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;
```