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		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 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		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 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';
```

Imie	Nazwisko
Janina	Nowak
Mariusz	Perkob

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

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;
```

Nazwa
Kadr
Inwestycji
Informatyczny
Finansowy
Administracyjny

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

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

1. `COUNT` - ilość wyrażeń w kolumnie,
2. `SUM` - suma wyrażeń w kolumnie,
3. `AVG` - średnia z wyrażeń w kolumnie,
4. `MIN` - najmniejsza wartość w kolumnie,
5. `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
FROM nazwa_tabela1 JOIN nazwa_tabela2
ON warunek_złączenia;
```

```
SELECT lista_kolumn
FROM nazwa_tabela1 JOIN nazwa_tabela2
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		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, 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 ROOLUP

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 ROOLUP

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 ROOLUP

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		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 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 ROOLUP

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 ROOLUP

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

GROUPING(Stanowisko)	Stanowisko	SUM(Pensja)
0	programista	9800
0	sekretarka	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 Odzial,
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 Odzial,
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 całościowe

```
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)
```

jest równoważne

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

$n + 1$ - grupowań

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

jest równoważne

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

Kompozycja kolumn

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

jest równoważne

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

2^n - grupowań

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

jest równoważne

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

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 $0, .., n-1$, 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 `GROUPING(a) = 1`, `GROUPING(b) = 0`, `GROUPING(c) = 1` wtedy `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 $\leqslant 1$)
- `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	2400C
Neena Kochhar	AD_VP	17000	2	3400C
Lex De Haan	AD_VP	17000	2	3400C
John Chen	FI_ACCOUNT	8200	5	3960C
Daniel Faviet	FI_ACCOUNT	9000	5	3960C
Ismael Sciarra	FI_ACCOUNT	7700	5	3960C
Jose Manuel Urman	FI_ACCOUNT	7800	5	3960C

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

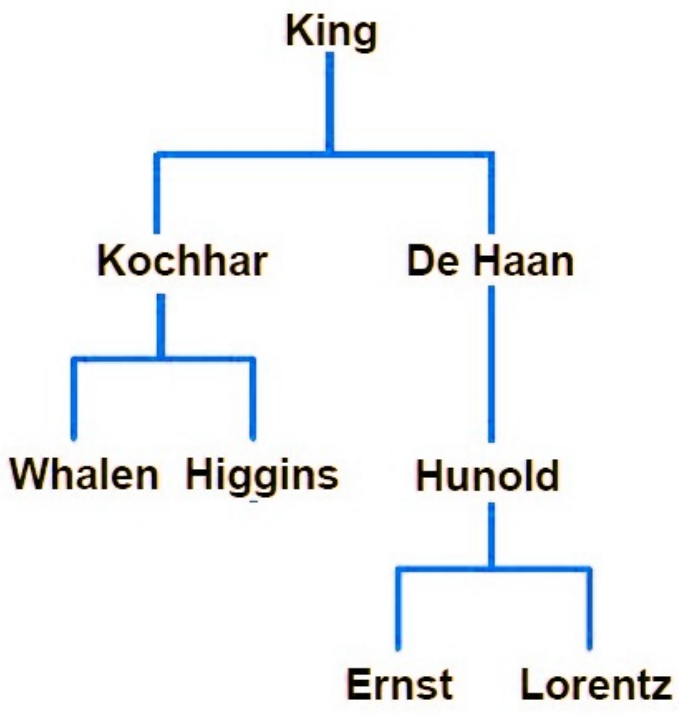
Joanna Kapusta
e-mail: joanna.kapusta@kul.pl

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 ...
CONECT 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 selekcjonujący 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

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.

- Jądrem 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 niezmienione, 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, pozwalajace na zidentyfikowanie klientów, którzy mogą odejść
- zarządzanie zasobami - posiadanie właściwych towarów, we właściwym miejscu i czasie

Business Intelligence

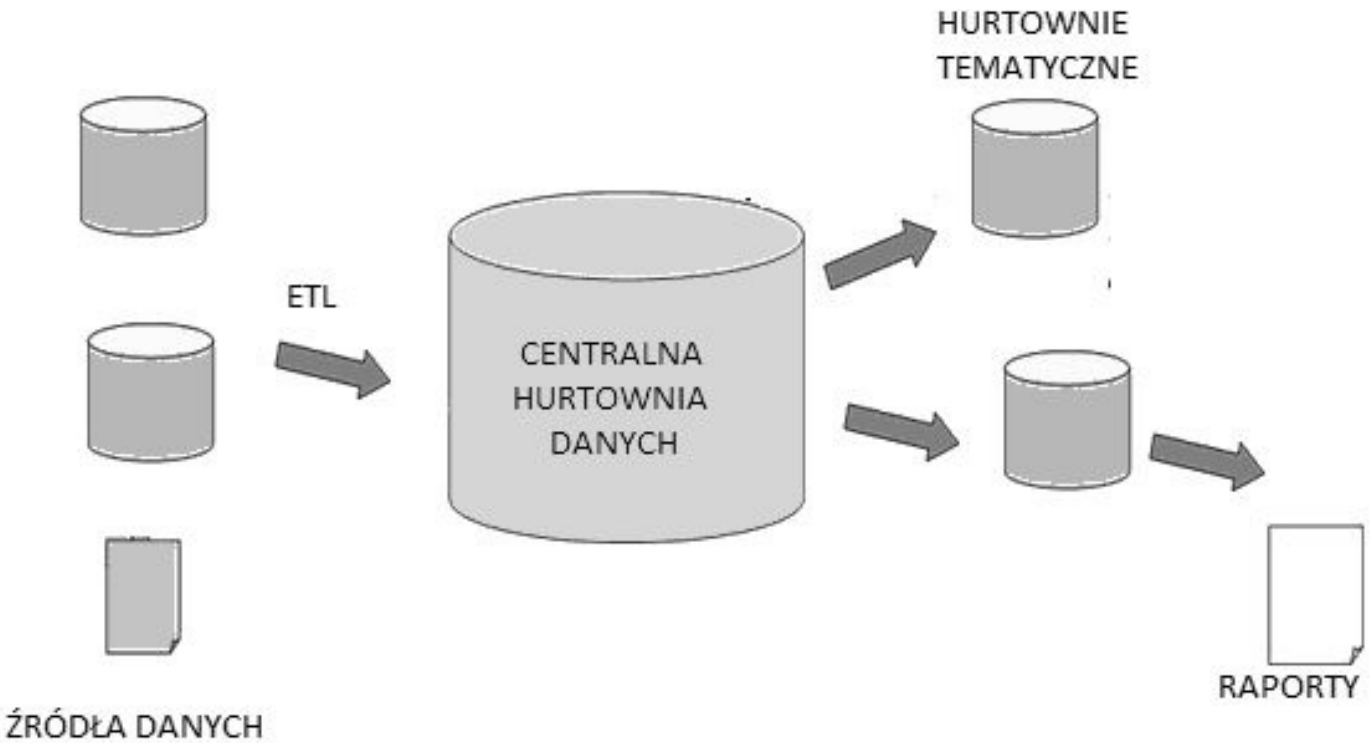
Implementacja systemu BI:

- 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 ilościowo 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

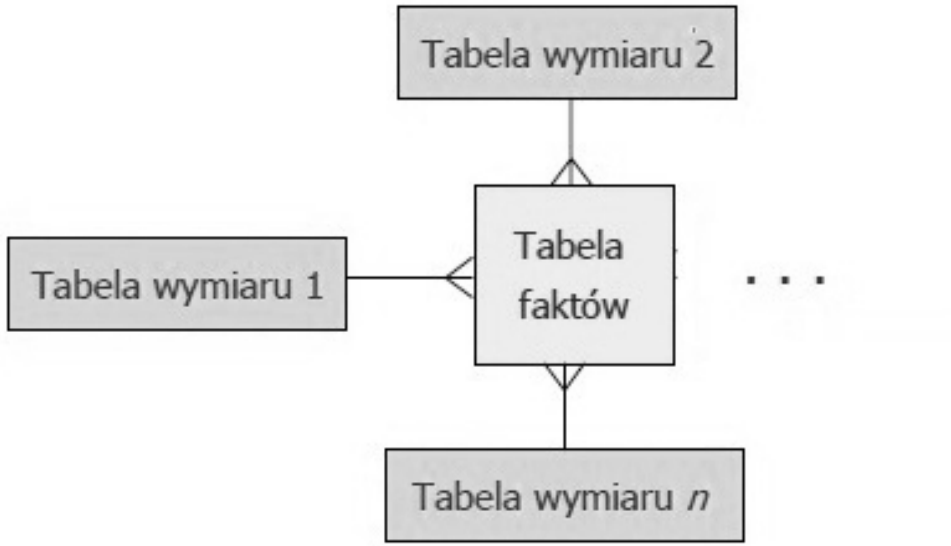
- ROLAP (relacyjny)
  - dane przechowywane w tabelach relacyjnych (przy czym schemat BD zaprojektowany jest tak aby odzwierciedlić wielowymiarową strukturę danych)
  - fakty przechowywane w tabelach faktów
  - wymiary przechowywane w tabelach wymiarów
- MOLAP (wielowymiarowy) - dane przechowywane w wielowymiarowych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

Implementacja ROLAP

Schematy logiczne:

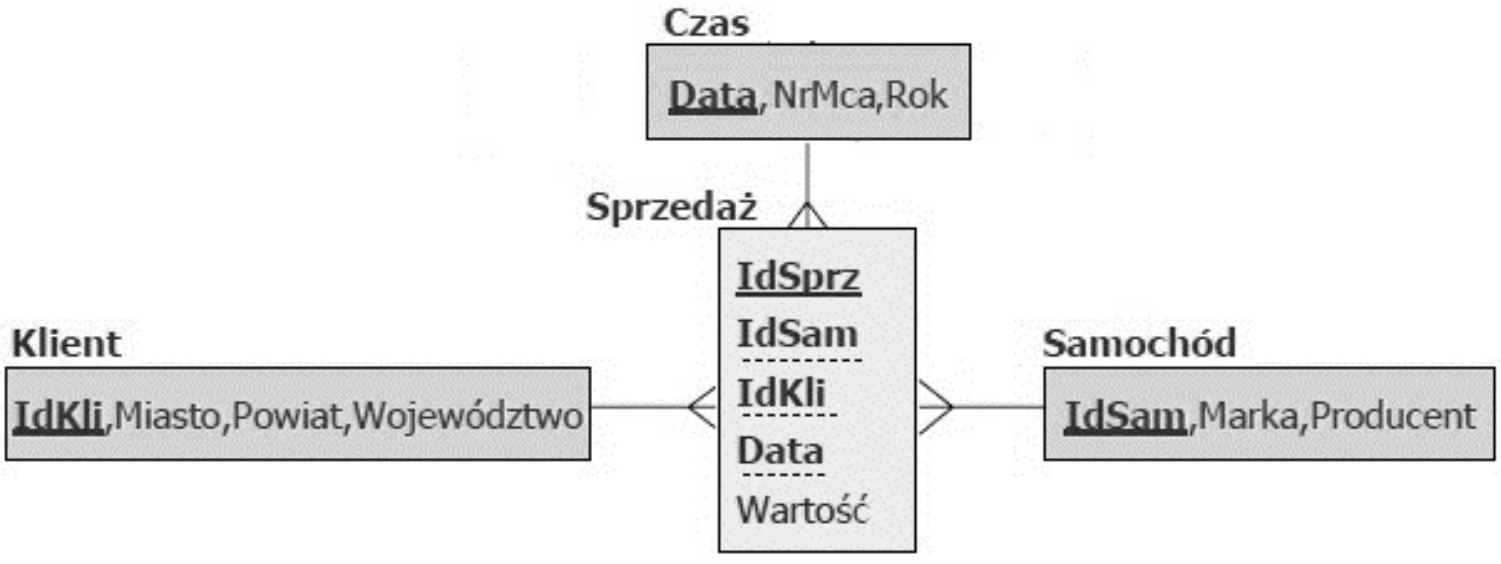
- schemat gwiazdy (ang. star schema)
- schemat płatka śniegu (ang. snowflake schema)
- konstelacja faktów - schemat gwiazdy lub płatka śniegu, w którym ten sam wymiar jest powiązany z wieloma tabelami faktów

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 płatka śniegu)
- tabela faktów zawiera klucze obce (powiązane z tabelami wymiarów) oraz atrybuty zwane miarami

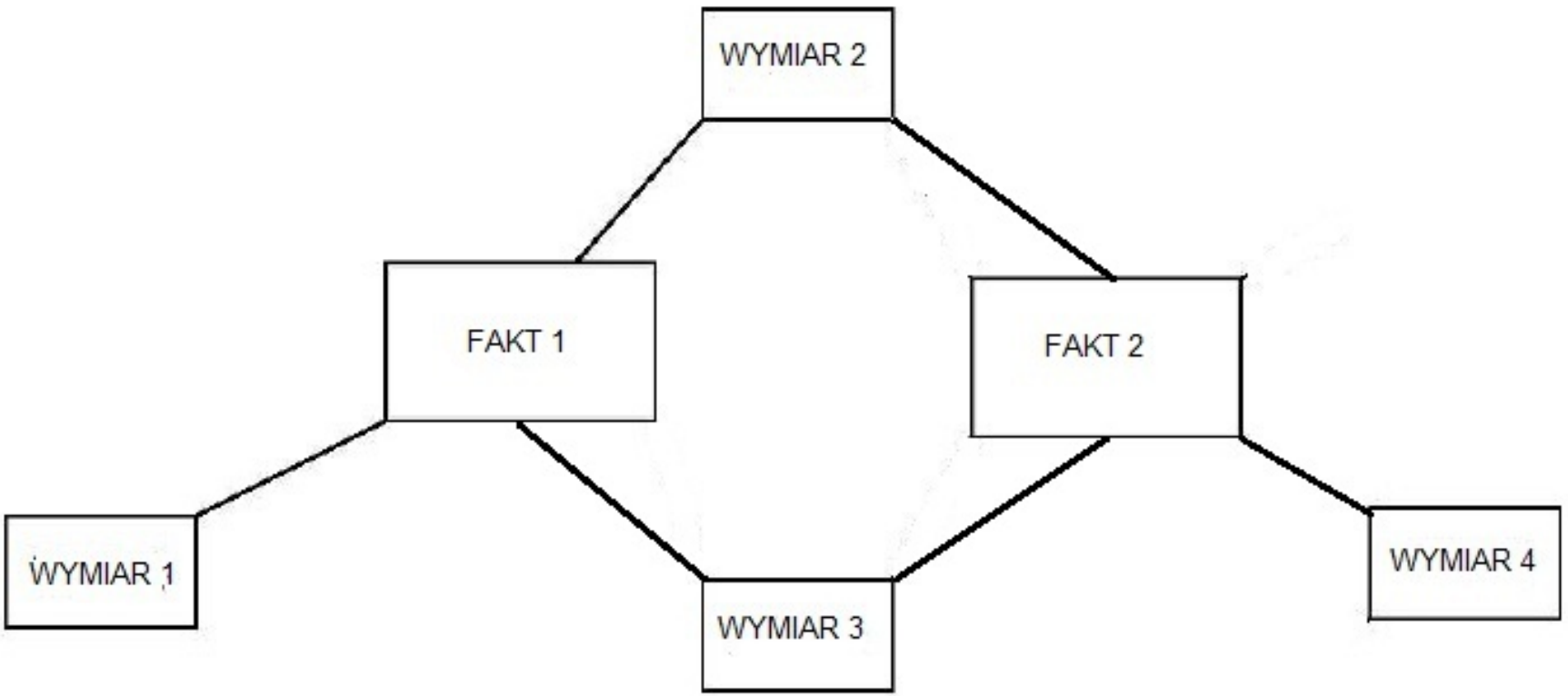
Schemat gwiazdy - przykład



Schemat płatka śniegu:

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

Schemat konstelacji faktów



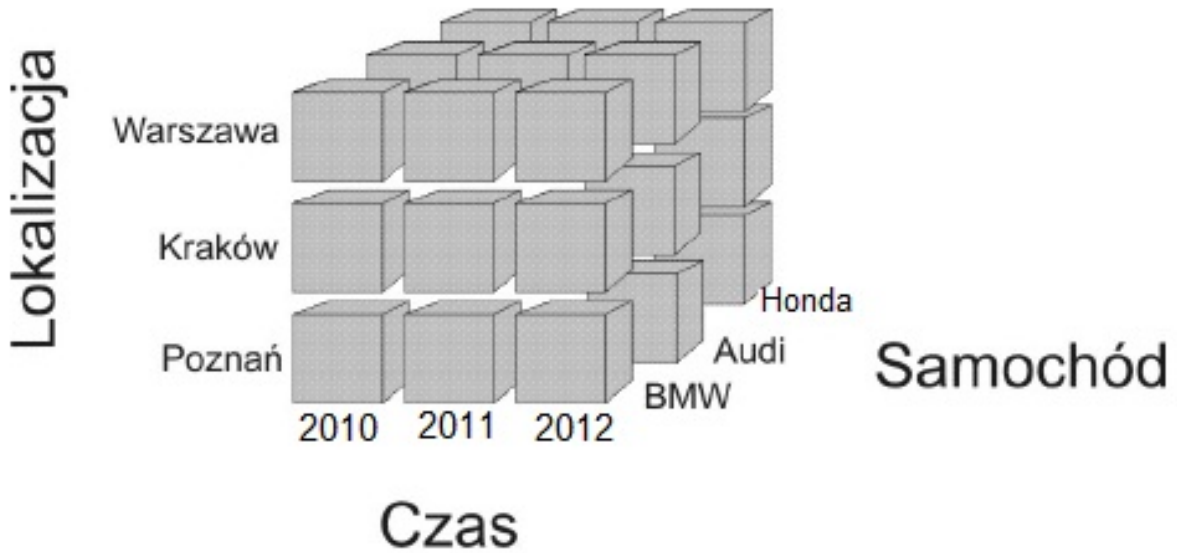
Schemat będący kombinacją schematów gwiazd, które współdzielą niektóre wymiary.

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 zasilenie 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.Chądzyńska-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)

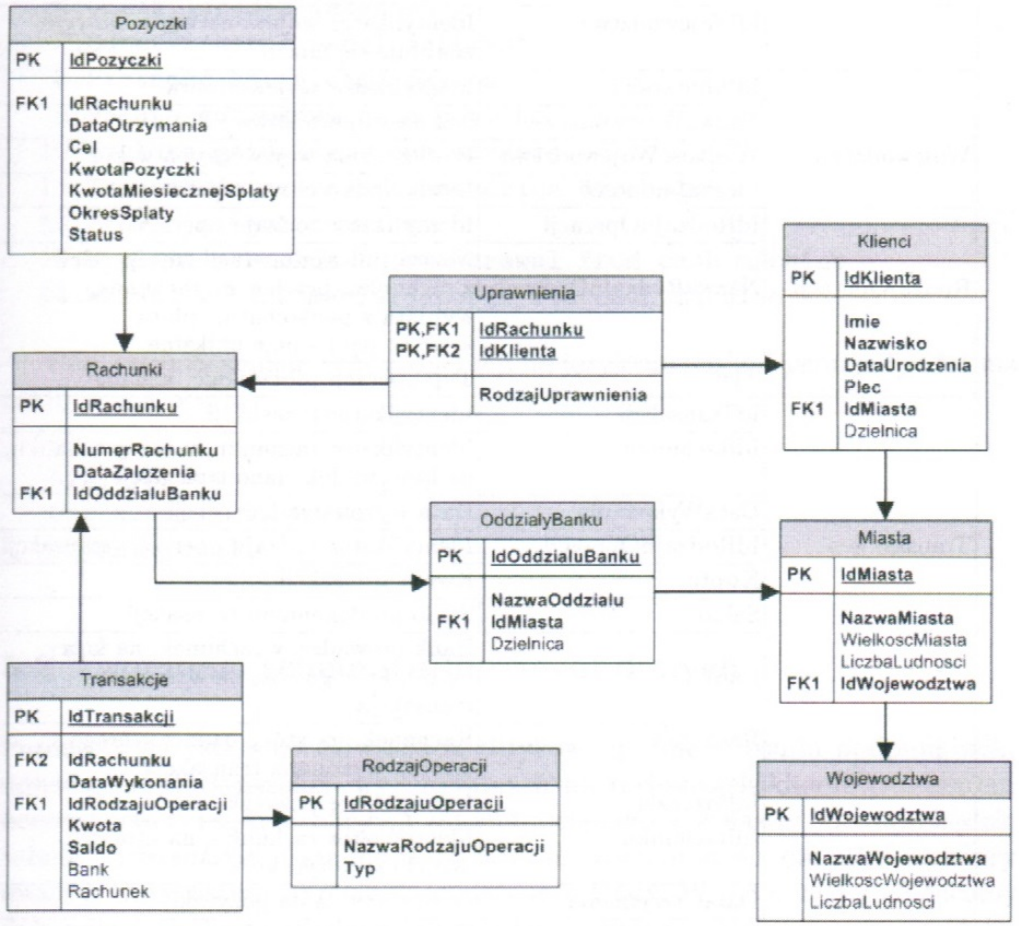
Raporty

- 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

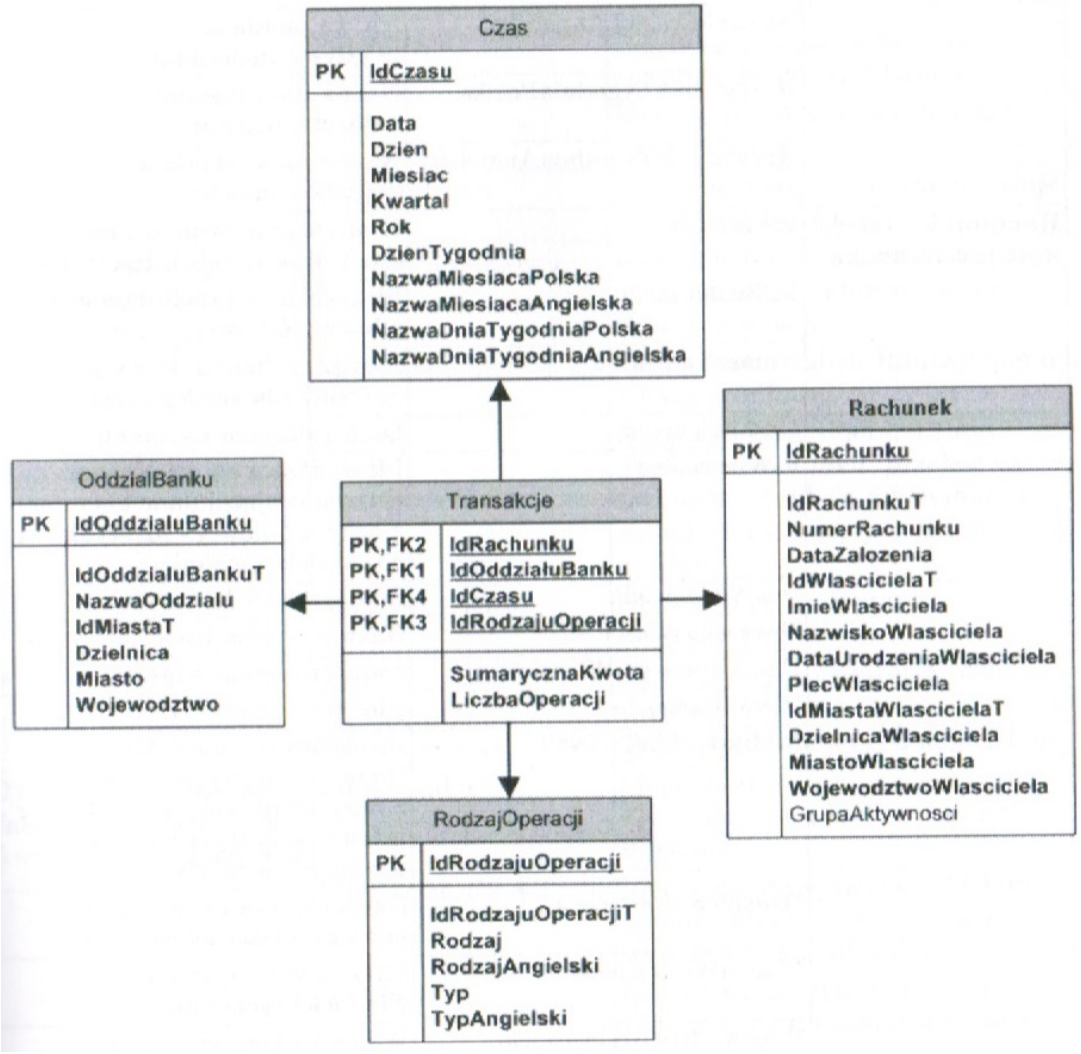


- pliki z tłumaczeniami nazw typów i rodzajów operacji na język angielski
  1; Przelew z rachunku; transfer from account
  2; Przelew na rachunek; transfer into account

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) - wydobycie 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 podzapytaniami, 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)
VALUES (1234, 'Handlowy');
```

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

Wstawianie danych z wykorzystaniem podzapytań

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

- wstawia 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 waruneku 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 zewnetrzne).

```
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')
```