Uniwersytet Wrocławski

Bazy danych 2022

Wykład 3

Normalizacja, zależności funkcyjne; zapytania agregujące i grupujące

Bartosz Brzostowski

Wydział Fizyki i Astronomii UWr
semestr letni r. akad. 2021/22

17 marca 2022

Motywacja

- Utrzymanie „porządku” w bazie danych przez wybranie właściwego *schematu*
- Uniknięcie anomalii:
  - Anomalia aktualizacji: jeśli jakaś dana występuje w kilku miejscach, to możliwa jest jej niekonsekwentna modyfikacja
  - Anomalia wstawiania: niektóre dane mogą być niemożliwe do wstawienia, bo są niekompletne (tylko z punktu widzenia bazy, nie rzeczywistości!)
  - Anomalia usuwania: usuwanie pewnych danych może być możliwe tylko jeśli usunie się coś więcej
- Schemat powinien być łatwy do zmodyfikowania przy przeprojektowywaniu bazy
- Normalizacja: stopniowe modyfikowanie schematu przez rozbijanie relacji (tabel) na mniejsze
- Zachowuje atrybuty, informację (tj. powstałe tabele można złączyć i otrzymać tę samą wiedzę) i *zależności funkcyjne*

Relacja (bardzo) nieznormalizowana

Zakład	Pracownicy
ZFN	Sobczyk, Golan, Graczyk, Juszczak, ...
ZISFS	Koza, Bancewicz, Durka, Grech, Kondrat, ...
	...

- ... albo wręcz:

Zakład	Pracownicy
ZFN	Stopień | Nazwisko prof | Sobczyk dr | Golan ...
ZISFS	Stopień | Nazwisko prof | Koza mgr | Bancewicz ...
...	

Pierwsza postać normalna (1NF)

- Dane są *atomowe* — kolumny zawierają pojedyncze wartości
- Dla tabel zagnieżdżonych — wyrzucamy „na zewnątrz”, dodajemy klucz obcy wskazujący odpowiedni wiersz tabeli oryginalnej
- Atomizacja wartości (głupio):

Zakład	Prac 1	Prac 2	...
ZFN	Sobczyk	Golan	
ZISFS	Koza	Durka	
	...		

- Atomizacja wartości (mądrze):

Zakład	Pracownicy
ZFN	Sobczyk
ZFN	Golan
ZFN	Graczyk
	...
ZISFS	Koza
ZISFS	Bancewicz
ZISFS	Durka
	...

Zależności funkcyjne

- A, B — podzbiory schematu relacji R (zbiory atrybutów). B zależy funkcyjnie od A, lub A wyznacza (funkcyjnie) B (ozn. $A \to B$), jeśli dla dowolnych wierszy x, y zachodzi

$$\forall x, y \qquad x.A = y.A \quad \Rightarrow \quad x.B = y.B$$

- Są własnością modelowanej rzeczywistości, a nie bazy!
- Przykład: dla relacji o schemacie

$$R = \{\text{pesel}, \text{nazwisko}, \text{imię}, \text{kodPocz}, \text{miasto}, \text{ulica}, \text{dom}\}$$

zachodzą (między innymi) zależności

$$\begin{aligned} \text{pesel} &\to \{\text{nazwisko}, \text{imię}\} \\ \{\text{miasto}, \text{ulica}, \text{dom}\} &\to \text{kodPocz} \\ \text{kodPocz} &\to \text{miasto} \end{aligned}$$

A także: pesel → nazwisko (ale ona wynika z tych) a także trywialne, np. {pesel, kodPocz, dom} → kodPocz (trywialne to A → B dla B ⊆ A)

- Nie zachodzą:

$$\begin{aligned} \{\text{nazwisko}, \text{imię}\} &\not\to \{\text{miasto}, \text{ulica}, \text{dom}\} \\ \{\text{miasto}, \text{ulica}, \text{dom}\} &\not\to \{\text{nazwisko}, \text{imię}\} \\ \text{miasto} &\not\to \text{kodPocz} \end{aligned}$$

Klucze

- *Nadklucz* — zbiór atrybutów A, który wyznacza wszystkie atrybuty relacji: $A \rightarrow R$. Alternatywnie

$$\forall x, y \qquad x \neq y \quad \Rightarrow \quad x.A \neq y.A$$

- Trywialnym nadkluczem jest zawsze całe R
- *Klucz* — minimalny nadklucz, tj. taki, którego nietrywialne podzbiory już nie są nadkluczami (jeden można wyróżnić jako klucz główny, pozostałe — kandydujące)
- *Atrybut główny* — atrybut należący do *dowolnego* klucza

np. z zależnością {miasto, ulica, dom} → kodPocz z poprzedniego slajdu w relacji R'={miasto, ulica, dom, kodPocz} (nad)kluczem jest {miasto, ulica, dom}, ale żadne podzbiory ({miasto, ulica}, {miasto, dom}, {ulica, dom} i tym bardziej 1-elementowe) już nie!

Zależności częściowe i 2NF

- *Zależność częściowa*: $A \to b$, gdzie A — nietrywialny podzbiór jakiegoś klucza, b — atrybut nie będący głównym (nie należy do żadnego klucza)
- Dla $R = \{\text{produkt}, \text{nrZamow}, \text{cena}, \text{dataZamow}\}$ kluczem jest $\{\text{produkt}, \text{nrZamow}\}$
- Zachodzą zależności częściowe:

$$\text{produkt} \to \text{cena} \qquad \text{nrZamow} \to \text{dataZamow}$$

- Relacja jest w *drugiej postaci normalnej*, jeśli jest w 1NF i nie występują zależności częściowe
- Rozkład dla R:

$$\begin{aligned} R_1 &= \{\text{produkt}, \text{nrZamow}\} \\ R_2 &= \{\text{produkt}, \text{cena}\} \\ R_3 &= \{\text{nrZamow}, \text{dataZamow}\} \end{aligned}$$

Zależności przechodnie i 3NF

nrZamow	klient	adresKlienta
001	UWr	pl. Uniwersytecki 1
002	UWr	~~pl. Uniwersytecki 1~~ pl. Borna 9

Mogę zmienić adres na "pl. Borna 9" w dokładnie jednym z tych 2 wierszy → anomalia aktualizacji!

- *Zależność przechodnia* $X \to Y$ to taka, że istnieje Z takie, że $X \to Z$ i $Z \to Y$
- $R = \{\text{nrZamow}, \text{klient}, \text{adresKlienta}\}$, klucz: nrZamow
- Zależność nrZamow $\to$ adresKlienta jest przechodnia, bo istnieją nrZamow $\to$ klient i klient $\to$ adresKlienta
- Relacja jest w *trzeciej postaci normalnej*, jeśli jest w 2NF i nie ma przechodnich zależności, które z nadklucza wyznaczają atrybuty nie będące głównymi
- Tutaj problemem jest klient $\to$ adresKlienta
- Inne sformułowanie: zależności muszą albo mieć nadklucz po lewej, albo atrybuty główne po prawej
- Rozkład dla R:

$$R_1 = \{\text{nrZamow}, \text{klient}\} \qquad R_2 = \{\text{klient}, \text{adresKlienta}\}$$

Algorytm normalizacji do 3NF

- Weź zbiór F wszystkich zależności funkcyjnych relacji R i uprość go
  - Jeśli $\mathrm{A} \subseteq \mathrm{B}$, to $\mathrm{B} \to \mathrm{A}$ jest trywialna
  - Jeśli $\mathrm{A} \subseteq \mathrm{B}$ oraz $\mathrm{A} \to \mathrm{C}, \mathrm{B} \to \mathrm{C}$, to druga — „nadmiarowa”
  - $\mathrm{A} \to \{\mathrm{b}_1, \mathrm{b}_2, \ldots\}$ można rozbić na $\mathrm{A} \to \mathrm{b}_1, \mathrm{A} \to \mathrm{b}_2, \ldots$
- Otrzymujemy *minimalne pokrycie* F: $\mathrm{F}_{\mathrm{min}} = \{\mathrm{A}_i \to \mathrm{b}_i\}_i$ takie, że każdy A_i jest „minimalny”, $\mathrm{b}_i \notin \mathrm{A}_i$ oraz z $\mathrm{F}_{\mathrm{min}}$ da się wyprowadzić całe F
- Rozkładem R są relacje R_i o schematach $\mathrm{A}_i \cup \{\mathrm{b}_i\}$ (z pominięciem tych, które są podzbiorami innych); jeśli żadna z nich nie zachowała żadnego klucza R, to dodajemy jedną o atrybutach z (wybranego) klucza
- Taki rozkład zachowuje atrybuty i zależności funkcyjne (te z $\mathrm{F}_{\mathrm{min}}$ — jawnie) oraz informację (odtwarzana przez złączenie relacji), a każda R_i jest w 3NF

Dalsza normalizacja

- Każdą relację da się rozłożyć do 3NF
- Istnieją inne NF: Boyce'a-Codda, czwarta, piąta...
- Nie zawsze istnieje normalizacja!
- BCNF (czasem nazywana 3.5NF): wszystkie zależności funkcyjne $A \to B$ są trywialne ($B \subseteq A$) lub A jest nadkluczem
- Nie zawsze możliwe do osiągnięcia
  - $R = \{a, b, c\}$ z $F = \{ab \to c, c \to a\}$
  - Klucze: ab i bc
  - Nie jest w BCNF: $c \to a$ nietrywialna, ale c nie jest nadkluczem
  - Każdy rozkład gubi zależność funkcyjną $ab \to c$

Denormalizacja

- Czasami dokłada się do bazy pewne struktury (tabele, kolumny), aby zwiększyć wydajność
- Jest różnica między bazą nieznormalizowaną a zdenormalizowaną
- Przykład: atrybuty wyliczane, w tym np. statystyki, które wymagałyby agregacji dużej liczby danych, a można je modyfikować inkrementalnie
- Zmniejszenie liczby tabel w złączeniach (zwłaszcza jeśli mało odfiltrowujemy w pośrednich wynikach obliczeń): w bazie z pracowni moglibyśmy trzymać ID zamawiającego klienta w `detal_zamow`, żeby móc łączyć bez pośrednictwa `zamow`
- Minusy: za szybszy `SELECT` płaci się wolniejszym `INSERT` / `UPDATE` / `DELETE`
- Trzeba to wszystko dobrze napisać (np. wyzwalacze) i nie zepsuć — DBMS już nie dba (sam) o spójność danych

Co umiemy wyciągnąć z bazy?

- Informacje związane z konkretnym bytem (wierszem tabeli) lub ich kombinacją (dla złączeń)
- Co z informacjami o wielu bytach (wierszach) naraz?
- Przykład: ilu sklep ma klientów?

```
mysql> SELECT idk, nazwa, miasto FROM klienci;
+-----+--------------+----------+
| idk | nazwa        | miasto   |
+-----+--------------+----------+
...
11 rows in set (0,00 sec)
```

- Informacja wypisana przez klienta MySQL nie jest wynikiem zapytania!

Zliczanie wierszy

- Najprostszy i najogólniejszy przykład funkcji agregujących
- `COUNT(*)` — liczy wszystkie wiersze
- `COUNT(`*`[wyr]`*`)` — liczy wiersze z nie-NULLową wartością
- `COUNT(DISTINCT` *`[w1]`*`,` *`[w2]`*`, ...)` — liczy różne kombinacje wartości (*żadna* nie może być NULLem)

```
mysql> SELECT COUNT(ilosc), COUNT(DISTINCT ilosc)
    -> FROM produkty;
+--------------+----------------------+
| COUNT(ilosc) | COUNT(DISTINCT ilosc) |
+--------------+----------------------+
|           14 |                    9 |
+--------------+----------------------+
```

- Dla zbioru pustego (tj. złączenia z pustą tabelą lub gdy nic nie spełnia warunku filtrowania): 0
- Funkcje agregujące poza `COUNT(*)` ignorują NULLe (użyteczne, ale można uznać, że nieścisłe)

Funkcje agregujące

- Muszą być zdefiniowane dla dowolnej liczby argumentów (włącznie z 0)
- Muszą być zdefiniowane dla dowolnej kolejności argumentów (niekoniecznie z takim samym wynikiem, np. sklejanie napisów)
- Większość zdefiniowana tylko dla niektórych typów danych
- Dla typów „sortowalnych” (liczby, napisy, daty): `MAX`, `MIN`

```
mysql> SELECT MAX(cena), MIN(cena) FROM produkty;
+-----------+-----------+
| MAX(cena) | MIN(cena) |
+-----------+-----------+
|   3152.00 |    499.00 |
+-----------+-----------+
```

- Jest wariant `MAX(DISTINCT ...)` — dosyć bez sensu
- `MAX(∅)` zwraca ~~$-\infty$ INT_MIN~~ NULL

Liczbowe funkcje agregujące

- Dodawanie i mnożenie: łączne, przemienne — idealne do agregacji
- `SUM(`*[wyr]*`)`, `SUM(DISTINCT` *[wyr]*`)`
- Brak `PRODUCT` — mało życiowy; dla dodatnich można tak:

```
mysql> SELECT EXP(SUM(LN(ilosc))) FROM produkty;
+--------------------+
| EXP(SUM(LN(ilosc))) |
+--------------------+
|  1373718527999.9946 |
+--------------------+
```

- Statystyka: `AVG([DISTINCT] ...)`, `VARIANCE`, `STD`, `...`
- Dla $\emptyset$ wszystkie wymienione zwracają NULL (choć np. dla `SUM` sensowne byłoby 0)
- W różnych DBMS różne nazwy!

Inne funkcje agregujące

- Działania logiczne
  - Też łączne i przemienne
  - Warianty bitowe: `BIT_AND`, `BIT_OR`, `BIT_XOR`
  - `BIT_XOR` nie ma w PostgreSQLu! Ale są `BOOL_AND`, `BOOL_OR`
  - Dla $\emptyset$ zwracają ~~NULL~~ elementy neutralne, tj. wszystkie bity włączone (dla `BIT_AND`) lub wyłączone (pozostałe)
- Napisy
  - Konkatenacja: `GROUP_CONCAT(`*`[wyr1], [wyr2], ...`*`)`
  - Zmiana separatora z domyślnego `","`: `GROUP_CONCAT(`*`[wyr]`* `SEPARATOR` *`[sep]`*`)`
  - Konkatenacja nie jest przemienna!
  - Kontrola nad kolejnością argumentów: `GROUP_CONCAT(`*`[w1]`*`, ... ORDER BY` *`[w2]`* `[DESC], ...)`
  - Dla $\emptyset$ zwraca NULL, choć mogłaby pusty napis
- JSON (dowolny typ argumentów)
  - Tablica: `JSON_ARRAYAGG(`*`[wartości]`*`)`
  - Obiekt: `JSON_OBJECTAGG(`*`[klucze], [wartości]`*`)`

Grupowanie

- Umiemy wypisać zagregowane informacje o wszystkich bytach, ew. spełniających jakiś warunek (klauzula `WHERE`)
- Przykład: liczba klientów z Wrocławia. Ale co z liczbą klientów w rozbiciu na miasta?
- Agregacja w *grupach*

```
mysql> SELECT miasto, COUNT(*) FROM klienci GROUP BY miasto;
+----------+----------+
| miasto   | COUNT(*) |
+----------+----------+
| Toruń    |        1 |
| Warszawa |        2 |
| Wrocław  |        7 |
| Łódź     |        1 |
+----------+----------+
```

- Klauzula `GROUP BY` z listą wyrażeń kolumnowych

Co można SELECTować przy GROUP BY?

- Wyrażenia zależne od tych grupujących, agregaty
- Co poza tym?

```
mysql> SELECT miasto, COUNT(*), idk FROM klienci
    -> GROUP BY miasto;
ERROR 1055 (42000): Expression #3 of SELECT list is not
in GROUP BY clause and contains nonaggregated column
'test.klienci.idk' which is not functionally dependent
on columns in GROUP BY clause; this is incompatible with
sql_mode=only_full_group_by
```

ale:

```
mysql> SELECT nazwa, COUNT(idz)
    -> FROM klienci JOIN zamow ON idk = k_id GROUP BY idk;
+---------------+------------+
| nazwa         | COUNT(idz) |
+---------------+------------+
| Astro         |          1 |
...
```

- Kolumny *zależne funkcyjnie* od grupujących

Zależności funkcyjne

- Zbiór atrybutów B zależy funkcyjnie od A ($\mathrm{A} \to \mathrm{B}$) gdy stan krotki na atrybutach z A jednoznacznie określa stan na B:

$$\forall x, y \qquad x.\mathrm{A} = y.\mathrm{A} \quad \Rightarrow \quad x.\mathrm{B} = y.\mathrm{B}$$

- Przykład: od klucza tabeli zależy cała tabela
- Także dla kluczy kandydujących czy wielokolumnowych
- `SELECT nazwa ... FROM klienci ... GROUP BY idk` — `idk` to klucz, więc określa `nazwa`
- A gdyby zgrupować po `k_id`? Ze względu na równość w warunku złączenia również mamy `k_id` $\to$ `klienci.*`
- Zależności wykrywane automatycznie przez MySQL (z ograniczeniami: np. tylko dla kolumn, nie dla wyrażeń)
- Jak wymusić „reprezentanta” niezagregowanej, niezależnej funkcyjnie kolumny? `ANY_VALUE`

ale żeby mógł to robić, to TRZEBA deklarować klucze w schemacie bazy!

Grupowanie bez agregacji

- Agregacja bez grupowania już była
- Samo grupowanie — „mądrzejszy” `DISTINCT`
- Przykłady (choć trochę wydumane):
  - „Wypisz wszystkie miasta, w których mieszczą się siedziby klientów, którzy składali jakieś zamówienia”

```
SELECT DISTINCT miasto
  FROM klienci
    JOIN zamowienia
    ON idk = k_id
```

  - „Dla każdego klienta, który składał jakieś zamówienia, wypisz miasto, w którym ma siedzibę”

```
SELECT miasto
  FROM klienci
    JOIN zamowienia
    ON idk = k_id
  GROUP BY idk
```

HAVING, komplet klauzul SELECT

- HAVING — jak WHERE, tylko po agregacji / grupowaniu

```
mysql> SELECT miasto, COUNT(*) FROM klienci GROUP BY miasto
    -> HAVING miasto LIKE "W%";
+----------+----------+
| miasto   | COUNT(*) |
+----------+----------+
| Warszawa |        2 |
| Wrocław  |        7 |
+----------+----------+
```

To nie jest za dobry przykład, bo równie dobrze to filtrowanie mogłoby być z WHERE ...
Ale ... HAVING COUNT(*)>1 już nie!

- „Pełne” zapytanie:

5*. SELECT *[wyrażenia_kolumnowe]*
1. FROM *[wyrażenie_tabelowe]*
2. WHERE *[wyrażenie_filtrujące]*
3*. GROUP BY *[wyrażenia_kolumnowe]*
4*. HAVING *[wyrażenie_filtrujące]*
6. ORDER BY *[wyrażenia_kolumnowe]*

* W GROUP BY i HAVING działają aliasy z SELECT (nie-standardowe)