Systemy Baz Danych

Wykład II

Język SQL – polecenia DQL

Powtórzenie wiadomości – cz. 1

Materiał wykładu

Wykład zawiera przegląd podstawowych wiadomości o języku Structured Query Language (SQL), w zakresie poleceń należących do Data Query Language. Stanowi powtórzenie materiału wykładanego w ramach przedmiotu Relacyjne Bazy Danych dla studentów studiów inżynierskich Wydziału Informatyki PJWSTK.

Wykład jest przeznaczony dla studentów przedmiotu **Systemy Baz Danych** prowadzonego dla studiów inżynierskich Wydziału Informatyki PJWSTK.

Język SQL – geneza i struktura

Język SQL (*Structured Query Language*) powstał jako realizacja *postulatu pełnego języka* danych – jednego z postulatów Codd'a. Został opracowany w latach 70 XX wieku w firmie **IBM** i stał się standardem w komunikacji z serwerami relacyjnych baz danych. W 1996 roku stał się oficjalnym standardem ISO.

Język SQL jest językiem deklaratywnym – o sposobie przechowywania i operowania danymi decyduje SZBD.

Struktura języka:

SQL DML (ang. *Data Manipulation Language* – „język manipulacji danymi"),

SQL DDL (ang. *Data Definition Language* – „język definicji danych"),

SQL DCL (ang. *Data Control Language* – „język kontroli nad danymi").

SQL DQL (ang. *Data Query Language* – „język definiowania zapytań").

Powyższe tłumaczenia angielskich zwrotów (za Wikipedią) nie do końca precyzyjnie oddają ich istotę.

Język SQL – struktura

Użycie języka SQL polega na wydawaniu poleceń (instrukcji, zapytań) realizowanych przez SZBD. Standard języka gwarantuje dowolność użycia małych / wielkich liter w odniesieniu do słów kluczowych i nazw obiektów (tabel, kolumn, widoków etc.). Natomiast rozróżnianie wielkości liter (case sensitive) w obszarze danych zależy od SZBD i ustawień bazy danych.

DQL – Data Query Language

Zestaw poleceń odczytania danych z bazy – wszystkie polecenia rozpoczynają się od słowa kluczowego **SELECT**, a ich realizacja nie ma wpływu na stan danych.

DML – Data Manipulation Language

Zestaw poleceń odpowiedzialnych za operowanie danymi.

- **INSERT** – wpisywanie nowych danych (rekordów) do bazy danych
- **UPDATE** – aktualizacja (zmiana) danych już istniejących
- **DELETE** – usuwanie danych (rekordów) z bazy

Język SQL – struktura

DDL – Data Definition Language

Zestaw poleceń odpowiedzialnych za operacje na obiektach bazy danych

- **CREATE** – polecenie utworzenia nowego obiektu bazy danych
- **ALTER** – zmiana struktury obiektu już istniejącego
- **DROP** – polecenie usunięcia z bazy istniejącego obiektu

DCL – Data ControlLanguage

Zestaw poleceń odpowiedzialnych za nadawanie uprawnień do operacji na bazie danych

- **GRANT** – przyznawanie uprawnień do operacji na obiektach bazy danych
- **REVOKE** – odebranie uprawnień do operacji na obiektach bazy danych
- **DENY** – zabrania operacji na obiektach bazy danych

Język SQL – typy danych

Typy danych przewidziane przez standard języka:

Typy napisowe:

`Charakter(n)` – napis o stałej długości n znaków

`Varying Charakter(n)` – napis o zmiennej długości max. n znaków

`Bit(n)` – napis o określonej długości n bitów

`Bit Varying(n)` – napis o długości do n bitów

Typy liczb całkowitych:

`Integer` – liczba całkowita, dziesiętna lub binarna, ze znakiem

`Smallint` – liczba całkowita, dziesiętna lub binarna, ze znakiem

Język SQL – typy danych

Typy liczb zmiennoprzecinkowych:

`Numeric(p, q)` – liczba dziesiętna ze znakiem, złożona z *p* cyfr, z kropką dziesiętną po *q* cyfrach (licząc od prawej)

`Decimal(p,q)` – liczba dziesiętna ze znakiem, złożona z *p* cyfr przed i *q* cyfr po przecinku

`Float(p)` – liczba zmiennopozycyjna zapisywana w postaci wykładniczej

Typy daty i czasu:

`Date`

`Time`

`Timestamp`

`Interval` – specyfikacja przedziału czasowego

MS SQL Server – podstawowe typy danych

Char(n)

Varchar(n)

Bit

Int

Numeric

Decimal

Money – liczba zmiennoprzecinkowa formatowana na typ walutowy

Date (od v. 2008)

Time (od v. 2008)

Datetime

Binary(n) – dane w postaci binarnej zapisane na dokładnie *n* bajtach

Varbinary(n) - dane w postaci binarnej zapisane na max. *n* bajtach

ORACLE – podstawowe typy danych

Char(n)

Varchar2(n)

Clob – do 8 terabajtów znaków

Int

Number

Number(p, s)

Blob – do 128 terabajtów danych w postaci binarnej

Date

Timestamp

Jak widać, implementacje realizują założenia przyjęte w opisie standardu języka, traktując wymogi standardu rozszerzająco.

Instrukcja *SELECT*

Instrukcja **SELECT** składa się z kilku tzw. klauzul, występujących w ściśle określonej kolejności, z których każda zaczyna się od słowa kluczowego. Pierwsza klauzula, rozpoczynająca się od słowa kluczowego **SELECT** oraz druga rozpoczynająca się od **FROM**, są obligatoryjne i muszą wystąpić w składni całego polecenia przynajmniej jeden raz (mogą pojawiać się wielokrotnie).

Całość instrukcji **SELECT** definiuje:

- z jakich źródeł mają zostać odczytane dane,
- jakie warunki muszą spełniać dane, aby pojawiły się w wyniku wykonania instrukcji,
- w jakiej postaci wynik instrukcji ma zostać zwrócony do użytkownika (aplikacji).

Standard języka wymaga zakończenia instrukcji **SELECT** (oraz każdej innej) średnikiem. **MS SQL Server** traktuje ten wymóg opcjonalnie.

Instrukcja *SELECT*

Struktura instrukcji **SELECT**:

```
SELECT [DISTINCT] wyrażenie, ...
FROM nazwa_tabeli, ...
[WHERE warunek]
[GROUP BY wyrażenie, ...]
[HAVING warunek]
...
[ORDER BY wyrażenie, ...];
```

Każda z klauzul, poza **ORDER BY**, może w składni całej instrukcji pojawić się więcej niż jeden raz. Klauzule **SELECT** i **FROM** muszą pojawić się co najmniej jeden raz.

Instrukcja *SELECT* klauzula *SELECT*

Struktura instrukcji **SELECT**:

```
SELECT [DISTINCT] wyrażenie, ...
FROM              nazwa_tabeli, ...
                  ...;
```

Klauzula **SELECT** definiuje dane, które mają zostać odczytane z bazy, oraz sposób ich prezentacji. Mogą nimi być nazwy kolumn, wyrażenia (także odwołujące się do nazw kolumn) oraz stałe, niezwiązane z danymi zapisanymi w bazie. Wyrażenia oddzielane są przecinkami. Nazwy kolumn są poprzedzone (kwalifikowane) nazwą tabeli z której pochodzą:

Nazwa_tabeli.Nazwa_kolumny

Nazwa tabeli poprzedzająca nazwę kolumny może zostać pominięta, jeśli nie wprowadzi to niejednoznaczności.

Opcjonalna dyrektywa **DISTINCT** eliminuje z wyniku powtarzające się rekordy.

Instrukcja *SELECT* klauzula *FROM*

Struktura instrukcji **SELECT**:

```
SELECT [DISTINCT]wyrażenie, ...
FROM             nazwa_tabeli, ...
[WHERE           warunek]
                 ...;
```

Klauzula **FROM** definiuje źródła, z których dane będą odczytywane. Mogą nimi być tabele, widoki i inne instrukcje **SELECT**. Nazwy tabel, widoków i instrukcji **SELECT** oddzielane są przecinkami. Instrukcje w klauzuli **FROM** (nazywane tu podzapytaniami) muszą być ujęte w okrągłe nawiasy.

Instrukcja *SELECT* – elementarny przykład

Wypisać z tabeli EMP nazwiska, zarobki i stanowiska pracowników firmy:

```
SELECT  Ename, Sal, Job
FROM    Emp;
```

Wypisać wszystkie wartości wszystkich kolumn tabeli EMP:

```
SELECT  *
FROM    Emp;
```

lub

```
TABLE Emp;
```

(Składnia przewidziana przez standard języka, nie implementowana w **MS SQL Server**).

Wyrażenia w instrukcji *SELECT*

Wyrażenia w klauzuli **SELECT** mogą tworzyć struktury złożone, w których mogą zostać użyte:

- Stałe tekstowe (literały) i liczbowe, wyrażenia arytmetyczne,
- nazwy kolumn,
- operatory algebraiczne: dodawania (+), odejmowania (-), mnożenia (*), dzielenia (/) ,
- funkcje (wybrane funkcje zostaną omówione w dalszej części wykładu).
- nawiasy,
- operatory konkatenacji (łączenia tekstów) – w **MS SQL Server** jest to znak +, w **ORACLE** ||.

Wyróżnikiem tekstu w wyrażeniu jest apostrof (pojedynczy cudzysłów).

Wyrażenia w instrukcji *SELECT*

W przypadku konkatenacji wyrażeń różnych typów, **MS SQL Server** wymaga jawnej ich konwersji. **ORACLE** w przypadkach oczywistych stosuje konwersję automatyczną.

Funkcje konwertujące typy danych:

ORACLE i **MS SQL Server**:

```
Cast(Wyrażenie AS Typ_danych)
```

ORACLE

```
To_char(wyrażenie), To_date(wyrażenie)
```
i inne

MS SQL Server:

```
Convert(Typ_danych, Wyrażenie, [styl daty])
```

Np.

```
SELECT ename + ' ' + Cast(sal AS Varchar)
FROM   emp;
```

Instrukcja *SELECT* – aliasy nazw kolumn

Wyrażeniom na liście **SELECT** mogą zostać nadane nazwy czyli *aliasy:*

- prosty identyfikator - napis złożony z liter, cyfr i znaków podkreślenia
- ograniczony identyfikator - dowolny napis ograniczony podwójnymi cudzysłowami, np.

"Zarobki pracownika:"

W ograniczonym identyfikatorze mogą występować spacje, które są niedozwolone w prostym identyfikatorze.

Alias może zostać poprzedzony słowem **AS** przewidzianym przez standard języka, dopuszczalnym, ale nie wymaganym w większości implementacji.

W przypadku braku aliasu dla złożonego wyrażenia, **ORACLE** jako nazwę kolumny wynikowej zastosuje definicję wyrażenia, **MS SQL Server** pozostawi ją nienazwaną.

NULL w wyrażeniach

Jeżeli wartość wyrażenia jest **NULL**, wówczas na takim wyrażeniu nie można przeprowadzić żadnej operacji, gdyż wykonywanie jakiejkolwiek operacji (arytmetycznej, logicznej, konkatenacji) na wartości **NULL** daje w wyniku **NULL** W takim przypadku, aby można było wykonać operację na wyrażeniu, należy zamienić **NULL** na wartość znaczącą. W tym celu używa się funkcji:

- **ISNULL**(wyrażenie, zamienić_na) w **MS SQL Serwer**
- **NVL**(wyrażenie, zamienić_na) w **ORACLE**
- **NZ**(wyrażenie, zamienić_na) w **MS Access**

Funkcje te zwracają wartość pierwszego argumentu jeśli nie jest on NULL, lub drugi argument.

UWAGA: oba argumenty muszą być tych samych typów. W przypadku niezgodności **ORACLE** skonwertuje je niejawnie, w **MS SQL Serwer** trzeba użyć funkcji konwertujących.

Instrukcja *SELECT* klauzula *ORDER BY*

Wyniki zapytania mogą zostać posortowane w porządku rosnącym – **ASCENDING** (domyślnie) lub malejącym – **DESCENDING** (dopuszczalne skróty **ASC** i **DESC**).

```
SELECT [DISTINCT]   wyrażenie [[AS] alias],...
FROM                nazwa_tabeli
[WHERE              warunek]
ORDER BY            wyrażenie1 [ASC|DESC],...;
```

Klauzula **ORDER BY** może się pojawić w składni polecenia tylko jeden raz, zawsze jako ostatnia. Na jej liście mogą występować wyrażenia, w tym nazwy kolumn i ich aliasy a także numery wyrażeń w kolejności ich występowania na liście **SELECT**.

Sortowanie może odbywać się według kilku wyrażeń. Hierarchia sortowania wynika z kolejności umieszczenia wyrażeń na liście klauzuli.

Instrukcja *SELECT* klauzula *ORDER BY*

Przykład.

```
SELECT    ename
          ,job
          ,sal*12 + Isnull(comm,0) "Roczny dochód"
FROM      emp
ORDER BY 2, "Roczny dochód" DESC;
```

W powyższym przykładzie sortowanie wierszy wynikowych zostanie wykonane według nazwy stanowiska rosnąco, w obrębie stanowisk według dochodów rocznych malejąco.

Instrukcja *SELECT* klauzula *WHERE*

Klauzula **WHERE** pozwala zdefiniować warunek logiczny, ograniczający rekordy zwracane w wyniku działania instrukcji **SELECT** do tych tylko, dla których przyjmuje on wartość logiczną **TRUE**. Rekordy, dla których warunek przyjmuje wartość **FALSE** lub **NULL** są z wyniku eliminowane.

Warunek **WHERE** może być koniunkcją (**AND**), alternatywą (**OR**) bądź negacją (**NOT**) innych warunków logicznych. Hierarchia operatorów **NOT**, **AND**, **OR** może zostać zmieniona przy użyciu nawiasów.

Przykład.

Wypisz rekordy pracowników, których zarobki są większe lub równe 1100 i którzy pracują na stanowisku `'CLERK'`

```
SELECT Empno, Ename, Job, Sal
FROM   Emp
WHERE  Sal >= 1100 AND Job='CLERK';
```

Instrukcja *SELECT* klauzula *WHERE* - operatory

W konstrukcji warunku logicznego w klauzuli WHERE mogą zostać użyte operatory:

- arytmetyczne **+, -, *, /**
- operator konkatenacji (łączenia) napisów **||** (**Oracle**) lub **+** (**MS SQL Server),**
- operatory porównań **=**, **<>**, **<**, **<=**, **>**, **>=** Standard języka SQL dopuszcza użycie w roli argumentów porównań list wartości (list wyrażeń) - porównania odbywają się po odpowiednich składowych. Jest to zaimplementowane w **ORACLE**. **W MS SQL Server** argumenty operatorów porównań muszą być wyrażeniami,
- operator testujący Null `x` **IS [NOT] NULL**,
- operatory logiczne **NOT, AND, OR**,

Instrukcja *SELECT* klauzula *WHERE* - operatory

- Operator przynależności do listy wartości: `x [NOT] IN (x1,....)` np.

```
Kolor IN ('Czarny', 'Biały', 'Czerwony')
```

Lub

```
Grupa IN (201, 202, 203)
```

- Operator zawierania w przedziale: `x [NOT] BETWEEN z AND y` np.

```
Sal BETWEEN 1000 AND 2000
```

W Standardzie i w Oracle x,y,z mogą być listami wyrażeń tej samej długości – porównania odbywają się po odpowiadających sobie składowych. W MS SQL Server muszą być wyrażeniami.

```
…WHERE (deptno, job) IN ((10, ,'CLERK'), (20, 'MANAGER'));
```

Instrukcja *SELECT* klauzula *WHERE* - operatory

- Operator wzorca w tekście **x** [**NOT**] **LIKE** **y** gdzie **y** zawiera:
  - znak podkreślenia _ oznaczający dowolny jeden znak oraz ,
  - znak % oznaczający dowolny ciąg znaków.

  Np.

  ```
  ename LIKE 'Kowal%'
  ```

  Zwróci w wyniku nazwiska Kowal, Kowalski, Kowalska itd.

  ```
  Ename LIKE ,Kr_l'
  ```

  zwróci w wyniku nazwiska Krul i Król

  Operator **LIKE** w implementacjach działa na typach tekstowych, liczbowych i daty (!).

Instrukcja *SELECT* na wielu źródłach rekordów

Dotychczas rozważaliśmy odczytywanie danych z jednego źródła – tabeli. Jednak operacja ta może dotyczyć kilku źródeł, którymi mogą być tabele, widoki i inne instrukcje **SELECT** (podzapytania).

Rozważmy następujący przykład:

```
SELECT ename, dname, emp.deptno, dept.deptno
FROM   emp, dept;
```

W wyniku tego polecenia otrzymamy iloczyn kartezjański wartości odczytanych z tabel `emp` i `dept`, czyli z każdym nazwiskiem i numerem departamentu odczytanym z tabeli `emp`, zostanie połączony każda nazwa i numer departamentu odczytane z tabeli `dept`.

Instrukcja *SELECT* na wielu źródłach rekordów

Wynik uzyskany w powyższym przykładzie, jako całość jest oczywiście nieprawdziwy, natomiast zawiera w sobie wiersze prawdziwe. Są to te wiersze, w których `emp.deptno = dept.deptno`. Stąd wniosek, że dodanie do polecenia klauzuli

```
... WHERE emp.deptno = dept.deptno
```

pozostawi w wyniku wyłącznie wiersze spełniające wstępne założenie powiązania wartości w tabelach przez układ *klucz główny – klucz obcy*.

W przykładzie zastosowaliśmy powiązanie tabel `emp` i `dept` poprzez predykat złączenia umieszczony w klauzuli **`WHERE`**, w której pojawia się on obok predykatów ograniczających, takich jak np.

```
... sal >= 1000
```

Instrukcja *SELECT* na wielu źródłach rekordów

Istnieje możliwość innego sposobu zapisu warunku złączenia – przeniesienia go do klauzuli FROM

```
SELECT      ename, dname, emp.deptno, dept.deptno
FROM        emp
INNER JOIN dept
ON          emp.deptno, dept.deptno
WHERE       sal >= 1000;
```

Wynik w obu przypadkach będzie jednakowy, jednak rozdzielenie predykatów złączenia od predykatów ograniczających czyni cały kod bardziej czytelnym. Wszystkie serwery oba typy złączeń realizują w sposób identyczny.

Zarówno w przypadku klauzuli **WHERE** jak i **INNER JOIN** eliminowane są rekordy, dla których warunek przyjmuje wartość logiczną **FALSE** albo **NULL**.

Instrukcja *SELECT* na wielu źródłach rekordów

Złączenie tabel **INNER JOIN** nie musi być definiowane poprzez porównanie wartości klucz główny – klucz obcy. Jako porównanie może być użyta każda formuła która spowoduje zwrócenie oczekiwanego wyniku. Na przykład tabele `emp` i `salgrade` nie są połączone układem kluczy, lecz zależnością „biznesową” – grupa zarobkowa pracownika określona jest przez zawieranie się jego płacy w przedziale pomiędzy dolną i górną wartością określoną dla grupy.

```
SELECT      ename, grade, losal, sal, hisal
FROM        emp
INNER JOIN salgrade
ON          sal BETWEEN losal AND Hisal;
```

Instrukcja *SELECT* złączenie *INNER* i *OUTER JOIN*

Obok złączenia **INNER JOIN** istnieje drugi typ złączenia **OUTER JOIN**. Występuje w trzech wariantach:

RIGHT [OUTER] JOIN

LEFT [OUTER] JOIN

FULL [OUTER] JOIN

Słowo **OUTER** jest opcjonalne, może zostać pominięte.

```
SELECT      ename, dname
FROM        emp
LEFT JOIN   dept
ON          emp.deptno = dept.deptno;
```

Instrukcja *SELECT* złączenie *INNER* i *OUTER JOIN*

```
...emp
LEFT / RIGHT / FULL JOIN dept
ON emp.deptno = dept.deptno;
```

Użycie złączenia **LEFT JOIN** zwróci w wyniku wszystkie rekordy spełniające warunek powiązania `emp.deptno = dept.deptno` oraz te rekordy z tabeli po lewej stronie związku (`emp`), które tego warunku nie spełniają. Odpowiednio **RIGHT JOIN** dołączy rekordy z tabeli po prawej stronie (`dept`) nie spełniające warunku powiązania. Wreszcie **FULL JOIN** do wyniku dołączy niespełniające tego warunku rekordy z obu tabel.

Można powiedzieć, że **OUTER JOIN** uzupełnia wynik **INNER JOIN** o rekordy nie spełniające warunku powiązania, pochodzące odpowiednio z tabeli po lewej (**LEFT**), prawej (**RIGHT**) lub obu tabel (**FULL**).

Instrukcja *SELECT* – samozłączenie tabel

Związek może być zdefiniowany na jednej tabeli – mamy do czynienia wówczas ze związkiem **rekurencyjnym** - ta sama tabela występuje po obu stronach związku. Aby odczytać dane z rekordów powiązanych w ten sposób, należy w zapytaniu dwukrotnie przywołać nazwę tabeli, nadając przynajmniej jednej z jej instancji alias, czyli tymczasową nazwę.

```
SELECT      Prac.Ename As Pracownik
            ,Kier.Ename AS Kierownik
FROM        Emp Prac
INNER JOIN  Emp Kier
ON          Prac.Mgr = Kier.Empno;
```

Operatory algebraiczne

Wyniki kilku zapytań można połączyć operatorami algebraicznymi na zasadzie:

Instrukcja_SELECT **operator** instrukcja_SELECT

Zapytania muszą być kompatybilne tzn. zwracać tą samą liczbę kolumn i te same typy danych w odpowiadających sobie kolumnach.

Dostępne operatory:

- **UNION** – sumowanie wyników obu zapytań z usunięciem duplikatów
- **UNION ALL** – sumowanie bez usuwania duplikatów
- **EXCEPT** – różnica (wiersze zwracane przez pierwsze zapytanie, a nie zwracane przez drugie); w **ORACLE MINUS**
- **INTERSECT** – część wspólna (wiersze zwracane w obu zapytaniach)

Zapytania sumaryczne (podsumowujące)

Dane zwracane w wyniku zapytania operującego na jednym lub kilku źródłach rekordów, mogą zostać przeliczone przy użyciu jednej z funkcji sumarycznych (agregujących):

- **`Count`** – liczba wszystkich rekordów, zwykle **`COUNT(1)`**
- **`Avg`** – średnia
- **`Sum`** – suma
- **`Max`** – maksymalna wartość
- **`Min`** – minimalna wartość

Argumentem tych funkcji może być wyrażenie (odpowiedniego typu) lub **`DISTINCT`** wyrażenie. Argumentem funkcji **`COUNT`** może być cokolwiek – liczone są wiersze wchodzące w skład wyniku.

Pseudowartości **Null** nie są brane pod uwagę przy obliczaniu wartości funkcji

Zapytania sumaryczne (podsumowujące)

Rekordy wynikowe zapytania mogą zostać podzielone na grupy, w celu wykonania obliczeń funkcji agregujących nie na całym wyniku, lecz na grupach rekordów. Grupy definiowane są w klauzuli **GROUP BY**, według jednakowych wartości zawartych w niej wyrażeń. Na liście może znajdować się więcej niż jedno wyrażenie.

Rekordy wynikowe, po operacji grupowania i obliczeniach funkcji agregujących, mogą zostać poddane weryfikacji, poprzez zastosowanie warunku logicznego umieszczonego w klauzuli **HAVING**. Do wyniku ostatecznego zostaną włączone tylko te grupy, które spełniają ten warunek (przyjmuje on wartość TRUE).

Elementami listy **SELECT**, klauzuli **HAVING** i **ORDER BY** mogą być tylko:

- stała,
- funkcja agregująca,
- Wyrażenie grupujące (występujące w klauzuli **GROUP BY**)

Jeżeli w wyrażeniu grupującym pojawia się wartość **NULL** jest tworzona dla niej oddzielna grupa.

Zasady wykonania zapytania grupującego

1. Powtórz kroki 2-7 dla każdego składnika operatora algebraicznego.
2. Rozważ kolejno wszystkie kombinacje wierszy tabel występujących w klauzuli **FROM**.
3. Do każdej kombinacji zastosuj warunek złączenia **JOIN** i warunek ograniczający **WHERE**. Pozostaw tylko kombinacje dające wartość ***True*** (usuwając wiersze dające ***False*** lub ***Null***).
4. Podziel pozostające kombinacje na grupy.
5. Dla każdego pozostającego wiersza reprezentującego grupę oblicz wartości wyrażeń na liście **SELECT**.
6. Do każdej grupy zastosuj warunek w klauzuli **HAVING**. Pozostaw tylko grupy, dla których wartość warunku jest ***True***.
7. Jeśli po **SELECT** występuje **DISTINCT**, usuń duplikaty wśród wynikowych wierszy.
8. Jeśli trzeba, zastosuj odpowiedni operator algebraiczny.
9. Jeśli występuje klauzula **ORDER BY**, wykonaj sortowanie wierszy.

Podzapytania

Wewnątrz klauzul **WHERE**, **HAVING** i **FROM**, mogą wystąpić podzapytania, mające taką samą postać jak zapytania (tylko są ujęte w nawiasy). W klauzuli **WHERE** może znajdować się więcej niż jedno podzapytanie.

Podzapytanie może wystąpić jako prawy argument predykatów **=, <,** <=, >, >=, <>, **IN**, **NOT IN**, przy czym w przypadku predykatów =, <, <=, >, >=, <>, powinno zwracać jedną wartość, a w przypadku predykatów **IN** oraz **NOT IN** wartości zwracanych przez podzapytanie może być wiele.

```
… WHERE job IN (SELECT job FROM emp WHERE deptno = 10);
```

Standard języka przewiduje możliwość dokonywania porównań list wartości tych samych rozmiarów. Porównywane są odpowiednie składniki. Implementuje to ORACLE, MS SQL Server nie.

```
… WHERE (job, sal) IN
  (SELECT job, MAX(sal) FROM emp GROUP BY deptno);
```

Podzapytania

W podzapytaniu nie można używać klauzul **`ORDER BY`**.

W podzapytaniu ***zwykłym*** zbiór wynikowych wierszy podzapytania nie zależy od wierszy w głównym zapytaniu.

W podzapytaniu dostępne są nazwy kolumn wprowadzone w głównym zapytaniu (jest to wykorzystywane w podzapytaniach ***skorelowanych***).

Podzapytanie w klauzuli FROM

Jak wspomniano, podzapytanie może pojawić się w klauzuli **FROM**. Pełni ono wówczas rolę dostarczyciela rekordów, budowanego „w locie” widoku, który może zostać powiązany z innymi tabelami lub widokami użytymi w poleceniu **SELECT**.

Przykład:

Znajdź pracowników zarabiających powyżej średnich w ich działach. W wyniku podaj średnią dla działu.

```
SELECT       ename, sal, Avsal, X.deptno
FROM         emp
INNER JOIN   (SELECT deptno, AVG(sal) Avsal
              FROM emp
              GROUP BY deptno) X
ON      emp.deptno = X.deptno
WHERE   sal > Avsal;
```

Common Table Expression

Struktura CTE stanowi alternatywę (wydajniejszą) dla rozwiązań używających składni z podzapytaniem w klauzuli FROM.

```
WITH nazwa (Kol_1, Kol_2, …)
AS
(SELECT Wyr_1, Wyr_2, …
FROM …[pełna składnia instrukcji])
Instrukcja SQL odwołująca się do instrukcji
zdefiniowanej w klauzuli WITH
```

Konstrukcja ta tworzy tymczasowy zestaw wyliczonych rekordów, możliwy do wykorzystania przez instrukcję **SELECT**, **INSERT**, **UPDATE**, **DELETE** zdefiniowaną bezpośrednio po **WITH** i dostępną tylko dla tej jednej instrukcji. Zaletą jej jest efektywność, czyli krótki czas wykonywania operacji.

Użycie CTE nie musi ograniczać się do wykorzystania przez następującą po niej instrukcję **SELECT**. Może zostać użyta do realizacji instrukcji DML, ze szczególnym uwzględnieniem skorelowanej instrukcji **UPDATE**.

Common Table Expression

Poniższy przykład to rozwiązanie alternatywne dla przykładu poprzedniego, gdzie podzapytanie zostało umieszczone w klauzuli FROM. Teraz zostaje ono zastąpione przez CTE.

```
WITH X (deptno, Avsal)
AS
(SELECT    deptno, AVG(sal)
 FROM      emp
 GROUP BY deptno)
SELECT      X.deptno, Avsal, ename, sal
FROM        emp
INNER JOIN  X
ON          emp.deptno = X.deptno
WHERE       sal > Avsal;
```

Podzapytania – kwantyfikatory *Some* (*Any*) i *All*

Gdy podzapytanie zwraca więcej niż jedną wartość (wiele wartości) możemy określić, że interesują nas wartości większe (lub mniejsze) od **<u>każdej</u>** wartości zwróconej przez podzapytanie:

```
sal >= ALL (SELECT Sal FROM Emp WHERE Deptno = 30)
```

Czyli poszukujemy płacy większej lub równej od zarobków każdego z pracowników działu 30, czyli dokonujemy porównania z płacą maksymalną w dziale 30.

Możemy też poszukiwać wartości większych (lub mniejszych) od jakiejkolwiek wartości zwróconej przez podzapytanie:

```
sal >= SOME (SELECT Sal FROM Emp WHERE Deptno = 30)
```

Czyli poszukujemy płacy większej lub równej od zarobków dowolnego z pracowników działu 30, czyli dokonujemy porównania z płacą minimalną w dziale 30.

SOME i **ANY** działają identycznie (mogą być używane wymiennie).

Podzapytania skorelowane

Konstrukcja zapytania z podzapytaniem może uzależniać zbiór wyników podzapytania od wartości występujących w wierszach głównego zapytania. Taką konstrukcję nazywamy zapytaniem skorelowanym z podzapytaniem.

Przykład:

Dla każdego działu znajdź osobę, która zarabia najwięcej w tym dziale.

Rozwiązanie możemy realizować według koncepcji:

```
Maksymalne zarobki w danym dziale =
SELECT Max(Sal)
FROM   Emp
WHERE  Deptno =
                  <Deptno określony w głównym zapytaniu>
```

Podzapytania skorelowane

Całe zapytanie wygląda następująco:

```
SELECT    a.Deptno,
          a.Ename,
          a.Sal
FROM      Emp a
WHERE     Sal =
                  (SELECT Max(b.Sal)
                  FROM Emp b
                  WHERE b.Deptno = a.Deptno);
```

Podkreślenie w warunku **WHERE** podzapytania wskazuje na porównywanie wartości z zapytania nadrzędnego i podrzędnego.

Predykaty *EXISTS* i *NOT EXISTS*

Nie zawsze wykorzystujemy podzapytania do poszukiwania konkretnych wartości. W celu sprawdzenie, czy podzapytanie zwraca zbiór pusty (czy nie), używane są predykaty **EXISTS** i **NOT EXISTS**.

np.

```
EXISTS (SELECT 'x' FROM Emp WHERE Deptno = 10)
```

"istnieje co najmniej jeden pracownik zatrudniony w dziale o numerze 10".

Dla wyniku nie jest istotne co napiszemy na liście **SELECT** w ramach predykatu **EXISTS** – najprostsza obliczeniowo jest wartość stała taka jak 'x' lub 1, gdyż istotne jest tylko, czy podzapytanie zwraca „coś" czy też „nic".

W podzapytanie skorelowanym **EXISTS** i **NOT EXISTS** mogą być użyte w celu sprawdzenia, czy dla danej zależności pomiędzy zapytaniem nadrzędnym i podrzędnym zwracany jest pusty zbiór wyników.

Niemal zawsze używamy ich z podzapytaniami skorelowanymi.

CASE w klauzuli SELECT

Konstrukcja **CASE** użyta w klauzuli **SELECT** pozwala na zdefiniowanie listy wartości, której poszczególne elementy zostaną użyte w miejsce wyrażenia z listy **SELECT**, zależnie od jego wartości. Wyrażenie może być albo wartością stałą (simple **CASE**), albo wyrażeniem logicznym (searched **CASE**)

Simple CASE

```
CASE Alias = wyrażenie
WHEN wartość_wyrażenia THEN wartość_zwracana
WHEN ... THEN ...
[ELSE ...]
END;
```

Searched CASE

```
CASE Alias = wyrażenie
WHEN wyrażenie_logiczne THEN wartość_zwracana
WHEN ... THEN ...
[ELSE ...]
END;
```