Jan Jełowicki
Uniwersytet Przyrodniczy we Wrocławiu, Katedra Matematyki

Ściąga z SQL

materiały pomocnicze do zajęć z informatyki

Wrocław 2004–2014

0. Konwencje

Wszystkie wyrażenia języka SQL wyróżniono `krojem maszynowym` pisma. Słowa kluczowe SQL wyróżniono dodatkowo **`pismem pogrubionym`**. Wartości pochodzące od użytkownika (nazwy tabel, nazwy cech, wartości danych i wyrażeń wyróżniono *`pismem pochyłym`*. Należy je zastąpić czymś sensownym, bezmyślne przepisanie nic nie da.

Wzorce wyrażeń wyróżniono ponadto `kolorem tekstu`. Gotowe przykłady przeznaczone do dosłownego potraktowania, odnoszące się do konkretnej bazy danych, wyróżniono innym `kolorem`.

Osobny rodzaj wyróżnienia zarezerwowano dla odpowiedzi udzielanych przez system baz danych i dla prezentacji stanu bazy.

Do ilustracji pojęć i poleceń posłuży następująca maleńka baza danych:

kolory

kod	nazwa	name
r	czerwony	red
g	zielony	green
y	żółty	yellow
b	niebieski	blue

kwiaty

nazwa	name	barwa
chaber	cornflower	b
róża	rose	r
słonecznik	`null`	y
mak	poppy	r

Kluczem tabeli `kolory` jest pole `kod`. Pole `barwa` tabeli `kwiaty` jest kluczem obcym, odwołującym się do pola `kod` tabeli `kolory`.

Rozdziały 1–3. dotyczą praktycznych aspektów pracy z bazą danych. Rozdział 4. omawia niezbędne podstawy teoretyczne operacji na relacjach. Rozdział 5. omawia typy danych, składnię wyrażeń i składnię warunków wykorzystywanych w zapytaniach.

1. Zarządzanie tabelami

Na zajęciach operacje zarządzania tabelami nie będą potrzebne, ale dla porządku trzeba je wymienić.

1.1. Tworzenie tabeli

W najprostszym przypadku polecenie utworzenia nowej tabeli ma postać:

```
create table tabela (
    nazwa_pola typ_pola,
    nazwa_pola typ_pola,
```

```
    ...
    nazwa_pola typ_pola
);
```

Przykład: Do utworzenia dwóch tabel ilustrujących przykłady w tym opracowaniu można wykorzystać polecenia

```
create table kolory (
    kod char,
    nazwa text,
    name text
);

create table kwiaty (
    nazwa text,
    name text,
    barwa char
);
```

W rzeczywistości potrzebne są dodatkowe klauzule opisujące rolę pól w strukturze bazy danych: pewne pola pełnią rolę kluczy, inne są odwołaniami, jeszcze inne muszą spełniać dodatkowe warunki:

```
create table tabela (
    nazwa_pola typ_pola,
    ...
    nazwa_pola typ_pola,
    więzy_i_ograniczenia
);
```

Na przykład w tabeli `kolory` pole `kod` ma pełnić funkcję klucza. Pole takie nie może być w żadnym rekordzie puste, a jego wartości nie mogą się powtarzać wewnątrz tabeli.

Z kolei w tabeli `kwiaty` pole `barwa` służy do wskazania rekordu z tabeli `kolory`. Wartości tego pola nie mogą więc być dowolne, tylko muszą odpowiadać pewnej wartości klucza tabeli `kolory`.

Przykład:

```
create table kolory (
    kod char,
    nazwa text,
    name text,
    primary key (kod)
);

create table kwiaty (
    nazwa text,
    name text,
    barwa char,
    primary key (nazwa),
    foreign key (barwa) references kolory (kod)
);
```

1.2. Usuwanie tabeli

Tabelę usuwa się zawsze w ten sam sposób:

```
drop table tabela;
```

Tabelę da się usunąć tylko w sytuacji, kiedy żadne jej pole nie jest przedmiotem odwołania w żadnej innej tabeli.

2. Operacje manipulowania danymi w tabeli

Poleceń z tej grupy będziemy używać intensywnie.

2.1. Umieszczanie danych w tabeli

W sytuacji, kiedy znamy wartości wszystkich pól nowego rekordu:

```
insert into tabela values (
    wartość,
    wartość,
    ...
    wartość
);
```

Przykład:

```
insert into kolory values ('v', 'fioletowy', 'violet');
```

W sytuacji, kiedy znamy tylko wartości niektórych pól nowego rekordu, powinniśmy dodać specyfikację pól:

```
insert into tabela (
    nazwa_pola,
    nazwa_pola,
    ...
    nazwa_pola
) values (
    wartość,
    wartość,
    ...
    wartość
);
```

Pozostałe pola otrzymają wartości domyślne. Jeżeli dane pole nie ma wartości domyślnej, to zostanie mu nadana wartość `null`.

Przykład:

```
insert into kolory (kod, nazwa) values ('v', 'fioletowy');
```

W sytuacji, kiedy niektóre wartości nowego rekordu muszą być pobrane z kolumn klucza obcego, można wykorzystać zapytania pozyskiwania danych, o ile gwarantują one jednoznaczność odpowiedzi.

Przykład: Zamiast jawnie wskazywać wartość klucza obcego:

```
insert into kwiaty values (
    'fiołek',
    null,
    'v'                    );
```

lepiej jest skłonić system do samodzielnego odszukania wartości kluczowej.

Przykład:

```
insert into kwiaty values (
    'fiołek',
    null,
    (select kod from kolory where nazwa = 'fioletowy') );
```

Szczegółowa składnia żądań pobierania danych (**select**) zostanie omówiona w rozdziale 3.

Uwaga: polecenie **insert** wstawia zawsze nowy rekord (albo nowe rekordy). Nie da się za jego pomocą zmienić wartości już istniejących rekordów.

2.2. Wstawianie danych pochodzących z innych tabel

W tym przypadku dane pochodzą z zapytania **select**:

```
insert into tabela1 select * from tabela2;
```

Wariant z wyborem pól

```
insert into tabela1(pole1, pole2, ..., poleN)
select wartość1, wartość2, ..., wartośćN from tabela2;
```

Tabel docelowa musi istnieć, a liczba pól, typy danych oraz więzy muszą być zgodna z jej definicją.

2.3. Usuwanie danych z tabeli

Usunięcie z tabeli wszystkich rekordów spełniających zadany *warunek*:

```
delete from tabela where warunek;
```

Przykład:

```
delete from kolory where (kod='v');
```

Uwaga: polecenie **delete** usuwa z tabeli wszystkie rekordy spełniające podany *warunek*. W szczególności, jeżeli nie podano warunku, usunięte zostaną wszystkie rekordy.

2.4. Modyfikacja danych w tabeli

```
update tabela set
    nazwa_pola = wartość,
    nazwa_pola = wartość,
    ...
    nazwa_pola = wartość
where warunek;
```

Przykład:

```
update kolory set nazwa='fioletowy' where (kod='v');

update kwiaty set barwa='v', name='cornflower'
    where nazwa='chaber';
```

Uwaga: polecenie **update** modyfikuje zawartość wszystkich rekordów spełniających podany *warunek*. W szczególności, jeśli warunku nie określono, modyfikacją zostaną objęte wszystkie rekordy.

3. Wybór i prezentacja danych

Użytkowanie baz danych polega głównie na uzyskiwaniu zestawień informacji. Tabele wchodzące w skład bazy, jak i wyniki odpowiedzi, mają podobną budowę i są obrazami pewnych relacji. Argumentami zapytań są zawsze relacje. Nie jest istotne, czy pochodzą one z tabel przechowywanych w bazie, czy są jedynie etapem pośrednim uzyskiwania odpowiedzi. Ich źródłem mogą też być *perspektywy*, czyli udostępniane przez bazę relacje o zawartości odpowiadającej pewnemu zapytaniu. W opisach i przykładach z bieżącego rozdziału *relacja* może być nazwą tabeli, nazwą perspektywy, opatrzonym nazwą wynikiem zapytania podrzędnego albo wynikiem działań algebry relacji omawianych w rozdziale 4.

Do pobierania danych z bazy służy polecenie **select**.

3.1. Obliczanie wartości wyrażeń

Najprostszy przypadek użycia polecenia **select** polega na traktowaniu go jako kalkulatora obliczającego wartość wyrażeń niezwiązanych z żadną tabelą. Poniżej kilka przykładów ilustrujących tę możliwość.

Przykład:

```
select 10;
```

10

Przykład:

```
select 1+2+3;
```

6

Przykład:

```
select current_date;
```

2007-04-12

Przykład:

```
select 'Jan' || '␣' || 'Kowalski';
```

Jan Kowalski

W każdym z tych przypadków zwracany wynik ma postać jednokolumnowej relacji składającej się z pojedynczego rekordu.

3.2. Praca z pojedynczą relacją

Prezentacja całej relacji

```
select * from relacja;
```

Przykład:

```
select * from kolory;
```

W wyniku tego polecenia otrzymamy odpowiedź:

kod	nazwa	name
r	czerwony	red
g	zielony	green
y	żółty	yellow
b	niebieski	blue

Zastosowanie operatora rzutowania (patrz rozdział 4.1.)

```
select
    nazwa_pola,
    nazwa_pola,
    ...
    nazwa_pola
from relacja;
```

Przykład:

```
select name, nazwa from kolory;
```

W wyniku tego polecenia otrzymamy odpowiedź:

name	nazwa
red	czerwony
green	zielony
yellow	żółty
blue	niebieski

Zmianę nazw pól (kolumn) odpowiedzi wymuszamy używając słowa kluczowego **as**. Nową nazwę piszemy w podwójnych cudzysłowach technicznych.

```
select
    nazwa_pola as nazwa_kolumny,
    nazwa_pola as nazwa_kolumny,
    ...
    nazwa_pola as nazwa_kolumny
from relacja;
```

Przykład:

```
select
    name as "Nazwa angielska",
    nazwa as "Nazwa polska"
from kolory;
```

W wyniku tego polecenia otrzymamy odpowiedź:

Nazwa angielska	Nazwa polska
red	czerwony
green	zielony
yellow	żółty
blue	niebieski

Zastosowanie operatora wyboru (patrz rozdział 4.2.)

```
select
     nazwa_pola,
     nazwa_pola,
     ...
     nazwa_pola
from relacja
where warunek;
```

Przykład:

```
select nazwa as "kolor" from kolory where (kod <> 'y');
```

W wyniku tego polecenia otrzymamy odpowiedź:

kolor
czerwony
zielony
niebieski

Słowo kluczowe **`distinct`** użyte w odniesieniu do pola lub zespołu pól nakazuje pominąć powtórzenia wartości.

Przykład: I tak, żądanie

```
select distinct barwa from kwiaty;
```

doprowadzi do uzyskania odpowiedzi (z dokładnością do kolejności rekordów)

barwa
b
r
y

podczas gdy bez słowa **`distinct`** wartość `r` występowałaby dwukrotnie:

barwa
b
r
y
r

3.3. Pola relacji wynikowej

Przy wydawaniu zapytania **`select`** określamy listę pól odpowiedzi. Bardzo często jest ona po prostu wyliczeniem niektórych pól odpytywanej relacji. Jednak nie musi tak być zawsze. Pola odpowiedzi mogą być konstruowane za pomocą dowolnych wyrażeń angażujących pola odpytywanej relacji. Na przykład jest możliwe: obliczenie wieku osoby na podstawie różnicy daty bieżącej i daty urodzenia; otrzymanie długości w calach mimo, że baza przechowuje wartości metryczne; połączenie kilku pól tekstowych w jedno pole, itp.

W sytuacji, kiedy nie zależy nam na zawartości konkretnego rekordu, a tylko na stwierdzeniu jego istnienia (tak jest np. w przypadku zliczania), wystarczy spytać o wartość stałą, np. **`select`** `1` **`from`** `kolory` zwróci kolumnę jedynek, zaś **`select`** `count(1)` **`from`** `kolory`, podobnie jak **`select`** `count(*)` **`from`** `kolory` — liczbę rekordów tabeli `kolory`.

Możliwe jest też deklarowanie pól relacji wynikowej za pomocą osobnych zapytań. Technika ta będzie zaprezentowana w rozdziale 3.9.

3.4. Porządkowanie kolejności odpowiedzi

Możemy zażądać, by wyniki zapytania zostały uporządkowane według wartości podanego *`wyrażenia`*. Domyślnie porządkowanie przebiega od najmniejszej do największej wartości wyrażenia porządkującego.

```
select * from relacja order by wyrażenie;
```

Przykład:

```
select * from kolory order by kod;
```

W wyniku tego polecenia otrzymamy odpowiedź:

kod	nazwa	name
b	niebieski	blue
g	zielony	green
r	czerwony	red
y	żółty	yellow

Możliwe jest także uporządkowanie odpowiedzi zgodnie z wartościami *`wyrażenia`* od największej do najmniejszej:

```
select * from relacja order by wyrażenie desc;
```

Przykład:

```
select * from kolory order by nazwa desc;
```

W wyniku tego polecenia otrzymamy odpowiedź:

kod	nazwa	name
y	żółty	yellow
g	zielony	green
b	niebieski	blue
r	czerwony	red

3.5. Ograniczenie liczby rekordów odpowiedzi

Możemy zażądać, by wynik zapytania obejmował jedynie wskazaną liczbę rekordów.

```
select * from relacja order by wyrażenie limit liczba;
```

Ma to sens wtedy, kiedy żądamy uporządkowania relacji wynikowej. W przeciwnym razie zawartość odpowiedzi będzie zależeć od nie tylko sformułowania zapytania, ale także od sposobu jego realizacji.

Po określeniu **limit** 1 otrzymamy informacje o pojedynczym rekordzie — tym, dla którego *wyrażenie* porządkujące ma największą (względnie: najmniejszą) wartość.

Przykład:

```
select * from kolory order by kod limit 2;
```

W wyniku tego polecenia otrzymamy odpowiedź:

kod	nazwa	name
b	niebieski	blue
g	zielony	green

Jest to początkowa część odpowiedzi na pytanie opisane w pierwszym przykładzie z paragrafu 3.4.

3.6. Wybór i prezentacja danych z kilku relacji

Dla uzyskania informacji z wielu relacji najpierw trzeba zbudować relację, której postać pozwalałaby na odczytanie odpowiedzi z jej rekordów, a następnie zastosować do niej rzutowanie i wybór. Złożoną relację możemy zbudować za pomocą jednego z operatorów łączenia (patrz rozdział 4.6.): wewnętrznego

```
select
    nazwa_pola,
    nazwa_pola,
    ...
    nazwa_pola
from relacja_1 join relacja_2 on kryterium_łączenia
where warunek_wyboru;
```

lub zewnętrznego (np. lewostronnego):

```
select
    nazwa_pola,
    nazwa_pola,
    ...
    nazwa_pola
from
    relacja_1 left join relacja_2 on kryterium_łączenia
where warunek_wyboru;
```

Zamiast słowa kluczowego **join** możemy używać równoważnego, lecz dłuższego zwrotu **inner join**.

Często używaną regułą łączenia jest wiązanie rekordów na podstawie równości pól o tej samej nazwie (np. będących kluczami). W takim przypadku zamiast stwustronnego operatora **join** możemy użyć klauzuli **using**:

```
select
     nazwa_pola,
     nazwa_pola,
     ...
     nazwa_pola
from
     relacja_1 join relacja_2 using(nazwa_pola);
```

Przykład: Sensowne jest pytanie o nazwy barw kwiatów. Zapytanie

```
select
     kwiaty.nazwa as "kwiat",
     kolory.nazwa as "barwa"
from
     kwiaty join kolory
     on kwiaty.barwa=kolory.kod;
```

da nam następującą odpowiedź:

kwiat	barwa
chaber	niebieski
róża	czerwony
słonecznik	żółty
mak	czerwony

Przykład: Podobne zapytanie z wykorzystaniem operacji łączenia zewnętrznego

```
select
     kwiaty.nazwa as "kwiat",
     kolory.nazwa as "barwa"
from
     kwiaty full join kolory
     on kwiaty.barwa=kolory.kod;
```

da nieco inną odpowiedź:

kwiat	barwa
chaber	niebieski
róża	czerwony
słonecznik	żółty
mak	czerwony
`null`	zielony

gdyż tabela `kolory` zawiera rekord `('g', 'zielony', 'green')`, do którego nie ma odwołań w kolumnie `barwa` tabeli `kwiaty`.

Nic nie stoi na przeszkodzie, by do zapytań dotyczących połączeń stosować kryteria wyboru (czyli warunek po słowie kluczowym **`where`**).

Większą liczbę relacji łączy się przez zagnieżdżanie operatorów **`join`**, np.:

```
...
from
     (relacja_1
     left join relacja_2 on kryterium_łączenia_1)
     join relacja_3 on kryterium_łączenia_2
where ...
```

przy czym nawiasy są wymagane tylko w niektórych systemach, zaś standard pozwala je pomijać.

3.7. Grupowanie odpowiedzi

Kolejność rekordów w odpowiedzi na zapytanie wymusza się za pomocą kryterium porządkowania. Osobne kryterium służy do grupowania: grupę tworzą rekordy, dla których wyrażenie grupujące przyjmuje tę samą wartość. Żądanie grupowania powoduje, że każdą grupę opisze pojedynczy rekord.

3.7.1. Funkcje agregujące

Są to funkcje operujące na kolumnach danych, obliczające pojedynczą wartość: **avg**, **count**, **max**, **min** oraz **sum**.

Jeżeli w zapytaniu zawierającym taką funkcję nie zastosowano grupowania, to wynik ma postać pojedynczego rekordu dla całej tabeli.

Jeśli zastosowano grupowanie, to wynik zawiera rekordy odpowiadające każdej znalezionej wartości wyrażenia grupującego.

Zapytanie zawierające jednocześnie wartości zagregowane i niezagregowane jest nielegalne. Wyjątkiem jest zapytanie z żądaniem grupowania, zawierające kolumny zagregowane i kolumny obliczone na podstawie wyrażeń grupujących.

3.7.2. Kryteria grupowania

```
select * from relacja group by wyrażenie;
```

Przykład: Zapytanie

```
select * from kwiaty group by barwa;
```

nie ma sensu, bowiem żądane pola, oprócz pola `barwa`, w żaden sposób nie odnoszą się do grupy jako całości. Dobry system bazodanowy powinien odmówić realizacji takiego zapytania.

Przykład: Sensowniej jest zapytać o liczbę rekordów w poszczególnych grupach opisujących kwiaty tej samej barwy. Na zapytanie

```
select barwa "kolor", count(distinct nazwa) as "liczba"
from kwiaty
group by barwa;
```

otrzymamy odpowiedź:

kolor	liczba
b	1
r	2
y	1

Przykład: Jeszcze lepiej skierować to samo zapytanie to złączenia relacji: z zapytania

```
select
    kolory.nazwa as "kolor",
    count(distinct kwiaty.nazwa) as "liczba"
from kwiaty join kolory on (barwa=kod)
group by barwa, kolory.nazwa;
```

otrzymamy odpowiedź:

kolor	liczba
niebieski	1
czerwony	2
żółty	1

Uwagę zwraca grupowanie ze względu na wartość pola `nazwa` tabeli `kolory`, mimo że zdrowy rozsądek nie sugeruje takiej konieczności. Jednak w przeciwnym razie, tj. przy grupowaniu tylko ze względu na `barwa`, pierwsze pole odpowiedzi byłoby formalnie niepoprawne.

W powyższych przykładach do zliczania rekordów w obrębie grup zastosowaliśmy funkcję agregującą **`count`**. Jest to przykład *agregacji*, czyli obliczenia pewnej charakterystyki grupy na podstawie analizy należących do niej rekordów.

3.7.3. Kryteria wyboru grup

Dodatkowa klauzula **`having`** ogranicza zakres odpowiedzi do tych grup, które spełniają warunek zadany w poniższym wzorcu przez *`wyrażenie_2`*:

```
select * from relacja
group by wyrażenie_1
having wyrażenie_2;
```

Konstrukcja **`having`** działa w odniesieniu do grup podobnie, jak **`where`** w odniesieniu do pojedynczych rekordów.

Przykład: Rozbudujemy zapytanie z poprzedniego przykładu, dodając żądanie, by odpowiedź odnosiła się tylko do obiektów barwy czerwonej.

```
select
    kolory.nazwa as "kolor",
    count(distinct kwiaty.nazwa) as "liczba"
from kwiaty join kolory on (barwa=kod)
group by barwa, kolory.nazwa
having (kolory.nazwa='czerwony');
```

Otrzymamy następującą odpowiedź:

kolor	liczba
czerwony	2

3.8. Kolejność klauzul w zapytaniu

Jak wiemy z poprzednich podrozdziałów, zapytanie `select ... from` może zawierać dodatkowe klauzule określające charakter i zawartość odpowiedzi. Każdej z nich jest przypisane odpowiednie słowo kluczowe języka zapytań: `where`, `group by`, `having`, `order by`, `limit`. Kolejność dołączania tych klauzul do zapytania nie jest dowolna. Musi ona odpowiadać logice, jaką posługuje się system bazodanowy podczas realizacji żądania, a zatem musi być zgodna z następującym ogólnym wzorcem:

```
select lista_pól
from relacja
where warunek_wyboru_rekordów
group by wyrażenie_grupujące
having wyrażenie_wyboru_grup
order by wyrażenie_porządkujące
limit liczba_rekordów_odpowiedzi;
```

W pierwszej kolejności (`select`) ustalana jest postać relacji wynikowej, tj. jej lista kolumn. Dalej (`from`) ustalana jest relacja źródłowa, na podstawie której zostanie zbudowana odpowiedź (być może relację tę trzeba zbudować za pomocą odpowiednich operacji). Mając już tę relację, system przystąpi do selekcjonowania jej rekordów (`where`). Dysponując wynikami, w razie potrzeby połączy je w grupy (`group by`) i wyselekcjonuje grupy interesujące pytającego (`having`). Na koniec pozostaje posortowanie rekordów wynikowych (`order by`) i „ucięcie" ich ustalonej liczby (`limit`), o ile została ona określona w żądaniu.

Zaburzenie kolejności klauzul jest błędem składniowym języka i będzie w ten sposób raportowane przez system zarządzający bazą.

3.9. Zagnieżdżanie zapytań

Tak jak powiedziano we wstępie do rozdziału 3., relacja będąca argumentem zapytania może równie dobrze pochodzić wprost z pewnej tabeli, jak być wynikiem innego zapytania. Taki charakter mają omówione w podrozdziale 3.6. przykłady wybierania danych z wyniku złączenia dwóch lub więcej tabel.

Także wyniki polecenia **`select`** mogą służyć jako argument innego żądania lub zapytania, co zilustrują następne przykłady.

Przykład: Celem bieżącego zapytania jest uzyskanie spisu barw kwiatów umieszczonych w tabeli `kwiaty`:

```
select nazwa as "kolor"
from kolory
where kod in (select distinct barwa from kwiaty);
```

Zagnieżdżone zapytanie, zwane też *podzapytaniem*, powinno być umieszczone w nawiasie. Czasami nawias wolno pominąć.

Oto oczekiwana odpowiedź:

kolor
niebieski
czerwony
żółty

Do identycznego efektu doprowadziłoby pewne rzutowanie odpowiedniego złączenia obu tabel.

Przykład: Poniższe zlecenie nakazuje uzupełnić w tabeli `kolory` kody barw użytych w tabeli `kwiaty`:

```
insert into kolory (kod)
    (select distinct barwa from kwiaty
     where barwa not in (select kod from kolory));
```

Takie „wielopoziomowe" zapytania nie są niczym nadzwyczajnym i chociaż na zajęciach nie będziemy ich nadużywać, w realnych zastosowaniach bywają przydatne.

3.10. Wybór i prezentacja danych z relacji niebędącej tabelą

W dotychczasowych przykładach relacje będące przedmiotem zapytania były tabelami przechowywanymi w bazie albo wynikami operacji łączenia tabel. Nie zawsze musi tak być. „Relacja" musi być... właśnie relacją, czyli kolekcją rekordów o takim samym układzie pól. Oprócz tabel i wyników łączenia tabel, relacjami są także odpowiedzi na zapytania zadawane przez użytkownika, oraz zawczasu przygotowane przez projektantów systemu odpowiedzi na wybrane zapytania, zwane *perspektywami*.

Jeżeli zapytanie skierowane jest nie do tabeli, tylko do wyników innego zapytania podanego jako argument w klauzuli **from**, zapytanie to musi zostać ujęte w nawiasy i opatrzone nazwą tymczasową (za pomocą słowa kluczowego **as**), jak w poniższym przykładzie: **Przykład:**

```
select count(1) as "liczba",  kolor
from (select ...) as "robocza"
group by kolor;
```

Nasza przykładowa baza jest zbyt prosta, by dało się w niej sformułować sensowne zapytania tego typu. Jednak w rzeczywistych sytuacjach są one przydatne, szczególnie w razie konieczności grupowania ze względu na wyniki wymagające uprzedniego grupowania.

4. Operacje na relacjach

Zrozumienie operacji na relacjach jest konieczne dla swobodnego operowania bazami danych. Omawiamy następujące operacje:

— określone dla pojedynczej relacji: rzutowanie, wybór;
— określone dla par relacji o takiej samej postaci rekordów: suma, różnica, iloczyn;
— określone dla par dowolnych relacji: łączenie wewnętrzne, łączenie zewnętrzne (lewostronne, prawostronne i pełne).

Operacje relacyjne mają charakter algebraiczny i są blisko związane z algebrą zbiorów. Nieprzypadkowo: relacje są zbiorami, a operacje relacyjne nie są niczym innym, jak operacjami algebry zbiorów zastosowanymi do relacji.

Zastosowanie praktyczne operacji relacyjnych w systemach baz danych winno być poddane kryterium sensowności: nie wszystko, co da się zrobić, ma sens z punktu widzenia celowych działań na danych posiadających znaczenie poza sferą czysto formalną.

4.1. Operator rzutowania

Argumentem operatora rzutowania jest pojedyncza relacja. Wynikiem rzutowania jest nowa relacja, której polami są jedynie wybrane pola relacji pierwotnej lub pola obliczone na ich podstawie. Rzutowanie określa się podając listę pól w zapytaniu **select**.

4.2. Operator wyboru

Argumentami operatora wyboru są: pojedyncza relacja oraz kryterium wyboru opisane za pomocą formuły logicznej. Operator wyboru pomija te rekordy relacji, dla których zadany warunek nie jest spełniony (pamiętajmy o trójwartościowym rachunku logicznym). Warunek wyboru ustalany jest za pomocą frazy **where** *warunek* w zapytaniu **select**.

4.3. Suma relacji

Argumentem sumy jest para relacji. Dwie relacje z tej samej przestrzeni (o takiej samej postaci rekordów) można dodać do siebie. Wynikiem jest relacja, która zawiera wszystkie rekordy znajdujące się w przynajmniej jednej spośród relacji składowych.

```
relacja_1 union relacja_2
```

Operatora sumy używa się najczęściej do połączenia odpowiedzi uzyskanych z różnych źródeł.

4.4. Różnica relacji

Argumentem różnicy jest para relacji. Dwie relacje z tej samej przestrzeni (o takiej samej postaci rekordów) można odjąć od siebie. Wynikiem jest relacja, która zawiera wszystkie rekordy z pierwszej relacji, których nie ma w relacji drugiej.

```
relacja_1 except relacja_2
```

Operatora różnicy używa się najczęściej do wyodrębnienia interesującego nas zbioru z odpowiedzi uzyskanych z kilku różnych źródeł.

4.5. Iloczyn relacji

Argumentem iloczynu jest para relacji. Dla dwóch relacji z tej samej przestrzeni (o takiej samej postaci rekordów) można wyznaczyć ich iloczyn (część wspólną). Wynikiem jest relacja, która zawiera wszystkie rekordy znajdujące się jednocześnie w obu relacjach składowych.

```
relacja_1 intersect relacja_2
```

Operatora iloczynu używa się najczęściej do znalezienia części wspólnej odpowiedzi uzyskanych z różnych źródeł.

Jeżeli dane dwie relacje mają różne nagłówki tabel, to nie da się wyznaczyć ich sumy, różnicy ani iloczynu.

4.6. Łączenie relacji

Dowolne dwie relacje można połączyć posługując się kryterium dopasowania rekordów. Kryterium jest zadawane za pomocą formuły logicznej, której argumentami są nazwy pól relacji.

Relacja reprezentująca wynik złączenia zawiera wszystkie kolumny relacji składowych.

Rekordami relacji wynikowej te spośród wszystkich możliwych kombinacji rekordów relacji składowych, które spełniają kryterium połączenia.

Najczęściej mamy do czynienia z łączeniem przez dopasowanie wartości klucza obcego z jednej tabeli do odpowiadającej jej wartości klucza własnego w innej tabeli. Nie jest to jedyna możliwość. Inne warunki łączenia będą związane np. z kryteriami doboru obiektów w pary.

4.6.1. Łączenie wewnętrzne

Wynikiem złączenia wewnętrznego *relacja_1* **join** *relacja_2* **on** *kryterium* jest relacja złożona z wszystkich możliwych połączeń wierszy relacji składowych, dla których zdanie logiczne *kryterium* jest prawdziwe. Na ogół *kryterium* dotyczy równości wybranych pól w rekordach pierwszej i drugiej relacji.

4.6.2. Łączenie zewnętrzne

Istnieją trzy operatory łączenia zewnętrznego: **left join**, **right join**, **full join**.

Wynikiem złączenia zewnętrznego *relacja_1* **left join** *relacja_2* **on** *kryterium* jest relacja złożona z wszystkich możliwych połączeń wierszy relacji składowych, dla których zdanie logiczne *kryterium* jest prawdziwe. Dodatkowo relacja wynikowa zawiera rekordy odpowiadające jednej (pierwszej w przypadku operacji **left join**, zaś drugiej w przypadku operacji **right join**) lub obu (w przypadku operacji **full join**) relacjom składowym, do których zgodnie z kryterium połączenia nie udało się dopasować żadnego wiersza. Pozostałe pola złączenia dodawane do takich wierszy mają wartość `null`.

5. Składnia wyrażeń

Ten rozdział zawiera informacje dodatkowe, niezbędne do prawidłowego konstruowania kryteriów i warunków.

5.1. Typy danych i wyrażeń

Do najważniejszych typów danych należą:

— typ logiczny `boolean`. Wartości stałych logicznych `true` i `false` piszemy zawsze bez cudzysłowów; w przeciwnym razie będą traktowane jak napisy;
— typy liczbowe:
  — całkowity `integer`,
  — zmiennoprzecinkowy `float`,
  — stałoprzecinkowy `numeric`.
  Wartości stałych liczbowych piszemy bez cudzysłowów w postaci dziesiętnej lub półlogarytmicznej, np. `122`, `-1.23`, `6.81E+12`;
— typy tekstowe (napisowe):
  — do przechowywania ciągów znaków o stałej długości `char(`*n*`)`,
  — z ograniczeniem długości `varchar(`*n*`)`,
  — dla ciągów znaków dowolnej długości `text`.
  Wartości stałych tekstowych ujmujemy w pojedyncze apostrofy: `'wartość'`;
— typ datowy `date`. Wartości dat ujęte w cudzysłowach apostrofowych podajemy w formacie zgodnym z bieżącymi ustawieniami systemu, np. `'2004-04-22'`;
— typ danych dotyczących czasu `time`. Wartości ujęte w cudzysłowach apostrofowych podajemy w formacie zgodnym z bieżącymi ustawieniami systemu, np. `'12:05:0'`.

Istnieje wiele innych typów danych.

Wartość `null` można przypisywać polom rekordów niezależnie od ich typu. Oznacza ona, że pole pozostaje „puste”.

5.2. Wartości logiczne

W przeciwieństwie do tradycyjnej logiki posługującej się wyłącznie wartościami „prawda” i „fałsz”, systemy baz danych korzystają z trójwartościowego rachunku logicznego. Każde wyrażenie albo jest prawdziwe, albo jest fałszywe, albo ma nieokreśloną wartość logiczną. Ten trzeci przypadek wiąże się z operowaniem danymi o wartości `null` (`null` nie jest wartością logiczną, nie ma też ona żadnego innego typu).

Stała `true` oznacza prawdę; stała `false` oznacza nieprawdę; wartość `null` oznacza niemożność określenia wartości logicznej wyrażenia. Wyrażenie `true` jest zawsze spełnione; wyrażenie `false` nigdy nie jest spełnione; wyrażenie o wartości nieokreślonej, reprezentowane przez obiekt `null`, nie jest ani spełnione, ani niespełnione.

Trójwartościowy rachunek zdań stosuje się do wszystkich wyrażeń logicznych używanych w zapytaniach, w tym do kryteriów łączenia i wyboru.

5.3. Operatory logiczne

Argumentami operatorów logicznych są poprawnie sformułowane wyrażenia logiczne.

Operator koniunkcji:

```
warunek_1 and warunek_2
```

Operator alternatywy:

```
warunek_1 or warunek_2
```

Operator negacji:

```
not warunek
```

5.4. Operatory zwracające wartości logiczne

Operatory przyrównywania:

```
wartość_1 = wartość_2
```

oraz

```
wartość_1 <> wartość_2
```

Przyrównywane wartości muszą być tego samego typu.

Uwaga: `null` nie należy do zakresu wartości żadnego typu danych. Dlatego w pytaniach o wartości brakujące nie można używać operatorów przyrównywania — zwracają one wartość `null` w przypadku, kiedy choćby jeden z argumentów przyjmuje tę wartość. Do sprawdzania, czy pole jest puste, używa się specjalnego operatora **`is`** w postaci: **`not`** `(`*wartość* **`is`** `null)` lub `(`*wartość* **`is not`** `null)`.

Istnieją również cztery operatory porównywania:

```
wartość_1 < wartość_2
wartość_1 <= wartość_2
wartość_1 >= wartość_2
wartość_1 < wartość_2
```

o znaczeniu zgodnym z tradycyjną notacją matematyczną.

Porównywane wartości muszą być tego samego typu. Porównanie dotyczy porządku w zbiorze wartości tego właśnie typu, np. liczb, wartości logicznych, napisów czy też dat.

Uwaga: napis będący ciągiem cyfr nie jest liczbą! Przy porównywaniu napisów porównujemy ich znaki począwszy od pierwszego; przy porządkowaniu liczb porównujemy ich cyfry odpowiadających sobie rzędów wielkości, począwszy od największego. Uporządkowanie napisów składających się z cyfr daje zatem inny wynik (np. `'1', '10', '100', '11', '2'`), niż uporządkowanie odpowiadających im liczb (odpowiednio `1, 2, 10, 11, 100`).

Operator **`in`** służy do wyrażenia przynależności wartości do zbioru:

```
wartość in zbiór_wartoci
```

Często w roli zbioru występuje ciąg jawnie wyliczonych wartości albo jednokolumnowa odpowiedź na zapytanie.

```
wartość in (wartość_1, wartość_2, ..., wartość_n)
wartość in (select pole from relacja)
```

Operator podobieństwa **`like`** działa na danych typu tekstowego. Służy on do sprawdzania, czy testowany napis jest zbudowany zgodnie z podanym wzorcem:

```
wartość like wzorzec
```

W opisie wzorca można stosować następujące: znak „%” będzie dopasowany dowolnego ciągu znaków; znak „_” zostanie dopasowany do dowolnego pojedynczego znaku.

Przykład: W wyniku zapytania

```
select * from kolory
where (nazwa like '%y');
```

otrzymamy następującą odpowiedź:

kod	nazwa	name
r	czerwony	red
g	zielony	green
y	żółty	yellow

5.5. Funkcje działające na danych typu datowego

Do wyłuskiwania elementów daty z wyrażeń typu datowego służy funkcja **extract**. Używa się jej następująco:

```
extract (element_daty from wyrażenie)
```

Przykład:

```
select
    imie, nazwisko, extract(day from dataurodzenia)
    from pracownicy;

select
    imie, nazwisko, extract(month from dataurodzenia)
    from pracownicy;

select
    imie, nazwisko, extract(year from dataurodzenia)
    from pracownicy;
```

Poszczególne systemy bazodanowe mogą mieć własne niestandardowe funkcje przeznaczone do tego samego celu.

Do obliczania odstępu czasu między datami służy operator odejmowania, oznaczany symbolem −.

Przykład: Wartością wyrażenia

```
date('2006-01-13') - date('2004-05-16')
```

jest 607, czyli czas trwania od 16 maja 2004 do 13 stycznia 2006, wyrażony w dniach.