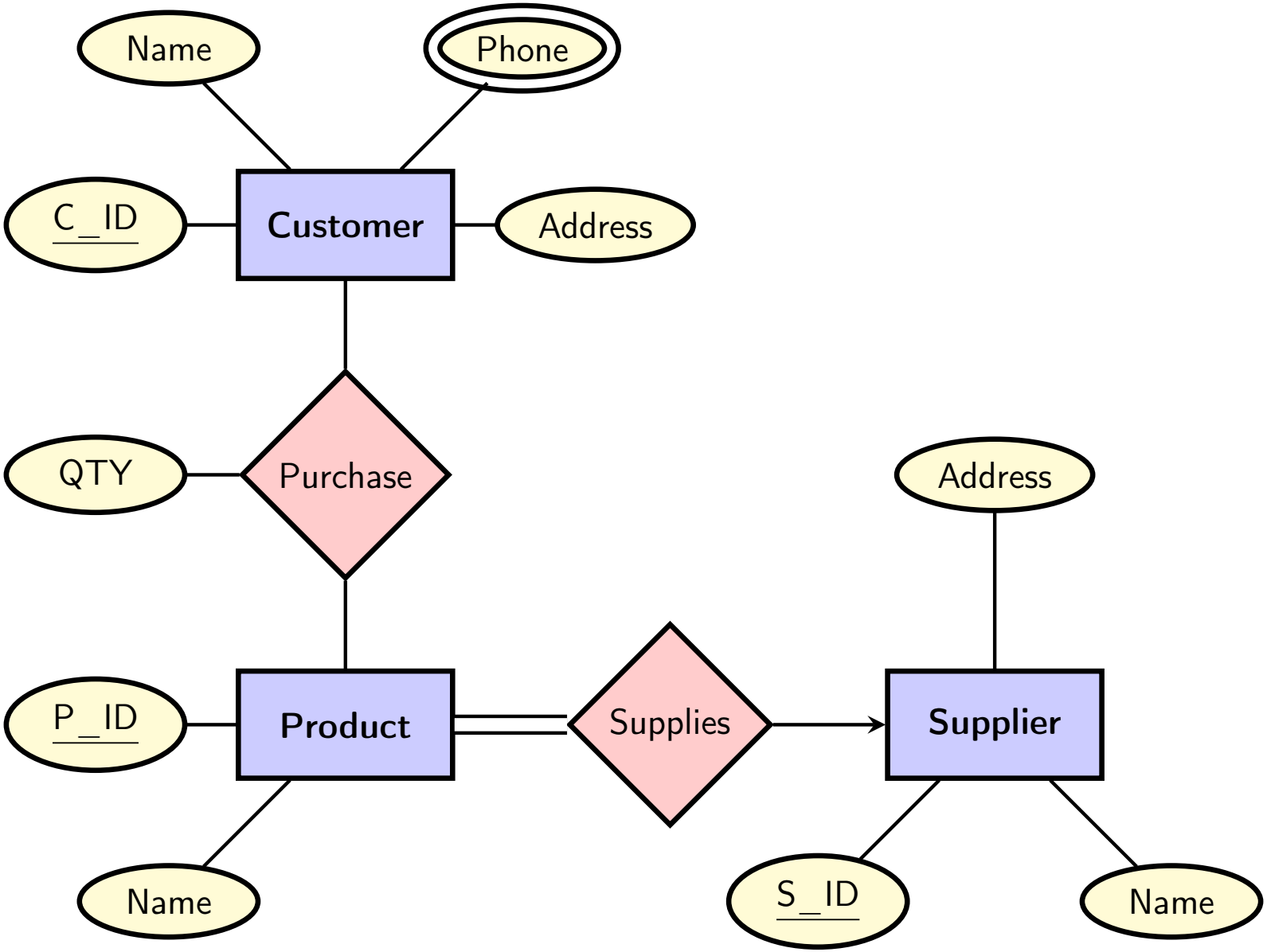


Dany **dostawca** musi **dostarczać** co najmniej jeden **produkt**



Dany **dostawca** musi **dostarczać** **dokładnie** jeden **produkt**

Pełen Przykład



Od Diagramu ER do Schematu Bazy Danych

Część I

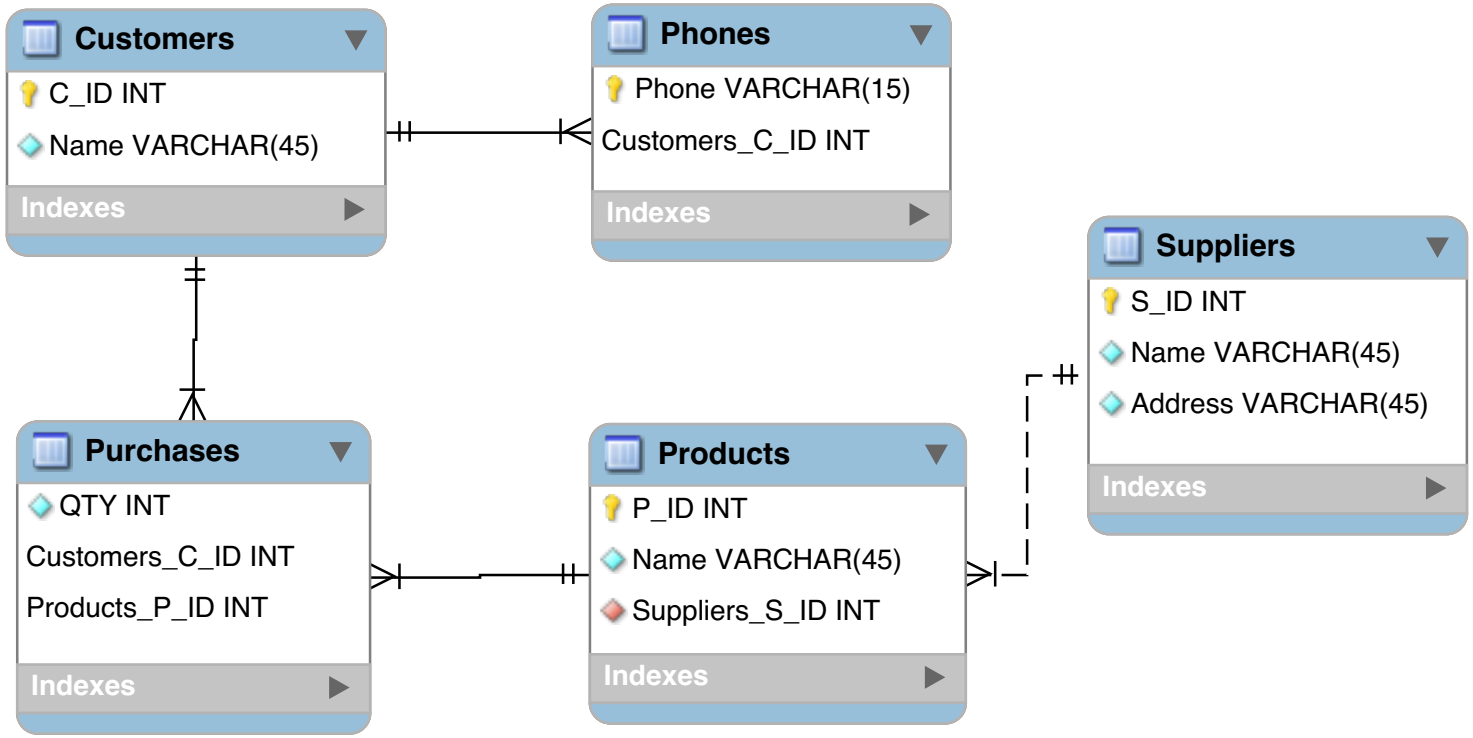
- Każdemu **atrybutowi** musimy przypisać typ wartości.
- Dla każdej **encji** i każdego **związku** (poza związkami binarnymi **jeden do wielu** lub **jeden do jeden** tworzymy zmienną relacyjną.
- **Atrybutami** tych zmiennych będą atrybuty jednowartościowe odpowiadających **encji** i **związków**. Dodatkowo
  - dla każdej tabeli odpowiadającej **związkowi** dodajemy klucze obce do wszystkich **encji** biorących udział w związku. Kombinacja tych kluczy będzie stanowiła zwykle klucz główny,
  - dla każdego **związku wiele do jeden** dodajemy dla tabeli odpowiadającej **encji** stojącej po stronie **wiele** klucz obcy do encji stojącej po stronie **jeden**.
  - W przypadku **związku jeden do jeden** wybieramy jedną z encji biorących udział w związku aby dodać do odpowiadającej tabeli klucz obcy na który dodatkowo nakładamy wiąz **UNIQUE**.

- Dla każdego **atrybutu wielowartościowego** A tworzymy zmienną relacyjną z następującymi dwoma atrybutami:
  1. atrybutem przechowującym pojedyńczą wartość A
  2. kluczem obcym do encji do której przynależy A.

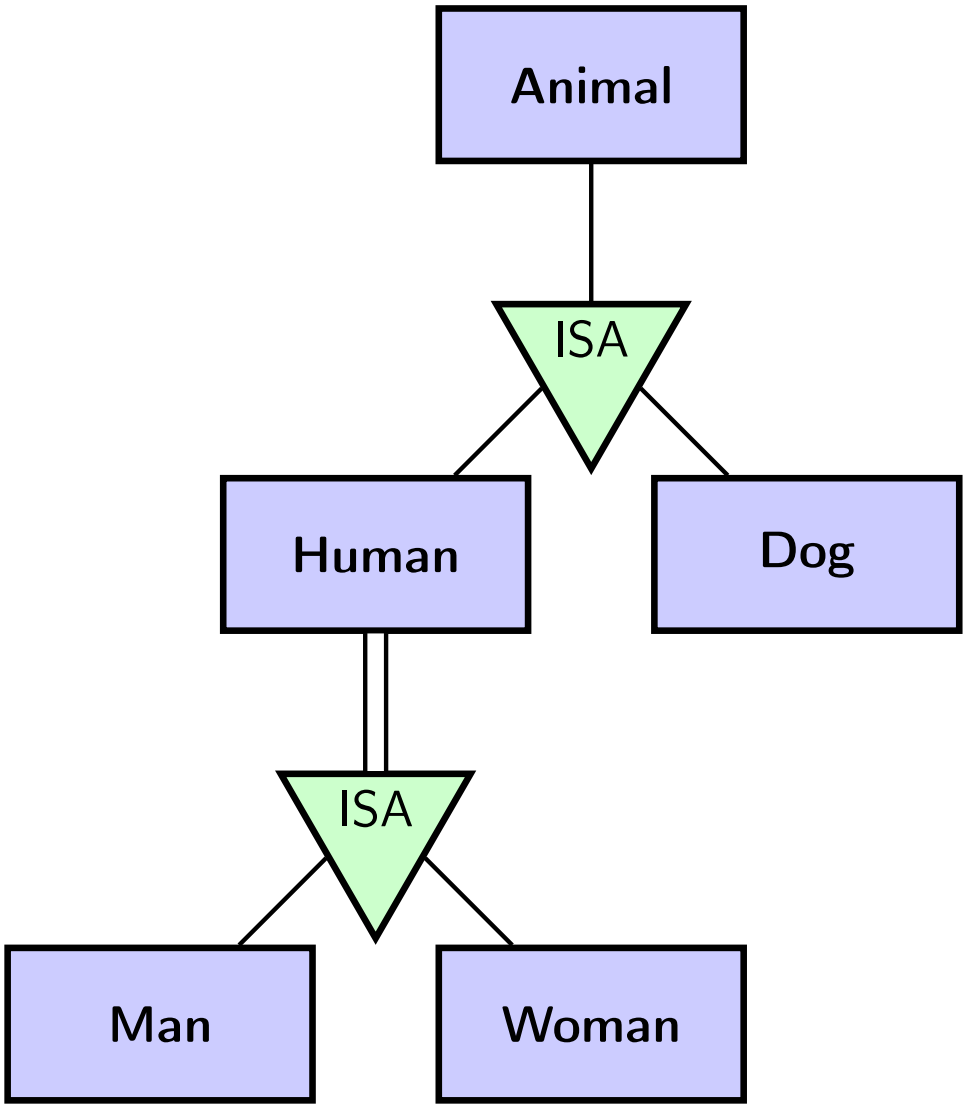
Diagramy ER i Normalizacja

- Na ogół, jeśli dokonamy poprawnego wyboru **encji**, **atrybutów** i **związków** (z odpowiednią krotnością), analizując strukturę danych które chcemy przechowywać w bazie, wówczas otrzymany bazę danych której schemat zawiera znormalizowane (co najmniej do **3NF**) zmienne relacyjne.
- Można to również odwrócić, mówiąc że cechą dobrego schematu **ER** jest to że daje się go bezpośrednio przetłumaczyć na odpowiednio znormalizowany schemat relacyjnej bazy danych.

Przykład


Customers
C_ID INT
Name VARCHAR(45)
Indexes
Phones
Phone VARCHAR(15)
Customers_C_ID INT
Indexes
Suppliers
S_ID INT
Name VARCHAR(45)
Address VARCHAR(45)
Indexes
Purchases
QTY INT
Customers_C_ID INT
Products_P_ID INT
Indexes
Products
P_ID INT
Name VARCHAR(45)
Suppliers_S_ID INT
Indexes

Specjalizacja/Generalizacja



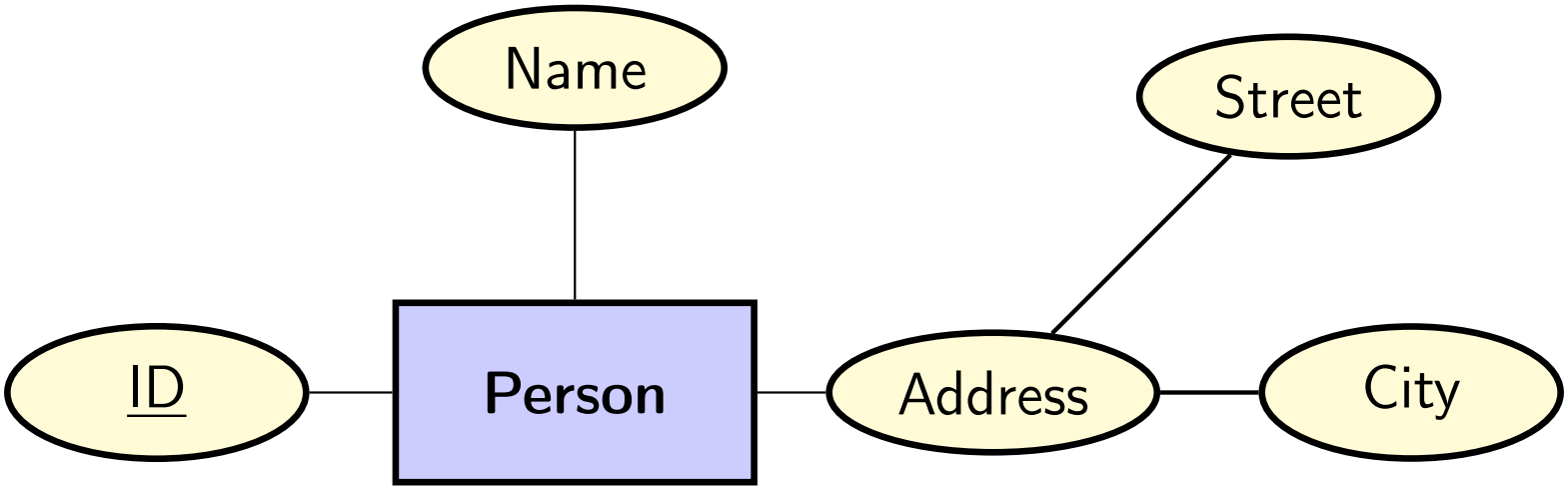
Związek **ISA** jest specjalnym rodzajem **związku** który można porównać do dziedziczenia w programowaniu obiektowym.

- Każdy **człowiek** i **pies** jest **zwierzęciem**
- Każdy człowiek jest **mężczyzną** albo **kobietą**

Encje będące **specjalizacją** innych encji dziedziczą atrybuty po tych encjach ale mogą też dodawać własne atrybuty.

- **Człowiek** jest specjalizacją **zwierzęcia**,
- **zwierzę** generalizuje **człowieka**

Złożone Atrybuty



- Atrybut mający własne atrybuty nazywa się **atrybutem złożonym**
- Można uznać że atrybut **Address** w przykładzie powyżej ma typ rekordowy z dwoma polami **Street** i **City**
- **Atrybutów złożonych** nie można reprezentować bezpośrednio w czystym modelu relacyjnym — encji **Person** będzie odpowiadała zmienna relacyjna z czterema atrybutami: **ID**, **Name**, **Street** i **City**

Obiektowo Relacyjne i Postrelacyjne DBMS-y

Wielowartościowych i złożonych atrybutów ani związków **ISA** pomimo ich naturalności i użyteczności nie da się reprezentować **bezpośrednio** w modelu relacyjnym, choć można to zrobić stosując dodatkowe zmienne relacyjne z nałożonymi więzami referencyjnymi.

Postrelacyjne i Obiektowo-Relacyjne DBMS-y których przykładem są **PostgreSQL** i **Oracle** pozwalają między innymi na

- Definiowanie własnych (także złożonych) typów przez użytkownika razem z operacjami domenowymi na nich wykonywalnymi
- Pozwalają na przechowywanie w atrybutach kolekcji elementów a nie tylko pojedyńczych wartości
- Pozwalają na deklarowanie zmiennych relacyjnych dziedziczących po innych tabelach.
- Pozwalają także na definiowanie kodu proceduralnego wykonywanego i przechowywanego przez DBMS w postaci **podprogramów składowanych** i **triggerów** (wyzwalaczy).

Systemy Baz Danych

Algebra Relacyjna

Bartosz Zieliński

WYDZIAŁ FIZYKI
i INFORMATYKI STOSOWANEJ
Uniwersytet Łódzki

Algebra Relacyjna

- Definiuje operacje na relacjach dające w wyniku relacje.
- Zbiór wszystkich możliwych relacji wraz z tymi operacjami tworzy algebrę w sensie matematycznym.
- Operacje można składać w bardziej złożone wyrażenia.
- Pomiędzy operacjami występują nietrywialne związki pozwalające na wyrażenie tej samej relacji na różne sposoby.
- Wyrażenia algebry relacyjnej definiują zawsze zapytania niezależne od dziedziny.

Algebra Relacyjna a Implementacja Wykonywania Zapytań w DBMS

- Wystarczy aby DBMS implementował operatory relacyjne.
- Aby wykonać dane zapytanie wystarczy teraz wykonać po kolei operacje z wyrażenia definiującego to zapytanie.

W rzeczywistych RDBMS-ach

- Interpreter SQL tłumaczy zapytanie SQL na wyrażenie **algebry relacyjnej**.
- Optymalizator może przekształcić to wyrażenie na **równoważne** (definiujące tą samą relację) ale wykonujące się szybciej, korzystając z reguł **algebry relacyjnej**.
- Dla niektórych operatorów istnieje kilka algorytmów je implementujących wybieranych przez optymalizator.

Operatory w Algebrze Relacyjnej

- **Przemianowanie** atrybutów (*rename*)
- **Projekcje** (*projections*)
- **Selekcje** (*selections*)
- Operatory teoriomnogościowe
  - **Unia**, **Przecięcie**, **Różnica**, **Iloczyn kartezjański**
- Operatory złączenia
  - **złączenie naturalne**, **złączenie równościowe** (*equijoin*), θ-złączenia, **złączenia zewnętrzne** (*outer joins*) itp.
- **Dzielenie** relacji
- **Obliczenia domenowe** (*domain computations*)
- **Agregacje**

Operacja Przemianowania Atrybutu $\rho_{B/A}$

Operacja $\rho_{B/A}(R)$ **przemianowania atrybutu** A na B zamienia w relacji R nazwę atrybutu A na B (aby wynik operacji był dobrze zdefiniowany wymagamy aby $A \in \text{Attr}(R)$ i $B \notin \text{Attr}(R)$).

Jobs

JobId	Name	MinSalary	MaxSalary
1	IT Specialist	8000	20000
2	Sales Specialist	5000	9000
3	Administration	7000	10000

$\rho_{\text{MinimalnaPensja}/\text{MinSalary}}(\textbf{Jobs})$

JobId	Name	MinimalnaPensja	MaxSalary
1	IT Specialist	8000	20000
2	Sales Specialist	5000	9000
3	Administration	7000	10000

Unia, Przecięcie i Różnica

Operacje **unii** $(R \cup S)$, **przecięcia** $(R \cap S)$ i **różnicy** $(R \setminus S)$ są zdefiniowane dla relacji R i S takich że $\mathrm{Attr}(R) = \mathrm{Attr}(S)$. Wtedy

$$\mathrm{Attr}(R \cap S) = \mathrm{Attr}(R \cup S) = \mathrm{Attr}(R \setminus S) := \mathrm{Attr}(R)$$

i

- $\mathrm{Rows}(R \cup S) := \mathrm{Rows}(R) \cup \mathrm{Rows}(S)$
  (krotki w $R \cup S$ to te i tylko te krotki które są w R lub S)
- $\mathrm{Rows}(R \cap S) := \mathrm{Rows}(R) \cap \mathrm{Rows}(S)$
  (krotki w $R \cup S$ to te i tylko te krotki które są w R i w S)
- $\mathrm{Rows}(R \setminus S) := \mathrm{Rows}(R) \setminus \mathrm{Rows}(S)$
  (krotki w $R \cup S$ to te i tylko te krotki które są w R i których nie ma w S)

Przykład Unii, Przecięcia i Różnicy

R

A	B
1	10
2	20
3	30
4	40

S

A	B
1	10
2	20
5	50

$R \cup S$

A	B
1	10
2	20
3	30
4	40
5	50

$R \cap S$

A	B
1	10
2	20

$R \setminus S$

A	B
3	30
4	40

Rzutowanie Krotek na Podzbiór Atrybutów

Definicja

Niech t będzie krotką i niech $X \subseteq \mathrm{Attr}(t)$. Wówczas **rzutowaniem krotki t na podzbiór atrybutów X** nazywamy krotkę $t|_X$ zdefiniowaną przez

$$\mathrm{Attr}(t|_X) := X, \quad (t|_X).A = t.A \text{ dla każdego } A \in X.$$

Przykład

$$t = \begin{array}{ccc} \mathbf{A} & \mathbf{B} & \mathbf{C} \\ \hline 1 & 2 & 3 \end{array}, \quad t|_{\{\mathbf{A},\mathbf{C}\}} = \begin{array}{cc} \mathbf{A} & \mathbf{C} \\ \hline 1 & 3 \end{array}.$$

Złączenia Krotek

Definicja

Niech t_1 i t_2 będą krotkami takimi że $t_1|_X = t_2|_X$, gdzie $X := \mathrm{Attr}(t_1) \cap \mathrm{Attr}(t_2)$ (warunek ten jest zawsze spełniony gdy $X = \emptyset$). Wówczas **złączeniem** t_1 **i** t_2 nazywamy krotkę $t_1 \bowtie t_2$ t.ż. $\mathrm{Attr}(t_1 \bowtie t_2) := \mathrm{Attr}(t_1) \cup \mathrm{Attr}(t_2)$ i

$$(t_1 \bowtie t_2).A := \begin{cases} t_1.A & \text{gdy } A \in \mathrm{Attr}(t_1) \\ t_2.A & \text{gdy } A \in \mathrm{Attr}(t_2) \end{cases}.$$

Przykład

$$\begin{array}{cc} A & B \\ \hline 1 & 2 \end{array} \bowtie \begin{array}{cc} B & C \\ \hline 2 & 3 \end{array} = \begin{array}{ccc} A & B & C \\ \hline 1 & 2 & 3 \end{array}, \qquad \begin{array}{cc} A & B \\ \hline 1 & 2 \end{array} \bowtie \begin{array}{c} C \\ \hline 3 \end{array} = \begin{array}{ccc} A & B & C \\ \hline 1 & 2 & 3 \end{array}$$

Iloczyn Kartezjański

Iloczyn kartezjański relacji jest określony dla par relacji o rozłącznych zbiorach atrybutów. Niech R i S będą relacjami takimi że $\mathrm{Attr}(R) \cap \mathrm{Attr}(S) = \emptyset$. Wówczas **iloczyn kartezjański** $R \times S$ jest relacją o atrybutach $\mathrm{Attr}(R \times S) := \mathrm{Attr}(R) \cup \mathrm{Attr}(S)$ i krotkach

$$\mathrm{Rows}(R \times S) := \{t_1 \bowtie t_2 \mid t_1 \in R, t_2 \in S\}$$

Zauważmy że $R \times S = S \times R$ (ponieważ $t_1 \bowtie t_2 = t_2 \bowtie t_1$).
Na przykład

R

A	B
1	10
2	20

S

C	D
3	30
4	40

$R \times S$

A	B	C	D
1	10	3	30
1	10	4	40
2	20	3	30
2	20	4	40

Projekcja na Podzbiór Atrybutów

Niech R będzie relacją i niech $X \subseteq \mathrm{Attr}(R)$. Wtedy $\pi_X(R)$ jest relacją o atrybutach X i krotkach

$$\mathrm{Rows}\big(\pi_X(R)\big) := \big\{ t|_X \;\big|\; t \in R \big\}$$

Przykład projekcji na podzbiór atrybutów:

Employees

Id	FirstName	LastName	Salary	JobId
1	Toru	Takemitsu	10000.11	1
2	Philip	Glass	9000.00	3
3	Michael	Nyman	10000.50	1
4	Henryk	Górecki	11000.00	1
5	Thomas	Tallis	8000.80	2
6	Arvo	Pärt	15000.70	1
7	Arnold	Schönberg	6000.00	2
8	Anton	Webern	6500.12	2
9	Alban	Berg	6750.50	2
10	Olivier	Messiaen	9500.00	3

$\pi_{\{\mathbf{JobId}\}}(\mathbf{Employees})$

JobId
1
2
3

Operacja Selekcji

Przypuśćmy że R jest relacją i niech ϕ będzie warunkiem określonym na krotkach o atrybutach $\mathrm{Attr}(R)$. Wówczas $\sigma_\phi(R)$ jest relacją o atrybutach $\mathrm{Attr}(\sigma_\phi(R)) := \mathrm{Attr}(R)$ i krotkach

$$\mathrm{Rows}(\sigma_\phi(R)) := \{t \mid t \in R \text{ i } t \text{ spełnia } \phi\}$$

Zauważmy że

$$\sigma_{\phi \text{ and } \psi}(R) = \sigma_\phi(R) \cap \sigma_\psi(R)$$
$$\sigma_{\phi \text{ or } \psi}(R) = \sigma_\phi(R) \cup \sigma_\psi(R)$$
$$\sigma_{\text{not } \phi}(R) = R \setminus \sigma_\phi(R)$$

Operacje Selekcji cd.

Atomowymi operacjami selekcji

nazywamy operacje postaci $\sigma_{C\theta D}(R)$ i $\sigma_{C\theta v}(R)$ gdzie $C, D \in \mathrm{Attr}(R)$, $v \in \mathcal{U}$, a θ jest operatorem porównania takim jak $<$, $\leq$, $=$, itp.

Proste operacje selekcji

tworzone są z operacji atomowych przy pomocy operatorów boolowskich.

Złożone operacje selekcji

mogą korzystać także z kwantyfikatorów po innych relacjach.

Przykład Atomowej Operacji Selekcji

Employees

Id	FirstName	LastName	Salary	JobId
6	Arvo	Pärt	15000.70	1
7	Arnold	Schönberg	6000.00	2
8	Anton	Webern	6500.12	2
9	Alban	Berg	6750.50	2
10	Olivier	Messiaen	11000.00	3

$\sigma_{\mathbf{Salary} \geq 10000}(\mathbf{Employees})$

Id	FirstName	LastName	Salary	JobId
6	Arvo	Pärt	15000.70	1
10	Olivier	Messiaen	11000.00	3

Przykład Prostej Operacji Selekcji

Employees

Id	FirstName	LastName	Salary	JobId
6	Arvo	Pärt	15000.70	1
7	Arnold	Schönberg	6000.00	2
8	Anton	Webern	6500.12	2
9	Alban	Berg	6750.50	2
10	Olivier	Messiaen	11000.00	3

$\sigma_{\mathbf{JobId}=3 \vee \mathbf{Salary}>15000}(\mathbf{Employees})$

Id	FirstName	LastName	Salary	JobId
6	Arvo	Pärt	15000.70	1
10	Olivier	Messiaen	11000.00	3

Operacja Złączenia Naturalnego

Przypuśćmy że relacje R i S są takie że $\mathrm{Attr}(R) \cap \mathrm{Attr}(S) \neq \emptyset$.

Złączenie naturalne $R \bowtie S$

jest relacją o atrybutach $\mathrm{Attr}(R \bowtie S) := \mathrm{Attr}(R) \cup \mathrm{Attr}(S)$ i krotkach

$$\mathrm{Rows}(R \bowtie S) = \left\{ t_1 \bowtie t_2 \mid t_1 \in R \wedge t_2 \in S \wedge t_1|_{\mathrm{Attr}_R \cap \mathrm{Attr}_S} = t_2|_{\mathrm{Attr}_R \cap \mathrm{Attr}_S} \right\}.$$

Przypomnijmy że

$$(t_1 \bowtie t_2)(A) := \begin{cases} t_1.A & \text{gdy } A \in \mathrm{Attr}(R) \\ t_2.A & \text{gdy } A \in \mathrm{Attr}(S) \end{cases} \qquad (= t_2 \bowtie t_1(A))$$

Zauważmy że $R \bowtie S = S \bowtie R$.

Przykład Operacji Złączenia Naturalnego

Employees

Id	FirstName	LastName	Salary	JId
1	Toru	Takemitsu	10000.11	1
2	Philip	Glass	9000.00	3
3	Michael	Nyman	10000.50	1
4	Henryk	Górecki	11000.00	1
5	Thomas	Tallis	8000.80	2
6	Arvo	Pärt	15000.70	1
7	Arnold	Schönberg	6000.00	2
8	Anton	Webern	6500.12	2
9	Alban	Berg	6750.50	2
10	Olivier	Messiaen	9500.00	3

Jobs

JId	Name	Min	Max
1	IT Specialist	8000	20000
2	Sales Specialist	5000	9000
3	Administration	7000	10000

Employees $\bowtie$ Jobs

Id	FirstName	LastName	Salary	JId	Name	Min	Max
1	Toru	Takemitsu	10000.11	1	IT Specialist	8000	20000
2	Philip	Glass	9000.00	3	Administration	7000	10000
3	Michael	Nyman	10000.50	1	IT Specialist	8000	20000
4	Henryk	Górecki	11000.00	1	IT Specialist	8000	20000
5	Thomas	Tallis	8000.80	2	Sales Specialist	5000	9000
6	Arvo	Pärt	15000.70	1	IT Specialist	8000	20000
7	Arnold	Schönberg	6000.00	2	Sales Specialist	5000	9000
8	Anton	Webern	6500.12	2	Sales Specialist	5000	9000
9	Alban	Berg	6750.50	2	Sales Specialist	5000	9000
10	Olivier	Messiaen	9500.00	3	Administration	7000	10000

Złączenie Naturalne a Inne Operacje

Złączenie naturalne można zdefiniować korzystając z operacji **projekcji**, **selekcji**, **przemianowania** oraz **iloczynu kartezjańskiego**.

Niech $\mathrm{Attr}(R) \cap \mathrm{Attr}(S) = \{A_1, \ldots, A_n\}$ i niech $\{B_1, \ldots, B_n\} \subseteq \mathcal{A} \setminus \mathrm{Attr}(R) \cup \mathrm{Attr}(S)$. Oznaczmy $\sigma_{\vec{A}=\vec{B}} := \sigma_{A_1=B_1 \wedge \cdots \wedge A_n=B_n}$ oraz $\rho_{\vec{B}/\vec{A}} := \rho_{B_1/A_1} \circ \cdots \circ \rho_{B_n/A_n}$. Wówczas

$$R \bowtie S = \pi_{\mathrm{Attr}(R) \cup \mathrm{Attr}(S)} \left(\sigma_{\vec{A}=\vec{B}} \left(R \times \rho_{\vec{B}/\vec{A}}(S) \right) \right)$$

Skoro **złączenie naturalne** (inne złączenia zresztą też) może zostać zdefiniowane przy pomocy innych operacji po co traktować je jako odrębną operację komplikując algebrę relacyjną?

Złączenia, Iloczyny Kartezjańskie i Implementacja

Przypuśćmy że $\mathrm{Attr}(R) \cap \mathrm{Attr}(S) = \{A\}$, $B \notin \mathrm{Attr}(R) \cup \mathrm{Attr}(S)$ $|R| = N$ i $|S| = M$. Wówczas $|R \times \rho_{B/A}(S)| = MN$ ale $|R \bowtie S| \leq N$ jeśli wartość A jednoznacznie identyfikuje krotkę w S.

Istnieje wiele algorytmów implementacji złączenia niewymagających tworzenia danych pośrednich o rozmiarze MN.

- Przykładem jest naiwny algorytm który wymaga MN operacji ale korzysta jedynie z $|R| + |S| + |R \bowtie S|$ rekordów w pamięci:

  for all $t_1 \in R$
  for all $t_2 \in S$
  if$(t_1(A) = t_2(A))$ **yield** $t_1 \bowtie t_2$

- istnieją też znacznie bardziej efektywne algorytmy zarówno korzystające jak i nie korzystające z **indeksów**.

θ-Złączenia i Złączenia Warunkowe

Przypuśćmy że R i S są relacjami takimi że $\text{Attr}(R) \cap \text{Attr}(S) = \emptyset$. Niech $C \in \text{Attr}_R$, $D \in \text{Attr}_S$ i niech θ będzie jednym z binarnych operatorów porównania $<$, $>$, $\leq$, $\geq$, $=$, itp.

θ-złączenie relacji R i S (na C, D)

jest zdefiniowane jako $R \bowtie_{C\theta D} S := \sigma_{C\theta D}(R \times S)$

- W ogólności mamy dla pewnego warunku ϕ będącego kombinacją boolowską warunków postaci $A\theta B$, gdzie $A \in \text{Attr}(R)$ a $B \in \text{Attr}(S)$, **złączenie warunkowe** $R \bowtie_\phi S := \sigma_\phi(R \times S)$.
- Szczególny przypadek $R \bowtie_{A_1=B_1 \wedge \cdots \wedge A_k=B_k} S$ nazywa się **złączeniem równościowym** (*equijoin*).

Dwa Oznaczenia Pomocnicze

Niech X będzie skończonym zbiorem atrybutów. Niech $\mathbf{null}_X$ będzie relacją zawierającą pojedyńczą krotkę o atrybutach X i wartości $\mathbf{NULL}$ dla każdego z tych atrybutów, tzn., $\mathrm{Attr}(\mathbf{null}_X) = X$ i $\mathrm{Rows}(\mathbf{null}_X) = \{t\}$, gdzie $t.A = \mathbf{NULL}$ dla każdego $A \in X$.

Przypuśćmy że złączenie naturalne $R \bowtie S$ jest określone. Oznaczmy przez R_{-S} relację składającą się z krotek należących do R dla których nie istnieją odpowiadające (w sensie złączenia) krotki w relacji S:

$$R_{-S} := R \setminus \pi_{\mathrm{Attr}_R}(R \bowtie S)$$

Dodatkowo oznaczmy $R^{\bowtie}_{-S} := R_{-S} \times \mathbf{null}_{\mathrm{Attr}(S) \setminus \mathrm{Attr}(R)}$.

Złączenia Zewnętrzne

Złączenia zewnętrzne, zdefiniowane dla tych samych par relacji co złączenia naturalne, oprócz złączonych krotek zawierają także wszystkie krotki z jednej lub obu relacji biorących udział w złączeniu dla których nie istnieje odpowiadająca krotka z drugiej relacji

Wyróżniamy następujące rodzaje złączeń zewnętrznych:

- **Lewe złączenie zewnętrzne** (*left outer join*)

$$R \overset{(+)}{\bowtie} S := R \bowtie S \cup R^{\bowtie}_{-S}$$

- **Prawe złączenie zewnętrzne** (*right outer join*)

$$R \overset{(+)}{\bowtie} S := R \bowtie S \cup S^{\bowtie}_{-R}$$

- **Pełne złączenie zewnętrzne** (*full outer join*)

$$R \overset{(+)}{\bowtie}{}^{(+)} S := R \bowtie S \cup R^{\bowtie}_{-S} \cup S^{\bowtie}_{-R}$$

Przykłady Złączeń Zewnętrznych

R

X	Y
10	1
20	2
30	2
100	3

S

Y	Z
1	X1
2	X2
4	X4

$R \overset{(+)}{\bowtie} S$

X	Y	Z
10	1	X1
20	2	X2
30	2	X2
100	3	**NULL**

$R \bowtie^{(+)} S$

X	Y	Z
10	1	X1
20	2	X2
30	2	X2
NULL	4	X4

$R \overset{(+)}{\bowtie}{}^{(+)} S$

X	Y	Z
10	1	X1
20	2	X2
30	2	X2
100	3	**NULL**
NULL	4	X4

$R \bowtie S$

X	Y	Z
10	1	X1
20	2	X2
30	2	X2

Warunkowe Złączenia Zewnętrzne

Także złączenia warunkowe występują w (trzech) wersjach zewnętrznych

R

X
1
2
10

S

Y
0
9

$R^{(+)}\bowtie_{X<Y} S$

X	Y
1	9
2	9
10	NULL

$R \bowtie^{(+)}_{X<Y} S$

X	Y
NULL	0
1	9
2	9

$R^{(+)}\bowtie^{(+)}_{X<Y} S$

X	Y
NULL	0
1	9
2	9
10	NULL

$R \bowtie_{X<Y} S$

X	Y
1	9
2	9

Operator Dzielenia Relacji

Chcemy mieć operator $\div$ będący odwrotnością iloczynu kartezjańskiego w tym sensie że

$$(K \times S) \div S = K$$

Przypuśćmy że R is S są relacjami takimi że $\mathrm{Attr}(S) \subsetneq \mathrm{Attr}(R)$. Oznaczmy $X := \mathrm{Attr}(R) \setminus \mathrm{Attr}(S)$. Wówczas $R \div S$ jest relacją o atrybutach X zdefiniowaną jako największa relacja K taka że

$$K \subseteq \pi_X(R) \quad \text{i} \quad K \times S \subseteq R.$$

Przykład Użycia Operatora Dzielenia Relacji

- **Zadanie**: Podać identyfikatory programistów którzy znają wszystkie języki wymienione w **ImpLang**
- **Rozwiązanie**: $\mathbf{Programmers \div ImpLang}$

Programmers

Id	Language
1	C++
1	Java
1	Haskell
2	Haskell
2	Java
3	C++
3	Java

ImpLang

Language
C++
Java

$\mathbf{Programmers \div ImpLang}$

Id
1
3

Operator Dzielenia Relacji a Inne Operatory

Przypuśćmy że R is S są relacjami takimi że $\text{Attr}(S) \subsetneq \text{Attr}(R)$. Oznaczmy $X := \text{Attr}(R) \setminus \text{Attr}(S)$. Wówczas

$$R \div S = \pi_X(R) \setminus \pi_X\Big((\pi_X(R) \times S) \setminus R\Big)$$

gdzie $(\pi_X(R) \times S) \setminus R$ to zbiór elementów które powinny być w R gdyby był on iloczynem kartezjańskim rzutowania R na X z S, ale ich w R nie ma.

Przykłady Własności Operatorów Algebry Relacyjnej

Równości poniżej oznaczają że jeśli wyrażenie po jednej ze stron jest dobrze określone, to dobrze określone jest też wyrażenie po drugiej stronie i są one sobie równe.

- Naturalne złączenia i iloczyny kartezjańskie są **przemienne**, czyli $R \bowtie S = S \bowtie R$ i $R \times S = S \times R$.
- Naturalne złączenia i iloczyny kartezjańskie są **łączne**, czyli

  $$(R \bowtie S) \bowtie T = R \bowtie (S \bowtie T), \quad (R \times S) \times T = R \times (S \times T)$$

- Przypuśćmy że warunek ϕ zależy wyłącznie od atrybutów relacji R. Wówczas

  $$\sigma_\phi(R \bowtie S) = \sigma_\phi(R) \bowtie S.$$

  Podobna własność zachodzi dla iloczynów kartezjańskich.

Systemy Baz Danych

Zaawansowany SQL

Bartosz Zieliński

WYDZIAŁ FIZYKI
i INFORMATYKI STOSOWANEJ
Uniwersytet Łódzki

SQL a Model Relacyjny

Język zapytań (operacja **SELECT** w SQL jest wzorowany na kombinacji elementów z rachunku relacyjnego i algebry relacyjnej.

Należy jednak pamiętać że model danych zakładany przez SQL różni się nieco od czystego modelu relacyjnego:

- Zduplikowane wiersze: zmienne relacyjne przechowują, a zapytania zwracają multizbiory krotek.
- W "relacjach" zwracanych przez zapytania SQL kilka atrybutów może mieć te same nazwy, mogą też wystąpić atrybuty bez nazwy.
- W SQL kolejność kolumn w definicji zmiennej relacyjnej lub w zapytaniu może być istotna (inaczej niż w modelu relacyjnym).
- Niektóre operacje w SQL noszące nazwy operacji w algebrze relacyjnej mogą się różnić działaniem od swoich odpowiedników w algebrze.

Zapytanie SELECT z Lotu Ptaka

SELECT ⟨*lista wyrażeń definiujących atrybuty wynikowej relacji*⟩
FROM ⟨*opis zmiennych relacyjnych z których korzysta zapytanie*⟩
WHERE ⟨*(Opcj.) warunek selekcji (przed agregacją)*⟩
GROUP BY ⟨*(Opcj.) lista wyrażeń definiujących grupy przy agregacji*⟩
HAVING ⟨*(Opcj.) warunek selekcji (po agregacji)*⟩
ORDER BY ⟨*(Opcj.) opis sortowania wyników*⟩

To nie wszystkie elementy które mogą wystąpić w zapytaniu **SELECT**. Pomijamy klauzule **WITH** definiujące (także rekurencyjnie) lokalne nazywane podzapytania, hierarchiczne klauzule **CONNECT BY**, klauzule obracające kolumny (**PIVOT** i **UNPIVOT**), i wiele innych.

Zapytanie Zwracające Zawartość Wybranej Tabeli

SELECT *
FROM *NazwaZmiennejRelacyjnej*;

Uwaga

- Biblioteki programistyczne takie jak **JDBC** obsługujące operacje bazodanowe pozwalają na odwoływanie się do atrybutów zwracanych przez zapytanie krotek tak przez nazwę jak i przez numer kolejny.
- W przypadku zapytania powyżej kolejność (i nazwy) atrybutów są takie same jak w **CREATE TABLE** *NazwaZmiennejRelacyjnej* (...).

SELECT DISTINCT ... FROM ... WHERE ...

SELECT DISTINCT E_1 **AS** A_1, E_2 **AS** A_2, ..., E_n **AS** A_n
FROM R_1 t_1, R_2 t_2, ..., R_m t_m
WHERE *Warunek*;

- Relacje R_i mogą być zarówno odwołaniami do zmiennych relacyjnych jak i podzapytaniami **SELECT**.
- t_i nazywane są **aliasami tabel** lub **zmiennymi krotkowymi**.
- A_i nazywane są **aliasami kolumn**
- Pozycja atrybutu zwracanej krotki (przy pozycyjnych odwołaniach) jest taka sama jak na liście **SELECT**

Przykład SELECT DISTINCT ... FROM ... WHERE ...

Przykładowe zapytanie

```
1  SELECT DISTINCT e.FirstName||e.LastName AS Imie,
2         d.NazwaWydzialu AS Wydzial
3  FROM Employees e, (
4     SELECT d1.DepartmentName AS NazwaWydzialu,
5            d1.DepartmentId AS IDWydzialu
6     FROM Departments d;
7  ) d WHERE d.IDWydzialu = e.DepartmentId;
```

Rola DISTINCT

```
SELECT p.Y AS Z FROM R p;
```

Oznaczmy relację zwracaną przez powyższe zapytanie przez R' Jeśli

R jest dana przez

X	Y
1	10
2	10
3	20
4	20

to R' jest równa

Z
10
10
20
20

Projekcja w SQL domyślnie nie usuwa duplikatów! Aby wymusić usuwanie duplikatów trzeba skorzystać z **DISTINCT**

- Często nie jest to konieczne (a jest kosztowne), dlatego w dalszych przykładach będziemy pomijać **DISTINCT**.

Formy Uproszczone Zapytań
Pomijanie Nowej Nazwy Atrybutu

- Jeśli na liście **SELECT** pominiemy przemianowanie dla prostego odwołania do atrybutu A wówczas nazwą odpowiadającego atrybutu w wyniku zapytania będzie A.
- Jeśli nie nadamy nazwy atrybutowi zdefiniowanemu złożonym wyrażeniem nie będzie on miał nazwy. Jest to dozwolone w zewnętrznych zapytaniach (aplikacja może odwoływać się do kolumn wyniku pozycyjnie) ale nie w podzapytaniach

```
SELECT p.FirstName, p.Salary,
       p.Salary*0.2 AS Podatek,
       p.FirstName||p.LastName
FROM Employees p;
```

Powyższe zapytanie zwraca relację o atrybutach **FirstName**, **Salary** i **Podatek** oraz atrybut bez nazwy na czwartej pozycji.

Formy Uproszczone Zapytań
Pomijanie Zmiennych Krotkowych

Gdy nie prowadzi to do niejednoznaczności można nie deklarować **zmiennych krotkowych** dla **zmiennych relacyjnych** i odwoływać się do ich atrybutów bezpośrednio. **Zmienne krotkowe** są jednak **obowiązkowe** dla **podzapytań**.

```
SELECT FirstName, LastName, Salary, 0.2*Salary AS Tax
FROM Employees;
```

Gdy korzystamy z **różnych** zmiennych relacyjnych zamiast **zmiennymi krotkowymi** można kwalifikować nazwy atrybutów nazwami tabel.

```
SELECT Employees.Name, Departments.Name
FROM Employees, Departments;
```

Operatory Boolowskie i Atomowe Predykaty

- Warunki **WHERE** można budować łącząc atomowe predykaty przy pomocy operatorów boolowskich **AND**, **OR** i **NOT**.
- Wśród **atomowych predykatów** można wyróżnić operatory porównania =, <>, <, <=, >, >=, gdzie = i <> działają także dla łańcuchów znaków.
- Operator *napis* **LIKE** *wzorzec* porównuje *napis* ze *wzorcem*. We wzorcu można stosować znaki specjalne takie jak "%" (zero lub więcej dowolnych znaków) i "_" (pojedyńczy dowolny znak).
  - Np. zachodzi `'Berlin' LIKE '_erl%'`.
  - `x NOT LIKE y` to to samo co `NOT(x LIKE y)`
- *x* **BETWEEN** *a* **AND** *b* to to samo co `x>=a` **AND** `x<=b`.
- Sprawdzanie **NULL**-owatości: *x* **IS NULL** lub *x* **IS NOT NULL**.
- Sprawdzanie czy *x* należy do zbioru wartości: *x* **IN** $(v_1, v_2, \ldots, v_n)$. Można też skorzystać z podzapytania.

Operator IN z Podzapytaniem

```
SELECT * FROM Empls e
WHERE e.JobId IN (
    SELECT j.JobId FROM Jobs j
    WHERE j.MinSal >= 10000
)
```

Ważne

- Podzapytanie wewnątrz **IN** musi zwrócić relację z **pojedyńczym** atrybutem.
- Nazwa tego atrybutu nie ma znaczenia — może jej nawet nie być. Wynik podzapytania jest traktowany jako **zbiór wartości** a nie jako **relacja**.
- Te same uwagi dotyczą użycia podzapytań także w innych przypadkach gdy chcemy uzyskać **zbiór wartości**, np. wewnątrz operatorów **ALL** i **ANY**.

Operatory Oczekujące Zbioru: ALL i ANY

Z reguły poniższe operatory można zastąpić kombinacjami wywołań innych operatorów:

- $x > \mathbf{ANY}(M)$ to samo co $x > m$ dla któregoś $m \in M$
- $x > \mathbf{ALL}(M)$ to samo co $x > m$ dla każdego $m \in M$

Zamiast $>$ można skorzystać z innych operatorów porównania.

```
SELECT * FROM Empl e WHERE e.sal < ANY(1000,5000);
```

```
SELECT * FROM Empl e WHERE e.sal >= ALL(
   SELECT j.MinSal FROM Jobs j
);
```

Podzapytania Jako Wartości

Wszędzie tam gdzie w zapytaniu oczekiwana jest (pojedyńcza) wartość można użyć podzapytania zwracającego **pojedyńczą** krotkę z **pojedyńczym** atrybutem. Nazwa atrybutu nie jest istotna i może jej nie być.

Podzapytanie jako wartość w WHERE

```
SELECT * FROM Empls e WHERE e.sal = (
   SELECT 2*f.sal FROM Empls f WHERE f.Id=10
);
```

Podzapytanie jako wartość na liście SELECT

```
SELECT e.Name, (
   SELECT f.Name FROM Empls f WHERE f.Id=10
) AS BossName FROM Empls e;
```

Podzapytania Skorelowane

Podzapytania (**nie definiujące** źródłowych relacji w liście **FROM**) mogą zostać skorelowane z nad-zapytaniem przez odwołanie się do **zmiennych krotkowych** nad-zapytania.

- Podzapytania skorelowane można sobie wyobrazić jako wykonywane osobno dla każdej krotki zwracanej przez nad-zapytanie.
- Jeśli podzapytanie jest używane jako wartość wówczas powinno ono zwrócić **pojedyńczy wiersz** dla każdej krotki zapytania otaczającego.

Zapytanie skorelowane na liście SELECT

```
SELECT e.Name, (
    SELECT f.Name FROM Empls f
    WHERE f.Id = e.ManagerId
) AS BossName FROM Empls e;
```

Podzapytania Skorelowane

Inne Przykłady

Podzapytanie skorelowane wewnątrz operatora IN w WHERE

```
SELECT * FROM Empls e
WHERE e.JobId IN (
    SELECT j.JobId FROM Jobs j
    WHERE j.MinSal = e.sal
)
```

Podzapytanie skorelowane wewnątrz WHERE

```
SELECT * FROM Empls e
WHERE e.sal < (
    SELECT j.MinSal FROM Jobs j
    WHERE j.JobId=e.JobId
);
```

Operatory EXISTS i NOT EXISTS

Niech Q będzie podzapytaniem (zwracającym krotki o dowolnej ilości kolumn, niekoniecznie nazwanych. Wtedy $\mathbf{EXISTS}(Q)$ jest spełnione gdy relacja zdefiniowana przez Q jest niepusta.

- Użyteczne głównie gdy Q jest skorelowane z zewnętrznym zapytaniem.

```
SELECT * FROM Empls e WHERE EXISTS(
  SELECT * FROM Jobs j WHERE j.JobId=e.JobId
);
```

WHERE i Złączenia

Zapytanie typu

`SELECT * FROM` $R\ p_1$`,` $S\ p_2$
`WHERE` $p_1.A = p_2.B$

---

zostanie przez DBMS zaimplementowane jako odpowiednie złączenie warunkowe: $R \bowtie_{A=B} S$, nie zaś bezpośrednio jako złożenie selekcji z iloczynem kartezjańskim: $\sigma_{A=B}(R \times S)$

Lepiej jednak skorzystać jawnie z operatorów złączenie w SQL takich jak **NATURAL JOIN**, **INNER JOIN** itp.

Agregacja w SQL

- Klauzula **GROUP BY**
- Klauzula **HAVING**
- Funkcje agregujące, m.in: **Max**, **Min**, **Sum**, **Avg**, **Count**

GROUP BY

GROUP BY E_1, E_2, ..., E_n

- Klauzula **GROUP BY** definiuje podział relacji na rozłączne podzbiory (grupy) krotek według wartości wyrażeń $E_1, \ldots, E_n$.
- Każdy podzbiór zdefiniowany przez **GROUP BY** składa się z krotek o identycznej wartości n-tki $(E_1, E_2, \ldots, E_n)$.
- Wyrażenia na liście **SELECT** muszą mieć sens dla całych grup, zatem mogą się odwoływać wyłącznie do wartości $E_1, \ldots, E_n$ oraz wartości obliczanych przez funkcje agregujące.
- Wartość zwracana przez funkcję agregującą zależy od wszystkich krotek w grupie.
- Jeśli nie użyto klauzuli **HAVING** to relacja wynikowa zawiera po jednej krotce dla każdego podzbioru.
- Pominięcie **GROUP BY** przy jednoczesnym użyciu funkcji agregującej oznacza że mamy jeden podzbiór składający się ze wszystkich krotek.

Przykład Grupowania

GROUP BY X, Y*Y

R

X	Y	Z	Grupy
'A'	1	10	('A', 1)
'A'	-1	20	
'A'	2	20	
'A'	2	30	('A', 4)
'A'	-2	25	
'B'	2	5	('B', 4)
'B'	2	10	
'C'	3	35	('C', 9)

Na przykład:

```
SELECT p.X, p.Y*p.Y AS T,
       SUM(p.Z) + p.Y*p.Y AS S
FROM R p
GROUP BY p.X, p.Y*p.Y
```

zwraca relację:

X	T	S
'A'	1	31
'A'	4	79
'B'	4	19
'C'	9	44

HAVING

HAVING *Warunek*

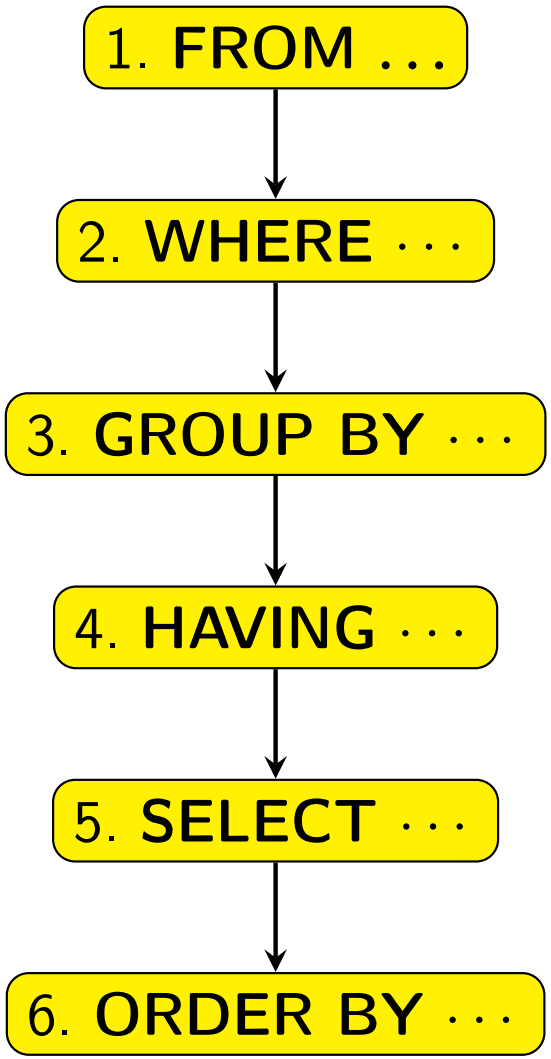
- Klauzula **HAVING** służy do eliminacji grup nie spełniających *Warunku*
- Oznacza to że *Warunek* musi być wyrażeniem logicznym mającym sens dla całej grupy (a nie dla poszczególnych krotek): musi odwoływać się jedynie do wartości $E_1, \ldots, E_n$ z listy **GROUP BY** oraz wartości obliczanych przez funkcje agregujące.
- W wielu DBMS-ach (np. Oracle) *Warunek* nie może się odwoływać do nazw kolumn zdefiniowanych na liście **SELECT**.
- Warunek **WHERE** obliczany jest przed grupowaniem.

Zapytanie SELECT z Lotu Ptaka Jeszcze Raz

SELECT ⟨*lista wyrażeń definiujących atrybuty wynikowej relacji*⟩
FROM ⟨*opis zmiennych relacyjnych z których korzysta zapytanie*⟩
WHERE ⟨*(Opcj.) warunek selekcji (przed agregacją)*⟩
GROUP BY ⟨*(Opcj.) lista wyrażeń definiujących grupy przy agregacji*⟩
HAVING ⟨*(Opcj.) warunek selekcji (po agregacji)*⟩
ORDER BY ⟨*(Opcj.) opis sortowania wyników*⟩

Kolejność różnych klauzul w zapytaniu **SELECT** jest myląca. Aby dobrze zrozumieć do czego można się odwoływać w różnych częściach zapytania należy wyobrazić sobie kolejność przetwarzania opisaną na następnym slajdzie.
Uwaga: Kolejności tej nie należy rozumieć jako definiującej faktyczną implementację wykonania zapytania

Kolejność Przetwarzania Klauzul Zapytania



1. Obliczana jest relacja wynikowa samego **FROM** (iloczyn kartezjański lub złączenie).
2. Z relacji zdefiniowanej powyżej eliminowane są krotki niespełniające warunku **WHERE**
3. Zbiór krotek które przetrwały **WHERE** jest dzielony na podzbiory według listy wyrażeń **GROUP BY**
4. Eliminowane są podzbiory niespełniające warunku **HAVING**
5. Obliczane są wyrażenia na liście **SELECT** dając w wyniku jedną krotkę dla każdego podzbioru który przetrwał **HAVING**.
6. Otrzymane krotki są sortowane.

Przykłady Zapytań z Agregacją

```
SELECT avg(p.Salary) AS Srednia,
       sum(p.Salary) AS Total
FROM Employees p;
```

```
SELECT p.DepartmentID,
       avg(p.Salary) AS Srednia
FROM Employees p
WHERE p.JobId<>'IT'
GROUP BY p.DepartmentId
HAVING min(p.Salary) > 5000
ORDER BY Srednia;
```

Błędne zapytanie:

```
SELECT p.FirstName, avg(p.Salary) AS Srednia
FROM Employees p GROUP BY p.DepartmentId
```

Standardowe Funkcje Agregujące w SQL

Każdy RDBMS implementuje następujące funkcje agregujące:

- $\mathbf{Min}(E)$ — dla grupy krotek liczy **minimalną** wartość wyrażeń E obliczonych dla każdej krotki grupy.
- $\mathbf{Max}(E)$— dla grupy krotek liczy **maksymaną** wartość wyrażeń E obliczonych dla każdej krotki grupy.
- $\mathbf{Sum}(E)$ — dla grupy krotek liczy sumę wyrażeń E obliczonych dla każdej krotki grupy.
- $\mathbf{Avg}(E)$ — dla grupy krotek liczy średnią wyrażeń E obliczonych dla każdej krotki grupy.
- $\mathbf{Count}()$ — dla grupy krotek liczy
  - $\mathbf{Count}(E)$ — ilość krotek w grupie dla których E nie jest **NULL**
  - $\mathbf{Count}(\mathbf{DISTINCT}\ E)$ — ilość różnych (i nie **NULL**) wartości E obliczonych dla każdej krotki grupy
  - $\mathbf{Count}(*)$ — ilość krotek w grupie

Operatory w SQL Pochodzące z Algebry Relacyjnej

- **CROSS JOIN** (iloczyn kartezjański)
- Operatory złączeń: **NATURAL JOIN** (i wersje zewnętrzne) oraz złączenia warunkowe **INNER JOIN** (i wersje zewnętrzne),
- Operatory teoriomnogościowe: **UNION**, **UNION ALL**, **INTERSECT**, **EXCEPT**.

CROSS JOIN

```
SELECT *
FROM Employees p CROSS JOIN Departments q;
```

to to samo co

```
SELECT *
FROM Employees p, Departments q;
```

Złączenia Naturalne

R **NATURAL JOIN** *S*

- Dokonuje złączenia naturalnego (jak w algebrze relacyjnej) na wspólnych (noszących te same nazwy) atrybutach z obu tabel.
- Relacja wynikowa będzie miała atrybuty obu relacji źródłowych (z usuniętymi duplikatami).
  - Nie jest legalne (w **SELECT**, **WHERE** itp.) odwoływanie się do wspólnych atrybutów obu relacji kwalifikowanych zmiennymi krotkowymi.
- Stosowanie tej operacji nie jest zalecane ponieważ atrybuty złączenia nie są podane jawnie i może dojść do złączenia na większej ilości atrybutów niż życzył sobie programista.
- Wersje zewnętrzne: **NATURAL LEFT OUTER JOIN**, **NATURAL RIGHT OUTER JOIN** i **NATURAL FULL OUTER JOIN**.
- Przykład:

```
SELECT * FROM R NATURAL JOIN S;
```

Złączenia Warunkowe

R p **INNER JOIN** *S q* **ON** *Warunek*

- Dokonuje złączenia warunkowego relacji *R* i *S* opisywanego *Warunkiem*
- Wersje zewnętrzne: **LEFT OUTER JOIN**, **RIGHT OUTER JOIN** i **FULL OUTER JOIN**.

Przykład

```
SELECT e.FirstName, e.LastName, e.Salary
       m.FirstName AS ManagerFN,
       m.LastName AS MangerLN
FROM Employees e LEFT OUTER JOIN Employees m
     ON m.EmployeeId=e.ManagerId
WHERE e.Salary > 5000;
```

Przykład Złączenia Warunkowego z Podzapytaniem

```
SELECT e.FirstName, e.LastName,
       e.Salary, m.AvgSalary, m.DepartmentId
FROM Employees e INNER JOIN (
   SELECT d.DepartmentId,
          Avg(d.Salary) AS AvgSalary
   FROM Employees d
   GROUP BY d.DepartmentId
) m ON e.DepartmentId = m.DepartmentId
  AND e.Salary > m.AvgSalary
```

UNION, INTERSECT i EXCEPT

Operatory mnogościowe łączą ze sobą kompletne zapytania **SELECT**:

SELECT ... FROM ... *Operator* **SELECT ... FROM ...**

gdzie *Operator* to jeden z:

- **UNION** — zwraca krotki z obu zapytań bez duplikatów. Usuwanie duplikatów jest kosztowne.
- **UNION ALL** — zwraca krotki z obu zapytań (możliwe duplikaty)
- **INTERSECT** — zwraca tylko krotki zwrócone przez oba zapytania
- **EXCEPT** — zwraca tylko te krotki zwrócone przez pierwsze zapytanie których nie zwróciło drugie.

Listy **SELECT** w obu zapytaniach muszą mieć tą samą długość a wyrażenia na odpowiadających pozycjach muszą zwracać wartości zgodnych typów. Nazwy kolumn wyniku zapytania są brane z pierwszego z zapytań składowych.

Przykład UNION ALL

```
SELECT e.FirstName, e.LastName, d.DepartmentName
FROM Employees e INNER JOIN Departments d
     ON e.DepartmentId = d.DepartmentId
UNION ALL
SELECT e.FirstName, e.LastName, NULL
FROM Employees e
WHERE e.DepartmentId NOT IN (
   SELECT dd.DepartmentId
   FROM Departments dd
)
ORDER BY FirstName, LastName;
```

Sortowanie Krotek

Do wybrania kolejności zwracania przez DBMS krotek wynikowych do aplikacji która wysłała zapytania służy klauzula **ORDER BY**. Może ona pojawić się **tylko** w najbardziej zewnętrznym zapytaniu

ORDER BY E_1 M_1, E_2 M_2, ..., E_n M_n

O kolejności krotek decyduje najpierw E_1, dla krotek o równych wartościach E_1 decyduje E_2, itd. Kolejność dla E_i jest rosnąca bądź malejąca zależnie od M_i którym może być albo **ASC** (rosnąca) albo **DESC** (malejąca).

```
SELECT e.FirstName||' '||e.LastName AS Name
       e.Salary
FROM Employees e
ORDER BY Salary ASC, Name DESC;
```

Zapytania a Operacje Modyfikujące Dane
INSERT

Zapytania **SELECT** mogą być też częścią składową operacji modyfikujących dane.

Wstawianie do zmiennej relacyjnej wierszy zwróconych przez zapytanie (dialekt SQL RDBMS Oracle)

```
INSERT INTO Employees(FirstName, LastName, Salary)
SELECT e.Imie, e.Nazwisko,e.Pensja
FROM Pracownicy e
WHERE e.Salary > 5000;
```

Zapytania a Operacje Modyfikujące Dane
DELETE i UPDATE

DELETE z podzapytaniem: skasować pracowników których wyniki są poniżej średniej dla departamentu w którym pracują:

```
DELETE FROM Employees e
WHERE Performance < (
   SELECT Avg(d.Performance) FROM Employees d
   WHERE d.DepartmentId = e.DepartmentId
);
```

UPDATE z podzapytaniem: podwyższyć wszystkim pracownikom pensję o 0.1 minimalnej pensji w departamencie w którym pracują:

```
UPDATE Employees e
SET Salary = Salary + 0.1 * (
   SELECT Min(d.Salary) FROM Employees d
   WHERE d.DepartmentId = e.DepartmentId
);
```

Widoki (Perspektywy)

W bazie danych oprócz zmiennych relacyjnych zawierających relacje mogą znajdować się także specjalne zmienne zwane **widokami** (albo inaczej **perspektywami** przechowujące zapytania **SELECT**.

Deklaracja perspektywy w SQL:

CREATE VIEW *NazwaWidoku* **AS**
SELECT ...

- Widoku można użyć wszędzie tam gdzie można użyć zmiennej relacyjnej **o ile widok nie jest przez to polecenie modyfikowany**
- **Modyfikowalne widoki** wykraczają poza zakres tych zajęć
- Wykonywanie polecenia odwołującego się do widoku zaczyna się od podstawienia zawartości widoku pod jego nazwę jako podzapytania.

Przykład Deklaracji Widoku

```
CREATE VIEW AvgSalaries AS
SELECT Avg(d.Salary) AS AvgSalary, d.DepartmentId
FROM Employees d GROUP BY d.DepartmentId;
```

Kiedy wykonywane jest zapytanie

```
SELECT * FROM Employees e INNER JOIN AvgSalaries m
ON e.DepartmentId = m.DepartmentId
```

najpierw podstawiane jest zapytanie przechowywane w **AvgSalaries**:

```
SELECT * FROM Employees e INNER JOIN (
   SELECT Avg(d.Salary) AS AvgSalary, d.DepartmentId
   FROM Employees d GROUP BY d.DepartmentId
) m ON e.DepartmentId = m.DepartmentId
```

Dalej zapytanie jest wykonywane normalnie.

Zastosowania Perspektyw

Wśród zastosowań perspektyw można wymienić:

- Modularyzacja złożonych zapytań:
  - Jeśli jakieś podzapytanie pojawia się w wielu poleceniach można je zapisać w widoku i wykorzystać w tych zapytaniach.
- Ograniczanie dostępu do danych:
  - Większość RDBMS pozwala ograniczyć dostęp użytkowników tylko do niektórych (całych) zmiennych relacyjnych.
  - Ograniczenie dostępu do części relacji przechowywanej w zmiennej relacyjnej jest trudniejsze — można tu skorzystać z widoków.
  - Odbieramy użytkownikowi prawa do czytania zmiennej relacyjnej i tworzymy widok (który użytkownik będzie mógł czytać) zdefiniowany zapytaniem zwracającym tą część relacji do której czytania użytkownik ma prawo.
  - Z operacjami modyfikującymi dane jest trudniej, w razie potrzeby można zaimplementować operacje modyfikujące dane przy pomocy **triggerów** (wyzwalaczy)

Transakcje

Operacje DML (także zapytania) na bazie danych odbywają się w ramach **transakcji**

- W ramach jednej transakcji można umieścić wiele operacji DML.
- Menedżer transakcji danego RDBMS-u zapewnia że wszystkie operacje danej transakcji odbywają się jako jedna całość w izolacji od innych transakcji, tzn. że spełnione są warunki **ACID** (*Atomicity, Consistency, Isolation, Durability*).
- Każdą transakcję można zakończyć na dwa sposoby:
  - **Zatwierdzając** ją — od tego momentu, nawet jeśli nastąpi awaria systemu, zapewniona jest trwałość zmian.
  - **Wycofując** wszystkie zmiany w bazie danych wprowadzone przez tą transakcję.

Rozpoczynanie i Kończenie Transakcji

Rozpoczynanie Transakcji

Transakcje rozpoczyna się (zależnie od dialektu SQL) jawnym poleceniem, np. **BEGIN** lub rozpoczyna się automatycznie pierwszym poleceniem DML wysłanym po zakończeniu poprzedniej transakcji.

Kończenie Transakcji

- **COMMIT** — zatwierdza ostatecznie bieżącą transakcję.
- **ROLLBACK** — wycofuje wszystkie efekty bieżącej transakcji.

Autocommit

W wielu przypadkach RDBMS pozwala na ustawienie **automatycznego zatwierdzania**. Oznacza to że każde polecenie DML jest wykonywane w osobnej transakcji która jest automatycznie zatwierdzana (lub wycofywana w przypadku błędu) po jego zakończeniu.

Własności ACID

- **A**tomicity (atomowość) — wykonają się albo wszystkie operacje transakcji albo żadna. Efekty działania przerwanych transakcji będą wycofane.
- **C**onsistency (spójność) — po zakończeniu transakcji spełnione będą wszystkie więzy spójności.
- **I**solation (izolacja) — Każda transakcja wykonuje się tak jakby była jedyną wykonywaną w bazie danych a dokonywane przez nią modyfikacje danych nie są widoczne dla innych transakcji do czasu jej zatwierdzenia.
- **D**urability (trwałość) — po zakończeniu transakcji nawet awaria systemu nie spowoduje utraty modyfikacji danych.

Współbieżność i Spójność

W szczególności RDBMS musi zapewnić spójny widok danych dla każdej transakcji przy współbieżności dostępu

Negatywne Zjawiska Którym Trzeba Zapobiec

- **Dirty reads** — transakcja czyta dane które zostały zmodyfikowane przez inną, jeszcze nie zatwierdzoną transakcję.
- **Nonrepeatable reads** — przy ponownym odczycie tych samych danych transakcja widzi że w międzyczasie inna **zatwierdzona** transakcja zmodyfikowała lub skasowała część wierszy.
- **Phantom reads** — przy ponownym wykonaniu tego samego zapytania transakcja widzi nowe wiersze wstawione przez inną, zatwierdzoną transakcję.

Poziomy Izolacji ANSI/ISO

Poziom Izolacji	**Dirty read**	**Nonrepeatable read**	**Phantom read**
Read uncommited	√	√	√
Read commited	—	√	√
Repeatable read	—	—	√
Serializable	—	—	—

- RDBMS-y udostępniają kilka poziomów izolacji (czasem różniących się od poziomów ANSI/ISO).
- Poziom izolacji można wybrać przy rozpoczęciu transakcji.
- Zwykle domyślnym poziomem jest **READ COMMITED**

Systemy Baz Danych

Więzy i Normalizacja

Bartosz Zieliński

WYDZIAŁ FIZYKI
i INFORMATYKI STOSOWANEJ
Uniwersytet Łódzki

Więzy i Normalizacja

- Atrybuty danych można bez straty informacji rozdzielić na wiele sposobów pomiędzy zmienne relacyjne.
- Niektóre z tych sposobów są lepsze niż inne ponieważ unikają pewnych **anomalii modyfikacji** — zmienne relacyjne są wtedy w jednej z **postaci normalnych**.
- Anomalie modyfikacji i postacie normalne zmiennych powiązane są z **zależnościami funkcyjnymi** i **wielowartościowymi**.
- Pogłębimy także zrozumienie **więzów** wprowadzonych wcześniej nieformalnie, a w szczególności więzów klucza głównego, referencyjnych i unikalności.

Zależności Funkcyjne dla Relacji

Definicja

Niech R będzie relacją i niech $X, Y \subseteq \text{Attr}(R)$. Powiemy że R **spełnia zależność funkcyjną** $X \to Y$, co oznaczamy $R \models X \to Y$, wtedy i tylko wtedy gdy dla każdych krotek $t, t' \in R$ jeśli $t|_X = t'|_X$ to $t|_Y = t'|_Y$.

Przykład Zależności Funkcyjnych

Rozważmy następującą relację R:

A	B	C
1	'K'	10
1	'K'	20
2	'L'	30
3	'M'	40

R spełnia zależność funkcyjną $\{\mathbf{A}\} \to \{\mathbf{B}\}$ ponieważ dla każdych krotek $t, t' \in R$ jeśli $t.\mathbf{A} = t'.\mathbf{A}$ to $t.\mathbf{B} = t'.\mathbf{B}$

R **nie** spełnia zależności funkcyjnej $\{\mathbf{A}, \mathbf{B}\} \to \{\mathbf{C}\}$ ponieważ istnieją dwie krotki $t, t' \in R$ (czerwone) dla których $t|_{\{\mathbf{A},\mathbf{B}\}} = \begin{array}{cc} \mathbf{A} & \mathbf{B} \\ \hline 1 & \text{'K'} \end{array} = t'|_{\{\mathbf{A},\mathbf{B}\}}$ ale $t.\mathbf{C} = 10 \neq 20 = t'.\mathbf{C}$, czyli $t|_{\{\mathbf{C}\}} \neq t'|_{\{\mathbf{C}\}}$.

Zależności Funkcyjne dla Zmiennych Relacyjnych

Definicja

Powiemy że zmienna relacyjna $\mathbf{R}$ spełnia **zależność funkcyjną** $X \to Y$, co oznaczamy $\mathbf{R} \models X \to Y$, jeśli wszystkie **legalne** relacje które **chcemy** przechowywać w $\mathbf{R}$ spełniają tą zależność.

Zauważmy że spełnianie zależności funkcyjnych dla zmiennych relacyjnych zależy od **znaczenia** tych zmiennych

Przykład

Przypuśćmy że $\mathbf{R}$ jest zmienną o atrybutach **PESEL** i **Imię** i chcemy aby każda krotka $\mathbf{R}$ przechowywała informacje o PESELu i imieniu rzeczywistego człowieka (np. pracownika). Wówczas $\mathbf{R}$ spełnia $\{\textbf{PESEL}\} \to \{\textbf{Imię}\}$ tylko gdy PESELe są w rzeczywistości unikalne (nie są).

Aksjomaty Armstronga Zależności Funkcyjnych

Aksjomaty Armstronga

Niech R będzie relacją lub zmienną relacyjną i niech $X, Y, Z \subseteq \text{Attr}(R)$.

1. Jeśli $Y \subseteq X$ to $R \models X \to Y$,
2. Jeśli $R \models X \to Y$ to $R \models X \cup Z \to Y \cup Z$
3. Jeśli $R \models X \to Y$ i $R \models Y \to Z$ to $R \models X \to Z$.

Twierdzenie

Niech $\mathbf{R}$ będzie zmienną relacyjną. Niech $\mathcal{X}$ będzie zbiorem zależności funkcyjnych z których każda jest spełniona przez R. Wówczas $\mathbf{R}$ spełnia zależność funkcyjną $X \to Y$ wtedy i tylko wtedy gdy można ją wyprowadzić z zależności ze zbioru $\mathcal{X}$ przy pomocy reguł danych **aksjomatami Armstronga**.

Przykład Wyprowadzania Zależności Funkcyjnych

Jako przykład udowodnimy korzystając z **aksjomatów Armstronga** że jeśli $R \models X \to Y$ i $R \models X \to Z$ to $R \models X \to Y \cup Z$.

Dowód

$$\dfrac{\dfrac{R \models X \to Y}{R \models X \cup X \to X \cup Y}\,\mathbf{A2} \quad \dfrac{R \models X \to Z}{R \models X \cup Y \to Y \cup Z}\,\mathbf{A2}}{R \models X \to Y \cup Z}\,\mathbf{A3}$$

Zauważmy że skorzystaliśmy tu z przemienności ($X \cup Y = Y \cup X$) oraz z idempotentości ($X \cup X = X$) operacji unii zbiorów.

Nieredukowalne Zależności Funkcyjne

Definicja

Zależność funkcyjną $X \rightarrow Y$ nazywamy **nieredukowalną w** R gdy

1. $R \models X \rightarrow Y$ i dla żadnego $Z \subsetneq X$ nie zachodzi $R \models Z \rightarrow Y$,
2. Y zawiera tylko jeden atrybut.

Wystarczalność Nieredukowalnych Zależności Funkcyjnych

Lemat

Następujące reguły można wyprowadzić z **aksjomatów Armstronga**:

1. $R \models X \to Y$ i $R \models X \to Z$ wtedy i tylko wtedy gdy $R \models X \to Y \cup Z$
2. Jeśli $X \subseteq X'$ i $R \models X \to Y$ to $R \models X' \to Y$.

Twierdzenie

Nietrudno zauważyć że gdy $R \models X \to Y$ to istnieje zbiór $\mathcal{S}$ zależności funkcyjnych nieredukowalnych w R taki że $R \models X \to Y$ można wyprowadzić z $\{R \models X' \to Y' \mid X' \to Y' \in \mathcal{S}\}$

Nadklucze

Definicja

Podzbiór X atrybutów zmiennej relacyjnej $\mathbf{R}$ nazywamy **nadkluczem** (*superkey*) zmiennej $\mathbf{R}$ gdy $\mathbf{R} \models X \rightarrow \text{Attr}(R)$

Uwaga

Nadkluczem każdej zmiennej relacyjnej $\mathbf{R}$ jest $\text{Attr}(\mathbf{R})$.

Klucze Kandydujące

Definicja

Klucz kandydujący X (*candidate key*) zmiennej relacyjnej $\mathbf{R}$ to minimalny **nadklucz** $\mathbf{R}$, tzn. $\mathbf{R} \models X \rightarrow \mathrm{Attr}(R)$ i dla żadnego $Y \subsetneq X$ nie zachodzi $\mathbf{R} \models Y \rightarrow \mathrm{Attr}(R)$.

Uwaga

Każdy **nadklucz** X zawiera w sobie co najmniej jeden **klucz kandydujący**: albo X jest minimalny (a zatem jest **kluczem kandydującym**) albo istnieje $Y \subsetneq X$ t.ż Y jest nadkluczem. Powtarzając rozumowanie dla Y (i kolejnych jego podzbiorów) w końcu musimy uzyskać klucz kandydujący.

Definicja

Pierwszorzędny atrybut (**prime attribute**) zmiennej $\mathbf{R}$ to atrybut należący do pewnego **klucza kandydującego** $\mathbf{R}$.

Klucz Główny

Definicja

Jeden z kluczy kandydujących danej zmiennej $\mathbf{R}$ projektant bazy może wybrać jako **klucz główny** (*primary key*).

Uwaga

Klucz główny pełni specjalną rolę w zmiennej relacyjnej. Jest on kanonicznym identyfikatorem wiersza w tabeli wykorzystywanym m.in. w więzach referencyjnych.

Wybrane Więzy dla Zmiennych Relacyjnych

Więzy CHECK i NOT NULL

Na zmiennej relacyjnej $\mathbf{R}$ można założyć więzy postaci

CHECK(ϕ)

każda krotka $t \in \mathbf{R}$ musi spełniać warunek ϕ zależny wyłącznie od t ale nie od innych krotek.

A NOT NULL

wartością atrybutu A nie może być **NULL**.

Wybrane Więzy dla Zmiennych Relacyjnych

Klucze (UNIQUE i PRIMARY KEY)

UNIQUE$(A_1, \ldots, A_n)$

dla każdych $t_1, t_2 \in \mathbf{R}$ jeśli $t_1.A_i \neq \mathbf{NULL}$, $i \in \{1, \ldots, n\}$ i $t_1|_{\{A_1,\ldots,A_n\}} = t_2|_{\{A_1,\ldots,A_n\}}$ to $t_1 = t_2$.

- **UNIQUE**$(A_1, \ldots, A_n)$ z A_i **NOT NULL**, $i \in \{1, \ldots, n\}$, deklaruje że $\{A_1, \ldots, A_n\}$ jest **nadkluczem** $\mathbf{R}$

PRIMARY KEY$(A_1, \ldots, A_n)$

deklaruje że $\{A_1, \ldots, A_n\}$ jest **kluczem głównym** $\mathbf{R}$ (choć warunek minimalności nie jest sprawdzany)

Przykład Deklaracji Więzów w SQL

```
1  CREATE TABLE Products (
2      vendor VARCHAR(30),
3      productId INTEGER,
4      -- NOT NULL jest zawsze umieszczany razem z atrybutem
5      productName INTEGER NOT NULL,
6      -- gdy wiąz odwołuje się tylko do jednego atrybutu
7      -- można go zdefiniować razem z atrybutem
8      unitPrice NUMBER(6,2) NOT NULL CHECK(unitPrice > 0),
9      quantity INTEGER NOT NULL CHECK(quantity >= 0),
10     price NUMBER(6,0) NOT NULL,
11     -- więzy odwołujące się do więcej niż jednego atrybutu
12     -- muszą być zdefiniowane osobno
13     PRIMARY KEY(vendor,productId),
14     UNIQUE(vendor,productName),
15     CHECK(price = unitPrice * quantity)
16 );
```

Więzy Referencyjne

Przypuśćmy że $\mathbf{S}$ jest zmienną relacyjną dla której określono wiąz klucza głównego poleceniem **PRIMARY KEY**$(A_1, \ldots, A_n)$.
Wówczas następujący wiąz referencyjny na zmiennej $\mathbf{R}$:

FOREIGN KEY$(B_1, \ldots, B_n)$ **REFERENCES** $\mathbf{R}(A_1, \ldots, A_n)$

deklaruje że dla każdej krotki $t \in \mathbf{R}$ zachodzi **jeden** z następujących warunków:

- $t.B_i = \mathbf{NULL}$ dla pewnego $i \in \{1, \ldots, n\}$
- istnieje $t' \in \mathbf{S}$ takie że $t'.A_i = t.B_i$ dla $i \in \{1, \ldots, n\}$.

- Zbiór atrybutów $\{B_1, \ldots, B_n\}$ nazywany jest **kluczem obcym**
- Dana zmienna relacyjna może może posiadać więcej niż jeden wiąz referencyjny.

Więzy Referencyjne a Związki Pomiędzy Tabelami

- Więzy referencyjne wyrażają związki pomiędzy krotkami z różnych tabel.
- Większość typowych złączeń to złączenia równościowe na atrybutach **klucza obcego** jednej tabeli i odpowiadających atrybutach **klucza głównego** drugiej.

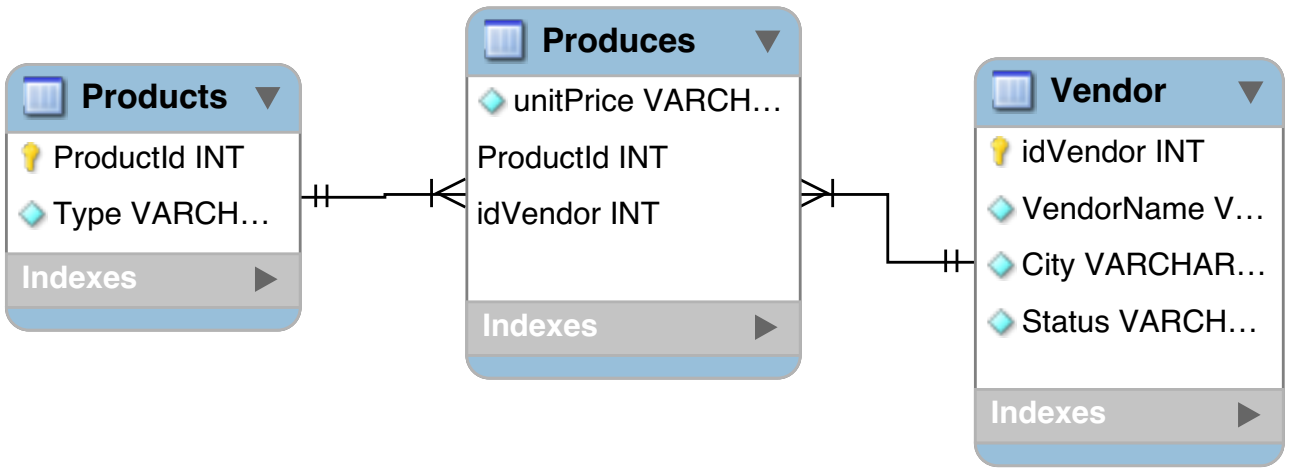
Przypuśćmy że $X \subseteq \mathrm{Attr}(R)$ jest kluczem obcym wskazującym na zmienną S. Załóżmy dla uproszczenia że kluczem głównym S jest także X. Wówczas

- W ogólności wiąz referencyjny wyraża związek 1 do wielu: każdemu $t \in S$ odpowiada 0 lub więcej $t' \in R$ t.ż. $t|_X = t'|_X$.
- Gdy na R założony jest dodatkowo wiąz **UNIQUE**(X) wówczas wiąz referencyjny wyraża relację 1 do co najwyżej 1: każdemu $t \in S$ odpowiada co najwyżej jeden $t' \in R$ t.ż. $t|_X = t'|_X$.
- Relacja 1 do 1 pomiędzy krotkami R i S wymaga by X w obu tabelach było zarówno kluczem kandydującym i obcym
- Relacja wiele do wielu wymaga zastosowania pośredniej tabeli.

Przykład Deklaracji Schematu Bazy

```
1  CREATE TABLE Products (
2    ProductId INTEGER NULL PRIMARY KEY,
3    Type VARCHAR(45) NOT NULL
4  );
5
6  CREATE TABLE Produces (
7    unitPrice NUMERIC(6,2) NOT NULL,
8    ProductId INTEGER,
9    idVendor INTEGER,
10   PRIMARY KEY (ProductId, idVendor),
11   FOREIGN KEY (ProductId) REFERENCES Products (ProductId),
12   FOREIGN KEY (idVendor) REFERENCES Vendor(idVendor)
13 );
14
15 CREATE TABLE Vendor (
16   idVendor INTEGER,
17   VendorName VARCHAR(45) UNIQUE,
18   City VARCHAR(45) NOT NULL,
19   Status VARCHAR(45) NOT NULL,
20   PRIMARY KEY (idVendor)
21 );
```

Przykład Diagramu Bazy



- Wykonany w programie SQL Workbench.
- Zwrócić uwagę na zastosowanie kruczych stóp po stronie "wiele" (czyli tam gdzie jest atrybut **klucz obcy**.
- Zmienna relacyjne **Produces** implementuje tu relację wiele do wielu pomiędzy **Products** a **Vendor** — każdy produkt jest produkowany przez wielu producentów, a każdy producent produkuje wiele produktów.

Modyfikacje Wskazywanych Tabel

```
CREATE TABLE Vend (
  VendId INTEGER PRIMARY KEY,
  VendName VARCHAR(30)
);
```

```
CREATE TABLE Prod (
  ProdId INTEGER PRIMARY KEY,
  VendId INTEGER REFERENCES Vend
);
```

Przypuśćmy że skasowaliśmy wiersz t z tabeli **Vend** lub zmodyfikowaliśmy wartość atrybutu **VendId** w krotce $t \in \mathbf{Vend}$. Co zrobić z krotkami $t' \in \mathbf{Prod}$ dla których $t.\mathbf{VendId} = t'.\mathbf{VendId}$? Można albo zabronić operacji, albo

- w przypadku **DELETE** usunąć także odpowiadające krotki $t' \in \mathbf{Prod}$,
- w przypadku **UPDATE** zmodyfikować także wartość atrybutu. $\mathbf{VendId}$ w odpowiednich krotkach $t' \in \mathbf{Prod}$.

Anomalie Modyfikacji

Supplies spełnia
$\{\#S, \#P\} \rightarrow \{QTY\}$,
$\{\#S\} \rightarrow \{City\}$

PRIMARY KEY$(\#S, \#P)$

Supplies

#S	City	#P	QTY
S1	London	P1	300
S1	London	P2	200
S1	London	P3	400
S1	London	P4	200
S1	London	P5	100
S1	London	P6	100
S2	Paris	P1	300
S2	Paris	P2	400
S3	Paris	P2	200
S4	London	P2	200
S4	London	P4	300
S4	London	P5	400

Dla tabeli **Supplies** mogą wystąpić **anomalie modyfikacji**, czyli trudności związane z operacjami:

- **INSERT** — nie można wstawić do **Supplies** danych dostawcy (np. (S5, Athens)) który niczego jeszcze nie dostarcza.
- **DELETE** — skasowanie ostatniej krotki z informacjami o dostawach pewnego dostawcy (np. S3) usuwa także informację o tym dostawcy (np. że S3 ma siedzibę w Paryżu co ciągle jest prawdą)
- **UPDATE** — ta sama informacja o siedzibie dostawcy x (atrybut **City**) musi się powtarzać w każdej krotce t dla której $t.\#S = x$. Zatem gdy x zmienia siedzibę musimy zmodyfikować tę informację w wielu krotkach jednocześnie.

Przyczyny Anomalii Modyfikacji

Supplies spełnia
$\{\#S, \#P\} \rightarrow \{QTY\}$,
$\{\#S\} \rightarrow \{City\}$

PRIMARY KEY$(\#S, \#P)$

Supplies

#S	City	#P	QTY
S1	London	P1	300
S1	London	P2	200
S1	London	P3	400
S1	London	P4	200
S1	London	P5	100
S1	London	P6	100
S2	Paris	P1	300
S2	Paris	P2	400
S3	Paris	P2	200
S4	London	P2	200
S4	London	P4	300
S4	London	P5	400

Przyczyną anomalii modyfikacji

jest umieszczanie w jednej zmiennej relacyjnej informacji dotyczących wielu różnych „obiektów", a w konsekwencji powtarzanie tych samych informacji w wielu krotkach. Symptomem obecności w jednej tabeli $\mathbf{R}$ danych dotyczących wielu obiektów jest spełnianie przez $\mathbf{R}$ zależności funkcyjnych postaci $X \rightarrow Y$ gdzie X nie jest nadkluczem a $Y \not\subseteq X$.

Przykład

Tabela $\mathbf{Supplies}$ zawiera dane zarówno o dostawcach jak i dostawach. Symptomem tego jest zależność funkcyjna $\{\#S\} \rightarrow \{City\}$ spełniana przez $\mathbf{Supplies}$: #City zależy od #S a nie od całego klucza kandydującego.

Usuwanie Anomalii Modyfikacji (Normalizacja)

Suppliers

#S	City
S1	London
S2	Paris
S3	Paris
S4	London

SP

#S	#P	QTY
S1	P1	300
S1	P2	200
S1	P3	400
S1	P4	200
S1	P5	100
S1	P6	100
S2	P1	300
S2	P2	400
S3	P2	200
S4	P2	200
S4	P4	300
S4	P5	400

Normalizacja

Anomalie modyfikacji usuwamy **normalizując** schemat bazy: zastępując każdą ze zmiennych relacyjnych dla których występują takie anomalie (zmienne takie nazywamy **nieznormalizowanymi**) dwoma lub więcej tabelami nie wykazującymi anomalii (każda z nich zawiera dane dotyczące tylko jednego rodzaju „obiektu").

Przykład

Należy zastąpić zmienną $\mathbf{Supplies}$ dwiema tabelami: $\mathbf{Suppliers} := \pi_{\{\#S,City\}}(\mathbf{Supplies})$ i $\mathbf{SP} := \pi_{\{\#S,\#P,QTY\}}(\mathbf{Supplies})$. Można pokazać że $\mathbf{Supplies} = \mathbf{Suppliers} \bowtie \mathbf{SP}$, zatem nowa baza ma tą samą zawartość informacyjną.

Postać Normalna Boyca/Codda (BCNF)

Jak widać z poprzedniego przykładu jedną z oznak redundacji w zmiennej relacyjnej $\mathbf{R}$ jest spełnianie przez $\mathbf{R}$ nietrywialnych zależności funkcyjnych $X \rightarrow Y$ dla których X nie jest nadkluczem $\mathbf{R}$.

Definicja

Mówimy że zmienna relacyjna $\mathbf{R}$ jest w **postaci normalnej Boyca/Codda (BCNF)** wtedy i tylko wtedy gdy dla każdej zależności funkcyjnej $X \rightarrow Y$ spełnianej przez $\mathbf{R}$ mamy że albo $Y \subseteq X$ (zależność jest trywialna) albo X jest nadkluczem $\mathbf{R}$.

Postać Normalna Boyca/Codda (BCNF)

Uwagi

- Gdy $\mathbf{R}$ jest w postaci **BCNF** wówczas w większości przypadków jest wolna od **anomalii modyfikacji**.
- Gdy $\mathbf{R}$ nie jest w postaci **BCNF** wówczas powinno się zastąpić $\mathbf{R}$ kilkoma zmiennymi relacyjnymi których wartości są wynikami projekcji $\mathbf{R}$ na odpowiednie podzbiory atrybutów.
- Powstaje pytanie czy możliwy jest taki wybór projekcji który gwarantuje że dane przechowywane w $\mathbf{R}$ można odzyskać z nowych zmiennych dokonując ich naturalnego złączenia?

Bezstratna Dekompozycja Zmiennych Relacyjnych

Definicja

Niech $X_i \subseteq \text{Attr}(\mathbf{R})$, $i \in \{1, \ldots, n\}$ będzie rodziną podzbiorów zbioru atrybutów zmiennej relacyjnej $\mathbf{R}$ taką że $\text{Attr}(\mathbf{R}) = \bigcup_{i \in \{1,\ldots,n\}} X_i$. Mówimy że $\mathbf{R}$ można **bezstratnie podzielić** pomiędzy projekcje $\pi_{X_i}(\mathbf{R})$, $i \in \{1, \ldots, n\}$ wtedy i tylko wtedy gdy dla każdej legalnej wartości $\mathbf{R}$ zachodzi $\mathbf{R} = \pi_{X_1}(\mathbf{R}) \bowtie \cdots \bowtie \pi_{X_n}(\mathbf{R})$.

Twierdzenie (Heath'a)

Przypuśćmy że zmienna relacyjna $\mathbf{R}$ spełnia zależność funkcyjną $X \to Y$. Oznaczmy $\bar{Y} := \text{Attr}(R) \setminus Y$. Wówczas dla dowolnej legalnej wartości $\mathbf{R}$ mamy $\mathbf{R} = \pi_{X \cup Y}(\mathbf{R}) \bowtie \pi_{X \cup \bar{Y})}(\mathbf{R})$.

Dowód Twierdzenia Heath'a

Twierdzenie (Heath'a)

Przypuśćmy że zmienna relacyjna $\mathbf{R}$ spełnia zależność funkcyjną $X \to Y$. Oznaczmy $\bar{Y} := \text{Attr}(R) \setminus Y$. Wówczas dla dowolnej legalnej wartości $\mathbf{R}$ mamy $\mathbf{R} = \pi_{X \cup Y}(\mathbf{R}) \bowtie \pi_{X \cup \bar{Y})}(\mathbf{R})$.

Oznaczmy $Z := X \cup Y$, $Z' = X \cup \bar{Y}$. Wtedy $\text{Attr}(\mathbf{R}) = Z \cup Z'$. Ponieważ $t = t|_Z \bowtie t|_{Z'} \in \pi_Z(\mathbf{R}) \bowtie \pi_{Z'}(\mathbf{R})$ dla każdego $t \in \mathbf{R}$ stąd $R \subseteq \pi_Z(\mathbf{R}) \bowtie \pi_{Z'}(\mathbf{R})$. Aby wykazać zawieranie w przeciwną stronę przypuśćmy teraz że $t \in \pi_Z(\mathbf{R}) \bowtie \pi_{Z'}(\mathbf{R})$. Wtedy istnieją $t', t'' \in \mathbf{R}$ takie że

- $t'|_X = t''|_X$ (zauważmy że $Z \cap Z' = X$)
- $t = (t'|_Z) \bowtie (t''|_{Z'})$.

Ponieważ $\mathbf{R} \models X \to Y$ więc z $t'|_X = t''|_X$ mamy $t'|_Z = t''|_Z$. Stąd $t = (t'|_Z) \bowtie (t''|_{Z'}) = (t''|_Z) \bowtie (t''|_{Z'}) = t'' \in \mathbf{R}$.

Dekompozycja Zmiennej Do Postaci BCNF

Mając daną tabelę $\mathbf{R}$ i zbiór $\mathcal{F}$ wszystkich zależności funkcyjnych spełnianych przez $\mathbf{R}$ postępujemy (w uproszczeniu) następująco:

- Jeśli istnieje jakiś $X \to Y \in \mathcal{F}$ taki taki że $Y \not\subseteq X$ i X nie jest nadkluczem, wówczas należy zastąpić zmienną $\mathbf{R}$ dwiema zmiennymi o wartościach równych projekcjom $\pi_{X \cup Y}(\mathbf{R})$ i $\pi_{X \cup (\mathsf{Attr}(\mathbf{R}) \setminus Y)}(\mathbf{R})$.
  - W razie potrzeby, zmienne te dzielić dalej.
- Jeśli taki $X \to Y$ nie istnieje oznacza to że $\mathbf{R}$ jest w **BCNF**

Proces (bezstratnego) zastępowania zmiennych relacyjnych ich odpowiednimi projekcjami w celu uzyskania sytuacji w której wszystkie tabele bazy znajdują się w odpowiedniej postaci normalnej (tu: **BCNF**) nazywa się **normalizacją** lub **sprowadzaniem do postaci normalnej**.

Jest jasne że algorytm powyższy nie określa ostatecznych projekcji jednoznacznie. Istnieją oczywiście rozkłady lepsze i gorsze.

Przykład Niejednoznacznej Dekompozycji

Rozważmy zmienną relacyjną $\mathbf{R}$ dla której

$$\text{Attr}(\mathbf{R}) = \{\#R, A, K\}, \quad \mathbf{R} \models \{\#R\} \to \{K\}, \quad \mathbf{R} \models \{K\} \to \{A\}$$

(z aksjomatów Armstronga otrzymujemy z powyższych zależności że $\mathbf{R} \models \{\#R\} \to \text{Attr}(R)$, tzn. $\{\#R\}$ jest **kluczem kandydującym** $\mathbf{R}$)

$\mathbf{R}$ nie jest w **BCNF** (z powodu $\mathbf{R} \models \{K\} \to \{A\}$).

Możemy rozbić R na dwie projekcje (w **BCNF**) na dwa sposoby:

$\pi_{\{\#R,A\}}(\mathbf{R})$ i $\pi_{\{\#R,K\}}(\mathbf{R})$

Nowe zmienne relacyjne spełniać będą jedynie zależności funkcyjne $\{\#R\} \to \{A\}$, $\{\#R\} \to \{K\}$. Zależność $\{K\} \to \{A\}$ została utracona.

$\pi_{\{\#R,K\}}(\mathbf{R})$ i $\pi_{\{K,A\}}(\mathbf{R})$

Nowe zmienne relacyjne spełniać będą zarówno $\{\#R\} \to \{K\}$ jak i $\{K\} \to \{A\}$. Wszystkie oryginalne zależności funkcyjne zostały odzwierciedlone w nowej bazie.

Dekompozycje Zachowujące Zależności

Definicja

Niech $\mathbf{R}$ będzie zmienną relacyjną i niech $X, Y \subseteq \text{Attr}(\mathbf{R})$ będą takie że $X \cup Y = \text{Attr}(R)$. Mówimy że zmienne $\mathbf{R}_1 := \pi_X(\mathbf{R})$ i $\mathbf{R}_2 := \pi_Y(\mathbf{R})$ są **niezależne** wtedy i tylko wtedy gdy

- Każda zależność funkcyjna spełniana przez $\mathbf{R}$ jest konsekwencją logiczną zależności funkcyjnych spełnianych przez $\mathbf{R}_1$ i $\mathbf{R}_2$.
- $\text{Attr}(\mathbf{R}_1) \cap \text{Attr}(\mathbf{R}_2)$ jest kluczem kandydującym co najmniej jednej ze zmiennych $\mathbf{R}_1$ i $\mathbf{R}_2$.

Dekompozycje Zachowujące Zależności

Uwagi

- Jeśli zmienną $\mathbf{R}$ zamienimy na **niezależne projekcje**, wówczas nie tylko zostaną zachowane wszystkie dane przechowywane w $\mathbf{R}$ (wartość $\mathbf{R}$ można odzyskać wykonując złączenie projekcji) ale również wszystkie zależności funkcyjne spełniane przez $\mathbf{R}$ zostaną odzwierciedlone w zależnościach funkcyjnych spełnianych przez te projekcje.
- Niestety istnieją zmienne nie będące w BCNF dla których nie istnieją niezależne projekcje i dla których zatem normalizacja skończy się utratą nie tyle danych co zależności w danych

Tabela Nie Normalizowalna Bez Utraty Zależności

$\mathbf{R}$

Student	**Subject**	**Teacher**
Smith	Math	Prof. White
Smith	Physics	Prof. Green
Jones	Math	Prof. White
Jones	Physics	Prof. Brown

Zależności funkcyjne na $\mathbf{R}$:
$\{Student, Subject\} \rightarrow \{Teacher\}$,
$\{Teacher\} \rightarrow \{Subject\}$

- $\mathbf{R}$ nie jest w BCNF (z uwagi na czerwoną zależność funkcyjną)
- Niestety żadne projekcje z $\mathbf{R}$ nie zachowują wszystkich zależności funkcyjnych. $\mathbf{R}$ jest przykładem zmiennej relacyjnej której nie uda się znormalizować do BCNF zachowując zależności funkcyjne.
- $\mathbf{R}$ jest natomiast w **trzeciej postaci normalnej**
- **3NF** choć dopuszcza więcej anomalii modyfikacji niż BCNF jest często stosowana w praktyce jako wystarczająca.

Trzecia Postać Normalna

Definicja

Zmienna relacyjna $\mathbf{R}$ znajduje się w **trzeciej postaci normalnej** (**3NF**) wówczas gdy dla każdej zależności funkcyjnej $X \rightarrow Y$ spełnianej przez $\mathbf{R}$ zachodzi jeden z warunków:

- X jest nadkluczem $\mathbf{R}$,
- $Y \subseteq X$,
- Każdy atrybut $A \in Y \setminus X$ jest **pierwszorzędny** (czyli jest elementem jakiegoś **klucza kandydującego** $\mathbf{R}$).

Twierdzenie

Dla dowolnej zmiennej relacyjnej zawsze istnieje rozkład na zmienne relacyjne w **3NF** dokonany przy pomocy ciągu niezależnych projekcji (a zatem zachowujący zależności funkcyjne).

Zależności Wielowartościowe. Przykład

CTX

Course	Teacher	Text
Physics	Prof. Green	Basic Mechanics
Physics	Prof. Green	Principles of Optics
Physics	Prof. Brown	Basic Mechanics
Physics	Prof. Brown	Principles of Optics
Math	Prof. Green	Basic Mechanics
Math	Prof. Green	Vector Analysis
Math	Prof. Green	Trigonometry

Zmienna relacyjna **CTX** przechowuje informacje o kursach, wykładowcach i podręcznikach przewidzianych dla danego kursu. Przypuśćmy że **CTX** nie spełnia żadnych nietrywialnych zależności funkcyjnych, natomiast dla każdego przedmiotu (który może być nauczany przez więcej niz jednego wykładowcę istnieje ustalony, niezależny od wykładowcy zbiór podręczników.

Zależności Wielowartościowe. Przykład cd.

Innymi słowy, dla każdych krotek

Course	**Teacher**	**Text**
c	*a*	*x*
c	*b*	*y*

$\subseteq \mathbf{CTX}$

także

Course	**Teacher**	**Text**
c	*a*	*y*
c	*b*	*x*

$\subseteq \mathbf{CTX}$

Mówimy że **CTX** spełnia **zależność wielowartościową** $\{\mathbf{Course}\} \twoheadrightarrow \{\mathbf{Text}\}$ (równoważnie $\{\mathbf{Course}\} \twoheadrightarrow \{\mathbf{Teacher}\}$) co zapisujemy $\mathbf{CTX} \models \{\mathbf{Course}\} \twoheadrightarrow \{\mathbf{Text}\}$

Zależności Wielowartościowe. Przykład cd.

CTX

Course	Teacher	Text
Physics	Prof. Green	Basic Mechanics
Physics	Prof. Green	Principles of Optics
Physics	Prof. Brown	Basic Mechanics
Physics	Prof. Brown	Principles of Optics
Math	Prof. Green	Basic Mechanics
Math	Prof. Green	Vector Analysis
Math	Prof. Green	Trigonometry

Nietrudno zauważyć że zależność wielowartościowa $\{\mathbf{Course}\} \twoheadrightarrow \{\mathbf{Text}\}$ na **CTX** powoduje **anomalie modyfikacji** na **CTX** pomimo że **CTX** jest w **BCNF** (i tym samym w **3NF**).

Przykład Anomalii Modyfikacji na CTX

Zmieniając tytuł książki „Basic Mechanics" na „Advanced Mechanics" dla kursu prowadzonego przez prof. Green'a, trzeba to samo zrobić dla kursu fizyki prowadzonego przez prof. Brown'a.

Zależności Wielowartościowe. Przykład cd.

Łatwo widać, że (dzięki zależności wielowartościowej $\{\mathbf{Course}\} \twoheadrightarrow \{\mathbf{Text}\}$) można rozbić zmienną relacyjną **CTX** bez utraty danych na dwie zmienne **CT** i **CX**, których wartości powinny być odpowiednimi projekcjami z **CTX**. Oryginalną wartość **CTX** można odzyskać jako złączenie $\mathbf{CT} \bowtie \mathbf{CX}$.

CT

Course	Teacher
Physics	Prof. Green
Physics	Prof. Brown
Math	Prof. Green

CX

Course	Text
Physics	Basic Mechanics
Physics	Principles of Optics
Math	Basic Mechanics
Math	Vector Analysis
Math	Trigonometry

Ogólna Definicja Zależności Wielowartościowych

Definicja

Niech $\mathbf{R}$ będzie zmienną relacyjną i niech $X, Y \subseteq \text{Attr}(\mathbf{R})$. Oznaczmy $\bar{Y} := \text{Attr}(\mathbf{R}) \setminus (X \cup Y)$. Wówczas

$$\mathbf{R} \models X \twoheadrightarrow Y$$

($\mathbf{R}$ spełnia **zależność wielowartościową** $X \twoheadrightarrow Y$) wtedy i tylko wtedy gdy dla każdej legalnej wartości $\mathbf{R}$, jeśli $t_1, t_2 \in \mathbf{R}$ są krotkami takimi że $t_1|_X = t_2|_X$ wówczas istnieją również krotki $t_3, t_4 \in \mathbf{R}$ takie że

$$t_3|_X = t_4|_X = t_1|_X,$$
$$t_3|_Y = t_1|_Y, \quad t_4|_Y = t_2|_Y, \quad t_3|_{\bar{Y}} = t_2|_{\bar{Y}}, \quad t_4|_{\bar{Y}} = t_1|_{\bar{Y}}.$$

Własności Zależności Wielowartościowych

Niech $\mathbf{R}$ będzie zmienną relacyjną i niech $X, Y \subseteq \text{Attr}(\mathbf{R})$. Oznaczmy $\bar{Y} := \text{Attr}(\mathbf{R}) \setminus (X \cup Y)$. Wówczas:

Twierdzenie

- Jeśli $\mathbf{R} \models X \to Y$ to $\mathbf{R} \models X \twoheadrightarrow Y$ (Czyli zależności funkcyjne są szczególnym rodzajem zależności wielowartościowych)
- $\mathbf{R} \models X \twoheadrightarrow Y$ wtedy i tylko wtedy gdy $\mathbf{R} \models X \twoheadrightarrow \bar{Y}$
- $\mathbf{R} \models X \twoheadrightarrow Y$ jeśli $X \cup Y = \text{Attr}(\mathbf{R})$ (czyli $\bar{Y} = \emptyset$)

Twierdzenie Fagina (uogólnia Twierdzenie Heatha)

$\mathbf{R} = \pi_{X \cup Y}(\mathbf{R}) \bowtie \pi_{X \cup \bar{Y}}(\mathbf{R})$ dla każdej legalnej wartości $\mathbf{R}$ wtedy i tylko wtedy gdy $\mathbf{R} \models X \twoheadrightarrow Y$.

Czwarta Postać Normalna (4NF)

Definicja

Mówimy że zmienna relacyjna $\mathbf{R}$ jest w **czwartej postaci normalnej** (**4NF**) wtedy i tylko wtedy gdy zawsze gdy $\mathbf{R} \models X \twoheadrightarrow Y$ wówczas albo X jest **nadkluczem** $\mathbf{R}$, albo $X \supseteq Y$, albo $X \cup Y = \text{Attr}(\mathbf{R})$.

Twierdzenie

Każda zmienna relacyjna która jest w **4NF** jest równiez w **BCNF**.

Istnieją zmienne relacyjne w **BCNF** które nie są w **4NF**.

Normalizacja do 4NF

Dzięki **Twierdzeniu Fagina** każdą zmienną relacyjną która nie jest w **4NF** można rozbić sukcesywnymi projekcjami na mniejsze tabele z których każda jest w **4NF**.

Systemy Baz Danych

Zapytania

Bartosz Zieliński

WYDZIAŁ FIZYKI
i INFORMATYKI STOSOWANEJ
Uniwersytet Łódzki

Zapytania do Bazy Danych

Zapytanie (kwerenda, *query*) do bazy danych

to polecenie wysłane do DBMS celem uzyskania informacji na podstawie danych zgromadzonych w bazie danych.

- W SQL służy do tego polecenie **SELECT**.
- Zapytanie może być też częścią polecenia modyfikującego dane.

Uwaga terminologiczna

Niektórzy autorzy używają terminu **zapytanie funkcjonalne** na oznaczenie poleceń które modyfikują dane lub ich strukturę (np. **DELETE, CREATE TABLE,** itp.). Jest też powszechną praktyką nazywanie zapytaniami dowolnych poleceń SQL. Na tym wykładzie jednak, o ile nie powiedziano inaczej, **zapytania służą wyłącznie do uzyskiwania danych a nie ich modyfikacji**.

Przykładowy Schemat Bazy Danych

Skorzystamy z przykładowej bazy danych ze zmiennymi utworzonymi poleceniami

```
1  CREATE TABLE Jobs (
2      JobId INTEGER PRIMARY KEY,
3      Name Varchar(20) NOT NULL UNIQUE,
4      MaxSalary Number(8,2) NOT NULL,
5      MinSalary Number(8,2) NOT NULL
6  );
7
8  CREATE TABLE Employees (
9      Id INTEGER PRIMARY KEY,
10     FirstName VARCHAR(30) NOT NULL,
11     LastName VARCHAR(30) NOT NULL,
12     Salary NUMERIC(6,2) NOT NULL CHECK(Salary >= 4000),
13     JobId INTEGER NOT NULL REFERENCES Jobs(JobId),
14     ManagerId INTEGER REFERENCES Employees(Id),
15     UNIQUE(FirstName, LastName)
16 );
```

Przykładowa Baza Danych

Employees

Id	FirstName	LastName	Salary	JobId	ManagerId
1	Toru	Takemitsu	10000.11	1	**null**
2	Philip	Glass	9000.00	3	1
3	Michael	Nyman	10000.50	1	1
4	Henryk	Górecki	11000.00	1	1
5	Thomas	Tallis	8000.80	2	3
6	Arvo	Pärt	15000.70	1	3
7	Arnold	Schönberg	6000.00	2	1
8	Anton	Webern	6500.12	2	2
9	Alban	Berg	6750.50	2	3
10	Olivier	Messiaen	9500.00	3	1

Jobs

JobId	Name	MinSalary	MaxSalary
1	IT Specialist	8000	20000
2	Sales Specialist	5000	9000
3	Administration	7000	10000

Przykładowe Zapytania (Nieformalne)

1. Podać imiona, nazwiska i pensje wszystkich pracowników.
2. Podać imiona, nazwiska i pensje wszystkich pracowników posortowane w porządku rosnącym według pensji.
3. Podać imiona i nazwiska pracowników zarabiających więcej niż 10000.
4. Podać imię, nazwisko i nazwę (**Name**) funkcji pełnionej przez pracownika z $\mathbf{Id} = 3$.
5. Dla każdej nazwy (**Name**) funkcji podać ilość pracowników pełniących tą funkcję i ich sumaryczne pensje.
6. Dla każdej nazwy funkcji podać imiona i nazwiska pracowników zarabiających maksymalną pensję wśród pracowników z tą funkcją.

Kwestie Do Rozstrzygnięcia

1. Czym właściwie jest odpowiedź na zapytanie?
2. Jakie są dopuszczalne zapytania?
3. Jak powinny być formułowane zapytania (skoro zapytania w języku naturalnym nie wchodzą w grę jako zbyt mało precyzyjne i jednoznaczne).

Odpowiedzi na Zapytania w Relacyjnych DBMS-ach

W relacyjnych DBMS-ach odpowiedzią na zapytanie jest zawsze relacja

Przykład

Odpowiedzią na zapytanie „*Dla każdej nazwy funkcji podać imiona i nazwiska pracowników zarabiających maksymalną pensję wśród pracowników z tą funkcją*" mogłaby być relacja:

Name	**FirstName**	**LastName**
Sales Specialist	Thomas	Tallis
IT Specialist	Arvo	Pärt
Administration	Olivier	Messiaen

Sortowanie Wyników Zapytania

Odpowiedzi na zapytania są **zbiorami** krotek, zatem krotki w odpowiedzi są z definicji **nieuporządkowane**. Co jednak zrobić z zapytaniami które jawnie żądają uporządkowania wyników?

Podać imiona, nazwiska i pensje wszystkich pracowników posortowane w porządku rosnącym według pensji.

- Formalnie krotki w odpowiedzi są nieuporządkowane, ale w jakiejś kolejności trzeba je wysłać do klienta.
- Można więc równie dobrze wysłać je w kolejności jakiej życzy sobie klient traktując to bardziej jako formę optymalizacji niż dodawanie informacji do wyniku.
- Wszystkie RDBMS-y pozwalają na sortowanie krotek odpowiedzi.

Dopuszczalne Zapytania

Pewne klasy zapytań można (niezależnie od języka zapytań) wykluczyć jako patologiczne. Zaczniemy od oczywistego przykładu.

Nieskończone odpowiedzi

Odpowiedź na zapytanie w RDBMS jest z definicji **skończonym** zbiorem krotek. Zatem musimy jako bezsensowne potraktować każde zapytanie którego wynikiem może być nieskończona relacja. Takie zapytanie jest zaskakująco łatwo zadać, np.

Podaj wszystkie pensje których nie zarabiają pracownicy

Okazuje się jednak że wymieniona wyżej patologia jest szczególnym przypadkiem ogólniejszej patologii opisanej dalej.

Przykład Nietrywialnie Patologicznego Zapytania

Przypuśćmy że baza danych zawiera zmienną relacyjną **V** z pojedyńczym atrybutem $\mathbf{A}$ której aktualną wartością jest relacja $\begin{array}{|c|}\hline \mathbf{A} \\ \hline \mathsf{f} \\ \mathsf{g} \\ \hline \end{array}$.

Rozważmy zapytanie

Czy **V** *zawiera wszystkie możliwe wartości w kolumnie* **A***?*

które może mieć odpowiedzi $\begin{array}{|c|}\hline ? \\ \hline \mathsf{tak} \\ \hline \end{array}$ (jeśli $\mathcal{U} = \{f, g\}$) lub $\begin{array}{|c|}\hline ? \\ \hline \mathsf{nie} \\ \hline \end{array}$ (w przeciwnym wypadku).

Zatem odpowiedź zależy tu nie tylko od danych przechowywanych w bazie ale również od zawartości uniwersum wartości $\mathcal{U}$. Co jest z tym nie tak?

Domain Independent Queries

Intuicyjnie wyniki zapytania powinny zależeć wyłącznie od

- wartości przechowywanych w bazie,
- wartości jawnie wymienionych w zapytaniu
- wartości które można obliczyć na podstawie tych z pierwszych dwóch punktów (o ile DBMS dopuszcza obliczenia domenowe)

nie zaś od (arbitralnie wybranego przez autora DBMS-u) uniwersum możliwych wartości.

Zapytania których wyniki spełniają ten warunek nazywają się **niezależnymi od dziedziny** (*domain independent*).

Zapytania Deklaratywne

Zapytania w języku naturalnym takie jak

Podać imiona i nazwiska pracowników zarabiających więcej niż 10000

są **deklaratywne** — definiują dane które są potrzebne, nie mówią jednak jak je uzyskać. Alternatywą byłoby podawanie dla każdego zapytania algorytmu uzyskania odpowiedzi, np.

$c := \mathbf{open}(\text{“Employees.dat”})$
$\mathbf{while}(e = \mathbf{read}(c))$
  $\mathbf{if}(e.salary \geq 10000)$

$$\mathbf{print}\left(\begin{array}{|ll|}\hline \textbf{Imię} & \textbf{Nazwisko} \\ \hline \text{e.}\textbf{FirstName} & \text{e.}\textbf{LastName} \\ \hline\end{array}\right)$$

Wiele wczesnych języków zapytań wyglądało właśnie w ten sposób: należało podać algorytm.

Formułowanie Zapytań

Podjęzyk zapytań SQL jest oparty na ideach z **rachunku relacyjnego** i **algebry relacyjnej**.

Rachunek relacyjny

Jest językiem deklaratywnym. Upraszczając, zapytania są tu tłumaczeniem na język logiki i teorii zbiorów deklaratywnych zapytań w języku naturalnym takich jak pokazane wcześniej.

Algebra Relacyjna

Definiuje podstawowe operacje i sposoby ich łączenia. Formalizm jest bardziej proceduralny (algorytmiczny): opisujemy jakie operacje trzeba wykonać aby uzyskać odpowiedź na zapytanie.

Zapytania Koniunktywne

Zanim przejdziemy do omówienia ogólnej postaci zapytań w SQL, najpierw zapoznamy się na następnym wykładzie z **algebrą relacyjną**. **Rachunek relacyjny** wykracza poza zakres tych zajęć. Zamiast ogólnego rachunku relacyjnego omówimy klasę szczególnie prostych zapytań SQL które bezpośrednio odpowiadają tzw. **zapytaniom koniunktywnym** (nieco uogólnionym) z **rachunku relacyjnego**.

Zapytania SELECT-FROM-WHERE w SQL

$$\textbf{SELECT } E_1 \textbf{ AS } A_1, E_2 \textbf{ AS } A_2, \ldots, E_m \textbf{ AS } A_m$$
$$\textbf{FROM } R_1\ t_1, R_2\ t_2, \ldots, R_n\ t_n$$
$$\textbf{WHERE } \phi;$$

$R_1, \ldots, R_n$ to nazwy zmiennych relacyjnych. Powyższe zapytanie zwraca relację zawierającą, dla każdej kombinacji krotek $t_1 \in R_1, \ldots, t_n \in R_n$ takich że warunek ϕ jest spełniony krotkę

A_1	A_2	$\ldots$	A_m
E_1	E_2	$\ldots$	E_m

.

Przykład Zapytania Koniunktywnego

Wypisać imiona, nazwiska, pensje i wartość podatku (20%) płaconego przez pracowników zarabiających ≥ 11000.

Zapytanie SQL

```
1  SELECT e.FirstName AS Imię,
2         e.LastName AS Nazwisko,
3         e.Salary AS Pensja,
4         0.2 * e.Salary AS Podatek
5  FROM Employees e
6  WHERE e.Salary >= 11000;
```

Wynik zapytania

Imię	Nazwisko	Pensja	Podatek
Henryk	Górecki	11000.00	2200.00
Arvo	Pärt	15000.70	3014.00

Przykład Zapytania Koniunktywnego

które odwołuje się do dwóch zmiennych relacyjnych

Podać dla każdego pracownika zarabiającego 11000 i więcej jego imię, nazwisko, wartość pensji, i nazwę stanowiska.

Zapytanie SQL

```
1  SELECT e.FirstName AS Imię,
2         e.LastName AS Nazwisko,
3         e.Salary AS Pensja,
4         j.Name as "Nazwa Stanowiska"
5  FROM Employees e, Jobs j
6  WHERE e.Salary >= 11000 AND j.JobId=e.JobId;
```

Wynik zapytania

Imię	Nazwisko	Pensja	Nazwa Stanowiska
Henryk	Górecki	11000.00	Administration
Arvo	Pärt	15000.70	Administration

Inny Przykład Zapytania Koniunktywnego

które odwołuje się do dwóch zmiennych relacyjnych

Podać imię, nazwisko i pensję każdego pracownika którego pensja nie mieści się w widełkach płacowych związanych ze stanowiskiem.

Zapytanie SQL

```
1  SELECT e.FirstName AS Imię,
2         e.LastName AS Nazwisko,
3         e.Salary AS Pensja
4  FROM Employees e, Jobs j
5  WHERE j.JobId=e.JobId AND (
6      e.Salary < j.MinSalary OR e.Salary > j.MaxSalary
7  );
```

Wynik zapytania

Imię	Nazwisko	Pensja

Przykład Zapytania Koniunktywnego

Które odwołuje się dwukrotnie do tej samej tabeli

Dla każdego pracownika podać jego imię i nazwisko i imię i nazwisko bezpośredniego przełożonego.

Zapytanie SQL

```
1  SELECT e.FirstName AS Imię,
2         e.LastName AS Nazwisko,
3         m.FirstName AS "Imię przełożonego",
4         m.LastName AS "Nazwisko przełożonego",
5  FROM Employees e, Employees m
6  WHERE e.ManagerId = m.EmployeeId;
```