

ZAAWANSOWANE SYSTEMY BAZ DANYCH

WYKŁAD

Joanna Kapusta
e-mail: joanna.kapusta@kul.pl

Katolicki Uniwersytet Lubelski Jana Pawła II

POBIERANIE DANYCH

```
SELECT [DISTINCT|ALL]
      {*[wyrażenie_kolumnowe [AS nazwa_kolumny] ],[...]}
FROM  nazwa_tabeli [alias], [...]
[WHERE  warunek_selekcji_wierszy]
[GROUP BY lista_kolumn]
[HAVING warunek_selekcji_grup]
[ORDER BY lista_kolumn];
```

gdzie

- wyrażenie_kolumnowe - nazwa kolumny lub wyrażenie,
- nazwa_tabeli - nazwa tabeli lub perspektywy,
- alias - nazwa zastępcza dla parametru nazwa_tabeli.

KLAUZULA WHERE

Klauzula WHERE - umożliwia wybranie wierszy spełniających pewne warunki.

Warunki selekcji:

- 1 porównanie,
- 2 sprawdzenie zakresu,
- 3 przynależność do zbioru,
- 4 dopasowanie do wzorca,
- 5 wartość pusta.

WARUNEK SELEKCJI: PORÓWNANIE

Operatory porównania:

- równe =,
- różne <>, !=,
- mniejsze <,
- mniejsze lub równe <=,
- większe >,
- większe lub równe >=.

Oddziały

| IdOd | Nazwa | Miasto |
|------|-----------------|----------|
| 321 | Finansowy | Warszawa |
| 322 | Administracyjny | Lublin |
| 323 | Kadr | Radom |
| 324 | Informatyczny | Warszawa |
| 325 | Inwestycji | Warszawa |

Wybrać oddziały z Warszawy.

```
SELECT *
FROM Oddziały
WHERE Miasto = 'Warszawa';
```

| IdOd | Nazwa | Miasto |
|------|---------------|----------|
| 321 | Finansowy | Warszawa |
| 324 | Informatyczny | Warszawa |
| 325 | Inwestycji | Warszawa |

WARUNEK SELEKCJI: SPRAWDZENIE ZAKRESU

Operator BETWEEN...AND służy do sprawdzenia, czy wartość znajduje się w przedziale.
NOT BETWEEN...AND - poza przedziałem

Pracownicy

| IdPrac | Imie | Nazwisko | Pensja | Premia | Stanowisko | DataZatr | IdOd |
|--------|---------|----------|--------|--------|-------------|------------|------|
| 123 | Jan | Kowalski | 4200 | 4800 | księgowy | 2001/11/03 | 321 |
| 124 | Marek | Kowalski | 2400 | | programista | 2008/12/04 | 324 |
| 144 | Janina | Nowak | 3200 | | sekretarka | 2005/02/12 | 322 |
| 145 | Halina | Abacka | 5400 | 2400 | kierownik | 2011/08/14 | 322 |
| 146 | Zenon | Warecki | 3200 | | programista | 2008/08/04 | 324 |
| 147 | Marta | Kos | 1400 | | asystent | 2011/08/14 | 323 |
| 149 | Mariusz | Perkob | 4200 | 1200 | programista | 2010/07/23 | 324 |

Wybrać imiona i nazwiska pracowników o pensji z przedziału [2000, 3200].

```
SELECT Imie, Nazwisko
FROM Pracownicy
WHERE Pensja BETWEEN 2000 AND 3200;
```

| Imie | Nazwisko |
|--------|----------|
| Marek | Kowalski |
| Janina | Nowak |
| Zenon | Warecki |

WARUNEK SELEKCJI: PRZYNALEŻNOŚĆ DO ZBIORU

Operator IN służy do sprawdzenia, czy wartość danych znajduje się na wyspecyfikowanej liści. NOT IN - wartość nie występuje na liście

Pracownicy

| IdPrac | Imie | Nazwisko | Pensja | Premia | Stanowisko | DataZatr | IdOd |
|--------|---------|----------|--------|--------|-------------|------------|------|
| 123 | Jan | Kowalski | 4200 | | księgowy | 2001/11/03 | 321 |
| 124 | Marek | Kowalski | 2400 | 4800 | programista | 2008/12/04 | 324 |
| 144 | Janina | Nowak | 3200 | | sekretarka | 2005/02/12 | 322 |
| 145 | Halina | Abacka | 5400 | | kierownik | 2011/08/14 | 322 |
| 146 | Zenon | Warecki | 3200 | 2400 | programista | 2008/08/04 | 324 |
| 147 | Marta | Kos | 1400 | | asystent | 2011/08/14 | 323 |
| 149 | Mariusz | Perkob | 4200 | 1200 | programista | 2010/07/23 | 324 |

Wybrać imiona i nazwiska kierowników i księgowych.

```
SELECT Imie, Nazwisko
FROM Pracownicy
WHERE Stanowisko IN ('kierownik', 'księgowy');
```

| Imie | Nazwisko |
|--------|----------|
| Jan | Kowalski |
| Halina | Abacka |

WARUNEK SELEKCJI: DOPASOWANIE DO WZORCA

Operator LIKE służy do wybierania wartości odpowiadających podanemu wzorcowi.

NOT LIKE - nie pasuje do wzorca

Symbole zastępcze:

- % - dowolny ciąg znaków (także pusty),
- _ - dokładnie jeden dowolny znak.

%a - dowolny ciąg znaków, w którym ostatni znak to 'a',

b__ - dokładnie trzy znaki, z których pierwszy to 'b',

Oddziały

| IdOd | Nazwa | Miasto |
|------|-----------------|----------|
| 321 | Finansowy | Warszawa |
| 322 | Administracyjny | Lublin |
| 323 | Kadr | Radom |
| 324 | Informatyczny | Warszawa |
| 325 | Inwestycji | Warszawa |

Wybrać nazwy działów rozpoczynające się od litery 'I'.

```
SELECT Nazwa
FROM Oddziały
WHERE Nazwa LIKE 'I%';
```

| |
|--------------|
| Nazwa |
|--------------|

| |
|---------------|
| Informatyczny |
| Inwestycji |

WARUNEK SELEKCJI: WARTOŚCI PUSTE

Operator IS NULL służy do wybierania wartości NULL.
IS NOT NULL - nie NULL

Pracownicy

| IdPrac | Imie | Nazwisko | Pensja | Premia | Stanowisko | DataZatr | IdOd |
|--------|---------|----------|--------|--------|-------------|------------|------|
| 123 | Jan | Kowalski | 4200 | 4800 | księgowy | 2001/11/03 | 321 |
| 124 | Marek | Kowalski | 2400 | | programista | 2008/12/04 | 324 |
| 144 | Janina | Nowak | 3200 | | sekretarka | 2005/02/12 | 322 |
| 145 | Halina | Abacka | 5400 | 2400 | kierownik | 2011/08/14 | 322 |
| 146 | Zenon | Warecki | 3200 | | programista | 2008/08/04 | 324 |
| 147 | Marta | Kos | 1400 | | asystent | 2011/08/14 | 323 |
| 149 | Mariusz | Perkob | 4200 | 1200 | programista | 2010/07/23 | 324 |

Wybrać imiona i nazwiska pracowników, dla których nie określono premii.

```
SELECT Imie, Nazwisko
FROM Pracownicy
WHERE Premia IS NULL;
```

| Imie | Nazwisko |
|--------|----------|
| Jan | Kowalski |
| Janina | Nowak |
| Halina | Abacka |
| Marta | Kos |

Tworzenie złożonych kryteriów

Do budowania warunków złożonych wykorzystuje się operatory AND i OR.

Pracownicy

| IdPrac | Imie | Nazwisko | Pensja | Premia | Stanowisko | DataZatr | IdOd |
|--------|---------|----------|--------|--------|-------------|------------|------|
| 123 | Jan | Kowalski | 4200 | | księgowy | 2001/11/03 | 321 |
| 124 | Marek | Kowalski | 2400 | 4800 | programista | 2008/12/04 | 324 |
| 144 | Janina | Nowak | 3200 | | sekretarka | 2005/02/12 | 322 |
| 145 | Halina | Abacka | 5400 | | kierownik | 2011/08/14 | 322 |
| 146 | Zenon | Warecki | 3200 | 2400 | programista | 2008/08/04 | 324 |
| 147 | Marta | Kos | 1400 | | asystent | 2011/08/14 | 323 |
| 149 | Mariusz | Perkob | 4200 | 1200 | programista | 2010/07/23 | 324 |

```
SELECT Imie, Nazwisko
FROM Pracownicy
WHERE Pensja>4000
      AND Stanowisko='programista'
      OR Stanowisko='sekretarka';
```

```
SELECT Imie, Nazwisko
FROM Pracownicy
WHERE Pensja>4000
      AND (Stanowisko='programista'
      OR Stanowisko='sekretarka');
```

| Imie | Nazwisko |
|---------|----------|
| Janina | Nowak |
| Mariusz | Perkob |

| Imie | Nazwisko |
|---------|----------|
| Mariusz | Perkob |

KLAUZULA ORDER BY

Do uporządkowania wyniku wykorzystuje się klauzulę ORDER BY z listą identyfikatorów (numerów, aliasów) kolumn według, których mają zostać uporządkowane dane.

ASC - rosnąco (domyślnie), DESC - malejąco.

Oddziały

| IdOd | Nazwa | Miasto |
|------|-----------------|----------|
| 321 | Finansowy | Warszawa |
| 322 | Administracyjny | Lublin |
| 323 | Kadr | Radom |
| 324 | Informatyczny | Warszawa |
| 325 | Inwestycji | Warszawa |

```
SELECT Nazwa
FROM Oddziały
ORDER BY Nazwa DESC;
```

```
SELECT Miasto, Nazwa
FROM Oddziały
ORDER BY 1, 2 DESC;
```

| Nazwa |
|-----------------|
| Kadr |
| Inwestycji |
| Informatyczny |
| Finansowy |
| Administracyjny |

| Miasto | Nazwa |
|----------|-----------------|
| Lublin | Administracyjny |
| Radom | Kadr |
| Warszawa | Inwestycji |
| Warszawa | Informatyczny |
| Warszawa | Finansowy |

OGRANICZENIE LICZBY ZWRACANYCH REKORDÓW

Klauzula `FETCH FIRST N ROWS ONLY` ogranicza liczbę zwracanych rekordów.

| Pracownicy | | | | | | | |
|------------|---------|----------|--------|--------|-------------|------------|------|
| IdPrac | Imie | Nazwisko | Pensja | Premia | Stanowisko | DataZatr | IdOd |
| 123 | Jan | Kowalski | 4200 | | księgowy | 2001/11/03 | 321 |
| 124 | Marek | Kowalski | 2400 | 4800 | programista | 2008/12/04 | 324 |
| 144 | Janina | Nowak | 3200 | | sekretarka | 2005/02/12 | 322 |
| 145 | Halina | Abacka | 5400 | | kierownik | 2011/08/14 | 322 |
| 146 | Zenon | Warecki | 3200 | 2400 | programista | 2008/08/04 | 324 |
| 147 | Marta | Kos | 1400 | | asystent | 2011/08/14 | 323 |
| 149 | Mariusz | Perkob | 4200 | 1200 | programista | 2010/07/23 | 324 |

```
SELECT Imie, Nazwisko
FROM Pracownicy
ORDER BY Nazwisko
FETCH FIRST 2 ROWS ONLY;
```

| Imie | Nazwisko |
|--------|----------|
| Halina | Abacka |
| Marta | Kos |

Uwaga:
`FETCH FIRST N ROWS ONLY`- Oracle 12c,
wcześniej `ROWNUM`
`LIMIT N` - MySQL,
`TOP N` - Microsoft SQL Server

FUNKCJE AGREGUJĄCE

- ❶ COUNT - ilość wyrażen w kolumnie,
- ❷ SUM - suma wyrażen w kolumnie,
- ❸ AVG - średnia z wyrażen w kolumnie,
- ❹ MIN - najmniejsza wartość w kolumnie,
- ❺ MAX - największa wartość w kolumnie.

Kwalifikatory:

DISTINCT - eliminuje powtórzenia

ALL - z powtórzeniami (domyślnie)

FUNKCJE AGREGUJĄCE

Pracownicy

| IdPrac | Imie | Nazwisko | Pensja | Premia | Stanowisko | DataZatr | IdOd |
|--------|---------|----------|--------|--------|-------------|------------|------|
| 123 | Jan | Kowalski | 4200 | | księgowy | 2001/11/03 | 321 |
| 124 | Marek | Kowalski | 2400 | 4800 | programista | 2008/12/04 | 324 |
| 144 | Janina | Nowak | 3200 | | sekretarka | 2005/02/12 | 322 |
| 145 | Halina | Abacka | 5400 | | kierownik | 2011/08/14 | 322 |
| 146 | Zenon | Warecki | 3200 | 2400 | programista | 2008/08/04 | 324 |
| 147 | Marta | Kos | 1400 | | asystent | 2011/08/14 | 323 |
| 149 | Mariusz | Perkob | 4200 | 1200 | programista | 2010/07/23 | 324 |

```
SELECT COUNT(*) AS ilosc
FROM Pracownicy
WHERE Stanowisko='programista';
```

| |
|--------------|
| ilosc |
| 3 |

```
SELECT COUNT(DISTINCT Stanowisko)
      AS ile
FROM Pracownicy;
```

| |
|------------|
| ile |
| 5 |

```
SELECT MIN(Pensja), MAX(Pensja)
FROM Pracownicy;
```

| | |
|--------------------|--------------------|
| MIN(Pensja) | MAX(Pensja) |
| 1400 | 5400 |

```
SELECT COUNT(IdPrac) AS ilosc,
      SUM(Pensja) AS suma
FROM Pracownicy
WHERE Stanowisko='programista';
```

| | |
|--------------|-------------|
| ilosc | suma |
| 3 | 9800 |

KLAUZULA GROUP BY

Zapytanie zawierające klauzule GROUP BY nazywamy **zapytaniem grupującym**.

W trakcie jego obliczania dane dzielone są na grupy i dla każdej grupy generowany jest wiersz wynikowy. Grupę stanowią wszystkie wiersze dla których wartości w podanych w klauzuli GROUP BY kolumnach są identyczne.

- Wszystkie nazwy kolumn wymienione na liście SELECT muszą występować w klauzuli GROUP BY. Wyjątek stanowią nazwy kolumn, które występują jedynie jako argumenty funkcji agregujących.
- Kolumny wymienione w GROUP BY nie muszą występować na liście SELECT.
- Z grupowania można wyeliminować pewne wiersze przy pomocy klauzuli WHERE.
- Wartości puste uznawane są za równe.

KLAUZULA GROUP BY

Pracownicy

| IdPrac | Imie | Nazwisko | Pensja | Premia | Stanowisko | DataZatr | IdOd |
|--------|---------|----------|--------|--------|-------------|------------|------|
| 123 | Jan | Kowalski | 4200 | | księgowy | 2001/11/03 | 321 |
| 124 | Marek | Kowalski | 2400 | 4800 | programista | 2008/12/04 | 324 |
| 144 | Janina | Nowak | 3200 | | sekretarka | 2005/02/12 | 322 |
| 145 | Halina | Abacka | 5400 | | kierownik | 2011/08/14 | 322 |
| 146 | Zenon | Warecki | 3200 | 2400 | programista | 2008/08/04 | 324 |
| 147 | Marta | Kos | 1400 | | asystent | 2011/08/14 | 323 |
| 149 | Mariusz | Perkob | 4200 | 1200 | programista | 2010/07/23 | 324 |

```
SELECT Stanowisko, MIN(Pensja) AS Min, MAX(Pensja) AS Max
FROM Pracownicy
GROUP BY Stanowisko;
```

| Stanowisko | Min | Max |
|-------------|------|------|
| księgowy | 4200 | 4200 |
| programista | 2400 | 4200 |
| sekretarka | 3200 | 3200 |
| kierownik | 5400 | 5400 |
| asystent | 1400 | 1400 |

KLAUZULA HAVING

Klauzulę HAVING stosuje się w celu wyselekcjonowania grup.

| Pracownicy | | | | | | | |
|------------|---------|----------|--------|--------|-------------|------------|------|
| IdPrac | Imie | Nazwisko | Pensja | Premia | Stanowisko | DataZatr | IdOd |
| 123 | Jan | Kowalski | 4200 | | księgowy | 2001/11/03 | 321 |
| 124 | Marek | Kowalski | 2400 | 4800 | programista | 2008/12/04 | 324 |
| 144 | Janina | Nowak | 3200 | | sekretarka | 2005/02/12 | 322 |
| 145 | Halina | Abacka | 5400 | | kierownik | 2011/08/14 | 322 |
| 146 | Zenon | Warecki | 3200 | 2400 | programista | 2008/08/04 | 324 |
| 147 | Marta | Kos | 1400 | | asystent | 2011/08/14 | 323 |
| 149 | Mariusz | Perkob | 4200 | 1200 | programista | 2010/07/23 | 324 |

Podaj średnią wartość pensji dla poszczególnych stanowisk - w wyniku uwzględnij tylko te stanowiska, dla których maksymalna pensja jest wyższa od 4000.

```
SELECT Stanowisko, AVG(Pensja)
FROM Pracownicy
GROUP BY Stanowisko
HAVING MAX(Pensja)>4000;
```

| Stanowisko | AVG(Pensja) |
|-------------|-------------|
| księgowy | 4200 |
| programista | 3266.66 |
| kierownik | 5400 |

ZŁĄCZENIA WEWNĘTRZNE

```

SELECT lista_kolumn                               SELECT lista_kolumn
FROM nazwa_tabela1 JOIN nazwa_tabela2             FROM nazwa_tabela1 JOIN nazwa_tabela2
ON warunek_złączenia;                             USING (kolumna_złączenia);
    USING można stosować, gdy w tabelach występują kolumny o takich samych nazwach

```

Oddziały

| IdOd | Nazwa | Miasto |
|------|---------------|----------|
| 321 | Finansowy | Warszawa |
| 324 | Informatyczny | Warszawa |

Pracownicy

| IdPrac | Imie | Nazwisko | Pensja | Premia | Stanowisko | DataZatr | IdOd |
|--------|---------|----------|--------|--------|-------------|------------|------|
| 123 | Jan | Kowalski | 4200 | | księgowy | 2001/11/03 | 321 |
| 124 | Marek | Kowalski | 2400 | 4800 | programista | 2008/12/04 | 324 |
| 146 | Zenon | Warecki | 3200 | 2400 | programista | 2008/08/04 | 324 |
| 149 | Mariusz | Perkob | 4200 | 1200 | programista | 2010/07/23 | 324 |

```

SELECT Imie, Nazwisko, Nazwa
FROM Pracownicy JOIN Oddziały
USING (IdOd);

```

| Imie | Nazwisko | Nazwa |
|---------|----------|---------------|
| Jan | Kowalski | Finansowy |
| Marek | Kowalski | Informatyczny |
| Zenon | Warecki | Informatyczny |
| Mariusz | Perkob | Informatyczny |

```

SELECT Imie, Nazwisko, Nazwa
FROM Pracownicy p JOIN Oddzialy o
ON p.IdOd = o.IdOd;

```

ZŁĄCZENIA ZEWNĘTRZNE

- LEFT OUTER JOIN
- RIGHT OUTER JOIN
- FULL OUTER JOIN

Lewostronne złączenie zewnętrzne LEFT OUTER JOIN - wybrane zostaną wiersze z pierwszej tabeli, które mają odpowiedniki w drugiej tabeli oraz dodatkowo wiersze z pierwszej tabeli, które nie mają odpowiedników w drugiej tabeli.

Oddziały

| IdOd | Nazwa | Miasto |
|------|---------------|----------|
| 321 | Finansowy | Warszawa |
| 324 | Informatyczny | Warszawa |
| 325 | Inwestycji | Warszawa |

Pracownicy

| IdPrac | Imie | Nazwisko | Pensja | Premia | Stanowisko | DataZatr | IdOd |
|--------|---------|----------|--------|--------|-------------|------------|------|
| 123 | Jan | Kowalski | 4200 | | księgowy | 2001/11/03 | 321 |
| 124 | Marek | Kowalski | 2400 | 4800 | programista | 2008/12/04 | 324 |
| 146 | Zenon | Warecki | 3200 | 2400 | programista | 2008/08/04 | 324 |
| 149 | Mariusz | Perkob | 4200 | 1200 | programista | 2010/07/23 | NULL |

```
SELECT Nazwa, Imie, Nazwisko
FROM Oddziały o LEFT OUTER JOIN Pracownicy p
ON o.IdOd = p.IdOd;
```

| Nazwa | Imie | Nazwisko |
|---------------|-------|----------|
| Finansowy | Jan | Kowalski |
| Informatyczny | Marek | Kowalski |
| Informatyczny | Zenon | Warecki |
| Inwestycji | NULL | NULL |

PODZAPYTANIA

Podzapytanie (zapytanie wewnętrzne) to zapytanie występujące w innym zapytaniu.

Pracownicy

| IdPrac | Imie | Nazwisko | Pensja | Premia | Stanowisko | DataZatr | IdOd |
|--------|---------|----------|--------|--------|-------------|------------|------|
| 123 | Jan | Kowalski | 4200 | | księgowy | 2001/11/03 | 321 |
| 124 | Marek | Kowalski | 2400 | 4800 | programista | 2008/12/04 | 324 |
| 144 | Janina | Nowak | 3200 | | sekretarka | 2005/02/12 | 322 |
| 145 | Halina | Abacka | 5400 | | kierownik | 2011/08/14 | 322 |
| 146 | Zenon | Warecki | 3200 | 2400 | programista | 2008/08/04 | 324 |
| 147 | Marta | Kos | 1400 | | asystent | 2011/08/14 | 323 |
| 149 | Mariusz | Perkob | 4200 | 1200 | programista | 2010/07/23 | 324 |

Podaj imiona i nazwiska pracowników, których pensja jest wyższa od średniej pensji w firmie.

```
SELECT Imie, Nazwisko
FROM Pracownicy
WHERE Pensja > (SELECT AVG(Pensja)
FROM PRACOWNICY);
```

```
SELECT Imie, Nazwisko
FROM Pracownicy      -ŹLE!!!!
WHERE Pensja > AVG(Pensja);
```

| Imie | Nazwisko |
|---------|----------|
| Jan | Kowalski |
| Halina | Abacka |
| Mariusz | Perkob |

PODZAPYTANIA SKORELOWANE

Podzapytanie wewnętrzne odwołuje się do jednej lub kilku kolumn zapytania zewnętrznego.

Pracownicy

| IdPrac | Imie | Nazwisko | Pensja | Premia | Stanowisko | DataZatr | IdOd |
|--------|---------|----------|--------|--------|-------------|------------|------|
| 123 | Jan | Kowalski | 4200 | 4800 | księgowy | 2001/11/03 | 321 |
| 124 | Marek | Kowalski | 2400 | | programista | 2008/12/04 | 324 |
| 144 | Janina | Nowak | 3200 | | sekretarka | 2005/02/12 | 322 |
| 145 | Halina | Abacka | 5400 | 2400 | kierownik | 2011/08/14 | 322 |
| 146 | Zenon | Warecki | 3200 | | programista | 2008/08/04 | 324 |
| 147 | Marta | Kos | 1400 | | asystent | 2011/08/14 | 323 |
| 149 | Mariusz | Perkob | 4200 | 1200 | programista | 2010/07/23 | 324 |

Podaj imiona i nazwiska pracowników, zarabiających więcej niż wynosi średnie wynagrodzenie dla danego stanowiska.

```
SELECT Imie, Nazwisko
FROM Pracownicy zewn
WHERE Pensja > (SELECT AVG(Pensja)
FROM PRACOWNICY wewn
WHERE wewn.Stanowisko=zewn.Stanowisko);
```

PRZYKŁADY

Oddziały(IdOd, Nazwa, Miasto)

Pracownicy(IdPrac, Imie, Nazwisko, Pensja, Premia, Stanowisko, DataZatr, IdOd)

- 1 Podaj imiona i nazwiska pracowników, których pensja mieści się w przedziale [2000, 5000].

```
SELECT  Imie, Nazwisko
FROM Pracownicy
WHERE Pensja BETWEEN 2000 AND 5000;
```

- 2 Podaj nazwy wszystkich stanowisk (bez powtórzeń).

```
SELECT DISTINCT Stanowisko
FROM Pracownicy;
```

- 3 Podaj ilość programistów.

```
SELECT COUNT(Stanowisko)
FROM Pracownicy
WHERE Stanowisko='programista';
```

PRZYKŁADY

Oddziały(IdOd, Nazwa, Miasto)

Pracownicy(IdPrac, Imie, Nazwisko, Pensja, Premia, Stanowisko, DataZatr, IdOd)

- 1 Podaj nazwy oddziałów rozpoczynające się od litery 'I'.

```
SELECT Nazwa  
FROM Oddziały  
WHERE Nazwa LIKE 'I%';
```

- 2 Podaj imiona i nazwiska pracowników, dla których nie określono premii.

```
SELECT Imie, Nazwisko  
FROM Pracownicy  
WHERE Premia IS NULL;
```

- 3 Podaj liczbę różnych stanowisk.

```
SELECT COUNT(DISTINCT Stanowisko)  
        AS ile  
FROM Pracownicy;
```

PRZYKŁADY

Oddziały(IdOd, Nazwa, Miasto)

Pracownicy(IdPrac, Imie, Nazwisko, Pensja, Premia, Stanowisko, DataZatr, IdOd)

- 1 Podaj imiona, nazwiska pracowników oraz wartości pensji po 10% podwyżce.

```
SELECT Imie, Nazwisko, Pensja*1.1 AS ''Pensja po podwyżce''
FROM Pracownicy;
```

- 2 Podaj nazwy oddziałów oraz imiona i nazwiska zatrudnionych w nich pracowników. Wyniki posortuj rosnąco wg nazwy oddziału.

```
SELECT Nazwa, Imie, Nazwisko
FROM Pracownicy p JOIN Oddziały o ON p.IdOd = o.IdOd
ORDER BY 1;
```

- 3 Podaj ilość zatrudnionych osób w oddziałach o liczbie pracowników większej niż 2. W wyniku należy wyświetlić nazwę oddziału i liczbę pracowników, posortowane w kolejności malejącej według nazw oddziałów.

```
SELECT Nazwa, COUNT(IdPrac) AS ilosc
FROM Pracownicy JOIN Oddziały USING (IdOd)
GROUP BY Nazwa
HAVING COUNT(IdPrac)>2
ORDER BY Nazwa DESC;
```

PRZYKŁADY

Oddziały(IdOd, Nazwa, Miasto)

Pracownicy(IdPrac, Imie, Nazwisko, Pensja, Premia, Stanowisko, DataZatr, IdOd)

- 1 Podaj imiona i nazwiska pracowników, których pensja jest wyższa od średniej pensji w firmie.

```
SELECT Imie, Nazwisko
FROM Pracownicy
WHERE Pensja > (SELECT AVG(Pensja) FROM Pracownicy);
```

- 2 Podaj nazwy oddziałów i średnie wynagrodzenia w oddziałach, ale tylko tych oddziałów dla których średnie wynagrodzenie jest wyższe niż średnie wynagrodzenie w firmie.

```
SELECT o.Nazwa, AVG(p.Pensja)
FROM Pracownicy p JOIN Oddziały o ON p.IdOd = o.IdOd
GROUP BY o.Nazwa
HAVING AVG(p.Pensja)> (SELECT AVG(Pensja) FROM Pracownicy);
```

- 3 Podaj imiona i nazwiska pracowników, pracujących na tym samym stanowisku co Kowalski.

```
SELECT Imie, Nazwisko
FROM Pracownicy
WHERE Stanowisko IN (SELECT Stanowisko FROM Pracownicy
WHERE Nazwisko='Kowalski');
```


PRZYKŁADY

Oddziały(IdOd, Nazwa, Miasto)

Pracownicy(IdPrac, Imie, Nazwisko, Pensja, Premia, Stanowisko, DataZatr, IdOd)

- 1 Dla każdego stanowiska podaj imiona i nazwiska pracowników zarabiających najmniej na danym stanowisku.

```
SELECT Stanowisko, Imie, Nazwisko
FROM Pracownicy
WHERE (Stanowisko, Pensja)
      IN (SELECT Stanowisko, min(Pensja)
          FROM PRACOWNICY
          GROUP BY Stanowisko);
```

- 2 Podaj imiona i nazwiska pracowników, zarabiających więcej niż wynosi średnia dla danego stanowiska (skorelowane).

```
SELECT Imie, Nazwisko
FROM Pracownicy zewn
WHERE Pensja > (SELECT AVG(Pensja)
                FROM PRACOWNICY wewn
                WHERE wewn.Stanowisko=zewn.Stanowisko);
```

ZAAWANSOWANE SYSTEMY BAZ DANYCH

WYKŁAD

Joanna Kapusta
e-mail: joanna.kapusta@kul.pl

Katolicki Uniwersytet Lubelski Jana Pawła II

WIELOKROTNE GRUPOWANIE - WSTĘP DO KOSTEK ANALITYCZNYCH

Rozszerzenia klauzuli GROUP BY

- ROLLUP - zwraca wiersze podsumowań częściowych dla poszczególnych grup kolumn oraz wiersz zawierający podsumowanie całościowe (istotna jest kolejność grupowania kolumn). Polecenie ROLLUP służy do konstruowania pół-kostek danych.
- CUBE - zwraca wiersze podsumowań częściowych dla wszystkich kombinacji kolumn oraz wiersz zawierający podsumowanie całościowe.

Dzięki rozszerzeniom klauzuli GROUP BY zapytania mogą wyznaczać wiele zbiorów agregacji na różnych poziomach grupowania.

OPERATOR ROLUP

Pracownicy

| IdPrac | Imie | Nazwisko | Pensja | Premia | Stanowisko | DataZatr | IdOd |
|--------|---------|----------|--------|--------|-------------|------------|------|
| 123 | Jan | Kowalski | 4200 | | księgowy | 2001/11/03 | 321 |
| 124 | Marek | Kowalski | 2400 | 4800 | programista | 2008/12/04 | 324 |
| 144 | Janina | Nowak | 3200 | | sekretarka | 2005/02/12 | 322 |
| 145 | Halina | Abacka | 5400 | | kierownik | 2011/08/14 | 324 |
| 146 | Zenon | Warecki | 3200 | 2400 | programista | 2008/08/04 | 324 |
| 147 | Marta | Kos | 1400 | | asystent | 2011/08/14 | 323 |
| 149 | Mariusz | Perkob | 4200 | 1200 | programista | 2010/07/23 | 324 |

Suma pensji dla poszczególnych stanowisk.

```
SELECT Stanowisko, SUM(Pensja)
FROM Pracownicy
GROUP BY Stanowisko;
```

| Stanowisko | SUM(Pensja) |
|-------------|-------------|
| programista | 9800 |
| sekretarka | 3200 |
| kierownik | 5400 |
| księgowy | 4200 |
| asystent | 1400 |

```
SELECT Stanowisko, SUM(Pensja)
FROM Pracownicy
GROUP BY ROLLUP(Stanowisko);
```

| Stanowisko | SUM(Pensja) |
|-------------|-------------|
| programista | 9800 |
| sekretarka | 3200 |
| kierownik | 5400 |
| księgowy | 4200 |
| asystent | 1400 |
| null | 24000 |

OPERATOR ROLUP

Pracownicy

| IdPrac | Imie | Nazwisko | Pensja | Premia | Stanowisko | DataZatr | IdOd |
|--------|---------|----------|--------|--------|-------------|------------|------|
| 123 | Jan | Kowalski | 4200 | | księgowy | 2001/11/03 | 321 |
| 124 | Marek | Kowalski | 2400 | 4800 | programista | 2008/12/04 | 324 |
| 144 | Janina | Nowak | 3200 | | sekretarka | 2005/02/12 | 322 |
| 145 | Halina | Abacka | 5400 | | kierownik | 2011/08/14 | 324 |
| 146 | Zenon | Warecki | 3200 | 2400 | programista | 2008/08/04 | 324 |
| 147 | Marta | Kos | 1400 | | asystent | 2011/08/14 | 323 |
| 149 | Mariusz | Perkob | 4200 | 1200 | programista | 2010/07/23 | 324 |

Suma pensji dla poszczególnych stanowisk w poszczególnych działach.

```
SELECT IdOd, Stanowisko, SUM(Pensja)
FROM Pracownicy
GROUP BY IdOd, Stanowisko;
```

| IdOd | Stanowisko | SUM(Pensja) |
|------|-------------|-------------|
| 322 | sekretarka | 3200 |
| 321 | księgowy | 4200 |
| 323 | asystent | 1400 |
| 324 | kierownik | 5400 |
| 324 | programista | 9800 |

WIELE KOLUMN W ROLLUP

W przypadku gdy do ROLLUP przekazanych zostaje wiele kolumn wiersze są grupowane w bloki z tymi samymi wartościami w kolumnach.

| Pracownicy | | | | | | | |
|------------|---------|----------|--------|--------|-------------|------------|------|
| IdPrac | Imie | Nazwisko | Pensja | Premia | Stanowisko | DataZatr | IdOd |
| 123 | Jan | Kowalski | 4200 | 4800 | księgowy | 2001/11/03 | 321 |
| 124 | Marek | Kowalski | 2400 | | programista | 2008/12/04 | 324 |
| 144 | Janina | Nowak | 3200 | | sekretarka | 2005/02/12 | 322 |
| 145 | Halina | Abacka | 5400 | 2400 | kierownik | 2011/08/14 | 324 |
| 146 | Zenon | Warecki | 3200 | | programista | 2008/08/04 | 324 |
| 147 | Marta | Kos | 1400 | | asystent | 2011/08/14 | 323 |
| 149 | Mariusz | Perkob | 4200 | | programista | 2010/07/23 | 324 |

Suma pensji dla poszczególnych stanowisk w poszczególnych działach.

```
SELECT IdOd, Stanowisko, SUM(Pensja)
FROM Pracownicy
GROUP BY ROLLUP(IdOd, Stanowisko);
```

wiersze z podsumowaniem dla poszczególnych IdOd i wiersz podsumowania całościowego

| IdOd | Stanowisko | SUM(Pensja) |
|------|-------------|-------------|
| 321 | księgowy | 4200 |
| 321 | null | 4200 |
| 322 | sekretarka | 3200 |
| 322 | null | 3200 |
| 323 | asystent | 1400 |
| 323 | null | 1400 |
| 324 | kierownik | 5400 |
| 324 | programista | 9800 |
| 324 | null | 15200 |
| null | null | 24000 |

ZAMIANA POZYCJI KOLUMN W ROLUP

| Pracownicy | | | | | | | |
|------------|---------|----------|--------|--------|-------------|------------|------|
| IdPrac | Imie | Nazwisko | Pensja | Premia | Stanowisko | DataZatr | IdOd |
| 123 | Jan | Kowalski | 4200 | | księgowy | 2001/11/03 | 321 |
| 124 | Marek | Kowalski | 2400 | 4800 | programista | 2008/12/04 | 324 |
| 144 | Janina | Nowak | 3200 | | sekretarka | 2005/02/12 | 322 |
| 145 | Halina | Abacka | 5400 | | kierownik | 2011/08/14 | 324 |
| 146 | Zenon | Warecki | 3200 | 2400 | programista | 2008/08/04 | 324 |
| 147 | Marta | Kos | 1400 | | asystent | 2011/08/14 | 323 |
| 149 | Mariusz | Perkob | 4200 | 1200 | programista | 2010/07/23 | 324 |

Suma pensji dla poszczególnych stanowisk w poszczególnych działach.

```
SELECT Stanowisko, IdOd, SUM(Pensja)
FROM Pracownicy
GROUP BY ROLLUP(Stanowisko, IdOd);
```

wiersze z podsumowaniem dla
poszczególnych stanowisk i wiersz
podsumowania całościowego

| Stanowisko | IdOd | SUM(Pensja) |
|-------------|------|-------------|
| asystent | 323 | 1400 |
| asystent | null | 1400 |
| kierownik | 324 | 5400 |
| kierownik | null | 5400 |
| księgowy | 321 | 4200 |
| księgowy | null | 4200 |
| sekretarka | 322 | 3200 |
| sekretarka | null | 3200 |
| programista | 324 | 9800 |
| programista | null | 9800 |
| null | null | 24000 |

OPERATOR ROLLUP

Z operatorem ROLLUP można używać dowolnych funkcji agregujących.

| Pracownicy | | | | | | | |
|------------|---------|----------|--------|--------|-------------|------------|------|
| IdPrac | Imie | Nazwisko | Pensja | Premia | Stanowisko | DataZatr | IdOd |
| 123 | Jan | Kowalski | 4200 | | księgowy | 2001/11/03 | 321 |
| 124 | Marek | Kowalski | 2400 | 4800 | programista | 2008/12/04 | 324 |
| 144 | Janina | Nowak | 3200 | | sekretarka | 2005/02/12 | 322 |
| 145 | Halina | Abacka | 5400 | | kierownik | 2011/08/14 | 324 |
| 146 | Zenon | Warecki | 3200 | 2400 | programista | 2008/08/04 | 324 |
| 147 | Marta | Kos | 1400 | | asystent | 2011/08/14 | 323 |
| 149 | Mariusz | Perkob | 4200 | 1200 | programista | 2010/07/23 | 324 |

```
SELECT Stanowisko, IdOd, SUM(Pensja) , MAX(Pensja), MIN(Pensja)
FROM Pracownicy
GROUP BY ROLLUP(Stanowisko, IdOd);
```

| Stanowisko | IdOd | SUM(Pensja) | MAX(Pensja) | MIN(Pensja) |
|-------------|------|-------------|-------------|-------------|
| asystent | 323 | 1400 | 1400 | 1400 |
| asystent | null | 1400 | 1400 | 1400 |
| kierownik | 324 | 5400 | 5400 | 5400 |
| kierownik | null | 5400 | 5400 | 5400 |
| księgowy | 321 | 4200 | 4200 | 4200 |
| księgowy | null | 4200 | 4200 | 4200 |
| sekretarka | 322 | 3200 | 3200 | 3200 |
| sekretarka | null | 3200 | 3200 | 3200 |
| programista | 324 | 9800 | 4200 | 2400 |
| programista | null | 9800 | 4200 | 2400 |
| null | null | 24000 | 5400 | 1400 |

OPERATOR CUBE

CUBE - rozszerza klauzulę GROUP BY zwracając dodatkowo wiersze podsumowań częściowych dla wszystkich kombinacji kolumn oraz wiersz zawierającego podsumowanie całościowe

Pracownicy

| IdPrac | Imie | Nazwisko | Pensja | Premia | Stanowisko | DataZatr | IdOd |
|--------|---------|----------|--------|--------|-------------|------------|------|
| 123 | Jan | Kowalski | 4200 | | księgowy | 2001/11/03 | 321 |
| 124 | Marek | Kowalski | 2400 | 4800 | programista | 2008/12/04 | 324 |
| 144 | Janina | Nowak | 3200 | | sekretarka | 2005/02/12 | 322 |
| 145 | Halina | Abacka | 5400 | | kierownik | 2011/08/14 | 324 |
| 146 | Zenon | Warecki | 3200 | 2400 | programista | 2008/08/04 | 324 |
| 147 | Marta | Kos | 1400 | | asystent | 2011/08/14 | 323 |
| 149 | Mariusz | Perkob | 4200 | 1200 | programista | 2010/07/23 | 324 |

```
SELECT Stanowisko, IdOd, SUM(Pensja)
FROM Pracownicy
GROUP BY CUBE(Stanowisko, IdOd);
```

sumy wynagrodzeń dla każdego stanowiska
i oddziału oraz podsumowanie całościowe

| Stanowisko | IdOd | SUM(Pensja) |
|-------------|------|-------------|
| null | null | 24000 |
| null | 321 | 4200 |
| null | 322 | 3200 |
| null | 323 | 1400 |
| null | 324 | 15200 |
| asystent | null | 1400 |
| asystent | 323 | 1400 |
| kierownik | null | 5400 |
| kierownik | 324 | 5400 |
| księgowy | null | 4200 |
| księgowy | 321 | 4200 |
| sekretarka | null | 3200 |
| sekretarka | 322 | 3200 |
| programista | null | 9800 |
| programista | 324 | 9800 |

FUNKCJA GROUPING

GROUPING - funkcja dla kolumny zwraca 0 albo 1; 1 jeżeli wartość kolumny wynosi null i 0 w przeciwnym przypadku. Używana w zapytaniach zawierających ROLLUP i CUBE do wskazania wierszy podsumowań - gdy chcemy wyświetlić wartość w miejscu w którym pojawiłby się null.

| Pracownicy | | | | | | | |
|------------|---------|----------|--------|--------|-------------|------------|------|
| IdPrac | Imie | Nazwisko | Pensja | Premia | Stanowisko | DataZatr | IdOd |
| 123 | Jan | Kowalski | 4200 | | księgowy | 2001/11/03 | 321 |
| 124 | Marek | Kowalski | 2400 | 4800 | programista | 2008/12/04 | 324 |
| 144 | Janina | Nowak | 3200 | | sekreterka | 2005/02/12 | 322 |
| 145 | Halina | Abacka | 5400 | | kierownik | 2011/08/14 | 324 |
| 146 | Zenon | Warecki | 3200 | 2400 | programista | 2008/08/04 | 324 |
| 147 | Marta | Kos | 1400 | | asystent | 2011/08/14 | 323 |
| 149 | Mariusz | Perkob | 4200 | 1200 | programista | 2010/07/23 | 324 |

```
SELECT GROUPING(Stanowisko), Stanowisko, SUM(Pensja)
FROM Pracownicy
GROUP BY ROLLUP(Stanowisko);
```

| GROUPING(Stanowisko) | Stanowisko | SUM(Pensja) |
|----------------------|-------------|-------------|
| 0 | programista | 9800 |
| 0 | sekreterka | 3200 |
| 0 | kierownik | 5400 |
| 0 | księgowy | 4200 |
| 0 | asystent | 1400 |
| 1 | null | 24000 |

WYRAŻENIE CASE

| Tytul | IdGatunku |
|------------|-----------|
| Miś | 1 |
| Drakula | 2 |
| Egzorcysta | 2 |

```
SELECT Tytul, IdGatunku,
CASE IdGatunku
WHEN 1 THEN 'komedia'
WHEN 2 THEN 'horror'
ELSE 'sensacyjny'
END Gatunek
from Filmy;
```

```
SELECT Tytul, IdGatunku,
CASE
WHEN IdGatunku = 1 THEN 'komedia'
WHEN IdGatunku = 2 THEN 'horror'
ELSE 'sensacyjny'
END Gatunek
from Filmy;
```

| Tytul | IdGatunku | Gatunek |
|------------|-----------|---------|
| Miś | 1 | komedia |
| Drakula | 2 | horror |
| Egzorcysta | 2 | horror |

FUNKCJA GROUPING Z WYRAŻENIEM CASE

Pracownicy

| IdPrac | Imie | Nazwisko | Pensja | Premia | Stanowisko | DataZatr | IdOd |
|--------|---------|----------|--------|--------|-------------|------------|------|
| 123 | Jan | Kowalski | 4200 | | księgowy | 2001/11/03 | 321 |
| 124 | Marek | Kowalski | 2400 | 4800 | programista | 2008/12/04 | 324 |
| 144 | Janina | Nowak | 3200 | | sekretarka | 2005/02/12 | 322 |
| 145 | Halina | Abacka | 5400 | | kierownik | 2011/08/14 | 324 |
| 146 | Zenon | Warecki | 3200 | 2400 | programista | 2008/08/04 | 324 |
| 147 | Marta | Kos | 1400 | | asystent | 2011/08/14 | 323 |
| 149 | Mariusz | Perkob | 4200 | 1200 | programista | 2010/07/23 | 324 |

```

SELECT
CASE GROUPING(Stanowisko)
WHEN 1 THEN 'RAZEM'
ELSE Stanowisko
END as Stanowisko, SUM(Pensja)
FROM Pracownicy
GROUP BY ROLLUP(Stanowisko);

```

| Stanowisko | SUM(Pensja) |
|-------------|-------------|
| asystent | 1400 |
| kierownik | 5400 |
| księgowy | 4200 |
| programista | 9800 |
| sekretarka | 3200 |
| RAZEM | 24000 |

FUNKCJA GROUPING Z WYRAŻENIEM CASE

Pracownicy

| IdPrac | Imie | Nazwisko | Pensja | Premia | Stanowisko | DataZatr | IdOd |
|--------|---------|----------|--------|--------|-------------|------------|------|
| 123 | Jan | Kowalski | 4200 | | księgowy | 2001/11/03 | 321 |
| 124 | Marek | Kowalski | 2400 | 4800 | programista | 2008/12/04 | 324 |
| 144 | Janina | Nowak | 3200 | | sekretarka | 2005/02/12 | 322 |
| 145 | Halina | Abacka | 5400 | | kierownik | 2011/08/14 | 324 |
| 146 | Zenon | Warecki | 3200 | 2400 | programista | 2008/08/04 | 324 |
| 147 | Marta | Kos | 1400 | | asystent | 2011/08/14 | 323 |
| 149 | Mariusz | Perkob | 4200 | 1200 | programista | 2010/07/23 | 324 |

```

SELECT
CASE GROUPING(IdOd)
WHEN 1 THEN 'Wszystkie oddzialy'
ELSE IdOd
END as Oddzial,
CASE GROUPING(Stanowisko)
WHEN 1 THEN 'Wszystkie stanowiska'
ELSE Stanowisko
END as Stanowisko,
SUM(Pensja) as Suma
FROM Pracownicy
GROUP BY ROLLUP(IdOd, Stanowisko);

```

| Oddzial | Stanowisko | Suma |
|--------------------|----------------------|-------|
| 321 | księgowy | 4200 |
| 321 | Wszystkie stanowiska | 4200 |
| 322 | sekretarka | 3200 |
| 322 | Wszystkie stanowiska | 3200 |
| 323 | asystent | 1400 |
| 323 | Wszystkie stanowiska | 1400 |
| 324 | kierownik | 5400 |
| 324 | programista | 9800 |
| 324 | Wszystkie stanowiska | 15200 |
| Wszystkie oddzialy | Wszystkie stanowiska | 24000 |

FUNKCJA GROUPING Z OPERATOREM CUBE

Pracownicy

| IdPrac | Imie | Nazwisko | Pensja | Premia | Stanowisko | DataZatr | IdOd |
|--------|---------|----------|--------|--------|-------------|------------|------|
| 123 | Jan | Kowalski | 4200 | | księgowy | 2001/11/03 | 321 |
| 124 | Marek | Kowalski | 2400 | 4800 | programista | 2008/12/04 | 324 |
| 144 | Janina | Nowak | 3200 | | sekretarka | 2005/02/12 | 322 |
| 145 | Halina | Abacka | 5400 | | kierownik | 2011/08/14 | 324 |
| 146 | Zenon | Warecki | 3200 | 2400 | programista | 2008/08/04 | 324 |
| 147 | Marta | Kos | 1400 | | asystent | 2011/08/14 | 323 |
| 149 | Mariusz | Perkob | 4200 | 1200 | programista | 2010/07/23 | 324 |

```

SELECT
CASE GROUPING(IdOd)
WHEN 1 THEN 'Wszystkie oddzialy'
ELSE IdOd
END as Oddzial,
CASE GROUPING(Stanowisko)
WHEN 1 THEN 'Wszystkie stanowiska'
ELSE Stanowisko
END as Stanowisko,
SUM(Pensja) as Suma
FROM Pracownicy
GROUP BY CUBE(IdOd ,Stanowisko)
order by 1, 2;

```

| Oddzial | Stanowisko | Suma |
|--------------------|----------------------|-------|
| 321 | księgowy | 4200 |
| 321 | Wszystkie stanowiska | 4200 |
| 322 | sekretarka | 3200 |
| 322 | Wszystkie stanowiska | 3200 |
| 323 | asystent | 1400 |
| 323 | Wszystkie stanowiska | 1400 |
| 324 | kierownik | 5400 |
| 324 | programista | 9800 |
| 324 | Wszystkie stanowiska | 15200 |
| Wszystkie oddzialy | asystent | 1400 |
| Wszystkie oddzialy | kierownik | 5400 |
| Wszystkie oddzialy | księgowy | 4200 |
| Wszystkie oddzialy | programista | 9800 |
| Wszystkie oddzialy | sekretarka | 3200 |
| Wszystkie oddzialy | Wszystkie stanowiska | 24000 |

GROUPING SETS

- umożliwia utworzenie tylko wierszy podsumowań częściowych (ten sam efekt UNION ALL - mniejsza wydajność)

Pracownicy

| IdPrac | Imie | Nazwisko | Pensja | Premia | Stanowisko | DataZatr | IdOd |
|--------|---------|----------|--------|--------|-------------|------------|------|
| 123 | Jan | Kowalski | 4200 | | księgowy | 2001/11/03 | 321 |
| 124 | Marek | Kowalski | 2400 | 4800 | programista | 2008/12/04 | 324 |
| 144 | Janina | Nowak | 3200 | | sekretarka | 2005/02/12 | 322 |
| 145 | Halina | Abacka | 5400 | | kierownik | 2011/08/14 | 324 |
| 146 | Zenon | Warecki | 3200 | 2400 | programista | 2008/08/04 | 324 |
| 147 | Marta | Kos | 1400 | | asystent | 2011/08/14 | 323 |
| 149 | Mariusz | Perkob | 4200 | 1200 | programista | 2010/07/23 | 324 |

| Stanowisko | Oddzial | Suma |
|-------------|---------|-------|
| programista | null | 9800 |
| sekretarka | null | 3200 |
| kierownik | null | 5400 |
| księgowy | null | 4200 |
| asystent | null | 1400 |
| null | 324 | 15200 |
| null | 323 | 1400 |
| null | 321 | 4200 |
| null | 322 | 3200 |

```
SELECT Stanowisko, IdOd, SUM(Pensja) as Suma
FROM Pracownicy
GROUP BY GROUPING SETS (Stanowisko, IdOd);
```

```
SELECT Stanowisko, null, SUM(Pensja) as Suma
FROM Pracownicy
GROUP BY Stanowisko
UNION ALL
```

```
SELECT null, IdOd, SUM(Pensja) as Suma
FROM Pracownicy
GROUP BY IdOd;
```

GROUPING SETS

Pracownicy

| IdPrac | Imie | Nazwisko | Pensja | Premia | Stanowisko | DataZatr | IdOd |
|--------|---------|----------|--------|--------|-------------|------------|------|
| 123 | Jan | Kowalski | 4200 | | księgowy | 2001/11/03 | 321 |
| 124 | Marek | Kowalski | 2400 | 4800 | programista | 2008/12/04 | 324 |
| 144 | Janina | Nowak | 3200 | | sekretarka | 2005/02/12 | 322 |
| 145 | Halina | Abacka | 5400 | | kierownik | 2011/08/14 | 324 |
| 146 | Zenon | Warecki | 3200 | 2400 | programista | 2008/08/04 | 324 |
| 147 | Marta | Kos | 1400 | | asystent | 2011/08/14 | 323 |
| 149 | Mariusz | Perkob | 4200 | 1200 | programista | 2010/07/23 | 324 |

| Stanowisko | Oddzial | Suma |
|-------------|---------|-------|
| asystent | 323 | 1400 |
| asystent | null | 1400 |
| kierownik | 324 | 5400 |
| kierownik | null | 5400 |
| księgowy | 321 | 4200 |
| księgowy | null | 4200 |
| programista | 324 | 9800 |
| programista | null | 9800 |
| sekretarka | 322 | 3200 |
| sekretarka | null | 3200 |
| null | null | 24000 |

```
SELECT Stanowisko, IdOd, SUM(Pensja) as Suma
FROM Pracownicy
GROUP BY GROUPING SETS ((Stanowisko, IdOd), Stanowisko, ());
```

() - podsumowanie całkowite

```
SELECT Stanowisko, IdOd, SUM(Pensja) as Suma
FROM Pracownicy
GROUP BY Stanowisko, IdOd
UNION ALL
SELECT Stanowisko, null, SUM(Pensja) as Suma
FROM Pracownicy
GROUP BY Stanowisko
UNION ALL
SELECT null, null, SUM(Pensja) as Suma
FROM Pracownicy
order by 1;
```


KOMPOZYCJA (ZŁOŻENIE) KOLUMN

- kompozycja kolumn stanowi kolekcje kolumn, które są traktowane jak jedność
ROLLUP (a, b, (c, d)) CUBE (a, (b, c), d)
- użycie złożenie kolumn z operatorami CUBE i ROLLUP powoduje pominięcie odpowiednich podsumowań cząstkowych

GROUP BY ROLLUP(a, b, c)

GROUP BY ROLLUP(a, (b, c))

jest równoważne

jest równoważne

GROUP BY a, b, c

GROUP BY a, b, c

UNION ALL

UNION ALL

GROUP BY a, b

GROUP BY a

UNION ALL

UNION ALL

GROUP BY a

GROUP BY ()

UNION ALL

GROUP BY ()

$n + 1$ - grupowań

KOMPOZYCJA KOLUMN

GROUP BY CUBE(a, b, c)

GROUP BY CUBE((a, b), c)

jest równoważne

jest równoważne

GROUP BY a, b, c

GROUP BY a, b, c

UNION ALL

UNION ALL

GROUP BY a, b

GROUP BY a, b

UNION ALL

UNION ALL

GROUP BY b, c

GROUP BY c

UNION ALL

UNION ALL

GROUP BY a, c

GROUP BY ()

UNION ALL

GROUP BY a

UNION ALL

GROUP BY b

UNION ALL

GROUP BY c

UNION ALL

GROUP BY ()

2^n - grupowań

GROUPING SETS A UNION ALL

GROUP BY GROUPING SETS(a,b,c)

GROUP BY a UNION ALL
GROUP BY b UNION ALL
GROUP BY c

GROUP BY GROUPING SETS(a,b,(b,c))

GROUP BY a UNION ALL
GROUP BY b UNION ALL
GROUP BY b,c

GROUP BY GROUPING SETS((a,b,c))

GROUP BY a, b,c

GROUP BY GROUPING SETS(a, ())

GROUP BY a UNION ALL
GROUP BY ()

GROUP BY GROUPING SETS(a, ROLLUP(b,c))

GROUP BY a UNION ALL
GROUP BY ROLLUP(b,c)

KILKUKROTNE UŻYCIE KOLUMNY W GROUP BY

W GROUP BY można kilkukrotnie wykorzystać tę samą kolumnę (można zmienić organizację danych lub uzyskać różne rodzaje grup).

| Pracownicy | | | | | | | |
|------------|---------|----------|--------|--------|-------------|------------|------|
| IdPrac | Imie | Nazwisko | Pensja | Premia | Stanowisko | DataZatr | IdOd |
| 123 | Jan | Kowalski | 4200 | | księgowy | 2001/11/03 | 321 |
| 124 | Marek | Kowalski | 2400 | 4800 | programista | 2008/12/04 | 324 |
| 144 | Janina | Nowak | 3200 | | sekretarka | 2005/02/12 | 322 |
| 145 | Halina | Abacka | 5400 | | kierownik | 2011/08/14 | 324 |
| 146 | Zenon | Warecki | 3200 | 2400 | programista | 2008/08/04 | 324 |
| 147 | Marta | Kos | 1400 | | asystent | 2011/08/14 | 323 |
| 149 | Mariusz | Perkob | 4200 | 1200 | programista | 2010/07/23 | 324 |

```
SELECT Stanowisko, IdOd, SUM(Pensja) as Suma
FROM Pracownicy
GROUP BY Stanowisko, ROLLUP (Stanowisko, IdOd);
```

| Stanowisko | IdOd | Suma |
|-------------|------|------|
| programista | null | 9800 |
| sekretarka | null | 3200 |
| kierownik | null | 5400 |
| księgowy | null | 4200 |
| asystent | null | 1400 |
| programista | null | 9800 |
| sekretarka | null | 3200 |
| kierownik | null | 5400 |
| księgowy | null | 4200 |
| asystent | null | 1400 |
| asystent | 323 | 1400 |
| programista | 324 | 9800 |
| kierownik | 324 | 5400 |
| księgowy | 321 | 4200 |
| sekretarka | 322 | 3200 |

FUNKCJA GROUP_ID

Funkcja GROUP_ID - funkcja bezargumentowa, zwraca liczby z zakresu $0, \dots, n - 1$, gdy dla określonego grupowania występuje n duplikatów.

Pracownicy

| IdPrac | Imie | Nazwisko | Pensja | Premia | Stanowisko | DataZatr | IdOd |
|--------|---------|----------|--------|--------|-------------|------------|------|
| 123 | Jan | Kowalski | 4200 | | księgowy | 2001/11/03 | 321 |
| 124 | Marek | Kowalski | 2400 | 4800 | programista | 2008/12/04 | 324 |
| 144 | Janina | Nowak | 3200 | | sekretarka | 2005/02/12 | 322 |
| 145 | Halina | Abacka | 5400 | | kierownik | 2011/08/14 | 324 |
| 146 | Zenon | Warecki | 3200 | 2400 | programista | 2008/08/04 | 324 |
| 147 | Marta | Kos | 1400 | | asystent | 2011/08/14 | 323 |
| 149 | Mariusz | Perkob | 4200 | 1200 | programista | 2010/07/23 | 324 |

```
SELECT GROUP_ID(), Stanowisko,
       IdOd, SUM(Pensja) as Suma
FROM Pracownicy
GROUP BY Stanowisko,
ROLLUP (Stanowisko, IdOd);
```

Funkcję GROUP_ID można wykorzystać do usunięcia duplikatów zwróconych przez GROUP BY.

| GROUP_ID() | Stanowisko | IdOd | Suma |
|------------|-------------|------|------|
| 0 | asystent | null | 1400 |
| 0 | programista | null | 9800 |
| 0 | sekretarka | null | 3200 |
| 0 | kierownik | null | 5400 |
| 0 | księgowy | null | 4200 |
| 1 | asystent | null | 1400 |
| 1 | programista | null | 9800 |
| 1 | sekretarka | null | 3200 |
| 1 | kierownik | null | 5400 |
| 1 | księgowy | null | 4200 |
| 0 | asystent | 323 | 1400 |
| 0 | programista | 324 | 9800 |
| 0 | kierownik | 324 | 5400 |
| 0 | księgowy | 321 | 4200 |
| 0 | sekretarka | 322 | 3200 |

FUNKCJA GROUP_ID

Pracownicy

| IdPrac | Imie | Nazwisko | Pensja | Premia | Stanowisko | DataZatr | IdOd |
|--------|---------|----------|--------|--------|-------------|------------|------|
| 123 | Jan | Kowalski | 4200 | | księgowy | 2001/11/03 | 321 |
| 124 | Marek | Kowalski | 2400 | 4800 | programista | 2008/12/04 | 324 |
| 144 | Janina | Nowak | 3200 | | sekretarka | 2005/02/12 | 322 |
| 145 | Halina | Abacka | 5400 | | kierownik | 2011/08/14 | 324 |
| 146 | Zenon | Warecki | 3200 | 2400 | programista | 2008/08/04 | 324 |
| 147 | Marta | Kos | 1400 | | asystent | 2011/08/14 | 323 |
| 149 | Mariusz | Perkob | 4200 | 1200 | programista | 2010/07/23 | 324 |

```

SELECT GROUP_ID(), Stanowisko,
       IdOd, SUM(Pensja) as Suma
FROM Pracownicy
GROUP BY Stanowisko,
ROLLUP (Stanowisko, IdOd)
HAVING GROUP_ID()<1;

```

| GROUP_ID() | Stanowisko | IdOd | Suma |
|------------|-------------|------|------|
| 0 | programista | null | 9800 |
| 0 | sekretarka | null | 3200 |
| 0 | kierownik | null | 5400 |
| 0 | księgowy | null | 4200 |
| 0 | asystent | null | 1400 |
| 0 | asystent | 323 | 1400 |
| 0 | programista | 324 | 9800 |
| 0 | kierownik | 324 | 5400 |
| 0 | księgowy | 321 | 4200 |
| 0 | sekretarka | 322 | 3200 |

FUNKCJA GROUPING_ID

- zastępuje wielokrotne użycie funkcji GROUPING dla każdej kolumny (konkatenacja),
- dla zestawu kolumn generowana jest postać binarna na podstawie poszczególnych wyników funkcji GROUPING (np. $111_2 = 7_{10}$)
- przykład: w wierszu mamy $\text{GROUPING}(a) = 1$, $\text{GROUPING}(b) = 0$, $\text{GROUPING}(c) = 1$ wtedy $\text{GROUPING_ID}(a, b, c) = 5$ - na podstawie wartości binarnej $101_2 = 5_{10}$

FUNKCJA GROUPING_ID

Pracownicy

| IdPrac | Imie | Nazwisko | Pensja | Premia | Stanowisko | DataZatr | IdOd |
|--------|---------|----------|--------|--------|-------------|------------|------|
| 123 | Jan | Kowalski | 4200 | | księgowy | 2001/11/03 | 321 |
| 124 | Marek | Kowalski | 2400 | 4800 | programista | 2008/12/04 | 324 |
| 144 | Janina | Nowak | 3200 | | sekretarka | 2005/02/12 | 322 |
| 145 | Halina | Abacka | 5400 | | kierownik | 2011/08/14 | 324 |
| 146 | Zenon | Warecki | 3200 | 2400 | programista | 2008/08/04 | 324 |
| 147 | Marta | Kos | 1400 | | asystent | 2011/08/14 | 323 |
| 149 | Mariusz | Perkob | 4200 | 1200 | programista | 2010/07/23 | 324 |

```

SELECT GROUPING(Stanowisko) GR_Stanowisko, GROUPING(IdOd) GR_IdOd,
        GROUPING_ID(Stanowisko, IdOd) GROUPING_ID, Stanowisko, IdOd, SUM(Pensja)
FROM Pracownicy
GROUP BY GROUPING SETS ((Stanowisko, IdOd ), Stanowisko, ());

```

| GR_Stanowisko | GR_IdOd | GROUPING_ID | Stanowisko | IdOd | Suma |
|---------------|---------|-------------|-------------|------|-------|
| 0 | 0 | 0 | programista | 324 | 9800 |
| ... | | | | | |
| 0 | 1 | 1 | asystent | null | 1400 |
| 0 | 1 | 1 | sekretarka | null | 3200 |
| ... | | | | | |
| 1 | 1 | 3 | null | null | 24000 |

ZAAWANSOWANE SYSTEMY BAZ DANYCH

WYKŁAD

Joanna Kapusta
e-mail: joanna.kapusta@kul.pl

Katolicki Uniwersytet Lubelski Jana Pawła II

FUNKCJE ANALITYCZNE

- wyznaczając wynik dla bieżącego wiersza zwykle wykorzystując informacje pochodzące z sąsiednich wiersz
- wiersze lub grupy odrzucone za pomocą klauzul WHERE lub HAVING nie są uwzględniane
- nie mogą być używane w klauzulach WHERE, GROUP BY, HAVING
- wykorzystywane są w klauzuli SELECT lub ORDER BY

FUNKCJE ANALITYCZNE

Etapy wykonania zapytania

- 1 złączenia, selekcja wierszy, grupowanie, selekcja grup
- 2 podział na partycje i zastosowanie funkcji analitycznej do każdego wiersza
- 3 sortowanie wyników

FUNKCJE ANALITYCZNE - PODZIAŁ

- funkcje rankingu – wyznaczają rankingi wierszy i zbiorów wierszy (tzw. n-tek)
- funkcje rankingu hipotetycznego – wyznaczają hipotetyczny ranking dla zadanych wartości (zbiór uporządkowany)
- funkcje okna – wyznaczają wartości agregatów dla zbiorów wierszy wyznaczanych przy użyciu definicji okna
- funkcje raportujące – wyznaczają wartości agregatów dla zbiorów wierszy w ramach tzw. partycji
- funkcja WIDTH_BUCKET – dzieli uporządkowany zbiór na określoną liczbę przedziałów
- funkcje LAG/LEAD – znajdują wartości określonych atrybutów w wierszach sąsiednich
- ...

TERMINOLOGIA

- bieżący rekord – rekord, dla którego wyliczana jest wartość funkcji analitycznej
- partycja – autonomiczna grupa rekordów w ramach której funkcja analityczna przetwarza dane
- okno - pozwala na zdefiniowanie ruchomego zakresu, definiowanego oddzielnie dla bieżącego wiersza partycji; określa, ile rekordów „przed” (początek okna) i ile „za” (koniec okna) bieżącym wierszem jest brane pod uwagę przy obliczeniach; okno może przesuwać się wraz ze zmianą bieżącego wiersza albo może być jedno okno dla całej partycji

FUNKCJE ANALITYCZNE

Ogólna składnia

NAZWA_FUNKCJI (PARAMETRY) OVER (DEFINICJA ZBIORU)

gdzie definicja zbioru to:

- definicja partycji (opcjonalne) - określa podział wierszy na partycje,
- określenie porządku sortowania wierszy w partycji (zależy od typu funkcji)
- definicja okna (tylko dla funkcji okna)

Przykłady:

RANK() OVER (PARTITION BY department_id ORDER BY salary)

AVG(salary) OVER (PARTITION BY department_id
ORDER BY salary ROWS UNBOUNDED PRECEDING)

PODZIAŁ NA PARTYCJE

Partycjonowanie określa podział zbioru rekordów na oddzielne grupy (partycje). Podział jest realizowany na podstawie wartości jednego lub wielu wyrażeń kolumnowych.

```
PARTITION BY <lista wyrażeń kolumnowych>
```

Przykłady:

```
PARTITION BY department_id
```

```
PARTITION BY department_name, job_title
```

```
PARTITION BY job_id, extract (year from hire_date)
```

PORZĄDEK WIERSZY W PARTYCJACH

- ustalenie porządku dla większości funkcji analitycznych jest obowiązkowe (wyjątek stanowią funkcje raportujące)
- podobnie jak w przypadku partycjonowania, porządkowanie może odbyć się w oparciu o jedno lub wiele wyrażeń
- klauzule `NULLS FIRST` i `NULLS LAST` pozwalają na wskazanie miejsca dla wierszy z pustymi wartościami (pominięcie tych klauzul powoduje, że wartości puste są traktowane jak największe)

```
ORDER BY <wyrażenie kolumnowe>[ASC|DESC]  
      [NULLS FIRST|NULLS LAST] [, ...]
```

Przykłady:

```
ORDER BY hire_date NULLS FIRST
```

```
ORDER BY extract (year from hire_date)
```

```
ORDER BY department_id desc, job_id
```


FUNKCJE RANKINGU (1)

- funkcje rankingu wyznaczają pozycję danego wiersza porównując go z wartościami innych wierszy w tej samej partycji (pozwalają na ustalenie rankingu w grupie)
- RANK i DENSE_RANK- wyznaczają klasyfikację w grupie (z przerwą w numeracji/ bez przerwy w przypadku dwóch lub większej liczby wierszy znajdujących się na tej samej pozycji)
- CUME_DIST i PERCENT_RANK – wyznaczają procentową rangę wartości w grupie (z uwzględnieniem/bez uwzględnienia bieżącego rekordu)
- ROW_NUMBER - wyznacza numer wiersza w grupie
- NTILE – podział partycji na grupy

FUNKCJE RANKINGU (2)

RANK i DENSE_RANK

- pozwalają na ustalenie rankingu w grupie
- funkcja RANK w przypadku dwóch lub większej liczby wierszy znajdujących się na tej samej pozycji pozostawia przerwy w numeracji, funkcja DENSE_RANK przerw nie zostawia

Wyznacz ranking wynagrodzeń pracowników w oddziale o identyfikatorze 50.

```
SELECT last_name, first_name, salary,
       RANK() OVER (ORDER BY salary) as rank,
       DENSE_RANK() OVER (ORDER BY salary) as dense_rank
FROM employees WHERE department_id = 50;
```

| LAST_NAME | FIRST_NAME | SALARY | RANK | DENSE_RANK |
|------------|------------|--------|------|------------|
| Olson | TJ | 2100 | 1 | 1 |
| Philtanker | Hazel | 2200 | 2 | 2 |
| Markle | Steven | 2200 | 2 | 2 |
| Gee | Ki | 2400 | 4 | 3 |
| Landry | James | 2400 | 4 | 3 |

FUNKCJE RANKINGU (3)

CUME_DIST i PERCENT_RANK

- pokazują procentowy ranking dla bieżącego rekordu (wartość ≤ 1)
- CUME_DIST(x) - ile procent rekordów w partycji poprzedza lub jest równe wartości z bieżącego rekordu (uwzględnia bieżący rekord)
- PERCENT_RANK(x) - ile procent rekordów w partycji poprzedza bieżący rekord

FUNKCJE RANKINGU (3)

CUME_DIST i PERCENT_RANK

Wyznacz procentowy ranking wynagrodzeń pracowników w oddziale o identyfikatorze 60 (ile procent rekordów poprzedza lub jest równe wartości wynagrodzenia z bieżącego rekordu, ile procent rekordów poprzedza wartość wynagrodzenia z bieżącego rekordu).

```
SELECT last_name, first_name, salary,
       CUME_DIST() OVER (ORDER BY salary) as cume_dist,
       PERCENT_RANK() OVER (ORDER BY salary) as percent_rank
from employees
where department_id = 60;
```

| LAST_NAME | FIRST_NAME | SALARY | CUME_DIST | PERCENT_RANK |
|-----------|------------|--------|-----------|--------------|
| Lorentz | Diana | 4620 | 0,2 | 0 |
| Austin | David | 5280 | 0,6 | 0,25 |
| Pataballa | Valli | 5280 | 0,6 | 0,25 |
| Ernst | Bruce | 6600 | 0,8 | 0,75 |
| Hunold | Alexander | 9900 | 1 | 1 |

FUNKCJE RANKINGU (4)

ROW_NUMBER i NTILE

- ROW_NUMBER- przypisuje każdemu rekordowi w partycji unikalny numer wynikający z jego porządku w partycji (od 1)
- NTILE(n)- dzieli wiersze w partycji na n grup; każdej grupie przypisuje numer wynikający z porządku grupy w partycji (liczba wierszy w grupie różni się maksymalnie o 1)
- funkcje niedeterministyczne (dla takich samych parametrów mogą zwracać różne wyniki)

FUNKCJE RANKINGU (5)

Wyznacz kolejne numery dla rekordów/grup w rankingu oddziałów, zbudowanym ze względu na sumę wynagrodzenia.

```
SELECT department_id, SUM(salary) AS suma,  
ROW_NUMBER() OVER (ORDER BY SUM(salary) DESC) AS row_num,  
NTILE(4) OVER (ORDER BY SUM(salary) DESC) AS ntile  
FROM employees  
GROUP BY department_id;
```

| DEPARTMENT_ID | SUMA | ROW_NUM | NTILE |
|---------------|--------|---------|-------|
| 80 | 304500 | 1 | 1 |
| 50 | 156400 | 2 | 1 |
| 90 | 58000 | 3 | 1 |
| 100 | 51608 | 4 | 2 |
| 60 | 31680 | 5 | 2 |

FUNKCJE RANKINGU (6)

```
SELECT last_name, first_name, department_id, salary,
RANK() OVER (PARTITION BY department_id ORDER BY salary) as rank,
DENSE_RANK() OVER (PARTITION BY department_id ORDER BY salary) as dense_rank,
ROW_NUMBER() OVER (PARTITION BY department_id ORDER BY salary) as row_number
from employees;
```

| LAST_NAME | FIRST_NAME | DEPARTMENT_ID | SALARY | RANK | DENSE_RANK | ROW_NUMBER |
|------------|------------|---------------|--------|------|------------|------------|
| Olson | TJ | 50 | 2100 | 1 | 1 | 1 |
| Philtanker | Hazel | 50 | 2200 | 2 | 2 | 2 |
| Markle | Steven | 50 | 2200 | 2 | 2 | 3 |
| Gee | Ki | 50 | 2400 | 4 | 3 | 4 |
| Landry | James | 50 | 2400 | 4 | 3 | 5 |
| Patel | Joshua | 50 | 2500 | 6 | 4 | 6 |

FUNKCJE RANKINGU HIPOTETYCZNEGO (1)

- wyznaczają hipotetyczny ranking wartości (co by było gdyby)
- RANK, DENSE_RANK, PERCENT_RANK, CUME_DIST -
argumentem jest wartość, dla której poszukujemy pozycji
w rankingu

`nazwa_funkcji(wyrażenie)`

`WITHIN GROUP (porządek sortowania)`

Wyznaczenie rankingu hipotetycznego dla pracownika o zarobkach 5000.

```
select RANK(5000) WITHIN GROUP (order by salary) wynagr  
from employees;
```


FUNKCJE RANKINGU HIPOTETYCZNEGO (2)

Na którym miejscu w rankingu oddziałów znalazłby się oddział, dla którego suma wynagrodzeń wynosi 10 000? Podaj ile procent oddziałów poprzedzałoby sumę wynagrodzeń równą 10 000.

```
SELECT  RANK(10000)
        WITHIN GROUP (ORDER BY SUM(salary))pozycja,
        PERCENT_RANK(10000)
        WITHIN GROUP (ORDER BY SUM(salary)) procentowo
FROM employees GROUP BY department_id;
```

| POZYCJA | PROCENTOWO |
|---------|------------|
| 3 | 0,181818 |

FUNKCJE OKNA

Wyliczają wartość dla bieżącego rekordu, biorąc pod uwagę wartości innych rekordów należących do tego samego okna co bieżący rekord.

Typ okna:

- okno fizyczne - ROWS – wyrażone w liczbie wierszy
- okno logiczne – RANGE - wyrażone przy użyciu wartości o typie zgodnym z atrybutem porządkującym wiersze w partycji

DEFINIOWANIE OKNA

- BETWEEN ... AND ... – jawnie zdefiniowany początek i koniec okna
- UNBOUNDED PRECEDING – początkiem okna będzie pierwszy wiersz w partycji
- UNBOUNDED FOLLOWING – końcem okna będzie końcowy wiersz w partycji
- CURRENT ROW – bieżący wiersz lub wartość atrybutu w bieżącym wierszu (w zależności od typu okna)
- FOLLOWING i PRECEDING - dla typu okna ROWS – przesunięcie wyrażone w liczbie rekordów od bieżącego rekordu do początku/końca okna; dla RANGE – logiczne przesunięcie zależne od wyrażenia wykorzystanego do uporządkowania wierszy w partycji

PRZYKŁADY DEFINICJI OKNA

- okno obejmujące 3 wiersze: bieżący i po jednym przed i po bieżącym wierszu
ROWS BETWEEN 1 PRECEDING AND 1 FOLLOWING
- okno obejmujące 2 wiersze przed bieżącym wierszem i kończące się na bieżącym wierszu
ROWS BETWEEN 2 PRECEDING AND CURRENT ROW
- okno obejmujące dwa "przeszłe"i trzy "przyszłe"miesiące
RANGE INTERVAL '2' MONTH PRECEDING
AND INTERVAL '3' MONTH FOLLOWING

Pominięcie definicji okna jest równoznaczne z wyrażeniem:
RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW

FUNKCJE OKNA - PRZYKŁAD

```

SELECT salary, hire_date,
SUM(salary) OVER (ORDER BY hire_date) suma,
TRUNC(AVG(salary) OVER (ORDER BY hire_date RANGE BETWEEN interval '6'
month PRECEDING AND interval '6' month FOLLOWING)) srednia,
MIN(salary) OVER (ORDER BY hire_date rows 3 PRECEDING) min3
FROM employees
WHERE department_id = 50
ORDER BY hire_date;

```

| ⚡ SALARY | ⚡ HIRE_DATE | ⚡ SUMA | ⚡ SREDNIA | ⚡ MIN3 |
|----------|-------------|--------|-----------|--------|
| 7900 | 03/05/01 | 7900 | 5000 | 7900 |
| 3600 | 03/07/14 | 11500 | 5000 | 3600 |
| 3500 | 03/10/17 | 15000 | 4640 | 3500 |
| 4200 | 04/01/27 | 19200 | 4600 | 3500 |
| 4000 | 04/02/04 | 23200 | 4600 | 3500 |

FUNKCJE RAPORTUJĄCE (1)

- pozwalają na wyznaczenie wartości funkcji agregujących w oparciu o zbiór wierszy uzyskanych w wyniku zapytania (tak jak funkcje okna)
- nie wprowadzają one definicji okna, ani porządku wierszy w partycji.
- wynik funkcji agregującej jest wyznaczany dla całej partycji i jest prezentowany dla każdego rekordu partycji (dla pojedynczego wiersza mamy dostęp do agregacji wyznaczonych w grupie)
- funkcje: MAX, MIN, AVG, SUM, COUNT, STDDEV, VARIANCE, RATIO_TO_REPORT

`nazwa_funkcji(wyrażenie1)`

`OVER (PARTITION BY wyrażenie2 - opcjonalnie)`

W przypadku braku definicji partycji wartość funkcji jest wyliczana na podstawie całego zbioru.

FUNKCJE RAPORTUJĄCE (2)

```
SELECT first_name||' '||last_name AS pracownik,
       job_id AS stanowisko,
       SUM(salary) AS wynagrodzenie,
       COUNT(job_id) OVER (PARTITION BY job_id) as ile,
       SUM(SUM(salary)) OVER (PARTITION BY job_id) AS suma_wynag_stan
FROM employees
GROUP BY first_name||' '||last_name, job_id
order by job_id;
```

```
SELECT first_name||' '||last_name AS pracownik,
       job_id AS stanowisko,
       SUM(salary) OVER (PARTITION BY job_id, last_name, first_name) AS wynagrodzenie,
       COUNT(job_id) OVER (PARTITION BY job_id) as ile,
       SUM(salary) OVER (PARTITION BY job_id) AS suma_wynag_stan
FROM employees;
```

| PRACOWNIK | STANOWISKO | WYNAGRODZENIE | ILE | SUMA_WYNAG_STAN |
|-------------------|------------|---------------|-----|-----------------|
| Steven King | AD PRES | 24000 | 1 | 24000 |
| Neena Kochhar | AD VP | 17000 | 2 | 34000 |
| Lex De Haan | AD VP | 17000 | 2 | 34000 |
| John Chen | FI ACCOUNT | 8200 | 5 | 39600 |
| Daniel Faviet | FI ACCOUNT | 9000 | 5 | 39600 |
| Ismael Sciarra | FI ACCOUNT | 7700 | 5 | 39600 |
| Jose Manuel Urman | FI ACCOUNT | 7800 | 5 | 39600 |

FUNKCJE RAPORTUJĄCE (3)

RATIO_TO_REPORT - wylicza stosunek wartości wyrażenia do sumy wartości wyrażenia ze wszystkich rekordów partycji $\frac{x}{SUM(x)}$ (udział)

```
SELECT first_name||' '||last_name AS pracownik,
       job_id AS stanowisko,
       SUM(salary) OVER (PARTITION BY job_id, first_name, last_name)
         AS wynagrodzenie,
       COUNT(job_id) OVER (PARTITION BY job_id) as ile,
       SUM(salary) OVER (PARTITION BY job_id) AS suma_wynag_stan,
       RATIO_TO_REPORT(salary) OVER (PARTITION BY job_id)
         AS udzial FROM employees;
```

| PRACOWNIK | STANOWISKO | WYNAGRODZENIE | ILE | SUMA_WYNAG_STAN | UDZIAL |
|---------------|------------|---------------|-----|-----------------|--------|
| William Gietz | AC ACCOUNT | 8300 | 1 | 8300 | 1 |
| Steven King | AD PRES | 24000 | 1 | 24000 | 1 |
| Neena Kochhar | AD VP | 17000 | 2 | 34000 | 0,5 |
| Lex De Haan | AD VP | 17000 | 2 | 34000 | 0,5 |

FUNKCJA WIDTH BUCKET

- dzieli uporządkowany zbiór wynikowy na n podzbiorów
- zbiór wynikowy zawiera rekordy z zadanego przedziału
- liczba rekordów w podzbiorach może się znacząco różnić (NTILE - różnica maksymalnie o 1)

```
select last_name, salary,  
WIDTH_BUCKET(salary, 10000, 20000, 4) as W_BUCKET  
from employees  
order by salary;
```

przedział $< 10000, 20000$) jest dzielony na 4: $< 10000, 12500$), $< 12500, 15000$), $< 15000, 17500$), $< 17500, 20000$) (< 10000 przypisuje 0, ≥ 20000 przypisuje 5)

| LAST_NAME | SALARY | W_BUCKET |
|-----------|--------|----------|
| Russell | 14000 | 2 |
| De Haan | 17000 | 3 |
| Kochhar | 17000 | 3 |
| King | 24000 | 5 |

FUNKCJE LAG I LEAD

- LAG - umożliwia dostęp do wartości atrybutów rekordów poprzedzających dany rekord
- LEAD - umożliwia dostęp do wartości atrybutów rekordów następujących po danym rekordzie
- argument wywołania funkcji określa przesunięcie w tył/przód względem bieżącego rekordu

FUNKCJE LAG I LEAD

```
SELECT extract(year from hire_date) rok, SUM(salary) zarobki,
LAG(SUM(salary),1) OVER (ORDER BY extract(year from hire_date))
zarobki_poprzedni_rok,
LEAD(SUM(salary),1) OVER (ORDER BY extract(year from hire_date))
zarobki_nastepny_rok
FROM employees
GROUP BY extract(year from hire_date);
```

porównuje sumy zarobków pracowników zatrudnionych w kolejnych latach

| ROK | ZAROBKI | ZAROBKI_POPRZEDNI_ROK | ZAROBKI_NASTEPNY_ROK |
|------|---------|-----------------------|----------------------|
| 2001 | 17000 | (null) | 68816 |
| 2002 | 68816 | 17000 | 46500 |
| 2003 | 46500 | 68816 | 86000 |
| 2004 | 86000 | 46500 | 198380 |

ZAAWANSOWANE SYSTEMY BAZ DANYCH

WYKŁAD

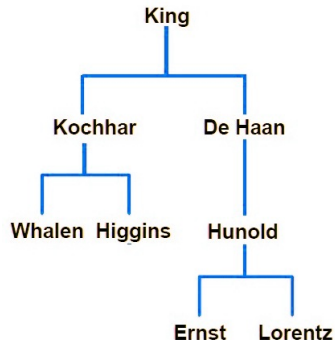
Joanna Kapusta
e-mail: joanna.kapusta@kul.pl

Katolicki Uniwersytet Lubelski Jana Pawła II

ZAPYTANIA HIERARCHICZNE

- pozwala wybierać wiersze z relacji w porządku hierarchicznym,
- wiersze powiązane ze sobą rekursywnym związkiem typu „jeden do wiele”,

| EMPLOYEE_ID | FIRST_NAME | LAST_NAME | JOB_ID | MANAGER_ID |
|-------------|------------|-----------|---------|------------|
| 102 | Lex | De Haan | AD_VP | 100 |
| 104 | Bruce | Ernst | IT_PROG | 103 |
| 205 | Shelley | Higgins | AC_MGR | 101 |
| 103 | Alexander | Hunold | IT_PROG | 102 |
| 156 | Janette | King | SA_REP | 146 |
| 100 | Steven | King | AD_PRES | (null) |
| 107 | Diana | Lorentz | IT_PROG | 103 |
| 200 | Jennifer | Whalen | AD_ASST | 101 |



ZAPYTANIA HIERARCHICZNE

```
SELECT [LEVEL], ...  
FROM ...  
WHERE ...  
CONNECT BY PRIOR ...  
START WITH ...
```

- CONNECT BY zawiera warunek określający związek rodzic-potomek w drzewie
- PRIOR odnosi się do nadrzędnego wiersza (rodzica)
- START WITH określa warunek selekcyjny wiersz, od którego rozpocznie się proces konstrukcji drzewa
- LEVEL - pseudokolumna, której wartość określa poziom zagnieżdżenia poszczególnych węzłów drzewa

ZAPYTANIA HIERARCHICZNE

Wyświetlenie pracownika o identyfikatorze 102 i jego podwładnych

```
SELECT level, employee_id, first_name, last_name, job_id, manager_id
FROM employees
CONNECT BY PRIOR employee_id = manager_id
START WITH employee_id=102;
```

| LEVEL | EMPLOYEE_ID | FIRST_NAME | LAST_NAME | JOB_ID | MANAGER_ID |
|-------|-------------|------------|-----------|---------|------------|
| 1 | 102 | Lex | De Haan | AD_VP | 100 |
| 2 | 103 | Alexander | Hunold | IT_PROG | 102 |
| 3 | 104 | Bruce | Ernst | IT_PROG | 103 |
| 3 | 105 | David | Austin | IT_PROG | 103 |
| 3 | 106 | Valli | Pataballa | IT_PROG | 103 |
| 3 | 107 | Diana | Lorentz | IT_PROG | 103 |

Wyświetlenie pracownika o identyfikatorze 102 i jego zwierzchników

```
SELECT level, employee_id, first_name, last_name, job_id, manager_id
FROM employees
CONNECT BY PRIOR manager_id = employee_id
START WITH employee_id=102;
```

| LEVEL | EMPLOYEE_ID | FIRST_NAME | LAST_NAME | JOB_ID | MANAGER_ID |
|-------|-------------|------------|-----------|---------|------------|
| 1 | 102 | Lex | De Haan | AD_VP | 100 |
| 2 | 100 | Steven | King | AD_PRES | (null) |

ZAPYTANIA HIERARCHICZNE

Wyświetlenie podwładnych Alexandara Hunnolda

```
SELECT level, employee_id, first_name, last_name
FROM employees
CONNECT BY PRIOR employee_id = manager_id
START WITH employee_id = (SELECT employee_id
FROM employees
WHERE first_name = 'Alexander'
AND last_name = 'Hunold');
```


ZAPYTANIA HIERARCHICZNE

Wyświetlenie pracowników na kolejnych poziomach drzewa

```
SELECT level, employee_id, first_name, last_name
FROM employees
CONNECT BY PRIOR employee_id = manager_id
START WITH employee_id = 100
ORDER BY LEVEL;
```

Wyświetlenie liczby poziomów w drzewie

```
SELECT COUNT(DISTINCT LEVEL)
FROM employees
CONNECT BY PRIOR employee_id = manager_id
START WITH employee_id = 100;
```

SYS_CONNECT_BY_PATH

SYS_CONNECT_BY_PATH

- pozwala na wyświetlenie wszystkich danych od wierzchołka do aktualnego poziomu (ścieżka),
- funkcja dwuparametrowa - pierwszy parametr wskazuje na dane które będą wyświetlane, drugi określa separator.

```
SELECT LEVEL, employee_id, last_name, sys_connect_by_path(last_name, '/')
FROM employees
START WITH employee_id = 100
CONNECT BY manager_id = PRIOR employee_id;
```

| LEVEL | EMPLOYEE_ID | LAST_NAME | SYS_CONNECT_BY_PATH(LAST_NAME, '/') |
|-------|-------------|-----------|-------------------------------------|
| 1 | 100 | King | /King |
| 2 | 101 | Kochhar | /King/Kochhar |
| 3 | 108 | Greenberg | /King/Kochhar/Greenberg |
| 4 | 109 | Faviet | /King/Kochhar/Greenberg/Faviet |
| 4 | 110 | Chen | /King/Kochhar/Greenberg/Chen |
| 4 | 111 | Sciarra | /King/Kochhar/Greenberg/Sciarra |
| 4 | 112 | Urman | /King/Kochhar/Greenberg/Urman |
| 4 | 113 | Popp | /King/Kochhar/Greenberg/Popp |
| 3 | 200 | Whalen | /King/Kochhar/Whalen |
| 3 | 203 | Mavris | /King/Kochhar/Mavris |
| 3 | 204 | Baer | /King/Kochhar/Baer |
| 3 | 205 | Higgins | /King/Kochhar/Higgins |
| 4 | 206 | Serge | /King/Kochhar/Higgins/Serge |

CONNECT BY ROOT

CONNECT_BY_ROOT

- pozwala na wyświetlenie danych z węzła najwyższego poziomu (korzenia),

```
SELECT LEVEL, EMPLOYEE_ID, last_name, CONNECT_BY_ROOT last_name AS ANCESTOR
FROM EMPLOYEES
START WITH EMPLOYEE_ID = 100
CONNECT BY manager_id = PRIOR EMPLOYEE_ID;
```

| LEVEL | EMPLOYEE_ID | LAST_NAME | ANCESTOR |
|-------|-------------|-----------|----------|
| 1 | 100 | King | King |
| 2 | 101 | Kochhar | King |
| 3 | 108 | Greenberg | King |
| 4 | 109 | Faviet | King |
| 4 | 110 | Chen | King |
| 4 | 111 | Sciarra | King |
| 4 | 112 | Urman | King |

FORMATOWANIE WYNIKÓW ZAPYTANIA HIERARCHICZNEGO

```
SELECT level, lpad(' ', 2*(level-1))||' '||first_name||' '||last_name
        as Pracownik
FROM employees
CONNECT BY PRIOR employee_id = manager_id
START WITH employee_id=100;
```

| LEVEL | PRACOWNIK |
|-------|-------------------|
| 1 | Steven King |
| 2 | Neena Kochhar |
| 3 | Nancy Greenberg |
| 4 | Daniel Faviet |
| 4 | John Chen |
| 4 | Ismael Sciarra |
| 4 | Jose Manuel Urman |
| 4 | Luis Popp |
| 3 | Jennifer Whalen |
| 3 | Susan Mavris |
| 3 | Hermann Baer |
| 3 | Shelley Higgins |
| 4 | William Gietz |
| 2 | Lex De Haan |

ELIMINOWANIE WĘZŁÓW

```
SELECT level, lpad(' ', 2*(level-1)) || ' ' || first_name || ' ' || last_name
       as Pracownik
FROM employees
WHERE last_name != 'Greenberg'
CONNECT BY PRIOR employee_id = manager_id
START WITH employee_id=100;
```

| LEVEL | PRACOWNIK |
|-------|-------------------|
| 1 | Steven King |
| 2 | Neena Kochhar |
| 4 | Daniel Faviet |
| 4 | John Chen |
| 4 | Ismael Sciarra |
| 4 | Jose Manuel Urman |
| 4 | Luis Popp |
| 3 | Jennifer Whalen |
| 3 | Susan Mavris |
| 3 | Hermann Baer |
| 3 | Shelley Higgins |
| 4 | William Gietz |
| 2 | Lex De Haan |
| 3 | Alexander Hunold |

ELIMINOWANIE GAŁĘZI

```
SELECT level, lpad(' ', 2*(level-1))||' '||first_name||' '||last_name
       as Pracownik
FROM employees
CONNECT BY PRIOR employee_id = manager_id
AND last_name != 'Greenberg'
START WITH employee_id=100;
```

| LEVEL | PRACOWNIK |
|-------|------------------|
| 1 | Steven King |
| 2 | Neena Kochhar |
| 3 | Jennifer Whalen |
| 3 | Susan Mavris |
| 3 | Hermann Baer |
| 3 | Shelley Higgins |
| 4 | William Gietz |
| 2 | Lex De Haan |
| 3 | Alexander Hunold |
| 4 | Bruce Ernst |
| 4 | David Austin |
| 4 | Valli Pataballa |
| 4 | Diana Lorentz |
| 2 | Den Raphaely |

ZAAWANSOWANE SYSTEMY BAZ DANYCH

WYKŁAD

Joanna Kapusta
e-mail: joanna.kapusta@kul.pl

Katolicki Uniwersytet Lubelski Jana Pawła II

BUSINESS INTELLIGENCE

BUSINESS INTELLIGENCE

Technologia informatyczna służąca do przekształcania dużych wolumenów danych w informacje, a następnie do przekształcania tych informacji w wiedzę. BI wspomaga proces podejmowania decyzji w przedsiębiorstwie.

- Jądem systemu BI jest hurtownia danych.
- Ogromne wymagania wydajnościowe.

OLTP vs. OLAP

OLTP (ang. OnLine Transaction Processing) - systemy przetwarzania transakcyjnego

OLAP (ang. OnLine Analytical Processing) - systemy przetwarzania analitycznego

| Cecha | OLTP | OLAP |
|---------------------------------|--|--|
| funkcja | ułatwienie pracownikom codziennej pracy | wspomaganie procesu podejmowania decyzji |
| czas odpowiedzi aplikacji | ułamki sekund, sekundy | minuty, godziny |
| wykonywane operacje | DML | select |
| czasowy zakres danych | miesiące | lata |
| organizacja danych | według aplikacji (zoorientowany na działanie) | tematyczna (zoorientowany na temat) |
| źródło danych | wprowadzane przez końcowych użytkowników systemu | dane ładowane z różnych źródeł (m.in. OLTP) |
| rozmiar | małe, duże | duże, wielkie |
| intensywność operacji dyskowych | mała, średnia | wielka |
| schemat bazy danych | znormalizowany z dużą liczbą tabel | normalizacja nie jest wymagana, liczba tabel relatywnie mała |

HURTOWNIE DANYCH

DEFINICJA (R. KIMBALL)

Hurtownia danych jest to system, który pozyskuje dane z systemów źródłowych, przekształca je i ładuje do wielowymiarowych struktur, a następnie dostarcza zapytania i analizy wspierające podejmowanie decyzji.

- Hurtownie należy traktować jako kompleksowe środowisko złożone z wielu elementów (odrębny projekt, narzędzia, metodologia).
- Hurtowni nie należy utożsamiać z kopią danych transakcyjnych ani wielowymiarowym modelem danych.

HURTOWNIE DANYCH

DEFINICJA (W. H. INMON)

Hurtownia danych to problemowo zorientowany, zintegrowany i trwały zbiór danych opisany wymiarem czasu.

- problemowo zorientowany - wokół głównego tematu zainteresowań - głównych obszarów działalności, np.: klientów, produktów, sprzedaży
- zintegrowany - integracja wielu, często heterogenicznych, źródeł danych związanych przedmiotem zainteresowań
- trwały - dane pozostają niezmiennicze, nowe dane są dołączane, hurtownia ma charakter przyrostowy
- opisane wymiarem czasu - dane opisują zdarzenia historyczne, a nie tylko stan aktualny

CELE TWORZENIA HURTOWNI DANYCH

- Wykonywanie analiz biznesowych bez ingerencji w systemy transakcyjne (raporty, wykresy, zestawienia statystyczne, śledzenie trendów). Analizy biznesowe wymagają złożonych i czasochłonnych obliczeń istotne jest oddzielenie systemu OLAP od OLTP.
- Wspomaganie decyzji - odkrywanie wiedzy, znajdowanie (niewidocznych dla człowieka) prawidłowości w danych zgromadzonych w hurtowniach danych.
- Całościowe gromadzenie danych firmy - gromadzenie w jednym miejscu zintegrowanych danych pochodzących z różnych źródeł daje pełniejszy obraz zdarzeń zachodzących w całej instytucji.
- Dostęp do danych historycznych - gromadzenie danych z długiego okresu, z jednoczesnym rejestrowaniem chwili zajścia danego zdarzenia.
- Ujednolicenie posiadanych informacji - doprecyzowanie pojęć, jednakowe wyliczanie i interpretowanie wskaźników w całej instytucji.

TYPOWE ZASTOSOWANIA

- prognozowanie - analiza trendów i zachowań (np. analiza trendów sprzedaży, analiza nakładów reklamowych i zysków)
- wykrywanie oszustw - nieregularność w przyjętych wzorcach postępowania (karty kredytowe, ubezpieczenia, operacje finansowe)
- wybór celu kampanii marketingowej - lepsze efekty przynoszą kampanie ukierunkowane na konkretny cel (wiek, płeć, obszar zamieszkania, grupa dochodowa)
- analiza rentowności klientów - wzmacnianie relacji z klientami dochodowymi
- zapobieganie odejściu klienta - opracowanie modeli oceny ryzyka, pozwalające na zidentyfikowanie klientów, którzy mogą odejść
- zarządzanie zasobami - posiadanie właściwych towarów, we właściwym miejscu i czasie

BUSINESS INTELLIGENCE

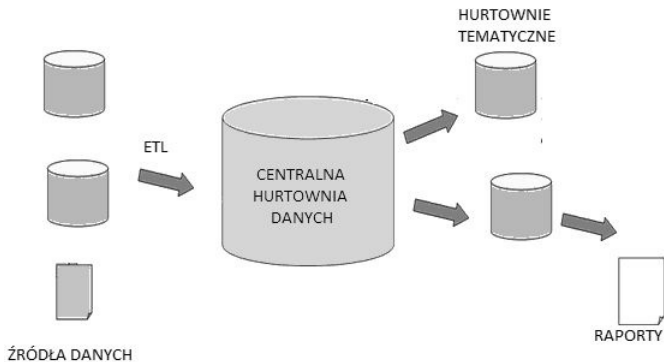
Implementacja systemu BI:

- analiza wymagań - zgromadzenie wiedzy o wymaganiach biznesowych w zakresie przetwarzania analitycznego
- projektowanie logiczne hurtowni danych - pojęciowa definicja wymaganych struktur danych
- implementacja struktur fizycznych hurtowni danych - tworzenie bazy danych, tabel, indeksów, perspektyw zmaterializowanych
- implementacja oprogramowania ETL - konstrukcja modułów programowych służących do zasilania hurtowni danych nowymi danymi
- implementacja aplikacji analitycznych - tworzenie programów dla użytkowników końcowych
- konserwacja hurtowni danych - dostrojenie serwera bazy danych (dodatkowe indeksy, perspektywy zmaterializowane)

ARCHITEKTURA HURTOWNI DANYCH

- Źródła danych - heterogeniczne i rozproszone.
- Warstwa ETL - zasila danymi hurtownie, tutaj odbywa się czyszczenie danych i ich transformacja do odpowiedniej postaci.
- Magazyn (centralna hurtownia danych) - na jej bazie powstają hurtownie tematyczne (ang. data marts). Zwykle hurtownia tematyczna (lokalna) zawiera wyselekcjonowane dane na wyższym poziomie agregacji niż hurtownia centralna i pozwala na sprawniejsze operowanie danymi.
- Aplikacje analityczne - analiza danych, trendów, anomalii, wyszukiwanie reguł zachowań

ARCHITEKTURA HURTOWNI DANYCH



ETL

Oprogramowanie ETL realizuje trzy fazy:

- Ekstrakcja (Extract) - odczyt danych ze źródeł
- Transformacja (Transform) - transformacja danych do wspólnego modelu wraz z usunięciem wszelkich niespójności i oznakowaniem czasowym
- Załadowanie (Load) - wczytanie danych do docelowej hurtowni danych.

PROJEKTOWANIE HURTOWNI DANYCH

- Model pojęciowy - to opis struktury, zawartości i przeznaczenia hurtowni danych z punktu widzenia celów biznesowych, przy użyciu nazw z języka naturalnego specjalistycznego, właściwego dla danej organizacji. Model pojęciowy może np. określić, jakie dane chcemy gromadzić, na jakie pytania chcemy poznać odpowiedzi, jaką postać mają mieć raporty wynikowe.
- Model logiczny - to opis odwołujący się do elementów logicznych baz danych i procesów hurtowni, w przypadku architektury relacyjnej określamy nazwy kolumn, tabel, relacji itp., dla architektury wielowymiarowej określamy postać kostek.
- Model fizyczny - to opis parametrów mających na celu optymalizację działania hurtowni danych, takich jak indeksy, partycje, perspektywy zmaterializowane, itp.

WIELOWYMIAROWY MODEL DANYCH

Modelem pojęciowym dla hurtowni danych jest wielowymiarowy model danych. W modelu wielowymiarowym analizujemy fakty wzdłuż wymiarów.

Podstawowe kategorie danych w modelu wielowymiarowym:

- fakty - reprezentują fakty podlegające analizie np. fakt sprzedaży towaru, fakt wykonania rozmowy telefonicznej, fakt wykonania usługi. Fakty mają charakter ilościowy i są określane za pomocą miar np. liczba zakupionych produktów, czas trwania rozmowy, zysk/koszt.
- wymiary - ustalają kontekst analizy, np. sprzedaż produktów w sieci hipermarketów w poszczególnych miesiącach jest dokonywana w wymiarach Produkt, Sklep, Czas, Klient. Wymiary składają się z poziomów, które tworzą hierarchię, np. Lokalizacja: Sklepy, Miasta i Województwa.

WIELOWYMIAROWY MODEL DANYCH

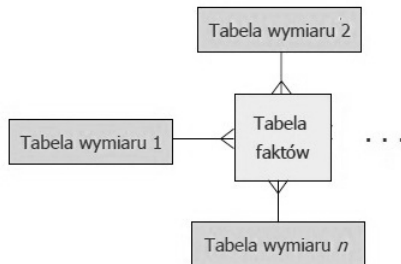
- ROLAP (relacyjny)
 - dane przechowywane w tabelach relacyjnych (przy czym schemat BD zaprojektowany jest tak aby odzwierciedlić wielowymiarową strukturę danych)
 - fakty przechowywane w tabelach faktów
 - wymiary przechowywane w tabelach wymiarów
- MOLAP (wielowymiarowy) - dane przechowywane w wielowymiarowych tabelach - kostkach
- HOLAP (hybrydowy - relacyjnowelowymiarowy)
 - dane elementarne przechowywane w tabelach
 - pozostałe dane przechowywane w kostkach

IMPLEMENTACJA ROLAP

Schematy logiczne:

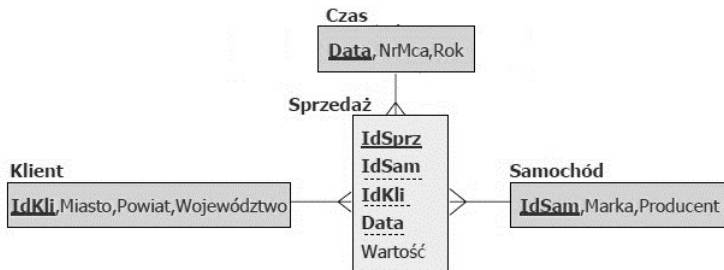
- schemat gwiazdy (ang. star schema)
- schemat płatka śniegu (ang. snowflake schema)
- konstelacja faktów - schemat gwiazdy lub płatka śniegu, w którym ten sam wymiar jest powiązany z wieloma tabelami faktów

SCHEMAT GWIAZDY



- centralna tabela faktów
- wielu tabel wymiarów; tabele wymiarów są w 1PN; kolumny mogą tworzyć hierarchię (jeśli wymiary spełniają przynajmniej 3 postać normalną wówczas mamy schemat płata śniegu)
- tabela faktów zawiera klucze obce (powiązane z tabelami wymiarów) oraz atrybuty zwane miarami

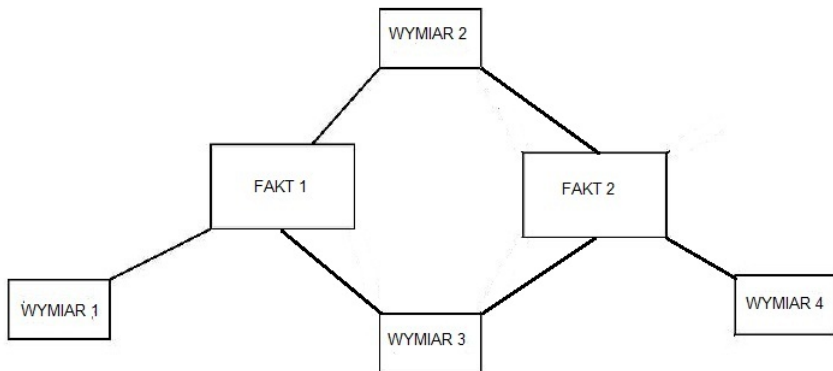
SCHEMAT GWIAZDY - PRZYKŁAD



Schemat płątka śniegu:

- tabela wymiaru Klient podzielona na tabele: Klient, Miasto, Powiat i Województwo,
- tabela wymiaru Samochód podzielona na tabele: Samochód, Marka, Producent

SCHEMAT KONSTELACJI FAKTÓW



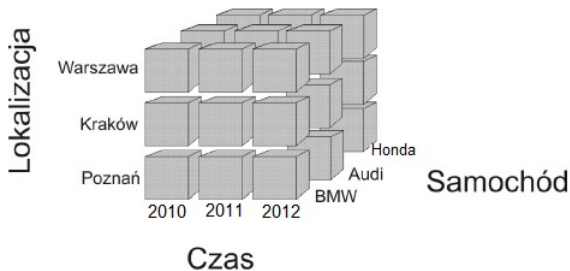
Schemat będący kombinacją schematów gwiazd, które współdzielą niektóre wymiary.

TABELE FAKTÓW I WYMIARÓW

- Tabela faktów:
 - zawiera dane numeryczne - miary
 - posiada wieloatrybutowy klucz główny złożony z kluczy obcych odwołujących się do tabel wymiarów
 - największa tabela - zwykle zawiera ponad 90% danych
 - szybko się powiększa.
- Tabela wymiarów:
 - zawiera atrybuty opisowe
 - nadaje znaczenie faktom (definiuje znaczenie faktów)
 - zawiera dane, które rzadko podlegają zmianom (np. pojawienie się nowych produktów)

IMPLEMENTACJA MOLAP

Dane są przechowywane w tablicach wielowymiarowych (kostkach), takie tablice zawierają wstępnie przetworzone dane pochodzące z wielu źródeł.



ZASILANIE HURTOWNI

- Hurtownia integruje dane z różnych źródeł - źródła danych zmieniają swoją zawartość
- Uaktualnianie zawartości hurtowni danych (aktualność danych) ma kluczowy wpływ na jakość wyników analiz, decyzje biznesowe.

Źródła danych (czas):

- produkcyjne - operacyjne bazy danych (Oracle, Sybase, ...), systemy plików, aplikacje,
- zarchiwizowane - dane historyczne, potrzebne do inicjalizacji hurtowni (mogą wymagać specjalnej transformacji),

Źródła danych (lokalizacja):

- wewnętrzne - wewnętrzne bazy danych, pliki, dokumenty, arkusze kalkulacyjne,
- zewnętrzne - komercyjne bazy danych, Internet - nieprzewidywalne (format, częstotliwość odświeżania).

ZASILANIE HURTOWNI

Wyróżnia się dwa rodzaje zasilania:

- pierwsze zasilanie pustej hurtowni,
- odświeżanie w trakcie eksploatacji - okresowo.

Sposoby odświeżania:

- pełne - ze źródła do hurtowni przesyłane są wszystkie dane
- przyrostowe - ze źródła do hurtowni przesyłane są tylko nowe dane lub dane zmodyfikowane od czasu ostatniego zasilenia

TRANSFORMACJA

Najtrudniejszy element ETL, zazwyczaj wymaga istnienia obszaru tymczasowego (ang. staging area), w którym następuje konsolidacja, czyszczenie, restrukturyzacja.

Anomalie:

- nazewnictwo i kodowanie
- semantyka danych
- literówki
- brak unikalnych kluczy podstawowych
- klucze złożone
- różne formaty danych wejściowych
- sprzeczne źródła danych
- brakujące bądź zduplikowane wartości
- brakujące więzy referencyjne

Techniki czyszczenia:

- eliminacja niespójności
- dodawanie i łączenie danych
- integracja danych

ŁADOWANIE DANYCH

Proces przesyłania danych z obszaru tymczasowego do docelowej hurtowni danych w czasie:

- inicjalizacji hurtowni - bardzo duża ilość danych,
- odświeżanie hurtowni - wykonywane cyklicznie - mniejsza ilość danych

Metody ładowania danych:

- specjalne narzędzia
- własne programy
- replikacja
- ręczne ładowanie

OLAP

Cel: dostarczanie informacji strategicznej i jej prezentacja zgodnie ze schematem poznawczym człowieka.

Typowe operacje OLAP:

- podsumowanie - roll up (drill-up):
 - przejście do wyższego poziomu w hierarchii lub redukcja wymiarów (sklep⇒miejscowość⇒województwo - *od szczegółu do ogółu*)
- rozwinięcie - roll down (drill-down):
 - przejście do niższego poziomu w hierarchii lub wprowadzanie nowych wymiarów - *od ogółu do szczegółu*
- rzut i selekcja (slice and dice) - wycinanie fragmentu danych poprzez określenie warunków na wartościach wymiarów
- zmiana orientacji kostki (pivot, rotate) - zmiana kolejności wymiarów

PRZYKŁAD - TWORZENIE HURTOWNI DANYCH DLA NIEWIELKIEGO BANKU

A.Chączyńska-Krasowska, E. Mrówka-Matejewska,
M.Jankowski-Lorek, Podstawy hurtowni danych, PJWSTK, 2013

WYMAGANIA BIZNESOWE (1)

Interesują nas odpowiedzi na pytania:

- Jak zmienia się w czasie liczba operacji wykonywanych na rachunkach w poszczególnych oddziałach banków?
- Jakie były kwoty operacji wykonywanych na rachunkach klientów w kolejnych miesiącach, kwartałach i latach w rozbiciu na wpłaty i wypłaty?
- Jakie były kwoty operacji wykonywanych na rachunkach klientów w kolejnych miesiącach, kwartałach i latach w rozbiciu na grupy wiekowe oraz na wpłaty i wypłaty?
- Jak przedstawia się rozkład korzystania z bankomatów (liczba transakcji, średnie pobierane kwoty) w zależności od dnia tygodnia i dnia miesiąca?
- Jak przedstawia się rozkład rodzajów wykonywanych operacji od grupy aktywności mierzonej średnimi ważonymi miesięcznym saldem klienta, średnią miesięczną kwotą wpłat oraz średnią miesięczną różnicą między wpłatami i wydatkami?
- Ilu mamy klientów w poszczególnych grupach aktywności?
- Jakie były przychody oddziałów banków z tytułu wprowadzenia opłat za prowadzenie kont w poszczególnych miesiącach?
- Jaki współczynnik klientów zakłada konto w swoim obszarze zamieszkania (województwo, miasto, dzielnica) i czy to się zmienia na przestrzeni czasu?

WYMAGANIA BIZNESOWE (2)

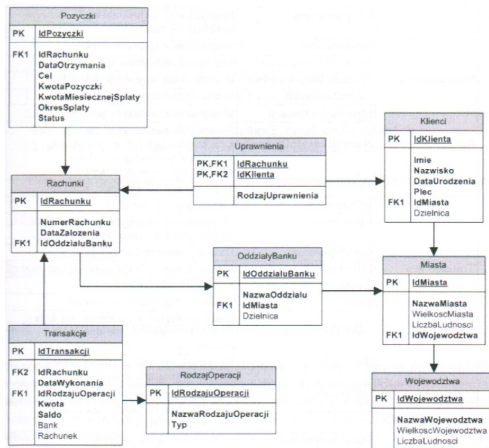
Raporty

- Raport umożliwiający analizę sumaryczną kwoty przeprowadzonych operacji bankowych (hierarchicznie - typ, rodzaj) w przygotowanych grupach wiekowych.
- Raport umożliwiający analizę liczby operacji bankowych z podziałem geograficznym w przygotowanych grupach wiekowych.
- Raport umożliwiający analizę średniej kwoty transakcji w rozbiciu na rodzaje transakcji i moment wykonania (w hierarchii kalendarzowej - rok, miesiąc, kwartał, data).
- Raport umożliwiający analizę zmian ilości i wartości operacji bankowych w poszczególnych województwach w zależności od miesiąca roku kalendarzowego.

Dodatkowo chcemy mieć możliwość obliczania różnic między wpłatami i wypłatami wzdłuż wybranych wymiarów i oglądania danych w angielskiej wersji językowej.

DOSTĘPNE ŹRÓDŁA

- baza transakcyjna o podanym schemacie

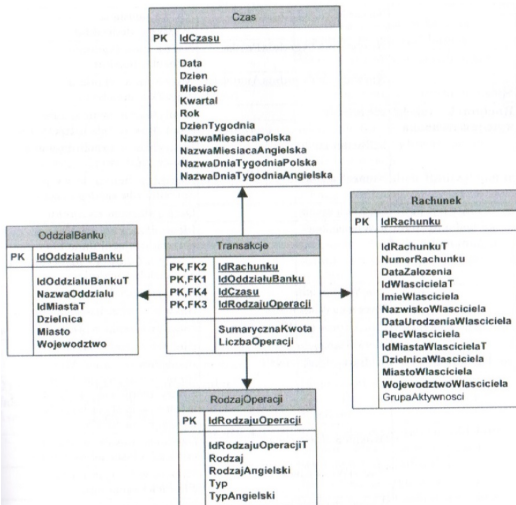


- pliki z tłumaczeniami nazw typów i rodzajów operacji na język angielski
 - 1; *Przelew z rachunku; transfer from account*
 - 2; *Przelew na rachunek; transfer into account*

Tworzenie schematu hurtowni danych

- ponieważ nasze pytania odnoszą się do transakcji wykonywanych na rachunkach klientów dlatego tematem hurtowni są **transakcje** - tabela faktów
- poziom szczegółowości danych i wymiary
 - wymiar **czas** - chcemy dokonywać analizy w rozbiciu na miesiące, kwartały i lata (pyt. 2 i 3), dodatkowo (pyt. 4) rozbicie na dni tygodnia i miesiąca - zatem szczegółowość danych powinna być dzienna
 - wymiar **rachunek** z danymi klienta
 - wymiar **oddział banku**
 - wymiar **rodzaj operacji**

SCHEMAT GWIAZDY DLA HURTOWNI DANYCH



ZAAWANSOWANE SYSTEMY BAZ DANYCH

WYKŁAD

Joanna Kapusta
e-mail: joanna.kapusta@kul.pl

Katolicki Uniwersytet Lubelski Jana Pawła II

EKSTRAKCJA, TRANSFORMACJA I ŁADOWANIE

ETL

- ekstrakcja (extract) - wydobywanie danych ze źródeł danych
- transformacja (transform) - dostosowanie formy i treści danych do potrzeb hurtowni
- ładowanie (load) - wprowadzenie danych do hurtowni

EKSTRAKCJA, TRANSFORMACJA I ŁADOWANIE

Przed wprowadzeniem danych do magazynu muszą one przejść proces integracji, tzn. ujednolicenia formy i treści pomiędzy różnorodnymi źródłami danych. Integracja to jeden z najważniejszych i najtrudniejszych etapów w cyklu życia hurtowni danych.

Dane mogą pochodzić z różnych źródeł (często niejednorodnych):

- baz danych: relacyjnych, sieciowych, hierarchicznych, ...
- plików tekstowych,
- arkuszy kalkulacyjnych,
- zewnętrznych (internet, komercyjne bazy danych),
- zdjęcia, mapy itp.

EKSTRAKCJA

- Wybranie informacji, które mają trafić do hurtowni (co?)
- Odczytanie informacji z baz źródłowych (jak?)

Sposób odczytu danych jest uzależniony od rodzaju źródła, np. relacyjne bazy danych mogą być odpytywane z wykorzystaniem poleceń języka SQL, niektóre źródła mogą wymagać specjalistycznego oprogramowania (np. zdjęcia RTG, USG).

ODCZYT NOWYCH DANYCH

- Dane mają jawnie podaną datę wprowadzenia do systemów źródłowych lub jest dostępny dziennik aktualizacji
- Możemy ingerować w systemy źródłowe - wprowadzamy wyzwalacze, które zapisują informacje, które rekordy zostały już wprowadzone, implementujemy tablicę różnic
- Nie możemy ingerować w systemy źródłowe - pamiętamy dane, które zostały załadowane (np. zakresy kluczy)

ODŚWIEŻANIE DANYCH

Odświeżanie danych - utrzymanie zgodności hurtowni z danymi źródłowymi. Modyfikacja danych źródłowych generuje konieczność opracowania metod rejestrowania historii zmian w hurtowni.

Monitorowanie zmian może się odbywać z wykorzystaniem:

- migawek - zewnętrzna kopia danych, porównywana co pewien czas ze stanem aktualnym
- dziennika aktywności - śledzenie przetwarzanych zapytań
- tablic różnic - źródłowa baza danych zapisuje w nich dodawane, usuwane i zmieniane rekordy (np. z wykorzystaniem wyzwalaczy)
- informacji przekazywanych przez źródła współpracujące - same informują hurtownię danych o wstawieniu/zmodyfikowaniu rekordów

TRANSFORMACJA

Transformacja i czyszczenie danych obejmuje:

- uzupełnianie brakujących wartości (np. brak kodu pocztowego)
- zmianę formatu (np. daty urodzenia)
- zmianę wartości (np. przeliczanie jednostek)
- ujednolicanie danych:
 - semantyka - ujednolicenie pojęć (np. fakt sprzedaży: złożenie zamówienia, wydanie towaru, wystawienie faktury)
 - format - np. wykrywanie literówek; płeć: K/M, 1/0, kobieta/mężczyzna; imiona i nazwiska w jednej lub dwóch kolumnach)
- utrzymanie integralności danych (więzy)
- usuwanie redundancji - wielokrotnie pojawiająca się informacja dot. jednego faktu

ŁADOWANIE

Ładowanie danych:

- problem przede wszystkim techniczny
- główne ograniczenie to wydajność całego procesu (wiele gigabajtów dziennie, konieczna optymalizacja obciążenia, zrównoleglanie prac)
- rodzaje:
 - pełne - jednorazowe, odnosi się do danych historycznych
 - przyrostowe - cykliczne, zwykle w cyklu dobowym podczas najmniejszego obciążenia hurtowni

ETL

Mechanizmy ETL:

- SQL Loader,
- tabele zewnętrzne,
- eksporty i importy
- DataPump
- SQL (merge, inserty wielotabelowe, inserty z podzapytaniem, tworzenie tabel z podzapytaniem, ...)
- PL/SQL

Tworzenie tabel z wykorzystaniem podzapytań

```
CREATE TABLE NazwaTabeli[(listaKolumn)] AS  
SELECT ...  
- tworzy tabelę i wstawia do niej dane będące wynikiem  
podzapytania
```

```
CREATE TABLE PracownicyBezPremii AS  
SELECT *  
FROM Pracownicy  
WHERE Premia IS NULL;
```

```
CREATE TABLE PracownicyOddzialy AS  
SELECT IdOd, Imie, Nazwisko, Nazwa  
FROM Pracownicy join Oddzialy using (IdOd);
```

WSTAWIANIE DANYCH - INSERT

Dodanie jednego wiersza

```
INSERT INTO NazwaTabeli[(listaKolumn)]  
VALUES (listaWartości)
```

Uwaga:

- Pominięcie listy kolumn wymaga wprowadzenia danych w takiej kolejności w jakiej występują w definicji tabeli (polecenie CREATE)
- Pominięcie danej dla kolumny wymaga aby w definicji tabeli była zdefiniowana wartość domyślna lub dopuszczalna wartość NULL

```
INSERT INTO Oddziały(IdOd, Nazwa, Miasto)  
VALUES (1234, 'Handlowy', 'Katowice');
```

Jeśli kolumna Miasto może zawierać wartości NULL

```
INSERT INTO Oddziały(IdOd, Nazwa) INSERT INTO Oddziały(IdOd, Nazwa, Miasto)  
VALUES (1234, 'Handlowy');          VALUES (1234, 'Handlowy', NULL);
```


WSTAWIANIE DANYCH Z WYKORZYSTANIEM PODZAPYTAŃ

```
INSERT INTO NazwaTabeli[(listaKolumn)]
```

```
SELECT ...
```

- wstawią do tabeli dane będące wynikiem podzapytania

```
INSERT INTO Programisci(IdPrac, Imie, Nazwisko)
```

```
SELECT IdPrac, imie, nazwisko
```

```
FROM Pracownicy
```

```
WHERE Stanowisko = 'programista';
```

ZARZĄDZANIE DUŻYMI ZBIORAMI DANYCH

- INSERT wielotabelowy umożliwia wstawienie danych uzyskanych przez podzapytanie do wielu tabel (transfer danych z jednego lub kilku źródeł do wielu tabel docelowych). Użyteczny w procesie ETL.
- Wstawienie danych do wielu tabel może być wykonane przez wielokrotne użycie polecenie INSERT ... SELECT - wymaga wielokrotnego przetwarzania źródła danych. INSERT wielotabelowy - jednokrotne przetwarzanie źródła.

ZARZĄDZANIE DUŻYMI ZBIORAMI DANYCH

INSERT wielotabelowy:

- bezwarunkowy

```
INSERT ALL
  INTO tab1 VALUES (lista wartości kolumn 1)
  INTO tab2 VALUES (lista wartości kolumn 2)
  INTO tab3 VALUES (lista wartości kolumn 3)
  ...
podzapytanie;
```

- warunkowy

```
INSERT opcja
  WHEN warunek1 THEN
    INTO tab1 VALUES (lista wartości kolumn 1)
  WHEN warunek2 THEN
    INTO tab2 VALUES (lista wartości kolumn 2)
  . . .
  ELSE
    INTO tab3 VALUES (lista wartości kolumn 3)
  podzapytanie;
```

- opcja: ALL albo FIRST
- domyślnie ALL

INSERT ALL

```
Pracownicy(IdPrac, Imie, Nazwisko, Pensja, Premia, Stanowisko, DataZatr, IdOd)  
Pracownicy1(IdPrac, Imie, Nazwisko, Pensja,...)  
Pracownicy2 (IdPrac, Imie, Nazwisko, IdOd, ...)
```

```
INSERT ALL  
INTO Pracownicy1 (IdPrac, Imie, Nazwisko, Pensja)  
VALUES (IdPrac, Imie, Nazwisko, Pensja)  
INTO Pracownicy2 (IdPrac, Imie, Nazwisko, IdOd)  
VALUES (IdPrac, Imie, Nazwisko, IdOd)  
SELECT * from Pracownicy;
```

INSERT ALL

Oddziały(IdOd, Nazwa, Miasto)

Pracownicy(IdPrac, Imie, Nazwisko, Pensja, Premia, Stanowisko, DataZatr, IdOd)

Srednie(IdOd, Nazwa, Srednia)

Minimalne(IdOd, Nazwa, Minimalna)

Maksymalne(IdOd, Nazwa, Maxi)

INSERT ALL

INTO Srednie

VALUES (IdOd, Nazwa, Sr)

INTO Minimalne

VALUES (IdOd, Nazwa, Mini)

INTO Maksymalne

VALUES (IdOd, Nazwa, Maxi)

SELECT IdOd, Nazwa, avg(Pensja) Sr, min(Pensja) Mini, max(Pensja) Maxi
from Pracownicy join Oddzialy using (IdOd)
group by IdOd, Nazwa;

INSERT ALL

Pracownicy(IdPrac, Imie, Nazwisko, Pensja, Premia, Stanowisko, DataZatr, IdOd)
 Pracownicy2005(IdPrac, Nazwisko, Imie)
 PracownicyPrzed2005(IdPrac, Nazwisko, Imie)
 PracownicyPo2005(IdPrac, Nazwisko, Imie) Pracownicy10000(IdPrac, Nazwisko)

```
INSERT ALL
WHEN (EXTRACT (YEAR FROM DataZatr) = 2005) THEN
INTO Pracownicy2005
VALUES (IdPrac, Nazwisko, Imie)
WHEN (EXTRACT (YEAR FROM DataZatr) < 2005) THEN
INTO PracownicyPrzed2005
VALUES (IdPrac, Nazwisko, Imie)
WHEN (EXTRACT (YEAR FROM DataZatr) > 2005) THEN
INTO PracownicyPo2005
VALUES (IdPrac, Nazwisko, Imie)
WHEN (Pensja < 10000) THEN
INTO Pracownicy10000
VALUES (IdPrac, Nazwisko)
SELECT IdPrac, Nazwisko, Imie, Pensja, DataZatr
from Pracownicy ;
```

INSERT FIRST

```
Pracownicy(IdPrac, Imie, Nazwisko, Pensja, Premia, Stanowisko, DataZatr, IdOd)  
Pracownicy2005(IdPrac, Nazwisko, Imie)  
Pracownicy10000(IdPrac, Nazwisko)  
PracownicyInni(IdPrac, Nazwisko, Imie)
```

```
INSERT FIRST  
WHEN (EXTRACT (YEAR FROM DataZatr) = 2005) THEN  
  INTO Pracownicy2005  
  VALUES (IdPrac, Nazwisko, Imie)  
WHEN (Pensja < 10000) THEN  
  INTO Pracownicy10000  
  VALUES (IdPrac, Nazwisko)  
ELSE  
  INTO PracownicyInni  
  VALUES (IdPrac, Nazwisko, Imie)  
SELECT IdPrac, Nazwisko, Imie, DataZatr  
from Pracownicy ;
```

INSERT ALL

Wydatki(IdOs, Rok, Styczen, Luty, Marzec)

Wydatki1(IdOs, Rok, Miesiac, Kwota)

INSERT ALL - można wykorzystać:

- do przekształcenia danych wierszowych w kolumnowe

```
INSERT ALL
  INTO Wydatki1
  VALUES (IdOs, Rok, 1, Styczen)
  INTO Wydatki1
  VALUES (IdOs, Rok, 2, Luty)
  INTO Wydatki1
  VALUES (IdOs, Rok, 3, Marzec)
SELECT * from Wydatki;
```

- do wstawienia wielu wierszy do tabeli

```
INSERT ALL
  INTO Wydatki1
  VALUES (1, 2013, 1, 110)
  INTO Wydatki1
  VALUES (1, 2013, 2, 220)
  INTO Wydatki1
  VALUES (1, 2013, 3, 330)
SELECT * from dual;
```


MERGE

- umożliwia scalenie wierszy z różnych tabel
- połączenie warunkowego polecenia INSERT, UPDATE i DELETE
- polecenie użyteczna w hurtowaniach danych (dane pochodzą z różnych źródeł zawierają duplikaty)

```

MERGE INTO tabela wynikowa
USING [tabela/perspektywa]
ON (warunek złączenia)
WHEN MATCHED THEN
    UPDATE SET
        kol1 = wartość1,
        ...
        koln = wartośćn
DELETE WHERE (warunek)
WHEN NOT MATCHED THEN
    INSERT [(lista kolumn)]
    VALUES (lista wartość);

```

- klauzula MERGE INTO określa nazwę tabeli w której będą scalane wiersze
- klauzula USING ... ON określa nazwę tabeli/perspektywy z której będą pobierane wiersze do scalenia oraz warunek złączenia tabel
- klauzula WHEN MATCHED określa czynność wykonywaną gdy wiersze spełniają warunek złączenia
- klauzula WHEN NOT MATCHED określa czynność wykonywaną gdy wiersze NIE spełniają warunku złączenia

MERGE - PRZYKŁAD

Towary(IdTow, Nazwa, Cena)

TowaryUaktualnione(IdTow, Nazwa, Cena)

- jeśli IdTow są równe to dane w tabeli Towary powinny zostać uaktualnione zgodnie z tym co jest zapisane w tabeli TowaryUaktualnione
- jeżeli w tabeli Towary nie występuje jakiś towar to powinien zostać dopisany do tabeli

```
MERGE INTO Towary t
USING TowaryUaktualnione tu
ON (t.IdTow = tu.IdTow)
WHEN MATCHED THEN
    UPDATE SET
        t.Nazwa = tu.Nazwa,
        t.Cena = tu.Cena
WHEN NOT MATCHED THEN
    INSERT
    VALUES (tu.IdTow, tu.Nazwa, tu.Cena);
```

KLAUZULA WITH

- użycie klauzuli WITH umożliwia wielokrotne wykorzystanie tego samego bloku zapytania w zapytaniu złożonym
- wynik klauzuli WITH jest przechowywany w tymczasowej przestrzeni tabel użytkownika
- korzyści: przejrzystość zapytań i zwiększenie wydajności

WITH

```
wynagr_dzial AS (SELECT Nazwa, SUM(Pensja) AS razem
                  FROM Pracownicy JOIN Oddzialy USING (Id_Od)
                  GROUP BY Nazwa),
srednia AS (SELECT AVG(Pensja) sr
            FROM Pracownicy
            )
SELECT * FROM wynagr_dzial
WHERE razem > (SELECT sr FROM srednia);
```

ZAAWANSOWANE SYSTEMY BAZ DANYCH

WYKŁAD

Joanna Kapusta
e-mail: joanna.kapusta@kul.pl

Katolicki Uniwersytet Lubelski Jana Pawła II

TABELE ZEWNĘTRZNE

- Pozwalają na korzystanie z danych zapisanych w plikach zewnętrznych tak jakby były one tabelami bazy danych
- Strukturę i lokalizację tabeli zewnętrznej definiuje się w systemie Oracle
- Do ładowania danych wykorzystuje mechanizm SQL*Loader lub Data Pump
- Odczyt danych z zewnętrznej tabeli odbywa się tak jak z tabel bazy danych (SELECT)
- Nie można aktualizować ani usuwać danych w plikach zewnętrznych z poziomu systemu
- Data Pump pozwala na jednorazowe zapisanie danych w pliku (CREATE TABLE AS SELECT)

TABELE ZEWNĘTRZNE

Utworzenie obiektu `directory` który odpowiada katalogowi, w którym znajdują się pliki (dane zewnętrzne).

```
CREATE OR REPLACE DIRECTORY z_dir  
AS 'C:\temp';
```

Nadanie uprawnień do odczytu i zapisu danych z katalogu

```
GRANT READ, WRITE ON DIRECTORY z_dir TO użytkownik;
```

TABELE ZEWNĘTRZNE

```
CREATE TABLE nazwa_tabeli (  
    kolumna1 typ_danych1, kolumna2 typ_danych2,...  
)  
ORGANIZATION EXTERNAL  
    (TYPE typ_sterownika  
    DEFAULT DIRECTORY nazwa_katalogu  
    ACCESS PARAMETERS  
    (...)  
)  
LOCATION (nazwa_pliku)  
PARALLEL  
REJECT LIMIT [0|liczba|UNLIMITED];
```

Tworzenie tabeli zewnętrznej

```
CREATE TABLE prac_pos (  
    imie char(25), nazwisko CHAR(25))  
    ORGANIZATION EXTERNAL  
    (TYPE ORACLE_LOADER  
    DEFAULT DIRECTORY z_dir  
    ACCESS PARAMETERS  
    (RECORDS DELIMITED BY NEWLINE  
    NOBADFILE  
    NOLOGFILE  
    FIELDS TERMINATED BY ','  
    (imie POSITION ( 1:20) CHAR,  
    nazwisko POSITION (22:41) CHAR))  
    LOCATION ('prac_pos.dat'))  
    PARALLEL 5  
    REJECT LIMIT 200;
```


TWORZENIE TABELI ZEWNĘTRZNEJ

```
CREATE TABLE prac_delim
(id_prac NUMBER(4),
 imie VARCHAR2(20),
 nazwisko VARCHAR2(25))
ORGANIZATION EXTERNAL
(TYPE ORACLE_LOADER DEFAULT DIRECTORY z_dir
ACCESS PARAMETERS
(
records delimited by newline
nobadfile
nologfile
fields terminated by ';'
(id_prac, imie, nazwisko))
LOCATION ('pracownicy.dat'))
PARALLEL REJECT LIMIT UNLIMITED;
```

TABELE ZEWNĘTRZNE

```
CREATE TABLE prac
  ORGANIZATION EXTERNAL
  (
    TYPE ORACLE_DATAPUMP
    DEFAULT DIRECTORY z_dir
    LOCATION
      ('prac.dat')
  )
  PARALLEL
AS
SELECT employee_id,
       first_name,
       last_name
FROM   employees;
```

MODYFKOWANIE TABELI ZEWNĘTRZNEJ

Parametry z klauzuli `access parameters` można modyfikować (nie trzeba usuwać i ponownie tworzyć tabeli zewnętrznej).

```
alter table prac
ACCESS PARAMETERS
(
records delimited by newline
skip 5
fields terminated by ';'
(id_prac, imie, nazwisko)
)
```

DODAWANIE, USUWANIE I MODYFIKOWANIE KOLUMN

Składnia poleceń taka sama jak w przypadku zwykłych tabel.

- Dodawanie kolumny

```
alter table prac add email varchar2(20);
```

- Modyfikowanie kolumny

```
alter table prac modify email varchar2(40);
```

- Usuwanie kolumny

```
alter table prac drop column email;
```

MODYFKOWANIE TABELI ZEWNĘTRZNEJ

- Zmiana katalogu domyślnego

```
alter table prac default directory new_dir;
```

- Zmiana plików zewnętrznych

```
alter table prac location ('prac1.txt, prac2.txt')
```

ZAAWANSOWANE SYSTEMY BAZ DANYCH

WYKŁAD

Joanna Kapusta
e-mail: joanna.kapusta@kul.pl

Katolicki Uniwersytet Lubelski Jana Pawła II

WPROWADZANIE DANYCH DO HURTOWNI

Do ładowania danych do hurtowni możemy wykorzystać np.:

- SQL*Loader
- DataPump
- Eksport/import
- Tabele zewnętrzne

SQL*Loader

SQL*Loader – narzędzie do szybkiego wypełniania tabel danymi pochodzącymi z plików tekstowych np. txt, csv.

Zwykle wykorzystuje dwa rodzaje plików:

- plik danych - zawiera dane, które mają zostać załadowane
- plik sterujący - zawiera informacje o formacie danych (wiersze kolumny), porządku ładowania, nazwy plików przechowujących dane, kryteria ładowania danych, itp.

W wyniku pracy SQL*Loadera powstają pliki:

- dziennika (log file *.log*) - zawiera informacje o przebiegu ładowania (np. liczba przetworzonych, zatwierdzonych wierszy)
- złych danych (bad file *.bad*) - zawiera wiersze, których ładowanie nie powiodło się ze względu na niepoprawność danych (duplikat w kolumnie klucza głównego)
- odrzuconych danych (discard file *.dsc*) - zawiera wiersze, które nie zostały wstawione do bazy, ponieważ nie zostały spełnione kryteria wstawiania

SQL*Loader dostarcza dwie metody ładowania danych:

- Conventional Path – wiele instrukcji insert
- Direct Path – bloki danych są dopisywane bezpośrednio do tabel, wydajność operacji ładowania jest dużo większa

PLIK STERUJĄCY

Zawiera informacje o:

- lokalizacji danych źródłowych,
- tabelach, do których mają zostać załadowane dane,
- regułach i kryteriach ładowania.

Przykładowe znaczniki:

- INFILE – wskazuje, w jakim pliku są dane do ładowania
- INTO TABLE – wskazuje, do których tabel załadować dane
- REPLACE – usuwa istniejące wiersze tabeli i ładuje na ich miejsce nowe
- APPEND – rozbudowuje tabelę, dodając do niej nowe wiersze
- INSERT – ładuje dane tylko do pustej tabeli (jeżeli tabela nie będzie pusta, to wystąpi błąd)
- FIELDS TERMINATED BY – znak oddzielający kolumny
- BAD FILE – nazwa pliku złych danych

PRZYKŁADY

```
LOAD DATA
INFILE 'c:\temp\prac.txt'
INTO TABLE PRACOWNICY(
id POSITION(1:5) CHAR,
imie POSITION(6:20) CHAR,
nazwisko POSITION(21:40) CHAR)

sqlldr user/haslo control = c:\temp\prac.ctr
      log = c:\temp\prac.log
```

PRZYKŁADY

```
LOAD DATA
INFILE 'c:\temp\prac.txt'
BADFILE 'c:\temp\prac.bad'
APPEND
INTO TABLE PRACOWNICY
FIELDS TERMINATED BY ","
(prac_id, imie, nazwisko)
```

Klauzula WHEN ogranicza ładowanie wierszy, tylko do tych, które spełniają warunek.

```
LOAD DATA
INFILE 'c:\temp\prac.txt'
BADFILE 'c:\temp\prac.bad'
APPEND
INTO TABLE PRACOWNICY
WHEN prac_id > 100
FIELDS TERMINATED BY ","
(prac_id, imie, nazwisko)
```

WYBRANE OPCJE URUCHAMIANIA SQL*LOADERA

- USERID – nazwa użytkownika i hasło wykorzystywane podczas ładowania, oddzielone ukośnikiem
- CONTROL – nazwa pliku sterującego
- LOG nazwa pliku dziennika
- BAD – nazwa pliku "złych danych"
- DATA – nazwa pliku danych
- DISCARD – nazwa pliku odrzuconych rekordów
- DISCARDMAX – maksymalna ilość odrzuconych rekordów przed zatrzymaniem ładowania. Domyślne dozwolone jest odrzucenie dowolnej liczby rekordów.
- SKIP – liczba logicznych rekordów w pliku wejściowym, które mają być pominięte przed rozpoczęciem ładowania danych. Domyślna wartość wynosi 0.
- LOAD – liczba logicznych rekordów do załadowania. Domyślnie ładowane są wszystkie rekordy.

EKSPORT I IMPORT DANYCH - DATA PUMP

- Programy EXPDP i IMPDP.
- Możemy eksportować/importować:
 - całą bazę
 - wybrany schemat
 - wybrane tabele
 - wybraną przestrzeń tabel
- Dane do uruchomienia:
 - użytkownik wykonujący eksport/import
 - katalog, w którym mają być utworzone pliki eksportu lub odczytane pliki do importu
 - nazwy plików eksportu/importu
 - co eksportujemy/importujemy

DIRECTORY

Utworzenie obiektu directory który odpowiada katalogowi, w którym dane zewnętrzne się znajdują.

```
CREATE OR REPLACE DIRECTORY m_dir  
AS 'C:\temp';
```

Nadanie uprawnień do odczytu danych z katalogu

```
GRANT READ ON DIRECTORY m_dir TO użytkownik;
```

Nadanie uprawnień do zapisu danych z katalogu

```
GRANT WRITE ON DIRECTORY m_dir TO użytkownik;
```

EKSPORT - DATA PUMP

Eksport całej bazy

```
EXPDP SYSTEM/sys DIRECTORY=DTPUMP_DIR  
DUMPFILE=BAZA.DMP LOGFILE=BAZA.LOG  
FULL=Y
```

Eksport schematu

```
EXPDP SYSTEM/sys DIRECTORY=DTPUMP_DIR  
DUMPFILE=HR.DMP LOGFILE=HR.LOG  
SCHEMAS=HR
```

Eksport tabel

```
EXPDP SYSTEM/sys DIRECTORY=DTPUMP_DIR  
DUMPFILE=HR_EMP_DEP.DMP LOGFILE=HR_EMP_DEP.LOG  
TABLES=HR.EMPLOYEES,HR.DEPARTMENTS
```

IMPORT - DATA PUMP

Import całej bazy

```
IMPDP SYSTEM/sys DIRECTORY=DTPUMP_DIR  
DUMPFILE=BAZA.DMP LOGFILE=BAZA.LOG  
FULL=Y
```

Import schematu

```
IMPDP SYSTEM/sys DIRECTORY=DTPUMP_DIR  
DUMPFILE=HR.DMP LOGFILE=HR.LOG  
SCHEMAS=HR
```

Import tabel

```
IMPDP SYSTEM/sys DIRECTORY=DTPUMP_DIR  
DUMPFILE=HR_EMP_DEP.DMP LOGFILE=HR_EMP_DEP.LOG  
TABLES=HR.EMPLOYEES,HR.DEPARTMENTS
```


IMPORT DANYCH ORACLE SQL DEVELOPER

- W oknie nawigacji tabel wybierz opcje Import Data (prawy klik myszy).
- Wskaż plik źródłowy.
- W oknie kreatora importu zostaną wyświetlone dane, zweryfikuj poprawność i kliknij Next.
- Wybierz metodę np. Insert i wpisz nazwę tabeli do której dane mają zostać zaimportowane.
- Wskaż kolumny, które mają zostać zaimportowane (przenieś z panelu Available Columns do Selected Columns).
- Dla każdej kolumny określ typ danych, rozmiar i inne właściwości
- Potwierdź import danych.

EKSPORT DANYCH ORACLE SQL DEVELOPER

- W oknie nawigacji kliknij na wybranej tabeli wybierz opcje Eksport... (prawy klik myszy).
- W oknie kreatora wskaż plik wynikowy.
- W kolejnym kroku wybierz dane które mają być eksportowane (Columns i Object Where np. salary > 1000).
- Zakończ.

EKSPORT DANYCH ORACLE SQL DEVELOPER

- Eksport bazy danych Tools/Database Export.
- Konfigurowanie środowiska Tools/Preferences
 - Database/Utilities/Import (Export)

ZAAWANSOWANE SYSTEMY BAZ DANYCH

Joanna Kapusta
e-mail: joanna.kapusta@kul.pl

Katolicki Uniwersytet Lubelski Jana Pawła II

PERSPEKTYWY ZMATERIALIZOWANE

- Perspektywa zmaterializowana to perspektywa, której zawartość jest fizycznie składowana w bazie danych.
- Struktura i zawartość perspektywy zmaterializowanej jest definiowana za pomocą zapytania SQL.
- Perspektywy zmaterializowane są odświeżane przyrostowo lub w pełni, synchronicznie (natychmiast) i asynchronicznie (z określonym okresem odświeżania).
- Przywileje systemowe: CREATE MATERIALIZED VIEW, CREATE ANY MATERIALIZED VIEW, CREATE TABLE, CREATE ANY TABLE

PERSPEKTYWY ZMATERIALIZOWANE

Perspektywy zmaterializowane vs. tabele podsumowań.

```
SELECT c.cust_id, SUM(amount_sold)
FROM   sales s, customers c
WHERE  s.cust_id = c.cust_id
GROUP BY c.cust_id;
```

```
CREATE TABLE podsumowanie_sp AS
SELECT c.cust_id, SUM(amount_sold) AS amount
FROM   sales s, customers c
WHERE  s.cust_id = c.cust_id
GROUP BY c.cust_id;
```

```
SELECT * FROM podsumowanie_sp;
```

PERSPEKTYWY ZMATERIALIZOWANE

Perspektywy zmaterializowane vs. tabele podsumowań.

--dba

```
CREATE MATERIALIZED VIEW podsumowanie_sp
ENABLE QUERY REWRITE AS
SELECT c.cust_id, SUM(amount_sold)
FROM   sales s, customers c
WHERE  s.cust_id = c.cust_id
GROUP BY c.cust_id;
```

--user

```
SELECT c.cust_id, SUM(amount_sold)
FROM   sales s, customers c
WHERE  s.cust_id = c.cust_id
GROUP BY c.cust_id;
```

--oracle server

```
SELECT * FROM podsumowanie_sp;
```

PERSPEKTYWY ZMATERIALIZOWANE

```
CREATE MATERIALIZED VIEW SPRZEDAZ_MV  
  BUILD IMMEDIATE  
  REFRESH COMPLETE  
  NEXT sysdate + 1  
AS  
  SELECT id_sklepu, id_produktu,  
    SUM(kwota) AS suma, AVG(kwota) AS srednia  
  FROM sprzedaz  
  GROUP BY id_sklepu, id_produktu;
```

- zapytanie SQL specyfikuje zawartość perspektywy
- sposób odświeżania zawartość COMPLETE - pełne

RODZAJE PERSPEKTYW ZMATERIALIZOWANYCH

- agregujące

```
CREATE MATERIALIZED VIEW podsumowanie_sp
  BUILD IMMEDIATE AS
  SELECT c.cust_id, s.channel_id, SUM(amount_sold)
  FROM   sales s, customers c
  WHERE  s.cust_id = c.cust_id
  GROUP BY c.cust_id, s.channel_id;
```

- bazujące tylko na złączeniach

```
CREATE MATERIALIZED VIEW produkty
  BUILD IMMEDIATE AS
  SELECT s.time_id, p.prod_name
  FROM sales s RIGHT OUTER JOIN products p
  WHERE s.prod_id = p.prod_id;
```

- zagnieżdżone

PERSPEKTYWY ZMATERIALIZOWANE

W momencie tworzenia perspektywy powstaje:

- tabela przechowująca dane (kontener)
- definicja perspektywy
- indeks zawierający funkcję agregującą tylko dla perspektyw agregujących

```
CREATE MATERIALIZED VIEW SPRZEDAZ_MV
```

```
BUILD IMMEDIATE
```

```
REFRESH COMPLETE
```

```
NEXT sysdate + 1
```

```
AS
```

```
SELECT id_sklepu, id_produktu,
```

```
SUM(kwota) AS suma, AVG(kwota) AS srednia
```

```
FROM sprzedaz
```

```
GROUP BY id_sklepu, id_produktu;
```

PERSPEKTYWY ZMATERIALIZOWANE

Zawartości perspektywy zmaterializowanej może być odświeżana na podstawie opcji (opcja podawana przy tworzeniu perspektywy):

- COMPLETE – odświeżanie pełne (powtórne wykonanie zapytania)
- FAST – odświeżanie przyrostowe (skorzystanie z dzienników zapisujących zmiany w tabelach bazowych MATERIALIZED VIEW LOG)
- ON DEMAND – przez wykonanie explicite instrukcji odświeżającej (DBMS_MVIEW.REFRESH)
- ON COMMIT – automatycznie po każdym COMMIT dotyczącym tabel bazowych
- START ... NEXT ... - od kiedy rozpocząć odświeżanie i z jaką częstotliwością

Przykład:

```
REFRESH FAST START WITH sysdate NEXT sysdate+1
```

DZIENNIKI ZMATERIALIZOWANYCH PERSPEKTYW

Definiowane w momencie tworzenia perspektywy:

```
CREATE MATERIALIZED VIEW LOG ON tabela  
WITH PRIMARY KEY/ROWID/SEQUENCE ...
```

- WITH PRIMARY KEY- w dzienniku rejestrowane wartości atrybutów wchodzących w skład klucza
- WITH ROWID - w dzienniku rejestrowane identyfikatorów rekordów (ROWID)
- WITH PRIMARY KEY, ROWID - w dzienniku rejestrowane wartości atrybutów kluczowych i ROWID
- WITH SEQUENCE - do odświeżania przyrostowego, gdy do tabeli bazowej są wstawiane rekordy, modyfikowane i usuwane

PERSPEKTYWY ZMATERIALIZOWANE

Jedna perspektywa wiele zapytań.

```
CREATE MATERIALIZED VIEW cust_sales_mv2
ENABLE QUERY REWRITE AS
SELECT cust_last_name, channel_id,
       SUM(amount_sold)
FROM   sales s, customers c
WHERE  s.cust_id = c.cust_id
GROUP BY c.cust_last_name, s.channel_id;
```

```
SELECT cust_last_name, channel_id,
       SUM(amount_sold)
FROM   sales s, customers c
WHERE  s.cust_id = c.cust_id
GROUP BY c.cust_last_name,
       s.channel_id;
```

```
SELECT cust_last_name,
       SUM(amount_sold)
FROM   cust_sales_mv2
GROUP BY c.cust_last_name;
```

```
SELECT channel_id, SUM(amount_sold)
FROM   sales s
GROUP BY channel_id;
```

```
SELECT channel_id, SUM(amount_sold)
FROM   cust_sales_mv2
GROUP BY c.channel_id;
```

SŁOWNIKI DANYCH

- USER_MVIEWS - zmaterializowane perspektywy
- USER_MVIEW_LOGS - dzienniki zmaterializowanych perspektyw
- USER_MVIEW_REFRESH_TIMES - informacje dotyczące odświeżania

KOMENTARZE

- dodanie komentarza

```
COMMENT ON MATERIALIZED VIEW sprzedaz_mv  
IS 'perspektywa zmalerializowana sprzedazy  produktu...
```

- odczytanie komentarza

```
SELECT mview_name, comments  
FROM user_mview_comments  
WHERE mview_name = 'sprzedaz_mv';
```

MODYFIKOWANIE I USUWANIE

- modyfikowanie (perspektywy, których jesteśmy właścicielami albo uprawnienie ALTER ANY MATERIALIZED VIEW)

```
ALTER MATERIALIZED VIEW sprzedaz_mv  
REFRESH FAST ON COMMIT;
```

- usuwanie

```
DROP MATERIALIZED VIEW sprzedaz_mv;
```


ZAAWANSOWANE SYSTEMY BAZ DANYCH

Joanna Kapusta
e-mail: joanna.kapusta@kul.pl

Katolicki Uniwersytet Lubelski Jana Pawła II

WYMIARY

Obiekt DIMENSION

- struktura przechowująca dodatkowe informacje dot. wymiaru (definicje hierarchii poziomów wymiaru)
- pozwala na lepszą optymalizację zapytań dla danych wymiaru (mechanizm przepisывania zapytań z użyciem perspektyw zmaterializowanych; odświeżanie przyrostowe zmaterializowanych perspektyw zawierających agregaty)
- opcjonalny, uzupełnia informacje przechowywane w relacji/relacjach wymiaru
- przed definicją obiektu DIMENSION musi istnieć relacja/relacje wymiaru

WYMIARY

Przywileje

- CREATE DIMENSION
- CREATE ANY DIMENSION
- SELECT - do obiektów do których odnoszą się wymiary
- ALTER ANY DIMENSION
- DROP ANY DIMENSION

OBIEKT DIMENSION

Wymiar - Czas

- wymiar
- hierarchia



OBIEKT DIMENSION

Wymiar - Czas

- tworzymy obiekt DIMENSION i wskazujemy poziomy hierarchii wymiaru:

```
CREATE DIMENSION CzasDim
  LEVEL Dzień IS Czas.Czas_Dzień
  LEVEL Miesiąc IS Czas.Czas_Miesiąc
  LEVEL Rok IS Czas.Czas_Rok
  HIERARCHY czas_hier
    (Dzień CHILD OF
      Miesiąc CHILD OF
        Rok);
```

- każdy poziom hierarchii odpowiada jednemu atrybutowi relacji wymiaru
- pomiędzy poziomami: nadrzędnym i podrzędnym hierarchii musi istnieć zależność 1:n
- cykle w hierarchii nie są dozwolone

OBIEKT DIMENSION

Wymiar - Produkt

| Produkty |
|------------------------|
| Prod_Id (PK) |
| Prod_Nazwa |
| Prod_Opis |
| Prod_Klasa_Wagowa |
| Prod_Klasa_Rozmiar |
| Prod_Min_Cena |
| Prod_Podkategoria |
| Prod_Podkategoria_Opis |
| Prod_Kategoria |
| Prod_Kategoria_Opis |

OBIEKT DIMENSION

Wymiar - Produkt

- tworzymy obiekt DIMENSION, wskazujemy poziomy hierarchii wymiaru, definiujemy zależności funkcyjne:

```
CREATE DIMENSION ProduktDim
  LEVEL produkt          IS (Produkty.Prod_Id)
  LEVEL podkategoria     IS (Produkty.Prod_Podkategoria)
  LEVEL kategoria        IS (Produkty.Prod_Kategoria)
  HIERARCHY prod_hier (
    produkt              CHILD OF
    podkategoria         CHILD OF kategoria)
  ATTRIBUTE produkt DETERMINES
    (Prod_Nazwa, Prod_Opis,
     Prod_Klasa_Wagowa, Prod_Klasa_Rozmiar,
     Prod_Min_Cena)
  ATTRIBUTE podkategoria DETERMINES
    (Prod_Podkategoria, Prod_Podkategoria_Opis)
  ATTRIBUTE kategoria DETERMINES
    (Prod_Kategoria, Prod_Kategoria_Opis);
```

- zależność funkcyjna określa powiązanie w stopniu 1:1 między zdefiniowanym poziomem hierarchii a zależnymi od niego innymi atrybutami poziomu

OBIEKT DIMENSION

Wymiar - Klient

- tabele

Kraje (Kraj_Id, Kraj_Nazwa_Państwa, Kraj_Region)

Klienci (Klient_Id, Klient_Miasto, Klient_Wojewodztwo, Kraj_Id)

- utworzenie obiektu DIMENSION

```
CREATE DIMENSION Klient_dim
```

```
  LEVEL klient      IS Klienci.Klient_Id
```

```
  LEVEL miasto      IS Klienci.Klient_Miasto
```

```
  LEVEL wojewodztwo IS Klienci.Klient_Wojewodztwo
```

```
  LEVEL kraj        IS Kraje.Kraj_Id
```

```
  LEVEL region      IS Kraje.Kraj_Region
```

```
HIERARCHY klient_hier (
```

```
klient CHILD OF miasto CHILD OF wojewodztwo CHILD OF
```

```
kraj CHILD OF region
```

```
JOIN KEY (Klienci.Kraj_Id) REFERENCES Kraje)
```

```
ATTRIBUTE klient DETERMINES
```

```
(Klient_Imie, Klient_Nazwisko)
```

```
ATTRIBUTE kraj DETERMINES Kraje.Kraj_Nazwa_Państwa;
```


WYMIARY

Informacje o wymiarach:

- USER_DIMENSIONS
- USER_DIM_ATTRIBUTES
- USER_DIM_CHILD_OF
- USER_DIM_HIERARCHIES
- USER_DIM_JOIN_KEY
- USER_DIM_LEVELS
- USER_DIM_LEVEL_KEY
- DBMS_DIMENSION.DESCRIBE_DIMENSION('CZAS_DIM')

WYMIARY

Walidacja wymiaru:

- dane, składowane w relacjach wymiaru, mogą nie być zgodne z informacjami zawartymi w obiekcie DIMENSION,
- obiekt DIMENSION nie wymusza poprawności danych relacji wymiarów
- jeśli dane w relacji wymiarów będą niepoprawne, niepoprawne mogą być również wyniki analiz (bazujących na wymiarach)
- Pakiet DBMS_DIMENSION.VALIDATE_DIMENSION
 - sprawdzenie poprawności danych relacji wymiarów na podstawie informacji z obiektu DIMENSION
 - wynik weryfikacji zostaje umieszczony z relacji DIMENSION_EXCEPTIONS
 - parametry procedury: nazwa obiektu DIMENSION, który ma być sprawdzony; czy sprawdzać tylko nowe rekordy, wprowadzone do relacji (TRUE) czy wszystkie rekordy (FALSE); czy sprawdzać, czy wszystkie atrybuty relacji wymiaru są obowiązkowe (TRUE).

WALIDACJA WYMIARU - PRZYKŁAD

```
EXECUTE DBMS_DIMENSION.VALIDATE_DIMENSION  
        ('CZAS_DIM', FALSE, TRUE);
```

```
SELECT *  
    FROM dimension_exceptions;
```

WALIDACJA WYMIARU - PRZYKŁAD

- Modyfikacja definicji obiektu DIMENSION:

```
ALTER DIMENSION <nazwa>  
[ADD [ATTRIBUTE | HIERARCHY | LEVEL] <definicja>] |  
[DROP [ATTRIBUTE | HIERARCHY | LEVEL] <nazwa>;
```

```
ALTER DIMENSION CZAS_DIM  
DROP LEVEL Rok;
```

- Usunięcie obiektu DIMENSION:

```
DROP DIMENSION <nazwa>;  
  
DROP DIMENSION CZAS_DIM;
```

STAR QUERY

Tabela faktów Sprzedaz i tabele wymiarów: Produkt, Klient, Czas

```
SELECT Produkt.Prod_Nazwa,  
       Klient.Klient_Miasto,  
       Czas.Miesiac,  
       SUM(Sprzedaz.Sprzedana_Ilosc) AS Total_Sprzedaz  
FROM Sprzedaz  
JOIN Produkt ON Sprzedaz.Produkt_Id = Produkt.Produkt_Id  
JOIN Klient ON Sprzedaz.Klient_Id = Klient.Klient_Id  
JOIN Czas ON Sprzedaz.Czas_Id = Czas.Czas_Id  
WHERE Czas.Rok = 2025  
GROUP BY Produkt.Prod_Nazwa,  
         Klient.Klient_Miasto,  
         Czas.Miesiac
```

- na kluczach obcych tabeli faktu powinny zostać założone indeksy bitmapowe
- parametr STAR_TRANSFORMATION_ENABLED musi być włączony
- świeże statystyki dla wszystkich obiektów biorących udział w zapytaniu

ZAAWANSOWANE SYSTEMY BAZ DANYCH

WYKŁAD

Joanna Kapusta
e-mail: joanna.kapusta@kul.pl

Katolicki Uniwersytet Lubelski Jana Pawła II

OBIEKTOWY MODEL DANYCH

Klasyczne rozwiązania:

- zbyt prosty model danych,
- tylko proste typy danych,
- brak mechanizmów do reprezentacji hierarchii i agregacji,
- brak mechanizmów do zarządzania temporalnymi aspektami bazy danych (czas, wersja obiektów, wersje schematów).

Wymagania:

- reprezentowanie i posługiwanie się złożonymi, zagnieżdżonymi obiektami
- definiowanie dowolnych typów danych i operowanie nimi
- reprezentowanie i zarządzanie zmianami w bazie danych (wersje obiektów,...)
- reprezentowanie i operowanie pojęciami hierarchii oraz agregacji

OBIEKTOWA BAZA DANYCH

OBIEKTOWY MODEL DANYCH

model danych, którego podstawą są pojęcia obiektowości, m.in.:
obiekt, klasa, dziedziczenie, hermetyzacja

OBIEKTOWA BAZA DANYCH

- zbiór obiektów, których stan, zachowanie i związki występujące między nimi są określone zgodnie z obiektowym modelem danych
- zorientowany obiektowo system, który umożliwia zarządzanie bazą danych

PODSTAWOWE POJĘCIA OBIEKTOWEGO MODELU DANYCH

- klasa
- obiekt
- dziedziczenie
- hermetyzacja
- agregacja

Przykład: Rachunek bankowy (właściciel, saldo, debet, Wpłać, Wypłać, Nalicz odsetki,...)

STANDARD ODMG

Standard ODMG (ang. Object Database Management Group) składa się z następujących części:

- opis modelu obiektowego
- ODL - język definicji schematu obiektowej bazy danych
- OQL - obiektowy język zapytań (wzorowany na SQL)
- OML – rozszerzenia obiektowych języków programowania (C++, Java) do przetwarzania trwałych obiektów w obiektowych bazach danych

RELACYJNE VS. OBIEKTOWE BAZY DANYCH

RELACYJNE

- niezależność od języka programowania
- dobrze zdefiniowana teoria, sprawdzona
- możliwość zarządzania wielką ilością danych (dla trudniejszych problemów bardzo dużo tabel)
- możliwość definiowania złożonych kryteriów wyszukiwania
- mało naturalna reprezentacja danych
- ograniczona podatność na zmiany
- brak złożonych typów danych
- trudność operowania na danych złożonych
- niezgodność z modelem używanym przez języki ogólnego przeznaczenia

OBIEKTOWE

- naturalna reprezentacja obiektów świata rzeczywistego
- naturalna reprezentacja zależności między obiektami
- łatwość operowania na złożonych obiektach
- duża podatność na zmiany
- możliwość definiowania własnych typów, metod
- brak konieczności konwersji na styku baza danych – aplikacja (integracja z językiem programowania)
- ujednolicony model pojęciowy - obiektowe podejście do analizy, projektowania i implementacji
- słaba obsługa przeszukiwania danych
- brak powszechnie zaakceptowanego języka zapytań

MODEL OBIEKTOWO-RELACYJNY

- Umożliwia wykorzystanie mechanizmów obiektowych w relacyjnej bazie danych
 - abstrakcyjne typy danych definiowane przez użytkownika
 - predefiniowane typy obiektowe dostarczane przez producentów
 - dziedziczenie, enkapsulacja, polimorfizm, klasy abstrakcyjne
 - dostępne mechanizmy relacyjne: SQL, przetwarzanie transakcyjne, ...
- Alternatywa dla systemów odwzorowania obiektowo-relacyjnego (O/RM)

BAZY OBIEKTOWO-RELACYJNE W ORACLE

- obiektywne typy danych użytkownika – metody, składowe, konstruktory, przeciążanie
- współdzielenie obiektów - referencje
- dziedziczenie – polimorfizm, przesłanianie metod, typy abstrakcyjne
- kolekcje – tabele o zmiennym rozmiarze, tabele zagnieżdżone

ABSTRAKCYJNE TYPY DANYCH UŻYTKOWNIKA

Abstrakcyjny typ danych umożliwia grupowanie atrybutów w obiekty.

```
CREATE OR REPLACE TYPE Pracownik AS OBJECT (  
    imie VARCHAR2(20),  
    nazwisko VARCHAR2(20),  
    pensja NUMBER(6,2),  
    data_ur DATE  
);
```

ABSTRAKCYJNE TYPY DANYCH UŻYTKOWNIKA

```
CREATE OR REPLACE TYPE Adres AS OBJECT (  
    miejscowosc VARCHAR2(20),  
    ulica VARCHAR2(20),  
    numer NUMBER(4)  
);
```

```
CREATE OR REPLACE TYPE Pracownik AS OBJECT (  
    imie VARCHAR2(20),  
    nazwisko VARCHAR2(20),  
    pensja NUMBER(6,2),  
    data_ur DATE,  
    adr Adres  
);
```

METODY TYPÓW OBIEKTOWYCH

Typ obiektowy może posiadać następujące rodzaje metod:

- CONSTRUCTOR - metoda tworząca nowy obiekt
- MEMBER - metoda wywołana na rzecz konkretnego obiektu
- MAP/ORDER - metoda wykorzystywana do odwzorowania/porównania obiektów
- STATIC - metody wołane na rzecz typu obiektowego

Uzyskiwanie informacji o typach obiektowych:

```
DESCRIBE nazwa_typu;
```


ABSTRAKCYJNE TYPY DANYCH UŻYTKOWNIKA

Typ obiektowy składa się z dwóch części:

- deklaracji (interfejsu): atrybuty i sygnatury metod
- definicji (ciało): implementacja metod

```
CREATE OR REPLACE TYPE Osoba AS OBJECT (
    imie VARCHAR2(20),
    nazwisko VARCHAR2(20),
    data_ur DATE,
    MEMBER FUNCTION Wiek RETURN NUMBER
);
```

```
CREATE OR REPLACE TYPE BODY Osoba AS
    MEMBER FUNCTION Wiek RETURN NUMBER IS
        w NUMBER;
    BEGIN
        w := EXTRACT (YEAR from SYSDATE)
            - EXTRACT (YEAR from data_ur);
        RETURN w;
    END;
END;
```

TYPY OBIEKTOWE W TABELACH BD

- Obiekty składuje się trwale w dwóch postaciach:
 - obiekty krotkowe (wierszowe),
 - obiekty atrybutowe (kolumnowe).
- Do przechowywania obiektów krotkowych wykorzystuje się tabele obiektowe.
- Obiekty kolumnowe są przechowywane w kolumnach zwyczajnej tabeli.

OBIEKTY ATRYBUTOWE

- Definicja typu obiektowego

```
CREATE TYPE TProdukt AS OBJECT (  
    nazwa VARCHAR(20),  
    cena NUMBER(4,2),  
    waga NUMBER(4,2)  
);
```

- Utworzenie tabeli z obiektami atrybutowymi

```
CREATE TABLE ProduktyTab (  
    produkt TProdukt,  
    symbolM VARCHAR(3)  
);
```

- Wstawienie obiektu atrybutowego do tabeli

```
INSERT INTO ProduktyTab(produkt, symbolM)  
VALUES (TProdukt('cukier',2.3,1), 'M01');
```

- Dostęp obiektów atrybutowych i ich składowych

```
SELECT p.produkt.nazwa, p.produkt.cena, symbolM  
FROM ProduktyTab p; -- wymagany alias!
```

OBIEKTY KROTKOWE

- Definicja typu obiektowego

```
CREATE TYPE TProduktK AS OBJECT (  
    nazwa VARCHAR(20),  
    cena NUMBER(4,2),  
    waga NUMBER(4,2)  
);
```

- Utworzenie tabeli obiektów

```
CREATE TABLE ProduktyTabOb  
OF TProduktK;
```

- Wstawienie obiektu krotkowego do tabeli

```
INSERT INTO ProduktyTabOb VALUES (TProduktK('chleb',2.35,1.2));
```

- Dostęp do obiektów krotkowych

```
SELECT p.nazwa FROM ProduktyTabOb p;  
SELECT * FROM ProduktyTabOb;
```

Tworzenie obiektów

- do utworzenia obiektu służy konstruktor
- nazwa konstruktora jest taka sama jak nazwa typu obiektowego
- każdy typ obiektowy posiada konstruktor atrybutowy - parametry zgodne z listą atrybutów typu obiektowego
- użytkownik może deklarować własne konstruktory, może przesłonić domyślny konstruktor atrybutowy
- tworzenia obiektów krotkowych i atrybutowych jest identyczne

IMPLEMENTACJA KONSTRUKTORA

```
CREATE OR REPLACE TYPE TProduktK AS OBJECT (  
    nazwa VARCHAR(20),  
    cena NUMBER(4,2),  
    waga NUMBER(4,2),  
    CONSTRUCTOR FUNCTION TProduktK(n VARCHAR) RETURN SELF AS RESULT  
);
```

```
CREATE OR REPLACE TYPE BODY TProduktK AS  
    CONSTRUCTOR FUNCTION TProduktK(n VARCHAR) RETURN SELF AS RESULT  
        IS BEGIN  
            nazwa := n;  cena := 1;  waga := 1; RETURN;  
        END;  
END;
```

```
CREATE TABLE ProduktyTabObK OF TProduktK;
```

```
INSERT INTO ProduktyTabObK VALUES (TProduktK('maslo'));
```

PORÓWNYWANIE WARTOŚCI OBIEKTÓW

- Dwa obiekty są równe gdy mają te same wartości składowych.
- Operatory = i != (<>) umożliwiają porównywanie obiektów.

```
SELECT p.nazwa, p.cena, p.waga  
FROM ProduktyTabOb p  
WHERE VALUE(p) <> TProdukt('cukier', 1, 1);
```

- Porównywanie obiektów najczęściej wymaga utworzenia funkcji odwzorowującej dla typu (<, >, <=, >=, BETWEEN ... AND ...).

FUNKCJA ODWZOROWUJĄCA MAP

```
CREATE OR REPLACE TYPE TTowar AS OBJECT (  
    nazwa VARCHAR(20),  
    cena NUMBER(6,2),  
    MAP MEMBER FUNCTION pobierz_cena RETURN NUMBER  
);
```

```
CREATE OR REPLACE TYPE BODY TTowar AS  
    MAP MEMBER FUNCTION pobierz_cena RETURN NUMBER IS  
    BEGIN  
        RETURN cena;  
    END;  
END;
```


FUNKCJA ODWZOROWUJĄCA MAP

```
CREATE TABLE TowaryTab0b  
OF TTowar;
```

```
INSERT INTO TowaryTab0b VALUES (TTowar('krzeslo', 202));  
INSERT INTO TowaryTab0b VALUES (TTowar('stol', 400));  
INSERT INTO TowaryTab0b VALUES (TTowar('stolek', 145));
```

```
SELECT p.nazwa, p.cena  
FROM TowaryTab0b p  
WHERE VALUE(p) BETWEEN TTowar('komoda', 200)  
                    AND TTowar('szafa', 600)  
ORDER BY VALUE(p);
```

FUNKCJA PORÓWNUJĄCA ORDER

```
CREATE OR REPLACE TYPE TTowar AS OBJECT (  
    nazwa VARCHAR(20),  
    cena NUMBER(6,2),  
    ORDER MEMBER FUNCTION porownaj(t TTowar) RETURN INTEGER  
);
```

```
CREATE OR REPLACE TYPE BODY TTowar AS  
    ORDER MEMBER FUNCTION porownaj(t TTowar) RETURN INTEGER IS  
    BEGIN  
        IF cena < t.cena THEN  
            RETURN -1;  
        ELSIF cena > t.cena THEN  
            RETURN 1;  
        ELSE  
            RETURN 0;  
        END IF;  
    END;  
END;
```

FUNKCJA PORÓWNUJĄCA ORDER

```
CREATE TABLE TowaryTabOb  
OF TTowar;
```

```
INSERT INTO TowaryTabOb VALUES (TTowar('krzeslo', 202));  
INSERT INTO TowaryTabOb VALUES (TTowar('stol', 400));  
INSERT INTO TowaryTabOb VALUES (TTowar('stolek', 145));
```

```
SELECT p.nazwa, p.cena  
FROM TowaryTabOb p  
ORDER BY VALUE(p);
```

ZAAWANSOWANE SYSTEMY BAZ DANYCH

WYKŁAD

Joanna Kapusta
e-mail: joanna.kapusta@kul.pl

Katolicki Uniwersytet Lubelski Jana Pawła II

DZIEDZICZENIE

Dziedziczenie polega na definiowaniu nowego typu obiektowego w oparciu o istniejący typ obiektowy.

```
CREATE OR REPLACE TYPE TOsoba AS OBJECT (  
    nazwisko VARCHAR2(20),  
    imie VARCHAR2(20),  
    MEMBER FUNCTION dane RETURN VARCHAR2  
) NOT FINAL;
```

```
CREATE OR REPLACE TYPE BODY TOsoba AS  
    MEMBER FUNCTION dane RETURN VARCHAR2 IS  
    BEGIN  
        RETURN imie||' '||nazwisko;  
    END;  
END;
```

DZIEDZICZENIE

```
CREATE OR REPLACE TYPE TStudent UNDER TOsoba (  
    kierunek VARCHAR2(20),  
    semestr INTEGER,  
    OVERRIDING MEMBER FUNCTION dane RETURN VARCHAR2  
);
```

```
CREATE OR REPLACE TYPE BODY TStudent AS  
    OVERRIDING MEMBER FUNCTION dane RETURN VARCHAR2 IS  
    BEGIN  
        RETURN imie||' '||nazwisko  
            ||' '||kierunek||' '||semestr;  
    END;  
END;
```

DZIEDZICZENIE

W celu uniknięcia powtarzania możliwe jest zastosowanie uogólnionego wywołania, tj. w typie podrzędnym wywołujemy metodę typu nadrzędnego.

```
CREATE OR REPLACE TYPE BODY TStudent AS
    OVERRIDING MEMBER FUNCTION dane RETURN VARCHAR2 IS
    BEGIN
        RETURN (SELF AS TOsoba).dane()
            || ' ' || kierunek || ' ' || semestr;
    END;
END;
```

POLIMORFIZM

Polimorfizm powoduje, że obiekty podtypu mogą zachowywać się jak obiekty swojego nadtypu

- zamiast obiektu nadtypu można wykorzystać obiekt podtypu
- zamiast obiektu podtypu można wymusić wykorzystanie obiektu nadtypu przez użycie operatora `TREAT`
- wybór wersji metody przeciążonej następuje w momencie wykonania programu (metody są wirtualne)

POLIMORFIZM

- Utworzenie tabeli obiektów typu TSoba
`CREATE TABLE TabOsobaObj OF TSoba;`
- Wstawienie obiektu typu TSoba do tabeli
`INSERT INTO TabOsobaObj
VALUES (TSoba('Nowak', 'Adam'));`
- Wstawienie obiektu typu TStudent do tabeli
`INSERT INTO TabOsobaObj
VALUES (TStudent('Kowalski', 'Marek',
 'informatyka', 3));`

FUNKCJE IS OF I TREAT

- IS OF umożliwia sprawdzenie czy obiekt jest podanego typu (lub podtypu)

```
SELECT VALUE(o) FROM TabOsobaObj o  
WHERE VALUE(o) IS OF (TStudent);
```

```
SELECT VALUE(o) FROM TabOsobaObj o  
WHERE VALUE(o) IS OF (ONLY TOsoba);
```

- TREAT umożliwia sprawdzenie czy obiekt może być traktowany jako obiekt określonego typu

```
SELECT NVL2(TREAT(VALUE(o) AS TStudent), 'TAK', 'NIE')  
FROM TabOsobaObj o;
```

ZAAWANSOWANE SYSTEMY BAZ DANYCH

WYKŁAD

Joanna Kapusta
e-mail: joanna.kapusta@kul.pl

Katolicki Uniwersytet Lubelski Jana Pawła II

KOLEKCJE

Kolekcje są zbiorami danych tego samego typu.

Typy kolekcji:

- tablica o zmiennej długości - varray,
- tablica zagnieżdżona - nested,
- tablica asocjacyjna - associative.

KOLEKCJE

Kolekcje mogą być tworzone:

- na poziomie SQL - tworzone na stałe w bazie danych
- na poziomie PL/SQL - lokalne deklaracje na potrzeby programu

TABLICE VARRAY

Tablice VARRAY:

- zbiór uporządkowany,
- każdy element ma indeks określający jego pozycję w tablicy,
- zdefiniowany maksymalny rozmiar
- numeracja elementów tablicy od 1 (bez dziur)

```
TYPE nazwa_typu IS VARRAY (rozmiar) OF typ_elementu;  
identyfikator nazwa_typu; -deklaracja  
identyfikator:=nazwa_typu();
```

DECLARE

```
TYPE t_liczb IS VARRAY(3) OF INTEGER;  
liczby t_liczb:= t_liczb(NULL,NULL,NULL);
```

BEGIN

```
liczby(1):=100;  
liczby(2):=115;
```

TWORZENIE KOLEKCJI - SQL

- Deklaracja tablic:

```
CREATE TYPE nazwa_typu AS VARRAY(rozmiar)  
OF typ_danych;
```

Modyfikowanie liczby elementów tablicy

```
ALTER TYPE nazwa_typu  
MODIFY LIMIT nowa_liczba_elem;
```

- Deklaracja tablic zanieżdżonych

```
CREATE TYPE nazwa_typu AS TABLE  
OF typ_danych;
```

UŻYCIE KOLEKCJI VARRAY

- Zdefiniowanie tabeli z kolumną typu VARRAY

```
CREATE TYPE t_vtab_miast AS VARRAY(3)  
OF VARCHAR2(20);
```

```
ALTER TYPE t_vtab_miast MODIFY LIMIT 10;
```

```
CREATE TABLE PanstwaV(  
id INTEGER PRIMARY KEY,  
nazwa VARCHAR2(20),  
miasta t_vtab_miast  
);
```

- Wstawienie danych

```
INSERT INTO PanstwaV  
VALUES(1, 'Polska', t_vtab_miast('Warszawa', 'Lublin', 'Krakow'));  
INSERT INTO PanstwaV  
VALUES(2, 'Wielka Brytania', t_vtab_miast('Londyn', 'Leeds'));
```


TABLICE ZAGNIEŻDŻONE

- zbiór nieuporządkowany,
- rozmiar dynamiczny,
- nie ma określonego maksymalnego rozmiaru,
- można modyfikować/usuwać/wstawiać pojedyncze elementy,
- dostęp do pojedynczych elementów tablicy.

```
TYPE nazwa_typu IS TABLE OF typ_elementu;  
identyfikator nazwa_typu; -deklaracja  
identyfikator:=nazwa_typu();
```

PRZYKŁAD

```
DECLARE
TYPE t_name IS TABLE OF VARCHAR2(10);
TYPE t_age IS TABLE OF INTEGER;
names t_name;
age t_age;
BEGIN
names := t_name('Ola', 'Ala', 'Iwona');
age:= t_age(36, 12, 28);
END;
```

UŻYCIE TABLICY ZAGNIEŹDŻONEJ

- Zdefiniowanie tabeli z kolumną będącą tablicą zagnieżdżoną

```
CREATE TYPE t_nest_tab AS TABLE  
OF VARCHAR2(20);
```

```
CREATE TABLE Panstwa(  
  id INTEGER PRIMARY KEY,  
  nazwa VARCHAR2(20),  
  miasta t_nest_tab  
)  
NESTED TABLE miasta  
STORE AS nested_miasta;
```

- Wstawienie danych

```
INSERT INTO Panstwa  
VALUES(1, 'Hiszpania', t_nest_tab('Madryt', 'Barcelona'));  
INSERT INTO Panstwa  
VALUES(2, 'Włochy', t_nest_tab('Rzym', 'Mediolan', 'Neapol'));
```

FUNKCJA TABLE

Funkcja TABLE pozwala zinterpretować kolekcję jako serię wierszy.

- z tablicą o zmiennej długości

```
SELECT m.*  
FROM PanstwaV p, TABLE(p.miasta) m;
```

- tablicą zagnieżdżoną

```
SELECT m.*  
FROM PanstwaN p, TABLE(p.miasta) m;
```

MODYFIKOWANIE ELEMENTÓW KOLEKCJI VARRAY

Tablica VARRAY może być modyfikowana tylko jako całość, tzn. że modyfikując jeden element musimy przekazać całą tablicę.

```
UPDATE PanstwaV  
SET miasta  
    = t_vtab_miast('Warszawa', 'Gdansk', 'Gdynia')  
WHERE id = 1;
```

MODYFIKOWANIE ELEMENTÓW TABLICY ZAGNIEŹDŻONEJ

W tablicach zagnieżdżonych - możemy modyfikować pojedyncze elementy.

- dodanie nowego elementu

```
INSERT INTO TABLE(  
  SELECT miasta FROM PanstwaN WHERE id = 2)  
VALUES('Bolonia');
```

- modyfikowanie pojedynczego elementu

```
UPDATE TABLE(  
  SELECT miasta FROM PanstwaN WHERE id = 2) m  
SET VALUE(m) = 'Turyn'  
WHERE VALUE(m) = 'Bolonia';
```

- usuwanie elementu

```
DELETE TABLE(  
  SELECT miasta FROM PanstwaN WHERE id = 2) m  
WHERE VALUE(m) = 'Neapol';
```

METODY KOLEKCJI

- COUNT - zwraca liczbę elementów kolekcji
- EXISTS(index) - zwraca true jeśli w kolekcji istnieje element o podanym indeksie
- DELETE: DELETE - usuwa wszystkie elementy, DELETE(index) - usuwa element o podanym indeksie, DELETE(p,k)- usuwa elementy o indeksach od p do k
- FIRST - zwraca indeks pierwszego elementu kolekcji (kolekcja pusta wynikiem jest null); w tablicy zagnieżdżonej pojedyncze elementy mogą być puste - metoda zwraca najmniejszy indeks niepustego elementu w tablicy zagnieżdżonej
- LAST - jw. tylko ostatni

METODY KOLEKCJI

- `NEXT(index)` - zwraca indeks elementu znajdującego się za elementem o podanym indeksie
- `PRIOR(index)` - jw. tylko przed elementem o podanym indeksie
- `LIMIT` - null dla tabel zagnieżdżonych, dla `VARRAY` - maksymalną liczbę elementów
- `EXTEND` - wstawia element na końcu kolekcji (`EXTEND` - jeden element o wartości null, `EXTEND(n)` - n elementów o wartości null, `EXTEND(n, index)` - n elementów o wartości skopiowanej z elementu o podanym indeksie)
- `TRIM` - usuwa elementy z końca kolekcji (`TRIM` - usuwa jeden element, `TRIM(n)` - usuwa n elementów)

TABLICE ASOCJACYJNE

Tablice asocjacyjna:

- nie wymagają inicjowania oraz konstruktora,
- indeksuje się je głównie liczbami (ewentualnie łańcuchami znaków),
- dynamiczny rozmiar tablicy,
- tablica składa się z dwóch kolumn: klucza oraz wartości.

TABLICE ASOCJACYJNE

Deklaracja tablic asocjacyjnych

```
TYPE nazwa IS TABLE OF typ_elementow
INDEX BY {PLS_INTEGER|BINARY_INTEGER|VARCHAR2(n)};

DECLARE
TYPE Ttab_nazwisk IS TABLE OF employees.last_name%TYPE
INDEX BY PLS_INTEGER;

t_nazwisk Ttab_nazwisk;

BEGIN
t_nazwisk(1):='Kowalski';
...
END;
```

TABLICE ASOCJACYJNE

```
create table prac(  
nazwisko  VARCHAR2(25)  
);  
  
DECLARE  
TYPE Ttab_nazwisk IS TABLE OF  
    prac.nazwisko%TYPE INDEX BY PLS_INTEGER;  
tab_naz  Ttab_nazwisk;  
i INTEGER := 1;  
BEGIN  
    tab_naz(1)      := 'Nowak';  
    tab_naz(2)      := 'Kowalski';  
    WHILE tab_naz.EXISTS(i) LOOP  
        insert into prac VALUES (tab_naz(i));  
        i := i + 1;  
    END LOOP;  
END;
```

TABLICE ASOCJACYJNE

```
DECLARE
    TYPE Ttab_prac is table of
        employees%ROWTYPE INDEX BY PLS_INTEGER;
    t_prac Ttab_prac;
    ile NUMBER(3) := 125;
BEGIN
    FOR i IN 120..ile
    LOOP
        SELECT * INTO t_prac(i)
        FROM employees
        WHERE employee_id = i;
    END LOOP;
    FOR i IN t_prac.FIRST..t_prac.LAST
    LOOP
        DBMS_OUTPUT.PUT_LINE(i||' '||t_prac(i).first_name
                               ||' '||t_prac(i).last_name);
    END LOOP;
END;
```

OPERACJE MASOWE

- operujemy wieloma zmiennymi jednocześnie jak jednym elementem,
- są znacznie wydajniejsze od wykonywanych w petli pojedynczych odczytów i zapisów,
- instrukcja FORALL
- klauzula BULK COLLECT

INSTRUKCJA FORALL

- służy do wykonywania instrukcji DML na podstawie zawartosci kolekcji
- znacznie skraca zapis

```
DECLARE
    TYPE Tnum_list IS VARRAY(20) OF NUMBER;
    k Tnum_list := Tnum_list(10, 20, 30, 40, 50);
BEGIN
    ...
FORALL j IN 3..5
    DELETE FROM de WHERE department_id = k(j);
end;
END;
```

KLAUZULA BULK COLLECT

- służy do odczytu wyniku zapytania dotyczącego kolekcji w jednej operacji
- może być stosowana w poleceniu SELECT
SELECT ... BULK COLLECT INTO ...
- może być stosowana w poleceniu FETCH
FETCH kursor BULK COLLECT INTO ...

KLAUZULA BULK COLLECT

```
DECLARE
    TYPE T_tab_dept IS TABLE OF departments%ROWTYPE;

    t_dept  T_tab_dept;
BEGIN

    SELECT *
    BULK COLLECT INTO t_dept
    FROM  departments;
    ...

END;
```


ZAAWANSOWANE SYSTEMY BAZ DANYCH

WYKŁAD

Joanna Kapusta
e-mail: joanna.kapusta@kul.pl

Katolicki Uniwersytet Lubelski Jana Pawła II

JSON

JSON (ang. JavaScript Object Notation)

- tekstowy format wymiany danych
- często wykorzystywany do przesyłania danych między systemami
- może zawierać dane konfiguracyjne na potrzeby aplikacji
- niezależny od języka, ale używający konwencji z innych języków np. C, C++, Java, JavaScript, Perl, Python, ...

Jaki typ dla danych typu JSON?

- VARCHAR2 dla małych dokumentów - poniżej 4000 bajtów (lub 32 767 bajtów dla baz z rozszerzonymi typami danych).
- BLOB dla większych dokumentów (CLOB zajmuje więcej miejsca, wymagana konwersja znaków).
- JSON od Oracle Database 21c (zoptymalizowany pod kątem przetwarzania zapytań i DML).

TWORZENIE TABELI

```
CREATE TABLE jobs_json (  
    id INTEGER GENERATED BY DEFAULT  
        AS IDENTITY (START WITH 100 INCREMENT BY 1),  
    job_id VARCHAR2(30),  
    json_data BLOB  
);
```

WALIDACJA POPRAWNOŚCI JSON-A

```
ALTER TABLE jobs_json  
ADD CONSTRAINT job_data_json  
CHECK (json_data IS JSON);
```

```
INSERT INTO jobs_json(job_id, json_data)  
VALUES ('JOB', utl_raw.cast_to_raw ( 'JSON...' ) );
```

ORA-02290: naruszono więzy CHECK (LAB01.JSON_DATA)

WSTAWIANIE DANYCH

```
INSERT INTO jobs_json(job_id, json_data)
VALUES ('IT_PROG', utl_raw.cast_to_raw (
'{ "job":"Programmer",
  "employees":[
    { "name":"Jones Alexander",
      "city":"Rome",
      "date":"2006-01-03T00:00:00",
      "salary":18000
    },
    { "name":"Smith Diana",
      "city":"Lublin",
      "date":"2007-02-07T00:00:00",
      "salary":14200
    } ]
}' ));
```

MODYFIKACJA - UPDATE

```
UPDATE jobs_json
SET json_data = utl_raw.cast_to_raw (
'{ "job":"Developer",
  "employees":[
    { "name":"Jones Alexander",
      "city":"Rome",
      "date":"2006-01-03T00:00:00",
      "salary":18000
    },
    { "name":"Smith Diana",
      "city":"Lublin",
      "date":"2007-02-07T00:00:00",
      "salary":14200
    } ]
}') where id = 108;
```


POBIERANIE DANYCH - SELECT

```
SELECT j.json_data.job  
FROM jobs_json j;
```

| JOB |
|------------|
| Developer |
| Programmer |


POBIERANIE DANYCH - SELECT

```
SELECT j.json_data.job  
FROM jobs_json j;
```



| JOB |
|------------|
| Developer |
| Programmer |

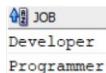
```
SELECT j.json_data.employees[0].name  
FROM jobs_json j  
WHERE id = 106;
```



| EMPLOYEES |
|-----------------|
| Jones Alexander |


POBIERANIE DANYCH - SELECT

```
SELECT j.json_data.job  
FROM jobs_json j;
```



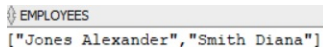
| JOB |
|------------|
| Developer |
| Programmer |

```
SELECT j.json_data.employees[0].name  
FROM jobs_json j  
WHERE id = 106;
```



| EMPLOYEES |
|-----------------|
| Jones Alexander |

```
SELECT j.json_data.employees[*].name  
FROM jobs_json j  
WHERE id = 106;
```



| EMPLOYEES |
|------------------------------------|
| ["Jones Alexander", "Smith Diana"] |

JSON_VALUE

JSON_VALUE - odczytuje wartość

```
SELECT JSON_VALUE (  
    json_data,  
    '$.employees[1].date' returning date  
    ) hire_date  
FROM jobs_json;
```

| HIRE_DATE |
|-----------|
| 07/02/07 |
| 07/02/07 |

JSON_VALUE - BRAK ATRYBUTU

NULL ON ERROR - domyślnie

```
SELECT JSON_VALUE (  
    json_data,  
    '$.nieIstnieacyAtrybut' returning date  
    ) hire_date  
FROM jobs_json;
```

| |
|-------------|
| ◆ HIRE_DATE |
| (null) |

JSON_VALUE - BRAK ATRYBUTU

NULL ON ERROR - domyślnie

```
SELECT JSON_VALUE (
    json_data,
    '$.nieIstnieacyAtrybut' returning date
) hire_date
FROM jobs_json;
```

| |
|----------------------------------|
| <div></div> <div>HIRE_DATE</div> |
| (null) |

ERROR ON ERROR

```
SELECT JSON_VALUE (
    json_data,
    '$.nieIstnieacyAtrybut' returning date
    ERROR ON ERROR
) hire_date
FROM jobs_json;
```

ORA-40462: JSON_VALUE evaluated to no value

JSON_QUERY

JSON_QUERY - odczytuje dokument, tablicę

- WITH WRAPPER — wszystkie wartości pasujące do ścieżki,
- WITHOUT WRAPPER — pojedynczy obiekt lub tablica, która pasuje do wyrażenia ścieżki (błąd, jeśli wyrażenie ścieżki pasuje do wartości skalarnej lub więcej niż jednej wartości)
- PRETTY - ładnie sformatowany ciąg znaków (wiersze, wcięcia).

Tablica pracowników:

```
SELECT JSON_QUERY(  
    json_data,  
    '$.employees[*]',  
    RETURNING VARCHAR2 PRETTY  
    WITH WRAPPER  
    ) employees  
FROM jobs_json d  
WHERE id = 106;
```

```
[  
  {  
    "name": "Jones Alexander",  
    "city": "Rome",  
    "date": "2006-01-03T00:00:00",  
    "salary": 18000  
  },  
  ]
```

PRETTY

```
SELECT JSON_QUERY(json_data, '$')  
FROM jobs_json;
```

```
{ "job": "Programmer", "employees": [ { "name": "Jones Alexander", "city": "Rome", "date": "2006-01-03T00:00:00", "salary": 18000 }, { "name": "Smith Diana", "city": "Lublin", "date": "2007-02-07T00:00:00", "salary": 14200 } ] }
```

```
SELECT JSON_QUERY(json_data, '$' pretty)  
FROM jobs_json;
```

```
{  
  "job": "Programmer",  
  "employees": [  
    {  
      "name": "Jones Alexander",  
      "city": "Rome",  
      "date": "2006-01-03T00:00:00",  
      "salary": 18000  
    },  
    {  
      "name": "Smith Diana",  
      "city": "Lublin",  
      "date": "2007-02-07T00:00:00",  
      "salary": 14200  
    }  
  ]  
}
```

JSON_TABLE

JSON_TABLE - umożliwia przekształcenie danych JSON w wiersze
(argumenty: dokument JSON oraz klauzula kolumn).

```
SELECT t.*
FROM jobs_json j, JSON_TABLE (j.json_data, '$' columns (
  job path '$.job',
  nested path '$.employees[*]'
    columns (
      name path '$.name',
      salary path '$.salary',
      hire_date path '$.date',
      city path '$.city'
    ) ) ) t
where j.id = 106;
```

| JOB | NAME | SALARY | HIRE_DATE | CITY |
|------------|-----------------|--------|---------------------|--------|
| Programmer | Jones Alexander | 18000 | 2006-01-03T00:00:00 | Rome |
| Programmer | Smith Diana | 14200 | 2007-02-07T00:00:00 | Lublin |

Klauzula kolumn:

- atrybuty na poziomie stanowiska (\$.job),
- zagnieżdżona ścieżka, zwracająca tablicę pracowników (employees[*]) z listą atrybutów pracowników.

WYSZUKIWANIE

- wyszukiwanie danych

```
SELECT *  
FROM jobs_json  
WHERE JSON_VALUE(json_data, '$.job') = 'Programmer';
```

- w którym dokumencie istnieje stanowisko Programmer

- wyszukiwanie atrybutów

```
SELECT id  
FROM jobs_json j  
WHERE JSON_EXISTS(json_data, '$.employees[*].salary');
```

- w którym dokumencie istnieje atrybut salary

JSON JAKO RELACJA - 1:1

Utworzenie indeksu wyszukiwania z opcją DATAGUIDE ON

```
CREATE SEARCH INDEX jobs_json_i ON
jobs_json (json_data )
FOR json;
```

```
ALTER INDEX jobs_json_i
REBUILD PARAMETERS ('dataguide on');
```

Dodanie wirtualnej kolumny do tabeli

```
EXEC dbms_json.add_virtual_columns
('jobs_json', 'json_data');
SELECT * FROM jobs_json;
```

| ID | JOB_ID | JSON_DATA | JSON_DATA\$job |
|-----|----------|-----------|----------------|
| 106 | IT_PROG | (BLOB) | Programmer |
| 107 | IT_PROG2 | (BLOB) | Programmer |

JSON JAKO RELACJA - 1:N

Uworzenie widoku (1:n - jedno stanowisko wielu pracowników)

BEGIN

```

dbms_json.create_view (
    'job_employees', 'jobs_json',
    'json_data',
    dbms_json.get_index_dataguide (
        'jobs_json',
        'json_data',
        dbms_json.format_hierarchical
    )
);

```

END;

SELECT * FROM job_employees;

| ID | JOB_ID | JSON_DATA\$job | JSON_DATA\$city | JSON_DATA\$date | JSON_DATA\$name | JSON_DATA\$salary |
|-----|---------|----------------|-----------------|---------------------|-----------------|-------------------|
| 106 | IT_PROG | Programmer | Rome | 2006-01-03T00:00:00 | Jones Alexander | 18000 |
| 106 | IT_PROG | Programmer | Lublin | 2007-02-07T00:00:00 | Smith Diana | 14200 |

GENEROWANIE JSON-A

- JSON_OBJECT tworzy obiekty JSON na podstawie wyrażeń
- JSON_ARRAY tworzy tablicę JSON na podstawie wyrażeń
- JSON_OBJECTAGG tworzy obiekt JSON, agregując informacje z wielu wierszy zgrupowanego zapytania
- JSON_ARRAYAGG tworzy tablicę JSON, agregując informacje z wielu wierszy zgrupowanego zapytania SQL. Kolejność elementów tablicy odzwierciedla kolejność wyników zapytania (można użyć klauzuli ORDER BY).

GENEROWANIE JSON-A

```

SELECT JSON_OBJECT(
    'job' VALUE j.job_title,
    'employees' VALUE JSON_ARRAYAGG (
        JSON_OBJECT(
            'name' VALUE last_name || ', ' || first_name,
            'salary' VALUE salary,
            'date' VALUE hire_date
        )
    )
)
FROM employees e
JOIN jobs j
ON e.job_id = j.job_id
WHERE j.job_id = 'IT_PROG'
GROUP BY j.job_title;

```

```

JSON_OBJECT('JOB'VALUEJ.JOB_TITLE,'EMPLOYEES'VALUEJSON_ARRAYAGG(JSON_OBJECT('NAME'VALUELAST_NAME||','||FIRST_NAME,'SALARY'VALUESALARY,'DATE'VALUEHIRE_DA
{"job":"Programmer","employees":[{"name":"Hunold, Alexander","salary":9900,"date":"2006-01-03T00:00:00"}, {"name"

```