



Język SQL

wersja 1.81

| | |
|---|-----------|
| 1. Uwagi wstępne..... | 3 |
| 2. Podstawowe grupy poleceń | 4 |
| 3. Składnia polecenia SELECT (uproszczona)..... | 4 |
| 4. Polecenie SELECT..... | 5 |
| 4.1. Najprostsze przykłady..... | 5 |
| 4.2. Klauzula ORDER BY | 7 |
| 4.3. Klauzula WHERE..... | 9 |
| 4.4. Operatory..... | 11 |
| 4.5. Aliasy | 12 |
| 4.6. Wyrażenia..... | 13 |
| 4.7. Wartości puste (NULL) | 14 |
| 4.8. Eliminowanie duplikatów..... | 16 |
| 4.9. Funkcje agregujące | 16 |
| 4.10. Klauzula GROUP BY | 17 |
| 4.11. Klauzula HAVING | 18 |
| 4.12. Złączanie tabel | 19 |
| 4.12.1. Iloczyn kartezjański (ang. <i>Cross Join</i>) | 19 |
| 4.12.2. Złączenia równościowe (ang. <i>Equi Join</i> lub <i>Inner Join</i>)..... | 20 |
| 4.12.3. Złączenia nierównościowe (ang. <i>Theta Join</i>)..... | 28 |
| 4.12.4. Złączenia zwrotne (ang. <i>Self Join</i>) | 30 |
| 4.12.5. Złączenia zewnętrzne (ang. <i>Outer Joins</i>) | 34 |
| 4.13. Operatory UNION, UNION ALL, INTERSECT, MINUS..... | 40 |
| 4.14. Podzapytania..... | 44 |
| 4.14.1. Podzapytania zwracające jeden rekord | 44 |
| 4.14.2. Podzapytania zwracające więcej niż jeden rekord | 45 |
| 4.14.3. Operatory ANY oraz ALL..... | 47 |
| 4.14.4. Podzapytania skorelowane, operatory EXISTS oraz NOT EXISTS..... | 48 |
| 4.14.5. Przykłady podzapytań, które można zastąpić złączeniami..... | 51 |
| 4.14.6. Podzapytania w klauzuli FROM..... | 54 |
| 5. Polecenie INSERT..... | 58 |
| 6. Polecenie UPDATE | 60 |
| 7. Polecenie DELETE..... | 64 |

| | |
|--|------------|
| 8. Wprowadzenie do mechanizmu transakcji | 66 |
| 9. Polecenie CREATE | 70 |
| 9.1. Tworzenie tabel | 70 |
| 9.2. Tworzenie ograniczeń integralnościowych (ang. <i>constraints</i>)..... | 74 |
| 9.3. Tworzenie i wykorzystywanie sekwencji | 84 |
| 9.4. Tworzenie i wykorzystywanie widoków (ang. <i>view</i>) | 89 |
| 10. Polecenie DROP | 91 |
| 11. Polecenie ALTER | 94 |
| 12. Słownik bazy danych | 97 |
| 13. Użytkownicy bazy danych | 98 |
| 13.1. Tworzenie, modyfikowanie i kasowanie użytkowników..... | 98 |
| 13.2. Uprawnienia..... | 98 |
| 13.3. Role bazodanowe | 99 |
| 13.4. Role predefiniowane | 100 |
| 14. Model SUMMIT2 | 101 |

1. Uwagi wstępne

Opracowanie omawia podstawowe elementy języka SQL na bazie systemu ORACLE. W zamierzeniu autora ma ono stanowić materiał pomocniczy do prowadzenia wykładu oraz ćwiczeń laboratoryjnych z przedmiotu *Bazy danych*. Opracowanie może być materiałem do samodzielnego studiowania, jednak należy mieć świadomość, że brak jest w nim bardziej systematycznego omówienia języka SQL. Język ten omówiono posługując się dużą liczbą (ponad 100) przykładów, ograniczając natomiast do minimum komentarz słowny. Starano się przedstawić jedynie jego najważniejsze elementy, najbardziej przydatne w praktyce. Wiele pomniejszych kwestii jest pominiętych lub omówionych tylko pobieżnie.

Wszystkie przykłady testowane były w systemie ORACLE, w wersji 8.1.7, jednak powinny bez żadnych zmian działać również w innych wersjach – zarówno wcześniejszych jak i późniejszych. Większość przykładów można wykonać również w innych niż ORACLE systemach bazodanowych (np. MySQL). Niektóre przykłady wykorzystują jednak specyficzne cechy ORACLE-a i uruchomienie ich na innej bazie danych może wymagać wprowadzenia mniejszych lub większych zmian. Głównie dotyczy to poleceń z grupy DDL, jako że bardzo mocno zależą one od architektury poszczególnych systemów bazodanowych.

Szczegółowy opis wszystkich poleceń języka SQL w systemie ORACLE można znaleźć w dokumentacji: [Oracle8i SQL Reference](#).

Zdecydowana większość przykładów operuje na demonstracyjnym modelu o nazwie `SUMMIT2`, który jest standardowo instalowany w systemie ORACLE, wersja 8.x. Dlatego też należy upewnić się, że skrypt `summit2.sql` wykonał się bezbłędnie. Krótkie omówienie modelu `SUMMIT2` zamieszczono na końcu niniejszego opracowania.

Wszystkie przykłady pokazane w opracowaniu zostały wykonane w programie `SQL*Plus`. Poniżej pokazano wygląd ekranu po prawidłowym uruchomieniu programu oraz po prawidłowym połączeniu się z jego pomocą do systemu ORACLE.

W zależności od konfiguracji systemu napisy mogą być w języku innym niż polski – domyślnie jest to oczywiście język angielski.

```
SQL*Plus: Release 8.1.7.0.0 - Production on Wto Mar 23 08:14:21 2004
```

```
(c) Copyright 2000 Oracle Corporation. All rights reserved.
```

```
Podaj nazwę użytkownika: summit2
```

```
Podaj hasło:
```

```
Połączony z:
```

```
Oracle8i Enterprise Edition Release 8.1.7.0.0 - Production
```

```
With the Partitioning option
```

```
JServer Release 8.1.7.0.0 - Production
```

```
SQL>
```

2. Podstawowe grupy poleceń

| Grupa | Polecenie | Opis | Dotyczy |
|-------|-----------|---|---------------------------|
| DML | select | wyświetlanie rekordów | tabele, widoki, sekwencje |
| DML | insert | wstawianie rekordów | tabele, widoki |
| DML | update | modyfikacja rekordów | tabele, widoki |
| DML | delete | kasowanie rekordów | tabele, widoki |
| DDL | create | tworzenie obiektów | wszystkie obiekty |
| DDL | alter | modyfikowanie obiektów | wszystkie obiekty |
| DDL | drop | kasowanie obiektów | wszystkie obiekty |
| DCL | commit | zatwierdza zmiany wprowadzone za pomocą poleceń DML | tabele, widoki |
| DCL | rollback | Wycofuje (anuluje) zmiany wprowadzone za pomocą poleceń DML | tabele, widoki |

Poszczególne polecenia SQL, w zależności od wykonywanych czynności, można zebrać w trzy podstawowe grupy. Są to:

- DML – Data **Manipulation** Language,
- DDL – Data **Definition** Language,
- DCL – Data **Control** Language.

3. Składnia polecenia SELECT (uproszczona)

```
SELECT
  [ DISTINCT | UNIQUE | ALL ]
  { * | nazwa_tabeli.atrybut [alias] , ... }
FROM
  nazwa_tabeli [alias] , ...
WHERE
  warunek [AND | OR warunek ...]
GROUP BY
  kolumna [ HAVING warunek ... ]
ORDER BY
  { kolumna | wyrażenie } [ASC | DESC ] , ...
;
```

4. Polecenie SELECT

4.1. Najprostsze przykłady

Przykład 1

```
SELECT * FROM region;
```

```
   ID NAME
-----
    1 North America
    2 South America
    3 Africa / Middle East
    4 Asia
    5 Europe
```

5 wierszy zostało wybranych.

```
SELECT * FROM "region";
```

```
SELECT * FROM "region"
          *
```

BŁĄD w linii 1:

```
ORA-00942: tabela lub perspektywa nie istnieje
          (table or view does not exist)
```

```
SELECT * FROM "REGION";
```

```
   ID NAME
-----
    1 North America
    2 South America
    3 Africa / Middle East
    4 Asia
    5 Europe
```

Komentarz:

Słowa kluczowe języka SQL nie są czułe na wielkość. Wielkości liter trzeba natomiast przestrzegać w odwoływaniu się do nazw obiektów (np. tabele) i atrybutów obiektów (np. nazwy kolumn). Zwykle jednak są one w bazie danych zapisywane dużymi literami i gdy odwołując się do nich nie używamy cudzysłowów, język SQL w systemie ORACLE przyjmuje domyślnie, że chodzi o duże litery (jak w pierwszym zapytaniu powyżej). Gdy decydujemy się na używanie cudzysłowów musimy pamiętać o wielkościach liter.

Dla zwiększenia czytelności można stosować dowolną ilość spacji, tabulatorów, znaków przejścia do nowej linii. Każde polecenie SQL musi być **zakończone średnikiem**.

We wszystkich przykładach będziemy stosować konsekwentnie zasadę, iż wszelkie słowa kluczowe pisane są **DUŻYMI LITERAMI**, natomiast pozostałe słowa (np. nazwy tabel, nazwy kolumn) pisane są małymi literami.

Gwiazdka zastępuje nazwy wszystkich kolumn. Zostaną one wyświetlone w dokładnie takiej kolejności, w jakiej występują w definicji tabeli. Nic nie stoi jednak na przeszkodzie, aby jawnie wymienić wszystkie kolumny – choćby po to, aby zmienić domyślną kolejność wyświetlania.

Nie jest możliwe jednoczesne używanie gwiazdki oraz specyfikowanie jawne nazw kolumn.

```
SQL> SELECT *, id FROM region;
SELECT *, id FROM region
      *
BŁĄD w linii 1:
ORA-00923: nie znaleziono słowa kluczowego FROM w oczekiwanym miejscu
(FROM keyword not found where expected)

SQL>
```

Przykład 2

```
SELECT name, id FROM region;
```

| NAME | ID |
|----------------------|----|
| North America | 1 |
| South America | 2 |
| Africa / Middle East | 3 |
| Asia | 4 |
| Europe | 5 |

5 wierszy zostało wybranych.

Komentarz:

Kolumny wyświetlane są od lewej do prawej w takiej kolejności, w jakiej były wymienione w wyrażeniu `SELECT` (chyba, że użyto znaku gwiazdki). Kolejność wyświetlania może więc być zupełnie inna niż rzeczywisty układ kolumn w tabeli. Trzeba być tego w pełni świadomym, gdyż wynik wyświetlany na ekranie zwykle nie odzwierciedla w widoczny sposób budowy poszczególnych tabel.

Zwróćmy uwagę, że nazwy kolumn zapisane są w bazie danych dużymi literami. W systemie ORACLE jest to powszechnie stosowana praktyka. Podając w poleceniu `SELECT` nazwy kolumn nie używaliśmy znaków cudzoalfabetycznych, więc system domyślnie przekonwertował ich nazwy na duże litery.

Przykład 3

```
SELECT id, id, id FROM region;
```

| ID | ID | ID |
|----|----|----|
| 1 | 1 | 1 |
| 2 | 2 | 2 |
| 3 | 3 | 3 |
| 4 | 4 | 4 |
| 5 | 5 | 5 |

5 wierszy zostało wybranych.

Komentarz:

Dowolne kolumny można wyświetlać dowolną ilość razy.

Przykład 4

```
SQL> DESCRIBE emp
Nazwa                                NULL?   Typ
-----
ID                                    NOT NULL NUMBER(7)
```

```

LAST_NAME                NOT NULL VARCHAR2 (25)
FIRST_NAME               VARCHAR2 (25)
USERID                   VARCHAR2 (8)
START_DATE               DATE
COMMENTS                 VARCHAR2 (255)
MANAGER_ID               NUMBER (7)
TITLE                    VARCHAR2 (25)
DEPT_ID                  NUMBER (7)
SALARY                   NUMBER (11, 2)
COMMISSION_PCT           NUMBER (4, 2)
SQL>

```

Komentarz:

Polecenie `DESCRIBE` (można je skracać do `DESC`) nie należy do języka SQL. Jest to polecenie programu SQL*Plus, który jest dostępny w ORACLE-u „od zawsze” i na każdej platformie systemowej i pozwala szybko wyświetlić szczegóły budowy tabeli (ale też tzw. widoków, synonimów, procedur i funkcji), takie jak:

- nazwy kolumn,
- czy dana kolumna ma granicznie `NOT NULL`,
- typy danych kolumn.

Więcej informacji na temat tego (w sumie bardzo prostego w użyciu) programu można znaleźć w dokumentacji:

- SQL*Plus User's Guide and Reference,
- SQL*Plus Quick Reference.

4.2. Klauzula ORDER BY

Przykład 5

```
SELECT last_name, salary FROM emp ORDER BY salary ASC;
```

| LAST_NAME | SALARY |
|-----------|--------|
| ----- | ----- |
| Newman | 750 |
| Palet | 795 |
| Palet | 795 |
| Chang | 800 |
| Markarian | 850 |
| Dancs | 860 |
| Smith | 940 |
| Biri | 1100 |
| Schwartz | 1100 |
| Urguhart | 1200 |
| Nozaki | 1200 |
| Menchu | 1250 |
| Catchpole | 1300 |
| Havel | 1307 |
| Nagayama | 1400 |
| Maduro | 1400 |
| Magee | 1400 |
| Ngao | 1450 |
| Dumas | 1450 |

| | |
|--------------|------|
| Quick-To-See | 1450 |
| Giljum | 1490 |
| Sedeghi | 1515 |
| Nguyen | 1525 |
| Ropeburn | 1550 |
| Velasquez | 2500 |

25 wierszy zostało wybranych.

Komentarz:

Klauzula `ORDER BY` służy do sortowania wyników według jeden lub kilku wybranych kolumn. Domyślnie dane sortowane są w porządku wzrastającym (1, 2, 3... oraz a, b, c...). Dlatego też słowo kluczowe `ASC` można pominąć – choć, gdy je jawnie wyspecyfikujemy nic złego się nie stanie.

Przykład 6

```
SQL> select last_name, salary FROM emp ORDER BY salary DESC;
```

| LAST_NAME | SALARY |
|--------------|--------|
| Velasquez | 2500 |
| Ropeburn | 1550 |
| Nguyen | 1525 |
| Sedeghi | 1515 |
| Giljum | 1490 |
| Ngao | 1450 |
| Dumas | 1450 |
| Quick-To-See | 1450 |
| Nagayama | 1400 |
| Magee | 1400 |
| Maduro | 1400 |
| Havel | 1307 |
| Catchpole | 1300 |
| Menchu | 1250 |
| Urguhart | 1200 |
| Nozaki | 1200 |
| Biri | 1100 |
| Schwartz | 1100 |
| Smith | 940 |
| Dancs | 860 |
| Markarian | 850 |
| Chang | 800 |
| Palet | 795 |
| Palet | 795 |
| Newman | 750 |

25 wierszy zostało wybranych.

Komentarz:

Chcą otrzymać dane posortowane „od największego do najmniejszego” musimy użyć słowa kluczowego `DESC`.

Przykład 7

```
SELECT name, region_id FROM dept ORDER BY region_id DESC, name ASC;
```

| NAME | REGION_ID |
|------------|-----------|
| Operations | 5 |

| | |
|----------------|---|
| Sales | 5 |
| Operations | 4 |
| Sales | 4 |
| Operations | 3 |
| Sales | 3 |
| Operations | 2 |
| Sales | 2 |
| Administration | 1 |
| Finance | 1 |
| Operations | 1 |
| Sales | 1 |

12 wierszy zostało wybranych.

Komentarz:

Słowo kluczowe ASC można pominąć, bo jest to opcja domyślna. Słowa kluczowego DESC pominąć nie można. Sortowanie odbywa się najpierw według pierwszej wymienionej kolumnie a następnie według drugiej.

4.3. Klauzula WHERE

Przykład 8

```
SELECT
    last_name, salary
FROM
    emp
WHERE
    salary > 1500
ORDER BY
    salary DESC;
```

| LAST_NAME | SALARY |
|-----------|--------|
| ----- | ----- |
| Velasquez | 2500 |
| Ropeburn | 1550 |
| Nguyen | 1525 |
| Sedeghi | 1515 |

4 wierszy zostało wybranych.

Komentarz:

Klauzula WHERE służy do ograniczania ilości wyświetlanych rekordów. Podany warunek logiczny może być w zasadzie dowolnie złożony. Można używać wszystkich dostępnych operatorów (będzie jeszcze o tym mowa poniżej).

W miarę zwiększania się wielkości zapytania SQL, warto stosować wcięcia i przejścia do nowej linii. Zdecydowanie zwiększa to czytelność kodu! W niniejszym opracowaniu zastosowane wcięcia i przejścia do nowej linii są kompromisem pomiędzy czytelnością a długością zapisu.

Przykład 9

```
SELECT name, credit_rating
FROM customer
WHERE credit_rating = 'EXCELLENT';
```

| NAME | CREDIT_RA |
|-------|-----------|
| ----- | ----- |

| | |
|----------------------------|-----------|
| Unisports | EXCELLENT |
| Womansport | EXCELLENT |
| Kam's Sporting Goods | EXCELLENT |
| Sportique | EXCELLENT |
| Beisbol Si! | EXCELLENT |
| Futbol Sonora | EXCELLENT |
| Kuhn's Sports | EXCELLENT |
| Hamada Sport | EXCELLENT |
| Big John's Sports Emporium | EXCELLENT |

9 wierszy zostało wybranych.

Komentarz:

Gdy w klauzuli WHERE odnosimy się do ciągów znaków, musimy ujmować je w apostrofy (nie cudzysłowy!). Sam język SQL nie rozróżnia wielkości liter, jednak uwaga ta nie odnosi się do danych zgromadzonych w tabelach. Przykładowo, gdy w klauzuli WHERE wpisujemy 'ExceLLent' zapytanie nie zwróci żadnego rekordu.

Nazwa drugiej kolumny została obcięta do 9 znaków, gdyż kolumna credit_rating jest typu VARCHAR2(9). Niedogodność tą można usunąć używając odpowiedniej funkcji formatującej dostępnej w SQL-u systemu ORACLE:

```
SELECT name, RPAD(credit_rating, 13) "CREDIT_RATING"
FROM customer
WHERE credit_rating = 'EXCELLENT';
```

| NAME | CREDIT_RATING |
|----------------------------|---------------|
| Unisports | EXCELLENT |
| Womansport | EXCELLENT |
| Kam's Sporting Goods | EXCELLENT |
| Sportique | EXCELLENT |
| Beisbol Si! | EXCELLENT |
| Futbol Sonora | EXCELLENT |
| Kuhn's Sports | EXCELLENT |
| Hamada Sport | EXCELLENT |
| Big John's Sports Emporium | EXCELLENT |

Przykład 10

```
SELECT
  first_name, last_name, start_date
FROM
  emp
WHERE
  start_date > '01-jan-1992';
```

| FIRST_NAME | LAST_NAME | START_DATE |
|------------|-----------|-------------|
| Antoinette | Catchpole | 09-feb-1992 |
| Henry | Giljum | 18-jan-1992 |
| Mai | Nguyen | 22-jan-1992 |
| Elena | Maduro | 07-feb-1992 |

Komentarz:

Podawanie dat w taki sposób, jak powyżej może być niebezpieczne. Gdyby bowiem datę podano w formacie innym niż obowiązujący w bieżącej sesji, SQL „pogubi się”. Lepiej więc zabezpieczyć się przed takim problemem używając jednej z tzw. funkcji formatujących, tutaj TO_DATE. Porównajmy:

```
SQL> SELECT first_name, last_name, start_date
2 FROM emp
3 WHERE start_date > '01-01-1992'
4 ;
WHERE start_date > '01-01-1992'
*
```

BŁĄD w linii 3:
ORA-01843: nieprawidłowy miesiąc
(not a valid month)

Gdy polecenie SQL-owe kończy się błędem, gwiazdka wskazuje miejsce, gdzie interpreter SQL-a po raz pierwszy „pogubił się”. Od tego miejsca należy więc rozpocząć poszukiwanie błędu. Analizowanie tylko komunikatów o błędach często jest bardzo mylące!

```
SQL> SELECT first_name, last_name, start_date
2 FROM emp
3 WHERE start_date > TO_DATE('01-01-1992', 'dd-mm-yyyy')
4 ;
```

| FIRST_NAME | LAST_NAME | START_DATE |
|------------|-----------|-------------|
| Antoinette | Catchpole | 09-feb-1992 |
| Henry | Giljum | 18-jan-1992 |
| Mai | Nguyen | 22-jan-1992 |
| Elena | Maduro | 07-feb-1992 |

Przykład 11

```
SELECT TO_CHAR(start_date, 'DD--MM--YYYY, HH24:::MI:::SS')
FROM emp
WHERE last_name = 'Velasquez';

TO_CHAR(START_DATE, '
-----
03--03--1990, 08:::30:::00

SQL>
```

Komentarz:

Typ danych DATE przechowuje wszystkie elementy, począwszy od tysiąclecia a skończywszy na sekundzie. Domyślnie wyświetlany jest tylko dzień, miesiąc oraz rok. Używając funkcji TO_CHAR można wyświetlić też pozostałe elementy. Decyduje o tym postać tzw. maski, czyli drugi parametr funkcji TO_CHAR.

4.4. Operatory

```
porównania:
= != ^= <> > < >= <=

arytmetyczne:
+ - * /

konkatenacja:
||

od określania przedziałów:
[NOT] BETWEEN ... AND
[NOT] IN
```

od określania wzorca:
[NOT] LIKE

logiczne:
AND OR NOT

do testowania wartości pustych:
IS [NOT] NULL

Przykład 12

```
SELECT last_name FROM emp WHERE salary >= 1500;

SELECT last_name FROM emp WHERE salary BETWEEN 1000 AND 2000;

SELECT last_name FROM emp WHERE salary >= 1000 AND salary <= 2000;

SELECT last_name FROM emp WHERE title IN ('President', 'VP, Sales');

SELECT last_name FROM emp
WHERE UPPER(title) IN (UPPER('President'), UPPER('VP, Sales'));

SELECT last_name FROM emp WHERE id IN (1, 10, 20);

SELECT last_name FROM emp WHERE last_name LIKE 'N%';

SELECT last_name FROM emp WHERE last_name LIKE '____'; -- 4 znaki podkreślenia
SELECT last_name FROM emp WHERE last_name LIKE 'N____'; -- 3 znaki podkreślenia
SELECT last_name FROM emp WHERE last_name LIKE 'N_a'; -- 2 znaki podkreślenia
SELECT last_name FROM emp WHERE manager_id = 3 AND salary > 1000;
```

Komentarz:

Operatory w klauzuli WHERE stosujemy do zawężania ilości wyświetlanych rekordów.

4.5. Aliasy

Przykład 13

```
SELECT
  first_name Imie,
  last_name "Nazwisko",
  start_date "Data zatrudnienia"
FROM
  emp
WHERE dept_id = 41;
```

| IMIE | Nazwisko | Data zatrud |
|---------|----------|-------------|
| LaDoris | Ngao | 08-mar-1990 |
| Molly | Urguhart | 18-jan-1991 |
| Elena | Maduro | 07-feb-1992 |
| George | Smith | 08-mar-1990 |

Komentarz:

Alias, który nie jest ujęty w cudzysłowy, zostanie wydrukowany dużymi literami. Gdy zostanie on ujęty w cudzysłowy, zostanie zachowana wielkość liter. O ile w drugim aliasie użycie cudzysłowów okazało się przydatne (ale nie było to niezbędne), o tyle w aliasie trzecim jest to konieczne. Jest tak dlatego, że definiowany alias zawiera białe znaki i bez cudzysłowów polecenie SQL byłoby błędne. Porównajmy:

```
SELECT first_name Imie, last_name "Nazwisko", start_date Data zatrudnienia
FROM emp WHERE dept_id = 41;
*
BŁĄD w linii 1:
ORA-00923: nie znaleziono słowa kluczowego FROM w oczekiwanym miejscu
(FROM keyword not found where expected)
```

Zwróćmy też uwagę, że trzeci alias został obcięty. O tym, jak usunąć tę niedogodność powiedzieliśmy już wyżej.

4.6. Wyrażenia

Przykład 14

```
SELECT last_name "Nazwisko", salary*12+100
FROM emp
WHERE dept_id = 41;
```

| Nazwisko | SALARY*12+100 |
|----------|---------------|
| Ngao | 17500 |
| Urguhart | 14500 |
| Maduro | 16900 |
| Smith | 11380 |

Komentarz:

W poleceniu SELECT można używać w zasadzie dowolnych wyrażeń – powyżej pole SALARY pomnożono przez 12 i dodano 100. W tym przykładzie aż prosi się o zastosowanie aliasu dla tego wyrażenia.

Przykład 15

```
SELECT id, name*100 FROM region;
```

```
select id, name*100 from region
*
BŁĄD w linii 1:
ORA-01722: nieprawidłowa liczba
(invalid number)
```

Komentarz:

Próba pomnożenia pola znakowego przez liczbę oczywiście nie powiedzie się. Gwiazdka pokazuje miejsce, gdzie interpreter SQL-a po raz pierwszy „pogubił się”.

Przykład 16

```
SELECT
  date_ordered D1,
  date_shipped D2,
  date_ordered - date_shipped "D1-D2"
FROM
  ord
```

```
ORDER BY
D1 - D2;
```

| D1 | D2 | D1-D2 |
|-------------|-------------|-------|
| 02-sep-1992 | 22-sep-1992 | -20 |
| 03-sep-1992 | 23-sep-1992 | -20 |
| 28-aug-1992 | 17-sep-1992 | -20 |
| 08-sep-1992 | 28-sep-1992 | -20 |
| 31-aug-1992 | 18-sep-1992 | -18 |
| 31-aug-1992 | 15-sep-1992 | -15 |
| 04-sep-1992 | 18-sep-1992 | -14 |
| 07-sep-1992 | 21-sep-1992 | -14 |
| 09-sep-1992 | 21-sep-1992 | -12 |
| 09-sep-1992 | 21-sep-1992 | -12 |
| 31-aug-1992 | 10-sep-1992 | -10 |
| 31-aug-1992 | 10-sep-1992 | -10 |
| 31-aug-1992 | 10-sep-1992 | -10 |
| 07-sep-1992 | 15-sep-1992 | -8 |
| 01-sep-1992 | 08-sep-1992 | -7 |
| 07-sep-1992 | 10-sep-1992 | -3 |

Komentarz:

Zapytanie wyświetla datę złożenia zamówienia, datę realizacji zamówienia oraz różnicę pomiędzy tymi datami (w dniach). Okazuje się więc, że operacje arytmetyczne można wykonywać nie tylko na liczbach, ale też na datach.

W klauzuli `ORDER BY` użyto wyrażenia. Dla skrócenia zapisu użyto aliasów. Alias `D1-D2` musi być ujęty w cudzysłowy, gdyż zawiera znak minusa.

4.7. Wartości puste (NULL)

Przykład 17

```
SELECT name FROM customer WHERE country IS NULL;
```

```
NAME
-----
Kam's Sporting Goods
```

Komentarz:

Wartość `NULL` oznacza, że w danej komórce nie ma żadnych danych. Wpisanie do komórki np. jednej spacji, mimo tego, że też jej nie widać na wydruku, nie jest tym samym co pozostawienie w niej wartości `NULL`. Testowania wartości `NULL` nie można wykonać w taki sposób, jak poniżej. Trzeba użyć zwrotu `IS NULL`:

```
SQL> SELECT name FROM customer WHERE country = ' ';
nie wybrano żadnych wierszy
```

Przykład 18

```
SELECT name FROM customer WHERE state IS NOT NULL;
```

```
NAME
-----
```

```
Womansport
Big John's Sports Emporium
Ojibway Retail
```

Komentarz:

Można też zrobić tak jak poniżej (tylko po co, skoro `IS NOT NULL` brzmi bardziej elegancko i jest chyba bardziej czytelnie):

```
SELECT name FROM customer WHERE state LIKE '%';

NAME
-----
Womansport
Big John's Sports Emporium
Ojibway Retail
```

Przykład 19

```
SELECT NVL(state, '-'), NVL(country, '?') FROM warehouse;
```

```
NVL (STATE, '-')      NVL (COUNTRY, '?')
-----
WA                    USA
-                    Czechoslovakia
-                    Brazil
-                    Nigeria
-                    ?
```

5 wierszy zostało wybranych.

Komentarz:

Funkcja `NVL` jest bardzo przydatna, gdy chcemy, aby wartość `NULL` była jakoś wyróżniona w wyświetlanym wyniku. Jest ona również bardzo przydatna, gdy obsługiwane są wartości numeryczne – wówczas jej użycie pozwala uniknąć wielu niespodziewanych błędów (wszelkie operacje arytmetyczne na wartościach `NULL` dają w wyniku też wartość `NULL`). Porównajmy:

```
SELECT commission_pct*10 FROM emp WHERE salary > 1500;
```

```
COMMISSION_PCT*10
-----
                100
                150
```

4 wierszy zostało wybranych.

```
SELECT NVL(commission_pct*10, 0) FROM emp WHERE salary > 1500;
```

```
NVL (COMMISSION_PCT*10, 0)
-----
                0
                0
                100
                150
```

4 wierszy zostało wybranych.

4.8. Eliminowanie duplikatów

Przykład 20

```
SELECT DISTINCT title FROM emp;
```

```
TITLE
-----
President
Sales Representative
Stock Clerk
VP, Administration
VP, Finance
VP, Operations
VP, Sales
Warehouse Manager

8 wierszy zostało wybranych.
```

Komentarz:

Słowo kluczowe `DISTINCT` (zamiennie można używać słowa kluczowego `UNIQUE` – są to równorzędne synonimy) pozwala usunąć z wyświetlanego wyniku duplikaty. W naszym zapytaniu interesowały nas wszystkie nazwy stanowisk a to, że pewne z nich są przypisane do więcej niż jednego pracownika nie jest dla nas w tym momencie istotne.

W wyniku użycia operatora `DISTINCT` wynikowe rekordy są dodatkowo sortowane.

Można też używać słowa kluczowego `ALL`, które powoduje wyświetlenie wszystkich rekordów w tabeli. Jest ono przyjmowane domyślnie i z tego powodu nie ma sensu wpisywać go jawnie.

4.9. Funkcje agregujące

Przykład 21

```
SELECT MAX(salary) "Zarobki MAX" FROM emp;
```

```
Zarobki MAX
-----
          2500
```

Komentarz:

Brak

Przykład 22

```
SELECT
  MAX(salary), MIN(salary), AVG(salary), SUM(salary), COUNT(salary)
FROM
  emp;
```

```
MAX(SALARY) MIN(SALARY) AVG(SALARY) SUM(SALARY) COUNT(SALARY)
-----
          2500          750      1255,08      31377          25
```

Komentarz:

Polecenie `COUNT(SALARY)` podaje całkowitą liczbę rekordów spełniających warunki zapytania.

Przykład 23

```
SELECT COUNT(*) FROM emp WHERE salary > 1500;
```

```
COUNT(*)
-----
         4
```

Komentarz:

Uzyskujemy informację o tym, ile rekordów spełnia warunek `WHERE`. Zamiast gwiazdki można użyć nazwy dowolnej kolumny z tabeli `EMP` a nawet dowolnego wyrażenia stałego. Porównajmy:

```
SQL> SELECT COUNT('cokolwiek') FROM emp WHERE salary > 1500;
```

```
COUNT('COKOLWIEK')
-----
                   4
```

```
SQL> SELECT COUNT(id) FROM emp WHERE salary > 1500;
```

```
COUNT(ID)
-----
         4
```

4.10. Klauzula GROUP BY

Przykład 24

```
SELECT dept_id, SUM(salary), 'wyrażenie stałe'
FROM emp
WHERE title = 'Warehouse Manager'
GROUP BY dept_id;
```

```
DEPT_ID SUM(SALARY) 'WYRAŻENIESTAŁE'
-----
         41         1200 wyrażenie stałe
         42         1250 wyrażenie stałe
         43         1100 wyrażenie stałe
         44         1300 wyrażenie stałe
         45         1307 wyrażenie stałe
```

```
SELECT dept_id, salary FROM emp WHERE title = 'Warehouse Manager'
GROUP BY dept_id;
```

```
BŁĄD w linii 1:
ORA-00979: to nie jest wyrażenie GROUP BY
          (not a GROUP BY expression)
```

```
SELECT dept_id, SUM(salary), last_name FROM emp WHERE title = 'Warehouse
Manager'GROUP BY dept_id;
```

```
BŁĄD w linii 1:
ORA-00979: to nie jest wyrażenie GROUP BY
          (not a GROUP BY expression)
```

Komentarz:

Gdy używamy funkcji grupującej `GROUP BY` musi ona wystąpić po klauzuli `WHERE`. W klauzuli `SELECT` mogą wówczas wystąpić tylko (porównaj przykład powyżej):

- funkcje agregujące (np. `SUM`, `MIN`, `MAX`, `AVG`, `COUNT`),
- nazwy kolumn występujące w funkcji grupującej `GROUP BY`,
- wyrażenia stałe.

Przykład 25

```
SELECT title, SUM(salary), count(*) FROM emp GROUP BY title;
```

| TITLE | SUM(SALARY) | COUNT(*) |
|----------------------|-------------|----------|
| President | 2500 | 1 |
| Sales Representative | 7380 | 5 |
| Stock Clerk | 9490 | 10 |
| VP, Administration | 1550 | 1 |
| VP, Finance | 1450 | 1 |
| VP, Operations | 1450 | 1 |
| VP, Sales | 1400 | 1 |
| Warehouse Manager | 6157 | 5 |

Komentarz:

W powyższym przykładzie wyświetliliśmy sumę zarobków wszystkich pracowników pracujących na poszczególnych stanowiskach. Dodatkowo wyświetliliśmy informację o tym, ilu pracowników pracuje na każdym ze stanowisk.

Przykład 26

```
SELECT title, SUM(salary)
FROM emp
WHERE SUM(salary) > 2000
GROUP BY title;
```

BŁĄD w linii 1:

```
ORA-00934: funkcja grupowa nie jest tutaj dozwolona
(group function is not allowed here)
```

Komentarz:

W tym przykładzie (niestety błędnym!) staraliśmy się wyświetlić tylko sumę zarobków pracowników z tych stanowisk, gdzie ta suma jest większa niż 2000. Okazuje się, że aby wykonać takie zadanie należy użyć klauzuli `HAVING` (patrz niżej).

4.11. Klauzula HAVING

Przykład 27

```
SELECT title, SUM(salary)
FROM emp
GROUP BY title
HAVING SUM(salary) > 2000;
```

| TITLE | SUM(SALARY) |
|----------------------|-------------|
| President | 2500 |
| Sales Representative | 7380 |

| | |
|-------------------|------|
| Stock Clerk | 9490 |
| Warehouse Manager | 6157 |

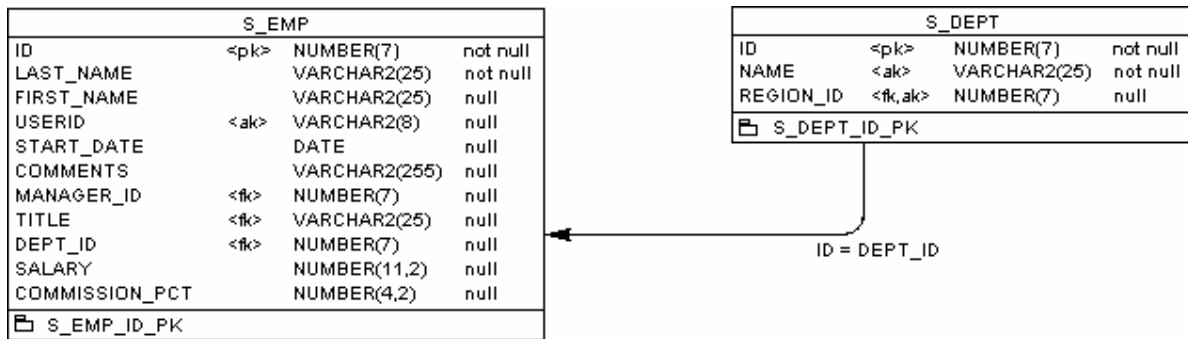
Komentarz:

Zasada działania klauzuli `HAVING` jest następująca: podczas tworzenia każdej grupy (za pomocą `GROUP BY`) obliczana jest suma zarobków (`SUM(salary)`) dla tej grupy. Kiedy warunek logiczny sprawdzany z pomocą klauzuli `HAVING` jest spełniony, taka grupa jest uwzględniana w wyniku i wyświetlana na konsoli.

4.12. Złączanie tabel

4.12.1. Iloczyn kartezjański (ang. *Cross Join*)

Przykład 28



```
SELECT
  E.first_name, E.last_name, D.name
FROM
  emp E, dept D;
```

| FIRST_NAME | LAST_NAME | NAME |
|------------|--------------|----------------|
| ----- | ----- | ----- |
| Carmen | Velasquez | Finance |
| LaDoris | Ngao | Finance |
| Midori | Nagayama | Finance |
| Mark | Quick-To-See | Finance |
| Audry | Ropeburn | Finance |
| ... | ... | ... |
| Vikram | Palet | Administration |
| Chad | Newman | Administration |
| Alexander | Markarian | Administration |
| Eddie | Chang | Administration |
| Radha | Palet | Administration |
| Bela | Dancs | Administration |
| Sylvie | Schwartz | Administration |

300 wierszy zostało wybranych.

Komentarz:

W tym przykładzie zapomniano o warunku złączeniowym. Efektem jest iloczyn kartezjański relacji `EMP` oraz `DEPT`. Wynik liczy 300 rekordów (25 rekordów w tabeli `EMP` razy 12 rekordów w tabeli `DEPT`).

DEPT). Gdyby złączanych tabel było więcej, należy liczyć się z ogromną liczbą (bezsensownych!) wynikowych rekordów. Może to czasami prowadzić do zawieszenia się całego systemu bazodanowego!

Zapytanie zwracające iloczyn kartezjański nazywa się w terminologii bazodanowej *cross-join*.

4.12.2. Złączenia równościowe (ang. *Equi Join* lub *Inner Join*)

Przykład 29

```
SELECT
  emp.first_name, emp.last_name, dept.name
FROM
  emp, dept
WHERE
  emp.dept_id = dept.id;
```

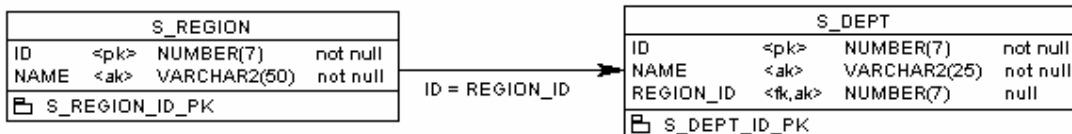
| FIRST_NAME | LAST_NAME | NAME |
|------------|--------------|----------------|
| Carmen | Velasquez | Administration |
| LaDoris | Ngao | Operations |
| Midori | Nagayama | Sales |
| Mark | Quick-To-See | Finance |
| Audry | Ropeburn | Administration |
| Molly | Urguhart | Operations |
| Roberta | Menchu | Operations |
| Ben | Biri | Operations |
| Antoinette | Catchpole | Operations |
| Marta | Havel | Operations |
| Colin | Magee | Sales |
| Henry | Giljum | Sales |
| Yasmin | Sedeghi | Sales |
| Mai | Nguyen | Sales |
| Andre | Dumas | Sales |
| Elena | Maduro | Operations |
| George | Smith | Operations |
| Akira | Nozaki | Operations |
| Vikram | Palet | Operations |
| Chad | Newman | Operations |
| Alexander | Markarian | Operations |
| Eddie | Chang | Operations |
| Radha | Palet | Sales |
| Bela | Dancs | Operations |
| Sylvie | Schwartz | Operations |

25 wierszy zostało wybranych.

Komentarz:

- dwie tabele wiążemy ze sobą używając tzw. kolumny wiążącej (najczęściej są to klucz główny w jednej tabeli i klucz obcy w drugiej). W naszym przykładzie wiążemy ze sobą każdy wiersz z tabeli EMP z odpowiadającym mu wierszem z tabeli DEPT,
- w poleceniu SELECT nazwy kolumn poprzedziliśmy nazwą tabeli. Gdy nazwy kolumn są unikalne, to nie jest to konieczne, aczkolwiek zalecane,
- w klauzuli WHERE nie możemy już opuścić nazw kolumn (chyba, że nazwy kolumn są unikalne),
- dla zwartości zapisu można zamiast całych nazw tabel zdefiniować aliasy,
- tego typu złączenie nazywane jest **złączeniem równościowym** (ang. *natural-join*).

Przykład 30



```
SQL> SELECT name, name FROM region, dept WHERE id = region_id;
SELECT name, name FROM region, dept WHERE id = region_id
*
```

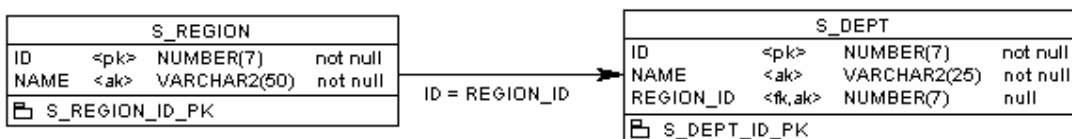
BŁĄD w linii 1:

**ORA-00918: kolumna zdefiniowana w sposób niejednoznaczny
(column ambiguously defined)**

Komentarz:

Tutaj nie można opuścić nazw tabel. System nie jest w stanie rozstrzygnąć, o którą kolumnę o nazwie ID chodzi – w obu użytych tabelach jest kolumna o takiej nazwie.

Przykład 31



```
SQL> SELECT name, name FROM region, dept WHERE region.id = dept.region_id;
SELECT name, name FROM region, dept WHERE region.id = dept.region_id
*
```

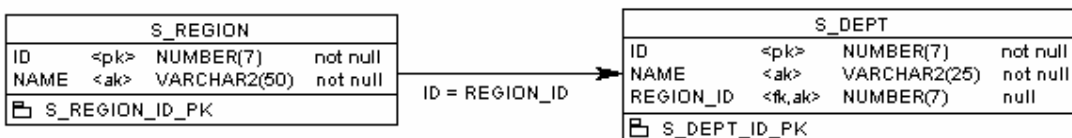
BŁĄD w linii 1:

**ORA-00918: kolumna zdefiniowana w sposób niejednoznaczny
(column ambiguously defined)**

Komentarz:

Tutaj nie można opuścić nazw tabel. System nie jest w stanie rozstrzygnąć, o którą kolumnę o nazwie NAME chodzi – w obu użytych tabelach jest kolumna o takiej nazwie.

Przykład 32



```
SELECT R.name, D.name
FROM region R, dept D
WHERE R.id = D.region_id
ORDER BY R.name;
```

| NAME | NAME |
|----------------------|------------|
| ----- | |
| Africa / Middle East | Sales |
| Africa / Middle East | Operations |
| Asia | Sales |
| Asia | Operations |
| Europe | Sales |

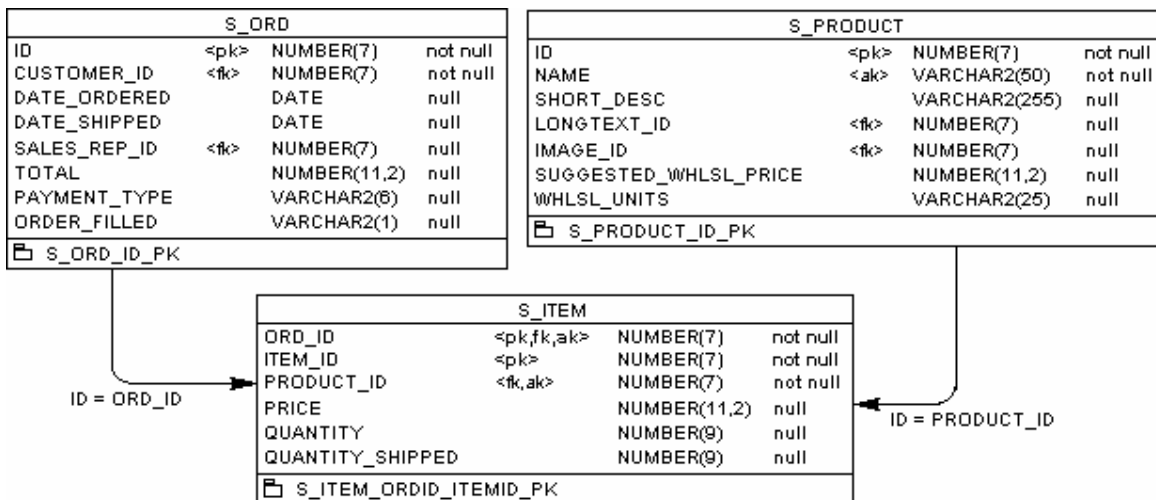
| | |
|---------------|----------------|
| Europe | Operations |
| North America | Finance |
| North America | Operations |
| North America | Sales |
| North America | Administration |
| South America | Sales |
| South America | Operations |

12 wierszy zostało wybranych.

Komentarz:

Zapytanie udało się wreszcie wykonać, gdyż we właściwy sposób odwołano się do poszczególnych kolumn w tabelach. Aby zapis zapytania był bardziej zwarty użyto aliasów dla nazw tabel.

Przykład 33



```

SELECT
  O.id, O.total, I.price, I.quantity, I.price * I.quantity, P.name
FROM
  ord O, item I, product P
WHERE
  O.id = I.ord_id AND
  P.id = I.product_id AND
  O.id = 100;
  
```

| ID | TOTAL | PRICE | QUANTITY | I.PRICE*I.QUANTITY | NAME |
|-----|--------|-------|----------|--------------------|------------------|
| 100 | 601100 | 135 | 500 | 67500 | Bunny Boot |
| 100 | 601100 | 380 | 400 | 152000 | Pro Ski Boot |
| 100 | 601100 | 14 | 500 | 7000 | Bunny Ski Pole |
| 100 | 601100 | 36 | 400 | 14400 | Pro Ski Pole |
| 100 | 601100 | 582 | 600 | 349200 | Himalaya Bicycle |
| 100 | 601100 | 20 | 450 | 9000 | New Air Pump |
| 100 | 601100 | 8 | 250 | 2000 | Prostar 10 Pound |

Komentarz:

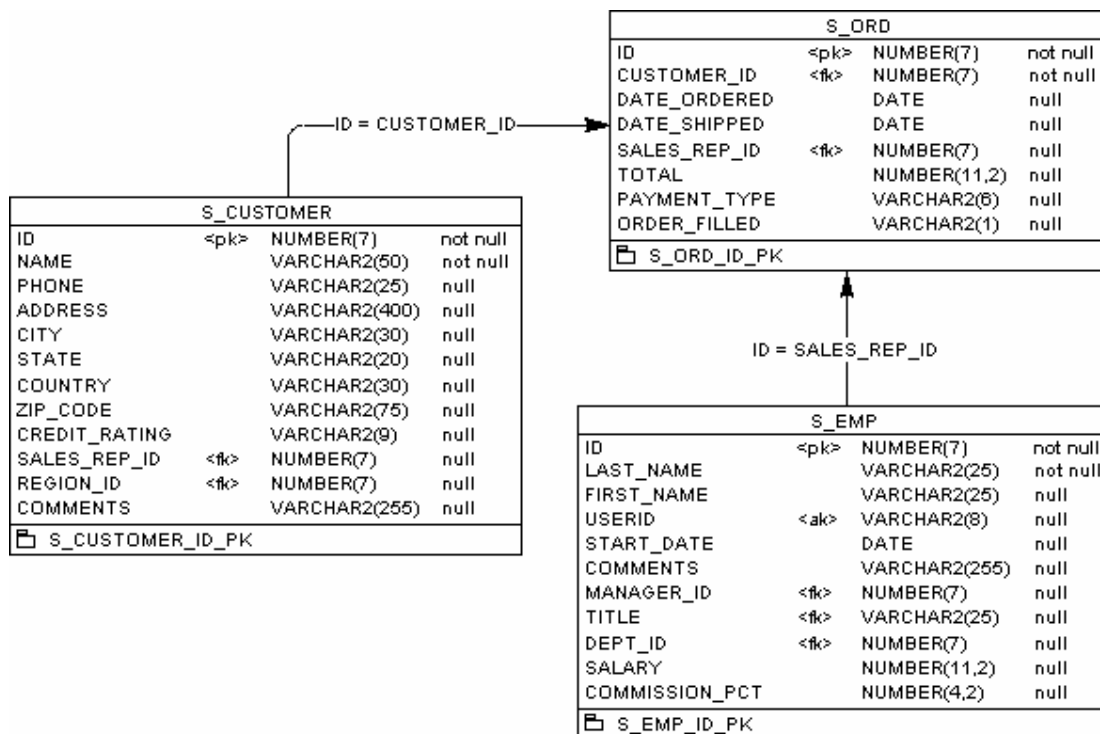
Bardzo częsty w praktyce przypadek, gdy łączymy dane z tabel, które są ze sobą w relacji N:N (tutaj tabele ORD oraz PRODUCT).

Dla sprawdzenia, czy dane w tabelach ORD oraz ITEM są spójne możemy wykonać poniższe zapytanie (czy domyślasz się o jaką spójność danych chodzi?):

```
SELECT SUM(price * quantity) FROM item WHERE ord_id = 100;
```

```
SUM(PRICE*QUANTITY)
-----
                601100
```

Przykład 34



```
SELECT
  O.id "Nr zam.", C.name "Klient", C.phone, E.first_name, E.last_name
FROM
  ord O, customer C, emp E
WHERE
  O.customer_id = C.id AND
  O.sales_rep_id = E.id AND
  O.id = 100;
```

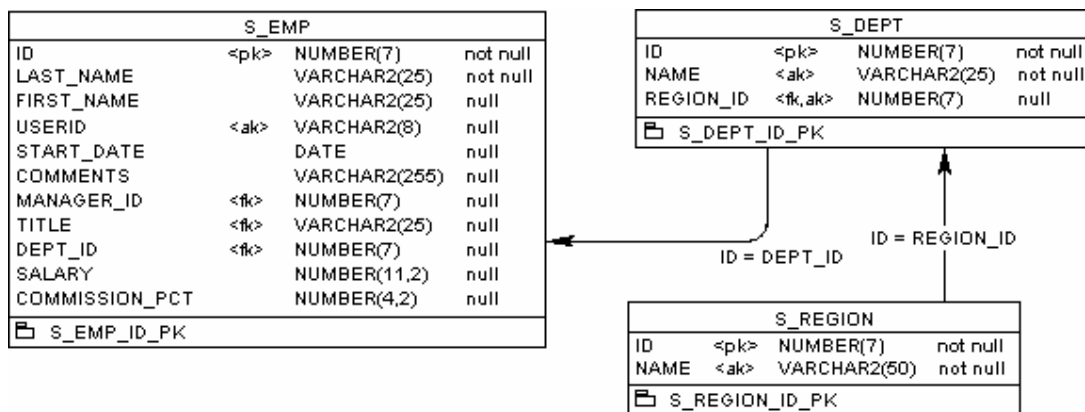
```
Nr zam. Klient      PHONE                FIRST_NAME  LAST_NAME
-----
    100 Womansport  1-206-104-0103  Colin      Magee
```

Komentarz:

Wyświetlamy dane o zamówieniu o numerze 100. Dane pobieramy z trzech tabel. Z tabeli ORD numer zamówienia, z tabeli CUSTOMER nazwę klienta i jego numer telefonu i wreszcie z tabeli EMP imię i nazwisko pracownika odpowiedzialnego za dane zamówienie.

Tabela CUSTOMER oraz EMP też są ze sobą połączone relacją (pola ID oraz SALES_REP_ID), jednak w kontekście tego przykładu to połączenie jest nieistotne, więc nie jest uwidocznione na rysunku.

Przykład 35



```

SELECT
  R.name "Region", D.name "Wydzial", SUM(salary) "Koszty placowe"
FROM
  emp E, dept D, region R
WHERE
  E.dept_id = D.id AND
  R.id = D.region_id
GROUP BY
  R.name, D.name
ORDER BY
  D.name;

```

| Region | Wydzial | Koszty placowe |
|----------------------|----------------|----------------|
| North America | Administration | 4050 |
| North America | Finance | 1450 |
| Africa / Middle East | Operations | 2700 |
| Asia | Operations | 2100 |
| Europe | Operations | 3267 |
| North America | Operations | 4990 |
| South America | Operations | 3245 |
| Africa / Middle East | Sales | 1515 |
| Asia | Sales | 2320 |
| South America | Sales | 1490 |
| North America | Sales | 2800 |
| Europe | Sales | 1450 |

12 wierszy zostało wybranych.

Komentarz:

Wyświetlamy listę wydziałów (pamiętajmy, że w różnych regionach występują wydziały o tych samych nazwach) wraz z sumą wszystkich ich kosztów płacowych (zarobki pracowników w danym wydziale). Aby wykonać to zadanie wymagane jest odpowiednie pogrupowanie danych.

Gdy w zapytaniu nieco zmienimy klauzulę GROUP BY (grupowanie następować będzie tylko według regionów) otrzymamy inny wynik.. Porównajmy:

```

SELECT
  R.name "Region", SUM(salary) "Koszty placowe"
FROM
  emp E, dept D, region R
WHERE
  E.dept_id = D.id AND
  R.id = D.region_id
GROUP BY
  R.name
-- teraz grupowanie tylko według jednej kolumny

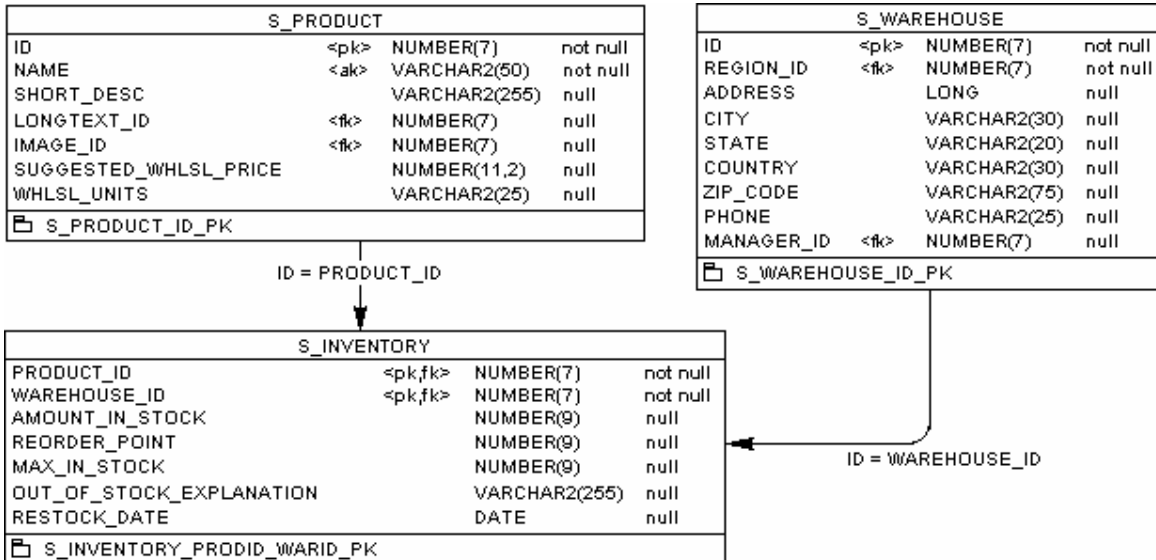
```



```
ORDER BY
  R.name;
```

| Region | Koszty placowe |
|----------------------|----------------|
| Africa / Middle East | 4215 |
| Asia | 4420 |
| Europe | 4717 |
| North America | 13290 |
| South America | 4735 |

Przykład 36



```

SELECT
  RPAD(P.name, 30) "Nazwa produktu",
  RPAD(NVL(W.country, '?')||', '||NVL(W.state, '?')||', '||W.city, 30)
  "Hurtownia (kraj, stan, miasto)",
  I.amount_in_stock "Stan biezacy",
  I.max_in_stock "Stan max"
FROM
  warehouse W, product P, inventory I
WHERE
  P.id = I.product_id AND
  W.id = I.warehouse_id AND
  I.max_in_stock - I.amount_in_stock < 10
ORDER BY
  W.id, P.name;
```

| Nazwa produktu | Hurtownia (kraj, stan, miasto) | Stan biezacy | Stan max |
|--------------------------|--------------------------------|--------------|----------|
| Junior Soccer Ball | USA, WA, Seattle | 993 | 1000 |
| Winfield Bat | USA, WA, Seattle | 173 | 175 |
| Alomar Glove | Brazil, ?, Sao Paulo | 98 | 100 |
| Black Hawk Knee Pads | Brazil, ?, Sao Paulo | 175 | 175 |
| Safe-T Helmet | Brazil, ?, Sao Paulo | 132 | 140 |
| Winfield Bat | Brazil, ?, Sao Paulo | 97 | 100 |
| Alexeyer Pro Lifting Bar | Nigeria, ?, Lagos | 70 | 70 |
| New Air Pump | Nigeria, ?, Lagos | 35 | 35 |
| Pro Curling Bar | Nigeria, ?, Lagos | 65 | 70 |
| Prostar 20 Pound Weight | Nigeria, ?, Lagos | 61 | 70 |

| | | | |
|----------------|-----------------|-----|-----|
| Himalaya Tires | ?, ?, Hong Kong | 135 | 140 |
| Safe-T Helmet | ?, ?, Hong Kong | 250 | 250 |

12 wierszy zostało wybranych.

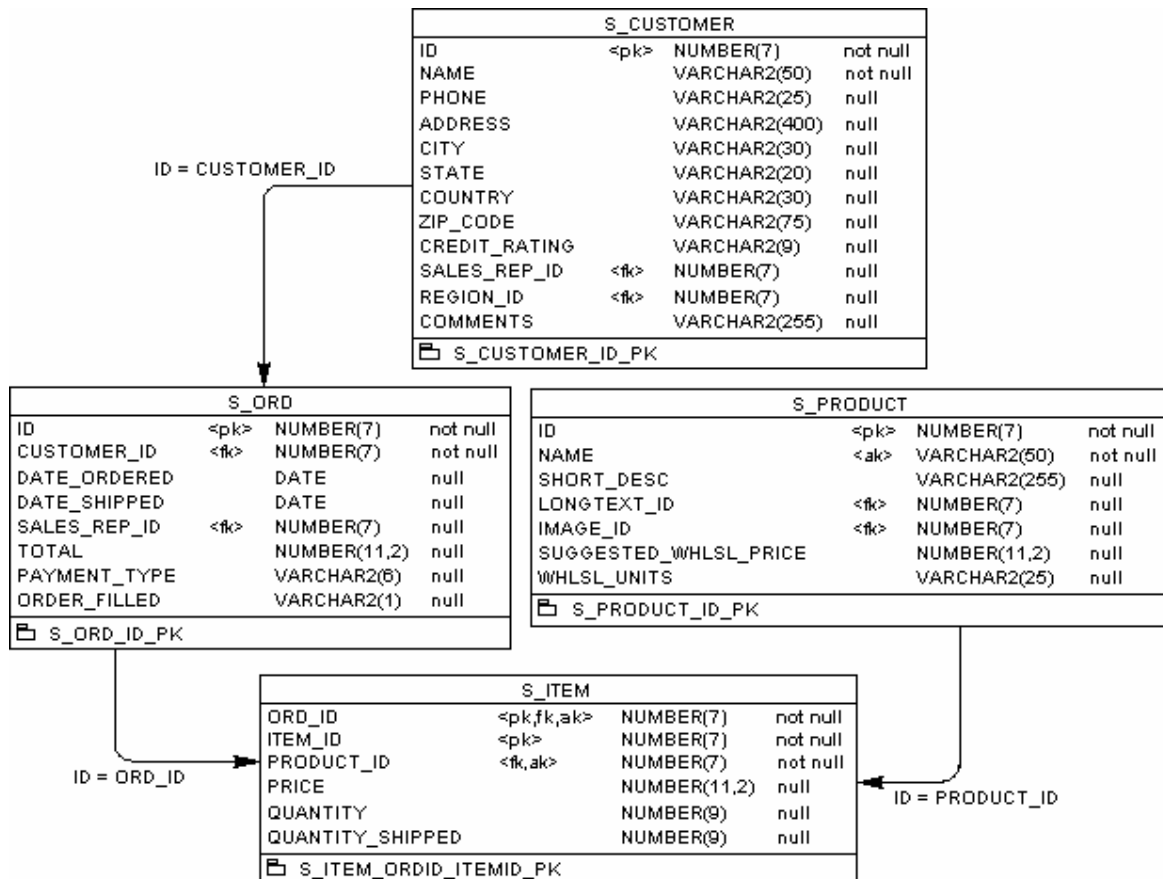
Komentarz:

Wyświetlamy stan magazynowy wszystkich produktów z rozbiciem na poszczególne hurtownie (tabele WAREHOUSE, PRODUCT, INVENTORY). Ograniczamy się tylko do tych produktów, których sprzedaż, czyli różnica wartości pól

`INVENTORY.max_in_stock - INVENTORY.amount_in_stock`

jest mniejsza niż 10.

Przykład 37



```

SELECT
  O.id "Nr zam.",
  RPAD(C.name, 15) "Klient",
  RPAD(P.name, 20) "Produkt",
  O.payment_type "Platnosc",
  TO_CHAR(O.date_ordered, 'DD-MM-YYYY') "Data",
  I.price "Cena",
  I.quantity "Ilosc"
FROM
  customer C,
  ord O,
  item I,
  product P
WHERE
  C.id = O.customer_id AND
  O.ord_id = I.ord_id AND
  O.product_id = I.product_id
  
```

```

O.id = I.ord_id AND
P.id = I.product_id AND
O.payment_type = 'CREDIT' AND
O.date_ordered BETWEEN
  TO_DATE('01-08-1992', 'DD-MM-YYYY') AND
  TO_DATE('31-08-1992', 'DD-MM-YYYY')
ORDER BY
O.id, P.name;

```

| Nr zam. | Klient | Produkt | Platno | Data | Cena | Ilosc |
|---------|----------------|----------------------|--------|------------|------|-------|
| 97 | Unisports | Grand Prix Bicycle | CREDIT | 28-08-1992 | 1500 | 50 |
| 97 | Unisports | Junior Soccer Ball | CREDIT | 28-08-1992 | 9 | 1000 |
| 99 | Delhi Sports | Black Hawk Elbow Pad | CREDIT | 31-08-1992 | 8 | 25 |
| 99 | Delhi Sports | Black Hawk Knee Pads | CREDIT | 31-08-1992 | 9 | 18 |
| 99 | Delhi Sports | Cabrera Bat | CREDIT | 31-08-1992 | 45 | 69 |
| 99 | Delhi Sports | Griffey Glove | CREDIT | 31-08-1992 | 80 | 53 |
| 100 | Womansport | Bunny Boot | CREDIT | 31-08-1992 | 135 | 500 |
| 100 | Womansport | Bunny Ski Pole | CREDIT | 31-08-1992 | 14 | 500 |
| 100 | Womansport | Himalaya Bicycle | CREDIT | 31-08-1992 | 582 | 600 |
| 100 | Womansport | New Air Pump | CREDIT | 31-08-1992 | 20 | 450 |
| 100 | Womansport | Pro Ski Boot | CREDIT | 31-08-1992 | 380 | 400 |
| 100 | Womansport | Pro Ski Pole | CREDIT | 31-08-1992 | 36 | 400 |
| 100 | Womansport | Prostar 10 Pound Wei | CREDIT | 31-08-1992 | 8 | 250 |
| 101 | Kam's Sporting | Cabrera Bat | CREDIT | 31-08-1992 | 45 | 50 |
| 101 | Kam's Sporting | Grand Prix Bicycle T | CREDIT | 31-08-1992 | 16 | 15 |
| 101 | Kam's Sporting | Griffey Glove | CREDIT | 31-08-1992 | 80 | 27 |
| 101 | Kam's Sporting | Major League Basebal | CREDIT | 31-08-1992 | 4,29 | 40 |
| 101 | Kam's Sporting | Pro Curling Bar | CREDIT | 31-08-1992 | 50 | 30 |
| 101 | Kam's Sporting | Prostar 10 Pound Wei | CREDIT | 31-08-1992 | 8 | 20 |
| 101 | Kam's Sporting | Prostar 100 Pound We | CREDIT | 31-08-1992 | 45 | 35 |
| 112 | Futbol Sonora | Junior Soccer Ball | CREDIT | 31-08-1992 | 11 | 50 |

21 wierszy zostało wybranych.

Komentarz:

Wyświetlamy szczegóły zamówień, które regulowane były karta kredytową i które złożone zostały w sierpniu 1992. Z wyniku zapytania mamy możliwość odczytu:

- danych klienta, który składał zamówienie,
- nazwy produktu, na który wystawiono zamówienie,
- ceny, za którą sprzedano produkt,
- ilość sprzedanych produktów każdego rodzaju.

Zapytanie pobiera dane z czterech tabel. Zwróćmy uwagę również na sposób formatowania polecenia SQL. Dzięki wprowadzeniu dużej ilości znaków końca linii zapytanie zyskało na czytelności. Przy dużych zapytaniach (łączyjących dane z więcej niż 4-5 tabel) nie powinniśmy zbyt mocno oszczędzać na ilościach linii zajętych przez zapytanie, gdyż w efekcie otrzymamy bardzo mało czytelny tekst.

Aby „odczulić” zapytanie na format podawanych dat użyto funkcji TO_DATE. Do sformatowania daty użyto funkcji TO_CHAR.

Przykład 38

```

SELECT
  O.id "Nr zam.",
  RPAD(C.name, 20) "Klient",
  RPAD(C.city, 20) "Miasto",
  SUM(I.price * I.quantity) "Suma"
FROM

```

```

customer C,
ord O,
item I,
product P
WHERE
C.id = O.customer_id AND
O.id = I.ord_id AND
P.id = I.product_id AND
O.payment_type = 'CREDIT' AND
O.date_ordered BETWEEN
    TO_DATE('01-08-1992','DD-MM-YYYY') AND
    TO_DATE('31-08-1992','DD-MM-YYYY')
GROUP BY
    O.id, C.name, C.city
ORDER BY
    O.id;

```

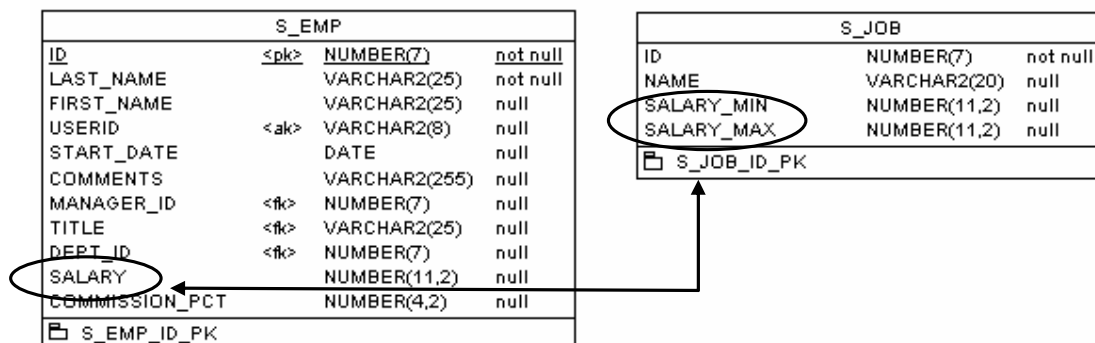
| Nr zam. | Klient | Miasto | Suma |
|---------|----------------------|-----------|--------|
| 97 | Unisports | Sao Paolo | 84000 |
| 99 | Delhi Sports | New Delhi | 7707 |
| 100 | Womansport | Seattle | 601100 |
| 101 | Kam's Sporting Goods | Hong Kong | 8056,6 |
| 112 | Futbol Sonora | Nogales | 550 |

Komentarz:

Na bazie zapytania z poprzedniego przykładu dokonaliśmy pogrupowania wyniku wg. zamówień. W wyniku umieściliśmy całkowitą sumę pieniędzy, na którą opiewało zamówienie. (przykładowo dla zamówienia o id=97 suma ta wynosi 84000. Łatwo sprawdzić poprawność tego wyniku korzystając z danych uzyskanych w poprzednim przykładzie – $50 \times 1500 + 9 \times 1000 = 84000$).

4.12.3. Złączenia nierównościami (ang. *Theta Join*)

Przykład 39



```

-- Tworzymy poniższą tabelę, aby móc zaprezentować
-- tzw. złączenia nierównościami.

DROP TABLE job;

CREATE TABLE job(
id          NUMBER(7) PRIMARY KEY,
name       VARCHAR2(20),
salary_min NUMBER(11,2),
salary_max NUMBER(11,2));

INSERT INTO job VALUES(1, 'President', 2000, 4000);

```

```

INSERT INTO job VALUES(2, 'Stock Clerk', 700, 1200);
INSERT INTO job VALUES(3, 'Sales Representative', 1200, 1400);
INSERT INTO job VALUES(4, 'Warehouse Manager', 1000, 1500);

```

```

SELECT * FROM job;

```

| ID | NAME | SALARY_MIN | SALARY_MAX |
|----|----------------------|------------|------------|
| 1 | President | 2000 | 4000 |
| 2 | Stock Clerk | 700 | 1200 |
| 3 | Sales Representative | 1200 | 1400 |
| 4 | Warehouse Manager | 1000 | 1500 |

4 wierszy zostało wybranych.

```

SELECT
  E.title, J.name, E.first_name, E.last_name,
  J.salary_min||' -- '||J.salary_max „Przedział”,
  E.salary
FROM
  emp E, job J
WHERE
  E.salary BETWEEN J.salary_min AND J.salary_max
ORDER BY
  E.last_name;

```

| TITLE | NAME | FIRST_NAME | LAST_NAME | Przedział | SALARY |
|----------------------|----------------------|------------|--------------|--------------|--------|
| Warehouse Manager | Stock Clerk | Ben | Biri | 700 -- 1200 | 1100 |
| Warehouse Manager | Warehouse Manager | Ben | Biri | 1000 -- 1500 | 1100 |
| Warehouse Manager | Sales Representative | Antoinette | Catchpole | 1200 -- 1400 | 1300 |
| Warehouse Manager | Warehouse Manager | Antoinette | Catchpole | 1000 -- 1500 | 1300 |
| Stock Clerk | Stock Clerk | Eddie | Chang | 700 -- 1200 | 800 |
| Stock Clerk | Stock Clerk | Bela | Dancs | 700 -- 1200 | 860 |
| Sales Representative | Warehouse Manager | Andre | Dumas | 1000 -- 1500 | 1450 |
| Sales Representative | Warehouse Manager | Henry | Giljum | 1000 -- 1500 | 1490 |
| Warehouse Manager | Sales Representative | Marta | Havel | 1200 -- 1400 | 1307 |
| Warehouse Manager | Warehouse Manager | Marta | Havel | 1000 -- 1500 | 1307 |
| Stock Clerk | Sales Representative | Elena | Maduro | 1200 -- 1400 | 1400 |
| Stock Clerk | Warehouse Manager | Elena | Maduro | 1000 -- 1500 | 1400 |
| Sales Representative | Sales Representative | Colin | Magee | 1200 -- 1400 | 1400 |
| Sales Representative | Warehouse Manager | Colin | Magee | 1000 -- 1500 | 1400 |
| Stock Clerk | Stock Clerk | Alexander | Markarian | 700 -- 1200 | 850 |
| Warehouse Manager | Sales Representative | Roberta | Menchu | 1200 -- 1400 | 1250 |
| Warehouse Manager | Warehouse Manager | Roberta | Menchu | 1000 -- 1500 | 1250 |
| VP, Sales | Sales Representative | Midori | Nagayama | 1200 -- 1400 | 1400 |
| VP, Sales | Warehouse Manager | Midori | Nagayama | 1000 -- 1500 | 1400 |
| Stock Clerk | Stock Clerk | Chad | Newman | 700 -- 1200 | 750 |
| VP, Operations | Warehouse Manager | LaDoris | Ngao | 1000 -- 1500 | 1450 |
| Stock Clerk | Stock Clerk | Akira | Nozaki | 700 -- 1200 | 1200 |
| Stock Clerk | Warehouse Manager | Akira | Nozaki | 1000 -- 1500 | 1200 |
| Stock Clerk | Sales Representative | Akira | Nozaki | 1200 -- 1400 | 1200 |
| Stock Clerk | Stock Clerk | Vikram | Palet | 700 -- 1200 | 795 |
| Stock Clerk | Stock Clerk | Radha | Palet | 700 -- 1200 | 795 |
| VP, Finance | Warehouse Manager | Mark | Quick-To-See | 1000 -- 1500 | 1450 |
| Stock Clerk | Stock Clerk | Sylvie | Schwartz | 700 -- 1200 | 1100 |
| Stock Clerk | Warehouse Manager | Sylvie | Schwartz | 1000 -- 1500 | 1100 |
| Stock Clerk | Stock Clerk | George | Smith | 700 -- 1200 | 940 |
| Warehouse Manager | Stock Clerk | Molly | Urguhart | 700 -- 1200 | 1200 |
| Warehouse Manager | Sales Representative | Molly | Urguhart | 1200 -- 1400 | 1200 |
| Warehouse Manager | Warehouse Manager | Molly | Urguhart | 1000 -- 1500 | 1200 |
| President | President | Carmen | Velasquez | 2000 -- 4000 | 2500 |

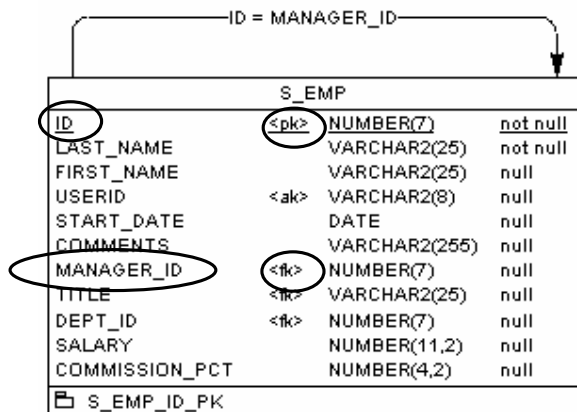
34 wierszy zostało wybranych.

Komentarz:

Powyższy przykład to tzw. **połączenie nierównościowe** (ang. *theta-join*). Złączamy ze sobą relacje, które nie są powiązane ze sobą więzami integralnościowymi. W powyższym przykładzie

wyświetlono listę pracowników oraz na bazie tabeli `JOB` sprawdzono, czy zarobki poszczególnych pracowników mieszczą się w „widełkach”. Można przykładowo zauważyć, że pracownik o nazwisku *Urguhart*, pracujący na stanowisku *Warehouse Manager* zarabia kwotę 1200, która to kwota mieści się w przedziale dla stanowisk *Stock Clerk* oraz *Sales Representative*. Z kolei pracownik o nazwisku *Biri* „podchodzi” pod zarobki dla stanowisk *Stock Clerk* oraz *Warehouse Manager*.

4.12.4. Złączenia zwrotne (ang. *Self Join*)



Przykład 40

```
SELECT last_name, title, id, manager_id FROM emp;
```

| LAST_NAME | TITLE | ID | MANAGER_ID |
|--------------|----------------------|----|------------|
| Velasquez | President | 1 | |
| Ngao | VP, Operations | 2 | 1 |
| Nagayama | VP, Sales | 3 | 1 |
| Quick-To-See | VP, Finance | 4 | 1 |
| Ropeburn | VP, Administration | 5 | 1 |
| Urguhart | Warehouse Manager | 6 | 2 |
| Menchu | Warehouse Manager | 7 | 2 |
| Biri | Warehouse Manager | 8 | 2 |
| Catchpole | Warehouse Manager | 9 | 2 |
| Havel | Warehouse Manager | 10 | 2 |
| Magee | Sales Representative | 11 | 3 |
| Giljum | Sales Representative | 12 | 3 |
| Sedeghi | Sales Representative | 13 | 3 |
| Nguyen | Sales Representative | 14 | 3 |
| Dumas | Sales Representative | 15 | 3 |
| Maduro | Stock Clerk | 16 | 6 |
| Smith | Stock Clerk | 17 | 6 |
| Nozaki | Stock Clerk | 18 | 7 |
| Patel | Stock Clerk | 19 | 7 |
| Newman | Stock Clerk | 20 | 8 |
| Markarian | Stock Clerk | 21 | 8 |
| Chang | Stock Clerk | 22 | 9 |
| Patel | Stock Clerk | 23 | 9 |
| Dancs | Stock Clerk | 24 | 10 |
| Schwartz | Stock Clerk | 25 | 10 |

25 wierszy zostało wybranych.

Komentarz:

W kolumnie `MANAGER_ID` wpisany jest identyfikator „szefa” danego pracownika. Zwróćmy uwagę, że pole `MANAGER_ID` u pracownika na stanowisku *President* ma wartość `NULL`, co oznacza, że nie ma on swojego zwierzchnika.

Przykład 41

```
SELECT
  E1.last_name, E1.title, E2.last_name, E2.title
FROM
  emp E1, emp E2
WHERE
  E1.manager_id = E2.id;
```

| LAST_NAME | TITLE | LAST_NAME | TITLE |
|--------------|----------------------|-----------|-------------------|
| Ngao | VP, Operations | Velasquez | President |
| Nagayama | VP, Sales | Velasquez | President |
| Quick-To-See | VP, Finance | Velasquez | President |
| Ropeburn | VP, Administration | Velasquez | President |
| Urguhart | Warehouse Manager | Ngao | VP, Operations |
| Menchu | Warehouse Manager | Ngao | VP, Operations |
| Biri | Warehouse Manager | Ngao | VP, Operations |
| Catchpole | Warehouse Manager | Ngao | VP, Operations |
| Havel | Warehouse Manager | Ngao | VP, Operations |
| Magee | Sales Representative | Nagayama | VP, Sales |
| Giljum | Sales Representative | Nagayama | VP, Sales |
| Sedeghi | Sales Representative | Nagayama | VP, Sales |
| Nguyen | Sales Representative | Nagayama | VP, Sales |
| Dumas | Sales Representative | Nagayama | VP, Sales |
| Maduro | Stock Clerk | Urguhart | Warehouse Manager |
| Smith | Stock Clerk | Urguhart | Warehouse Manager |
| Nozaki | Stock Clerk | Menchu | Warehouse Manager |
| Palet | Stock Clerk | Menchu | Warehouse Manager |
| Newman | Stock Clerk | Biri | Warehouse Manager |
| Markarian | Stock Clerk | Biri | Warehouse Manager |
| Chang | Stock Clerk | Catchpole | Warehouse Manager |
| Palet | Stock Clerk | Catchpole | Warehouse Manager |
| Dancs | Stock Clerk | Havel | Warehouse Manager |
| Schwartz | Stock Clerk | Havel | Warehouse Manager |

24 wierszy zostało wybranych.

Komentarz:

W pierwszej kolumnie wyświetlamy nazwisko pracownika, w drugiej stanowisko na jakim pracuje, w trzeciej nazwisko „szefa” a w czwartej stanowisko, na jakim pracuje „szef”. Użycie aliasów jest tutaj obowiązkowe.

Zwróćmy uwagę, że gdy w klauzuli `WHERE` zmienimy warunek na `E2.manager_id = E1.id` otrzymamy w wyniku inaczej posortowane rekordy. Ponadto w pierwszej oraz drugiej kolumnie są w tej chwili dane „szefa” a w trzeciej i czwartej dane pracownika. Trzeba o tym pamiętać, aby uniknąć trudnych do zdiagnozowania błędów.

| LAST_NAME | TITLE | LAST_NAME | TITLE |
|-----------|----------------|--------------|----------------------|
| Velasquez | President | Ngao | VP, Operations |
| Velasquez | President | Nagayama | VP, Sales |
| Velasquez | President | Quick-To-See | VP, Finance |
| Velasquez | President | Ropeburn | VP, Administration |
| Ngao | VP, Operations | Urguhart | Warehouse Manager |
| Ngao | VP, Operations | Menchu | Warehouse Manager |
| Ngao | VP, Operations | Biri | Warehouse Manager |
| Ngao | VP, Operations | Catchpole | Warehouse Manager |
| Ngao | VP, Operations | Havel | Warehouse Manager |
| Nagayama | VP, Sales | Magee | Sales Representative |
| Nagayama | VP, Sales | Giljum | Sales Representative |

| | | | |
|-----------|-------------------|-----------|----------------------|
| Nagayama | VP, Sales | Sedeghi | Sales Representative |
| Nagayama | VP, Sales | Nguyen | Sales Representative |
| Nagayama | VP, Sales | Dumas | Sales Representative |
| Urguhart | Warehouse Manager | Maduro | Stock Clerk |
| Urguhart | Warehouse Manager | Smith | Stock Clerk |
| Menchu | Warehouse Manager | Nozaki | Stock Clerk |
| Menchu | Warehouse Manager | Patel | Stock Clerk |
| Biri | Warehouse Manager | Newman | Stock Clerk |
| Biri | Warehouse Manager | Markarian | Stock Clerk |
| Catchpole | Warehouse Manager | Chang | Stock Clerk |
| Catchpole | Warehouse Manager | Patel | Stock Clerk |
| Havel | Warehouse Manager | Dancs | Stock Clerk |
| Havel | Warehouse Manager | Schwartz | Stock Clerk |

24 wierszy zostało wybranych.

Przykład 42

```

SELECT
  E.ID,
  E.manager_id,
  RPAD( RPAD(' ', 2*(LEVEL)-1, '.')||E.first_name ||' '||E.last_name, 30)
  "Imie, Nazwisko",
  E.title, LEVEL-1 "POZIOM"
FROM
  emp E
CONNECT BY PRIOR
  E.ID = E.manager_id
START WITH
  E.manager_id IS NULL;

```

| ID | MANAGER_ID | Imie, Nazwisko | TITLE | POZIOM |
|----|------------|--------------------------|----------------------|--------|
| 1 | | Carmen Velasquez | President | 0 |
| 2 | 1 | ..LaDoris Ngao | VP, Operations | 1 |
| 6 | 2 | ...Molly Urguhart | Warehouse Manager | 2 |
| 16 | 6 |Elena Maduro | Stock Clerk | 3 |
| 17 | 6 |George Smith | Stock Clerk | 3 |
| 7 | 2 | ...Roberta Menchu | Warehouse Manager | 2 |
| 18 | 7 |Akira Nozaki | Stock Clerk | 3 |
| 19 | 7 |Vikram Palet | Stock Clerk | 3 |
| 8 | 2 | ...Ben Biri | Warehouse Manager | 2 |
| 20 | 8 |Chad Newman | Stock Clerk | 3 |
| 21 | 8 |Alexander Markarian | Stock Clerk | 3 |
| 9 | 2 | ...Antoinette Catchpole | Warehouse Manager | 2 |
| 22 | 9 |Eddie Chang | Stock Clerk | 3 |
| 23 | 9 |Radha Palet | Stock Clerk | 3 |
| 10 | 2 | ...Marta Havel | Warehouse Manager | 2 |
| 24 | 10 |Bela Dancs | Stock Clerk | 3 |
| 25 | 10 |Sylvie Schwartz | Stock Clerk | 3 |
| 3 | 1 | ..Midori Nagayama | VP, Sales | 1 |
| 11 | 3 | ...Colin Magee | Sales Representative | 2 |
| 12 | 3 | ...Henry Giljum | Sales Representative | 2 |
| 13 | 3 | ...Yasmin Sedeghi | Sales Representative | 2 |
| 14 | 3 | ...Mai Nguyen | Sales Representative | 2 |
| 15 | 3 | ...Andre Dumas | Sales Representative | 2 |
| 4 | 1 | ..Mark Quick-To-See | VP, Finance | 1 |
| 5 | 1 | ..Audry Ropeburn | VP, Administration | 1 |

25 wierszy zostało wybranych.

Komentarz:

Polecenia `CONNECT BY PRIOR` oraz `START WITH` są specyficzną implementacją ORACLE-a, która pozwala w łatwy i wygodny sposób pobierać dane z tabel, gdzie zdefiniowano tzw. relację „sama do siebie”. Pseudokolumna `LEVEL` podaje „poziom”, na którym właśnie znajdujemy się.

Podwójne użycie `RPAD` spowodowane jest „dziwnym” zachowaniem się `SQL*PLUS` (zawija wiersze. Nie wiem – jak na razie – co jest tego przyczyną – AG).

Przykład 43

```
SELECT
  E.ID,
  E.manager_id,
  RPAD( RPAD(' ', 2*(LEVEL)-1, '.')||E.first_name ||' '||E.last_name,30)
  "Imie, Nazwisko",
  E.title, LEVEL-1 "POZIOM"
FROM
  emp E
CONNECT BY PRIOR
  E.ID = E.manager_id
START WITH
  E.manager_id IS NULL
ORDER BY LEVEL;
```

| ID | MANAGER_ID | Imie, Nazwisko | TITLE | POZIOM |
|----|------------|--------------------------|----------------------|--------|
| 1 | | Carmen Velasquez | President | 0 |
| 2 | 1 | ..LaDoris Ngao | VP, Operations | 1 |
| 4 | 1 | ..Mark Quick-To-See | VP, Finance | 1 |
| 3 | 1 | ..Midori Nagayama | VP, Sales | 1 |
| 5 | 1 | ..Audry Ropeburn | VP, Administration | 1 |
| 6 | 2 | ...Molly Urganhart | Warehouse Manager | 2 |
| 9 | 2 | ...Antoinette Catchpole | Warehouse Manager | 2 |
| 11 | 3 | ...Colin Magee | Sales Representative | 2 |
| 13 | 3 | ...Yasmin Sedeghi | Sales Representative | 2 |
| 15 | 3 | ...Andre Dumas | Sales Representative | 2 |
| 14 | 3 | ...Mai Nguyen | Sales Representative | 2 |
| 12 | 3 | ...Henry Giljum | Sales Representative | 2 |
| 10 | 2 | ...Marta Havel | Warehouse Manager | 2 |
| 8 | 2 | ...Ben Biri | Warehouse Manager | 2 |
| 7 | 2 | ...Roberta Menchu | Warehouse Manager | 2 |
| 16 | 6 |Elena Maduro | Stock Clerk | 3 |
| 24 | 10 |Bela Dancs | Stock Clerk | 3 |
| 25 | 10 |Sylvie Schwartz | Stock Clerk | 3 |
| 23 | 9 |Radha Palet | Stock Clerk | 3 |
| 22 | 9 |Eddie Chang | Stock Clerk | 3 |
| 21 | 8 |Alexander Markarian | Stock Clerk | 3 |
| 17 | 6 |George Smith | Stock Clerk | 3 |
| 18 | 7 |Akira Nozaki | Stock Clerk | 3 |
| 20 | 8 |Chad Newman | Stock Clerk | 3 |
| 19 | 7 |Vikram Palet | Stock Clerk | 3 |

25 wierszy zostało wybranych.

Komentarz:

W porównaniu do poprzedniego przykłady dane zostały posortowane wg. pseudokolumny `LEVEL`.

Przykład 44

```
SELECT
  E.ID,
  E.manager_id,
  RPAD(RPAD(' ', 2*(level)-1, '*')||E.first_name ||' '||E.last_name, 20)
  "Imie, Nazwisko",
  E.title, level-1 "POZIOM"
```

```

FROM
  emp E
CONNECT BY PRIOR
  E.ID = E.manager_id
START WITH
  E.title = 'VP, Sales';

```

| ID | MANAGER_ID | Imie, Nazwisko | TITLE | POZIOM |
|----|------------|------------------|----------------------|--------|
| 3 | 1 | Midori Nagayama | VP, Sales | 0 |
| 11 | 3 | **Colin Magee | Sales Representative | 1 |
| 12 | 3 | **Henry Giljum | Sales Representative | 1 |
| 13 | 3 | **Yasmin Sedeghi | Sales Representative | 1 |
| 14 | 3 | **Mai Nguyen | Sales Representative | 1 |
| 15 | 3 | **Andre Dumas | Sales Representative | 1 |

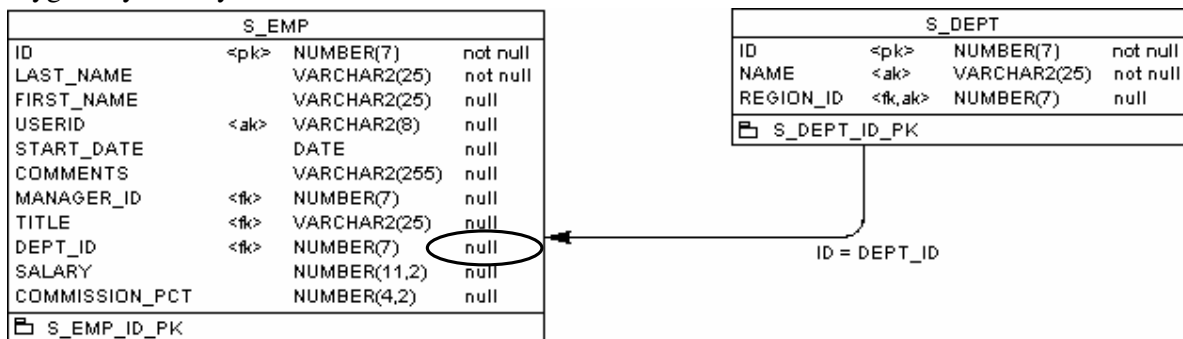
6 wierszy zostało wybranych.

Komentarz:

Tym razem wyświetlono tylko podległości dla stanowiska *VP, Sales*.

4.12.5. Złączenia zewnętrzne (ang. *Outer Joins*)

Aby omówić problem tzw. złączeń zewnętrznych wprowadzimy pewną modyfikację w oryginalnych danych modelu SUMMIT2.



W tabeli *EMP* w danych wybranych pracowników usuniemy informację o numerach działów, w których pracują. Wykonamy więc następujące polecenia:

```
UPDATE emp SET dept_id = NULL WHERE dept_id IN (41, 42, 43, 44, 45);
```

15 wierszy zostało zmodyfikowanych.

```
COMMIT;
```

Zatwierdzenie zostało ukończone.

```
SELECT first_name, last_name, dept_id FROM emp WHERE dept_id IS NULL;
```

| FIRST_NAME | LAST_NAME | DEPT_ID |
|------------|-----------|---------|
| LaDoris | Ngao | |
| Molly | Urguhart | |
| Roberta | Menchu | |
| Ben | Biri | |
| Antoinette | Catchpole | |
| Marta | Havel | |
| Elena | Maduro | |

| | |
|-----------|-----------|
| George | Smith |
| Akira | Nozaki |
| Vikram | Patel |
| Chad | Newman |
| Alexander | Markarian |
| Eddie | Chang |
| Bela | Dancs |
| Sylvie | Schwartz |

15 wierszy zostało wybranych.

Przykład 45

```
SELECT D.id, D.name, E.first_name, E.last_name
FROM dept D, emp E
WHERE D.id = E.dept_id
ORDER BY D.id;
```

| ID | NAME | FIRST_NAME | LAST_NAME |
|----|----------------|------------|--------------|
| 10 | Finance | Mark | Quick-To-See |
| 31 | Sales | Midori | Nagayama |
| 31 | Sales | Colin | Magee |
| 32 | Sales | Henry | Giljum |
| 33 | Sales | Yasmin | Sedeghi |
| 34 | Sales | Mai | Nguyen |
| 34 | Sales | Radha | Palet |
| 35 | Sales | Andre | Dumas |
| 50 | Administration | Carmen | Velasquez |
| 50 | Administration | Audry | Ropeburn |

10 wierszy zostało wybranych.

Komentarz:

Wiersze z obu relacji nie posiadające odpowiedników spełniających warunek połączenia nie są wyświetlane. W efekcie „gubimy” informację o pracownikach, którzy nie są przypisani do żadnego działu. Jest to z pewnością bardzo niekorzystne zjawisko. Gdy obecność pól z wartością NULL nie zostanie prawidłowo „obsłużona”, można spodziewać się wielu trudnych do zdiagnozowania błędów!

Przykład 46

```
SELECT D.id, D.name, E.first_name, E.last_name
FROM dept D, emp E
WHERE D.id(+) = E.dept_id
ORDER BY D.id;
```

| ID | NAME | FIRST_NAME | LAST_NAME |
|----|----------------|------------|--------------|
| 10 | Finance | Mark | Quick-To-See |
| 31 | Sales | Midori | Nagayama |
| 31 | Sales | Colin | Magee |
| 32 | Sales | Henry | Giljum |
| 33 | Sales | Yasmin | Sedeghi |
| 34 | Sales | Mai | Nguyen |
| 34 | Sales | Radha | Palet |
| 35 | Sales | Andre | Dumas |
| 50 | Administration | Carmen | Velasquez |
| 50 | Administration | Audry | Ropeburn |

| | |
|------------|-----------|
| LaDoris | Ngao |
| Molly | Urguhart |
| Roberta | Menchu |
| Elena | Maduro |
| Akira | Nozaki |
| Chad | Newman |
| Eddie | Chang |
| Sylvie | Schwartz |
| Bela | Dancs |
| Alexander | Markarian |
| Vikram | Palet |
| George | Smith |
| Ben | Biri |
| Antoinette | Catchpole |
| Marta | Havel |

25 wierszy zostało wybranych.

Komentarz:

Użycie operatora (+) spowodowało, że pojawiła się informacja o brakujących pracownikach (tych, którzy nie są przypisani do żadnego działu).

Operacja złączenia zewnętrznego **rozszerza możliwości** zwykłego złączenia. Zwraca ona te same rekordy co złączenie zwykle **plus** wszystkie te rekordy z tabeli EMP, które nie pasują do żadnego wiersza z tabeli DEPT.

Od wersji 9i Oracle wspiera również składnię zgodną z normą SQL, czyli operatory LEFT (RIGHT) OUTER JOIN. Więcej szczegółów patrz przypis ¹. Stara składnia nadal działa, ale powinno się z niej w miarę możliwości rezygnować.

Przykład 47

```
SELECT D.id, D.name, E.first_name, E.last_name
FROM dept D, emp E
WHERE D.id = E.dept_id(+)
ORDER BY D.id;
```

| ID | NAME | FIRST_NAME | LAST_NAME |
|-----------|-------------------|------------|--------------|
| 10 | Finance | Mark | Quick-To-See |
| 31 | Sales | Midori | Nagayama |
| 31 | Sales | Colin | Magee |
| 32 | Sales | Henry | Giljum |
| 33 | Sales | Yasmin | Sedeghi |
| 34 | Sales | Mai | Nguyen |
| 34 | Sales | Radha | Palet |
| 35 | Sales | Andre | Dumas |
| 41 | Operations | | |
| 42 | Operations | | |
| 43 | Operations | | |
| 44 | Operations | | |
| 45 | Operations | | |

¹ Artur Gramacki, Jarosław Gramacki: **Złączenia zewnętrzne i hierarchiczne w bazach danych**, W: Bazy danych - struktury, algorytmy, metody : architektura, metody formalne i eksploracja danych / red. S. Kozielski, B. Małysiak, P. Kasprowski, D. Mrozek .- Warszawa : Wydaw. Komunikacji i Łączności, 2006 - s. 237--248

Jarosław Gramacki, Artur Gramacki, **Złączenia wewnętrzne w bazach danych**, W: Bazy danych - struktury, algorytmy, metody : architektura, metody formalne i eksploracja danych / red. S. Kozielski, B. Małysiak, P. Kasprowski, D. Mrozek .- Warszawa : Wydaw. Komunikacji i Łączności, 2006 - s. 225--236

| | | |
|-------------------|--------|-----------|
| 50 Administration | Carmen | Velasquez |
| 50 Administration | Audry | Ropeburn |

15 wierszy zostało wybranych.

Komentarz:

Użycie operatora (+) spowodowało, że pojawiła się informacja o brakujących działach (tych o numerach od 41 do 45 nie są one przypisane do żadnego pracownika).

Operacja złączenia zewnętrznego **rozszerza możliwości** zwykłego złączenia. Zwracane są te same rekordy co złączenie zwykle **plus** wszystkie te rekordy z tabeli DEPT, które nie pasują do żadnego wiersza z tabeli EMP.

Uzyskany wynik również nie jest zadawalający, gdyż nie o to nam chodziło. Poprzedni przykład pokazuje wynik zgodny z oczekiwaniami. Okazuje się więc, że operator (+) musi być umieszczony z właściwej strony w klauzuli WHERE.

Przykład 48

```
SELECT D.id, D.name, E.first_name, E.last_name
FROM dept D, emp E
WHERE D.id(+) = E.dept_id(+)
ORDER BY D.id;
```

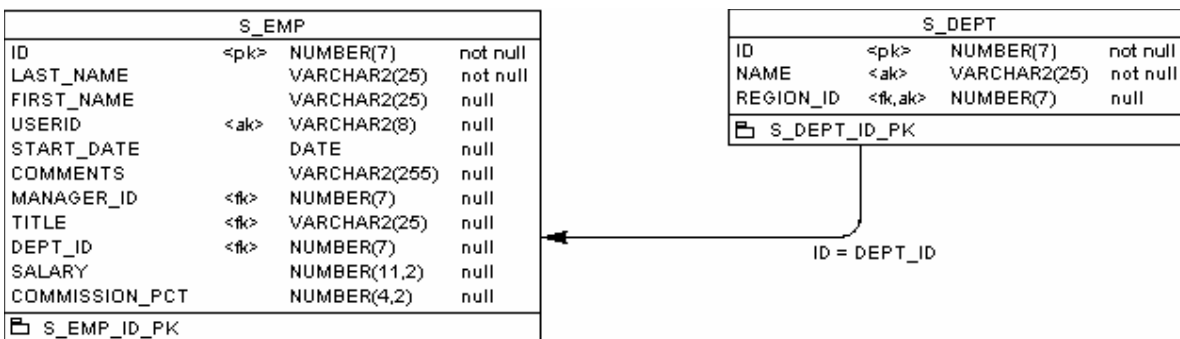
BŁĄD w linii 3:

ORA-01468: predykat może odwoływać się tylko do jednej łączonej zewnętrznie tabeli
(a predicate may reference only one outer-joined table)

Komentarz:

Użycie operatora (+) nie jest możliwe po obu stronach w klauzuli WHERE.

Przykład 49



```
SELECT D.id, SUM(E.salary) "Suma zar."
FROM dept D, emp E
WHERE D.id = E.dept_id
GROUP BY D.id;
```

| ID | Suma zar. |
|----|-----------|
| 10 | 1450 |
| 31 | 2800 |
| 32 | 1490 |
| 33 | 1515 |
| 34 | 2320 |
| 35 | 1450 |

```
50      4050
```

7 wierszy zostało wybranych.

Komentarz:

Tracimy informację o tym, że istnieją działy o numerach od 41 do 45. Jeżeli w tych działach nikt nie pracuje, to koszty płacowe będą tam zerowe. Taka informacja też jest istotna!

Przykład 50

```
SELECT D.id, SUM(E.salary) "Suma zar."
FROM dept D, emp E
WHERE D.id = E.dept_id(+)
GROUP BY D.id;
```

| ID | Suma zar. |
|-----------|-----------|
| 10 | 1450 |
| 31 | 2800 |
| 32 | 1490 |
| 33 | 1515 |
| 34 | 2320 |
| 35 | 1450 |
| 41 | |
| 42 | |
| 43 | |
| 44 | |
| 45 | |
| 50 | 4050 |

12 wierszy zostało wybranych.

Komentarz:

Pojawia się informacja o działach od 41 do 45. Mimo tego, że nie ma tam żadnych kosztów płacowych, to jednak informacja, że działy takie istnieją jest bardzo istotna.

Za pomocą funkcji NVL można zamiast wartości NULL wyświetlić np. liczbę zero, czyli zamiast SUM(E.salary) możemy napisać NVL(SUM(E.salary), 0).

Przykład 51

```
SELECT D.id, SUM(E.salary) "Suma zar."
FROM dept D, emp E
WHERE D.id(+) = E.dept_id
GROUP BY D.id;
```

| ID | Suma zar. |
|----|--------------|
| 10 | 1450 |
| 31 | 2800 |
| 32 | 1490 |
| 33 | 1515 |
| 34 | 2320 |
| 35 | 1450 |
| 50 | 4050 |
| | 16302 |

8 wierszy zostało wybranych.

Komentarz:

Tym razem operator (+) postawiono z drugiej strony niż w poprzednim przykładzie. Liczba w ostatnim wierszu jest po prostu sumą zarobków wszystkich pracowników, którzy nie są przypisani do żadnego działu (gdyż dokonujemy grupowania danych wg. pola dept.id). Aby sprawdzić ten wynik wykonajmy następujące zapytanie:

```
SELECT SUM(salary) FROM emp WHERE dept_id IS NULL;

SUM(SALARY)
-----
      16302
```

Przykład 52

```
SELECT D.id, TO_CHAR(SUM(E.salary)) "Suma zar."
FROM dept D, emp E
WHERE D.id = E.dept_id
GROUP BY D.id           -- brak średnika!
UNION
SELECT D.id, NULL       -- ta wartość NULL pojawi się w wyniku
FROM dept D
WHERE D.id NOT IN
      (SELECT dept_id FROM emp WHERE dept_id IS NOT NULL); -- podzapytanie
```

```
      ID  Suma zar.
-----  -
      10      1450
      31      2800
      32      1490
      33      1515
      34      2320
      35      1450
      41
      42
      43
      44
      45
      50      4050
```

12 wierszy zostało wybranych.

Komentarz:

Gdy operator (+) jest niedostępny (W ORACLE-u jest, w innych systemach niekoniecznie) należy użyć **operatora UNION**. Aby ułatwić zrozumienie istoty działania tego zapytania poniżej pokazano wyniki, jakie zwracają poszczególne jego fragmenty.

Jeżeli w wyniku działania zapytania nie zostaną znalezione wartości pasujące do D.dept_id, to wierszowi zostanie przypisana wartość NULL w kolumnie TO_CHAR(SUM(E.salary)).

Uwaga: gdy w drugim SELECT-ie nie będzie wartości numerycznej zostanie wygenerowany błąd:

ORA-01789: query block has incorrect number of result columns

Gdy zamiast zera będzie np. „a” (lub jakiś inny string) pojawi się błąd:

ORA-01790: expression must have same datatype as corresponding expression

| | | |
|--|-------------------------------------|---|
| SELECT D.id, SUM(E.salary) FROM dept D, emp E | SELECT D.id, 0 FROM dept D | SELECT dept_id FROM emp WHERE dept_id IS NOT NULL; |
|--|-------------------------------------|---|

| | | |
|---|--|--|
| <pre>WHERE D.id = E.dept_id GROUP BY D.id</pre> | <pre>WHERE D.id NOT IN (SELECT dept_id FROM emp WHERE dept_id IS NOT NULL);</pre> | |
| <pre> ID Suma zar. ----- 10 1450 31 2800 32 1490 33 1515 34 2320 35 1450 50 4050</pre> | <pre> ID 0 ----- 41 0 42 0 43 0 44 0 45 0</pre> | <pre> DEPT_ID ----- 50 31 10 50 31 32 33 34 35 34</pre> |

4.13. Operatory UNION, UNION ALL, INTERSECT, MINUS

Przykład 53

| | |
|--|---|
| <pre>SELECT region_id, name FROM dept ORDER BY name;</pre> | <pre>SELECT id, name FROM region;</pre> |
| <pre>REGION_ID NAME ----- 1 Administration 1 Finance 1 Operations 2 Operations 5 Operations 4 Operations 3 Operations 1 Sales 2 Sales 3 Sales 4 Sales 5 Sales 12 wierszy zostało wybranych.</pre> | <pre> ID NAME ----- 1 North America 2 South America 3 Africa / Middle East 4 Asia 5 Europe 5 wierszy zostało wybranych.</pre> |

```
SELECT name FROM region
UNION
SELECT name FROM dept;
```

```
NAME
-----
Administration
Africa / Middle East
Asia
Europe
Finance
North America
Operations
Sales
South America
```


9 wierszy zostało wybranych.

Komentarz:

Operator `UNION` umożliwia połączenie dwóch lub więcej instrukcji `SELECT` z jednoczesnym sumowaniem ich wyników. Wiersze wynikowe każdej instrukcji `SELECT` zostają obliczone i ustawione jeden pod drugim, po czym zostają posortowane w celu wyeliminowania powtarzających się wyników.

Zwykłe złączenie wierszy z dwóch lub więcej tabel powoduje, że wiersze układają się obok siebie. Operator `UNION` sprawia, że wiersze z różnych instrukcji `SELECT` układają się w wyniku jeden pod drugim.

Istnieje też następujący wariant: `UNION ALL` – w wyniku jego działania nie dochodzi do posortowania wierszy. Pozostają więc w nich powtarzające się wyniki.

Składnia `UNION` jest następująca:

```
Zapytanie_1 UNION zapytanie_2 ... [ORDER BY ...];
```

Przykład:

```
SELECT A, B FROM tabela1 WHERE ...  
UNION  
SELECT X, Y FROM tabela2 WHERE ...  
ORDER BY 2, 1;
```

Uwagi do powyższego:

- liczba wyrażeń w każdym zapytaniu musi być taka sama (u nas: A, B oraz X, Y),
- typy danych odpowiadających sobie wyrażeń w każdym zapytaniu muszą być zgodne (u nas: X musi mieć być tego samego typu co A, a Y tego samego typu co B),
- nazwy odpowiadających sobie kolumn w dwóch zapytaniach nie muszą być takie same (u nas: A i X oraz B i Y),
- ponieważ istnieje taka możliwość, że nazwy odpowiadających sobie kolumn w zapytaniach będą różne, sortowanie wartości za pomocą opcjonalnej klauzuli `ORDER BY` musi się odbywać według pozycji, a nie nazwy (u nas: `ORDER BY 2, 1`). Jedynym wyjątkiem jest sytuacja, gdy nazwy odpowiadających sobie kolumn, po których odbywa się sortowanie, są takie same (jak w bieżącym przykładzie),
- polecenie `UNION` może być używane do przeprowadzania włamań do systemów bazodanowych!

Przykład 54

```
SELECT name FROM region  
UNION ALL  
SELECT name FROM dept;
```

```
NAME  
-----  
North America  
South America  
Africa / Middle East  
Asia  
Europe  
Finance
```

```
Sales
Sales
Sales
Sales
Sales
Operations
Operations
Operations
Operations
Operations
Administration
```

17 wierszy zostało wybranych.

Komentarz:

Operator `UNION ALL` nie usuwa powtarzających się wartości, nie dochodzi też do sortowania wyników. Pierwsze 5 wierszy pochodzi z tabeli `REGION` a ostatnie 12 z tabeli `DEPT`.

Przykład 55

```
SELECT DISTINCT name FROM region
UNION ALL
SELECT DISTINCT name FROM dept
ORDER BY 1 DESC;
```

```
NAME
-----
South America
Sales
Operations
North America
Finance
Europe
Asia
Africa / Middle East
Administration
```

9 wierszy zostało wybranych.

Komentarz:

Tym razem użyliśmy dodatkowo operatora `DISTINCT`, który spowodował, że usunięte zostały duplikaty a wynik końcowy jest posortowany malejąco – jawnie przez użytkownika a nie automatycznie.

Przykład 56

```
SELECT dept_id, last_name FROM emp WHERE last_name LIKE 'N%'
UNION
SELECT id, name FROM region
ORDER BY 2;
```

```
DEPT_ID LAST_NAME
-----
        3 Africa / Middle East
        4 Asia
        5 Europe
       31 Nagayama
       43 Newman
       41 Ngao
       34 Nguyen
```

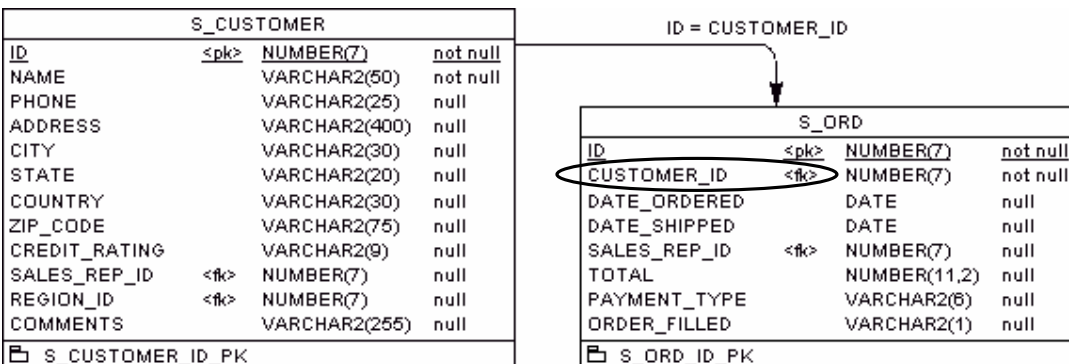
```
1 North America
42 Nozaki
2 South America
```

10 wierszy zostało wybranych.

Komentarz:

Tym razem kolumny w oby tabelach mają inne nazwy. Jest to dopuszczalne. Trzeba tylko pamiętać, aby odpowiadające sobie kolumny były tego samego typu. Dlatego też w klauzuli `ORDER BY` musimy posługiwać się numerami kolumn a nie ich nazwami.

Przykład 57



```
SELECT id FROM customer
INTERSECT
SELECT customer_id FROM ord;
```

```

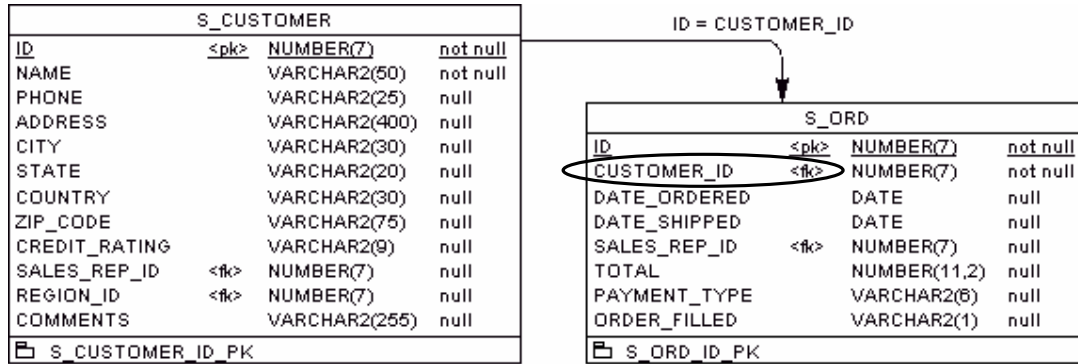
ID
-----
201
202
203
204
205
206
208
209
210
211
212
213
214
```

13 wierszy zostało wybranych.

Komentarz:

Operator `INTERSECT` zwraca te wiersze, które występują w oby zapytaniach `SELECT`. W naszym przypadku zwrócony wynik należy odczytać następująco: wyświetlone są numery ID tych klientów, którzy złożyli choć **jedno** zamówienie.

Przykład 58



```
SELECT id FROM customer
MINUS
SELECT customer_id FROM ord;
```

```
   ID
-----
   207
   215
```

Komentarz:

Operator MINUS zwraca te wiersze, które występują w pierwszym zapytaniu SELECT a nie występują w drugim. W naszym przypadku zwrócony wynik należy odczytać następująco: wyświetlone są numery ID tych klientów, którzy nie złożyli **ani jednego** zamówienia.

4.14. Podzapytania

4.14.1. Podzapytania zwracające jeden rekord

Przykład 59

```
SELECT first_name, last_name, salary
FROM emp
WHERE salary =
  (SELECT MIN(salary) FROM emp);
```

| FIRST_NAME | LAST_NAME | SALARY |
|------------|-----------|--------|
| Chad | Newman | 750 |

Komentarz:

Wyświetlamy dane pracownika zarabiającego najmniej.

Zapytanie z podzapytaniem działa w taki sposób, że jako pierwszy wyznaczany jest wynik podzapytania i jest on zapamiętywany w buforze tymczasowym. Następnie warunek w głównym zapytaniu jest sprawdzany z wynikiem podzapytania. Jeżeli wynik jest dodatni dane zostają zaliczone do wyniku ostatecznego. W przeciwnym wypadku są one odrzucane.

Przykład 60

```
SELECT first_name, last_name, salary
FROM emp
```

```
WHERE salary =
  (SELECT MIN(salary)
   FROM emp
   WHERE title = 'Warehouse Manager');
```

| FIRST_NAME | LAST_NAME | SALARY |
|------------|-----------|--------|
| Ben | Biri | 1100 |

Komentarz:

Wyświetlamy dane pracownika zarabiającego najmniej spośród tych, którzy pracują na stanowisku *Warehouse Manager*. Zwróćmy uwagę, na to, jak sformułowano warunek `WHERE`. Możemy też napisać tak jak poniżej i otrzymamy ten sam wynik (jednak zapytanie niepotrzebnie się komplikuje).

```
SELECT first_name, last_name, salary
FROM emp
WHERE (title, salary) =
  (SELECT 'Warehouse Manager', MIN(salary)
   FROM emp
   WHERE title = 'Warehouse Manager');
```

Przykład 61

```
SELECT first_name, last_name, salary
FROM emp
WHERE salary =
  (SELECT MIN(salary), dept_id
   FROM emp
   WHERE title = 'Warehouse Manager');
```

BŁĄD w linii 4:
ORA-00913: za duża liczba wartości
 (too many values)

```
SELECT first_name, last_name, salary
FROM emp
WHERE salary =
  (SELECT salary
   FROM emp
   WHERE title = 'Warehouse Manager');
```

BŁĄD w linii 4:
ORA-01427: jednowierszowe podzapytanie zwraca więcej niż jeden wiersz
 (single-row subquery returns more than one row)

Komentarz:

Musimy pamiętać o tym, aby liczba i typy wartości w klauzuli `SELECT` podzapytania była zgodna z tym, czego oczekuje klauzula `WHERE`.

4.14.2. Podzapytania zwracające więcej niż jeden rekord

Przykład 62

```
SELECT first_name, last_name, salary, title
FROM emp
WHERE (title, salary) IN
  (SELECT title, MIN(salary)
```

```
FROM emp
GROUP BY title);
```

| FIRST_NAME | LAST_NAME | SALARY | TITLE |
|------------|--------------|--------|----------------------|
| Carmen | Velasquez | 2500 | President |
| Colin | Magee | 1400 | Sales Representative |
| Chad | Newman | 750 | Stock Clerk |
| Audry | Ropeburn | 1550 | VP, Administration |
| Mark | Quick-To-See | 1450 | VP, Finance |
| LaDoris | Ngao | 1450 | VP, Operations |
| Midori | Nagayama | 1400 | VP, Sales |
| Ben | Biri | 1100 | Warehouse Manager |

8 wierszy zostało wybranych.

Komentarz:

Wyświetlamy dane pracowników, którzy zarabiają najmniej na poszczególnych stanowiskach. Istotna jest tutaj klauzula `GROUP BY`. Ponieważ podzapytanie zwraca więcej niż jeden rekord trzeba w klauzuli `WHERE` użyć operatora `IN`. Użycie operatora porównania = spowoduje wystąpienie błędu `ORA-01427`.

Przykład 63

```
SELECT first_name, last_name, salary
FROM emp
WHERE salary <
(SELECT AVG(salary) FROM emp);
```

| FIRST_NAME | LAST_NAME | SALARY |
|------------|-----------|--------|
| Molly | Urguhart | 1200 |
| Roberta | Menchu | 1250 |
| Ben | Biri | 1100 |
| George | Smith | 940 |
| Akira | Nozaki | 1200 |
| Vikram | Palet | 795 |
| Chad | Newman | 750 |
| Alexander | Markarian | 850 |
| Eddie | Chang | 800 |
| Radha | Palet | 795 |
| Bela | Dancs | 860 |
| Sylvie | Schwartz | 1100 |

12 wierszy zostało wybranych.

Komentarz:

Wyświetlamy dane o pracownikach, którzy zarabiają mniej niż wynosi średnia dla wszystkich pracowników.

Istnieją sytuacje, gdy wymaganego wyniku nie uzyskamy inaczej, jak z użyciem podzapytań. Taki przypadek jest przedstawiony w bieżącym przykładzie. Porównajmy:

```
SQL> SELECT first_name, salary FROM emp WHERE salary < AVG(salary);
SELECT first_name, salary FROM emp WHERE salary < AVG(salary)
```

BŁĄD w linii 1:

```
ORA-00934: funkcja grupowa nie jest tutaj dozwolona
(group function is not allowed here)
```

Mogę próbować uzyskać podobny efekt „na piechotę”:

```
SELECT AVG(salary) FROM emp
```

```
AVG(SALARY)
-----
      1255,08
```

```
SELECT first_name, last_name
FROM emp
WHERE salary < 1255.08
```

Przykład 64

```
SELECT first_name, last_name, salary
FROM emp
WHERE
  salary < (SELECT AVG(salary) FROM emp)
  AND
  dept_id IN (SELECT id FROM dept WHERE name = 'Sales');
```

| FIRST_NAME | LAST_NAME | SALARY |
|------------|-----------|--------|
| Radha | Palet | 795 |

Komentarz:

Wyświetlamy dane o pracownikach, którzy zarabiają mniej niż wynosi średnia dla wszystkich pracowników oraz (logiczne AND) pracują w dziale o nazwie *Sales*.

Podzapytania można zagłębiać, jednak należy robić to bardzo ostrożnie. Stają się one wówczas mało czytelne a ponadto bardzo wzrasta czas ich wykonywania.

4.14.3. Operatory ANY oraz ALL

Przykład 65

```
SELECT first_name, last_name, salary
FROM emp
WHERE salary >
  (SELECT salary FROM emp WHERE last_name = 'Patel');
```

BŁĄD w linii 4:

**ORA-01427: jednowierszowe podzapytanie zwraca więcej niż jeden wiersz
(single-row subquery returns more than one row)**

```
SELECT first_name, last_name, salary
FROM emp
WHERE salary > ANY
  (SELECT salary FROM emp WHERE last_name = 'Patel');
```

| FIRST_NAME | LAST_NAME | SALARY |
|------------|-----------|--------|
| Carmen | Velasquez | 2500 |
| ... | ... | ... |
| Sylvie | Schwartz | 1100 |

22 wierszy zostało wybranych.

Komentarz:

W zapytaniu chcieliśmy wyświetlić listę pracowników, którzy zarabiają więcej niż pracownik o nazwisku *Patel*. Niestety pojawił się błąd, gdyż istnieje dwóch pracowników o tym nazwisku.

W takich sytuacjach bardzo przydaje się operator `ANY`. W powyższym przykładzie warunek w `WHERE` jest prawdziwy, gdy liczba po lewej stronie jest większa niż jedna z liczb na liście. Zapis „> ANY” należy tłumaczyć jako „większe niż **przynajmniej jeden** element listy”.

Przykład 66

```
SELECT first_name, last_name, salary
FROM emp
WHERE salary > ALL
  (SELECT salary FROM emp WHERE last_name LIKE 'S%');
```

| FIRST_NAME | LAST_NAME | SALARY |
|------------|-----------|--------|
| Carmen | Velasquez | 2500 |
| Audry | Ropeburn | 1550 |
| Mai | Nguyen | 1525 |

Komentarz:

Istnieje też podobny do operatora `ANY` operator `ALL`, który służy do porównywania wartości ze wszystkimi wartościami zwracanymi przez podzapytanie. Zapis „> ALL” należy więc tłumaczyć jako „większe niż **każdy** element listy”.

Aby przekonać się, że wynik jest rzeczywiście poprawny wykonajmy polecenie `SELECT` z podzapytania i stwierdzamy, że rzeczywiście powyższe zapytanie wyświetliło dane tylko tych pracowników, którzy zarabiają więcej niż 1515 **oraz** więcej niż 940 **oraz** więcej niż 1100.

```
SELECT salary FROM emp WHERE last_name LIKE 'S%';

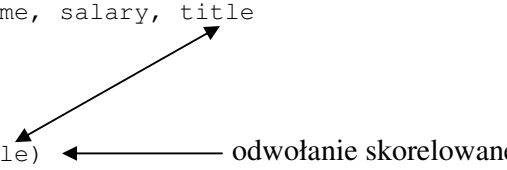
SALARY
-----
1515
940
1100
```

Używając operatora `ALL` należy ostrożnie używać go z operatorem równa się (np. `salary = ALL`), gdyż w przypadku, gdy lista zwracana w podzapytaniu zawiera różne wartości (np. 1515, 940, 1100 jak w powyższym przykładzie) całe zapytanie nigdy nie zwróci żadnego rekordu. Dzieje się tak dlatego, że liczba po lewej stronie nie może być **jednocześnie** równa 1515, 940 oraz 1100.

4.14.4. Podzapytania skorelowane, operatory `EXISTS` oraz `NOT EXISTS`

Przykład 67

```
SELECT first_name, last_name, salary, title
FROM emp E1
WHERE salary <
  (SELECT AVG(salary)
   FROM emp E2
   WHERE E2.title = E1.title)
ORDER BY title, salary;
```



| FIRST_NAME | LAST_NAME | SALARY | TITLE |
|------------|-----------|--------|----------------------|
| Colin | Magee | 1400 | Sales Representative |
| Andre | Dumas | 1450 | Sales Representative |
| Chad | Newman | 750 | Stock Clerk |
| Vikram | Patel | 795 | Stock Clerk |

| | | | |
|-----------|-----------|------|-------------------|
| Radha | Patel | 795 | Stock Clerk |
| Eddie | Chang | 800 | Stock Clerk |
| Alexander | Markarian | 850 | Stock Clerk |
| Bela | Dancs | 860 | Stock Clerk |
| George | Smith | 940 | Stock Clerk |
| Ben | Biri | 1100 | Warehouse Manager |
| Molly | Urguhart | 1200 | Warehouse Manager |

11 wierszy zostało wybranych.

Dla sprawdzenia wykonajmy:

```
SELECT AVG(salary), title FROM emp GROUP BY title;
```

```
AVG(SALARY)  TITLE
-----
          2500 President
          1476 Sales Representative
           949 Stock Clerk
          1550 VP, Administration
          1450 VP, Finance
          1450 VP, Operations
          1400 VP, Sales
          1231,4 Warehouse Manager
```

8 wierszy zostało wybranych.

Komentarz:

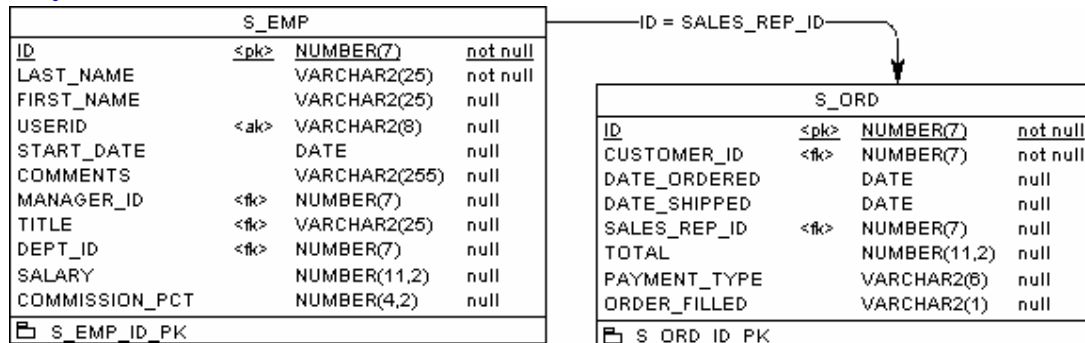
Wyświetlany dane pracowników, którzy zarabiają poniżej średniej zarobków dla swojego stanowiska (pole `title`).

Używając zwykłego podzapytania, jest ono obliczane jako pierwsze, a wyniki tymczasowe są przechowywane w buforach tymczasowych. Warunek w głównym zapytaniu jest sprawdzany z wynikiem podzapytania. Wiersze zostają zaliczone do wyniku zapytania, jeżeli spełniają warunek podzapytania.

Innym rodzajem podzapytań są tzw. **podzapytania skorelowane**. W takim podzapytaniu wartość z głównego zapytania jest **przekazywana** do podzapytania, aby mogło być ono wykonane.

W naszym przykładzie najpierw zostanie pobrany pierwszy rekord przez główne zapytanie. W kolejnym kroku zostanie wykonane podzapytanie na bazie danych zwróconych przez główne zapytanie. W zależności od otrzymanego wyniku, dane zostaną zaliczone bądź też odrzucone. Powyższa procedura zostanie powtórzona dla każdego wiersza zwracanego przez główne zapytanie.

Przykład 68



```
SELECT id, first_name, last_name
FROM emp E
WHERE EXISTS
  (SELECT 1 FROM ord O
   WHERE O.sales_rep_id = E.id);
```

| ID | FIRST_NAME | LAST_NAME |
|----|------------|-----------|
| 11 | Colin | Magee |
| 12 | Henry | Giljum |
| 13 | Yasmin | Sedeghi |
| 14 | Mai | Nguyen |
| 15 | Andre | Dumas |

Komentarz:

Wyrażenie z operatorem logicznym EXISTS jest prawdziwe, gdy w wyniku działania podzapytania zostanie zwrócony co najmniej jeden rekord. W przeciwnym wypadku jest ono fałszywe. Operator NOT EXISTS działa przeciwnie do EXISTS.

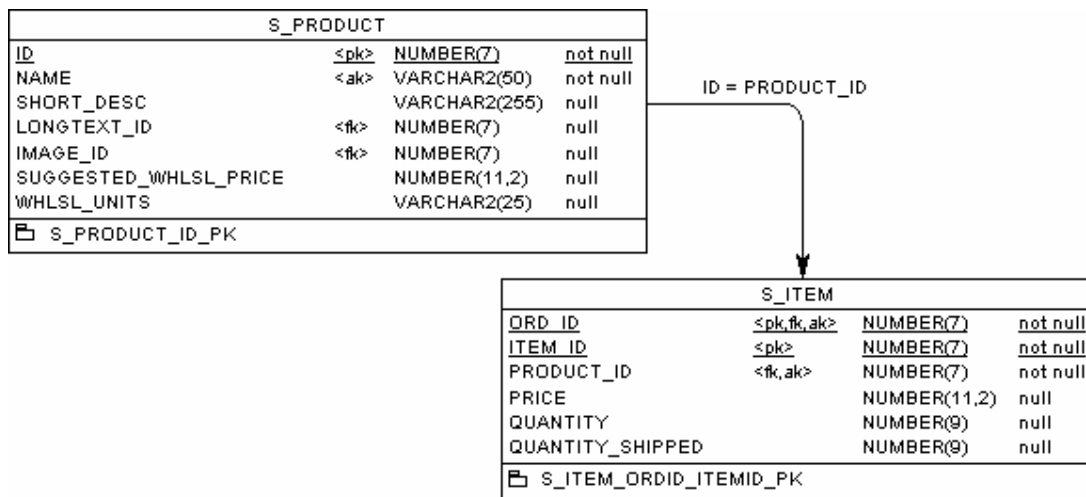
W przykładzie wyświetlamy dane pracowników, którzy choć raz „opiekowali” się złożonym przez klienta zamówieniem (czyli ich numer ID występuje choć raz w tabeli ORD). Dla sprawdzenia poprawności możemy wykonać poniższe zapytanie:

```
SQL> SELECT sales_rep_id, COUNT(*) FROM ord GROUP BY sales_rep_id;
```

| SALES_REP_ID | COUNT(*) |
|--------------|----------|
| 11 | 5 |
| 12 | 3 |
| 13 | 1 |
| 14 | 3 |
| 15 | 4 |

Z powyższego wynika, że tylko pięciu pracowników (z ich całkowitej liczby 25) brało udział w realizowaniu zamówień.

Przykład 69

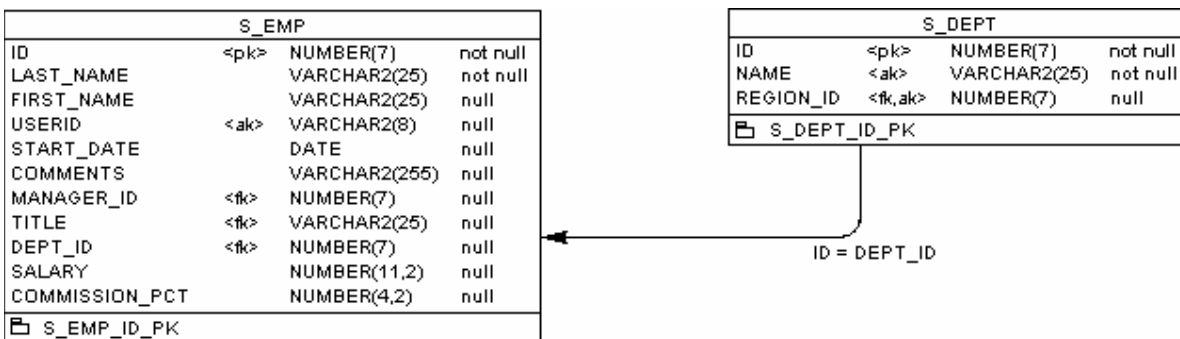


```
SELECT name FROM product P
WHERE NOT EXISTS
  (SELECT * FROM item I
   WHERE I.product_id = P.id);
```

```
NAME
-----
Prostar 20 Pound Weight
```

Komentarz:

Przykład analogiczny do poprzedniego. Wyświetlamy nazwy produktów, które nie pojawiły się w żadnym zamówieniu. Tutaj używamy operatora `NOT EXISTS`.

4.14.5. Przykłady podzapytań, które można zastąpić złączeniami**Przykład 70**

```
SELECT first_name, last_name, dept_id
FROM emp
WHERE dept_id IN
  (SELECT id FROM dept WHERE name = 'Sales');
```

| FIRST_NAME | LAST_NAME | DEPT_ID |
|------------|-----------|---------|
| Midori | Nagayama | 31 |
| Colin | Magee | 31 |
| Henry | Giljum | 32 |
| Yasmin | Sedeghi | 33 |
| Mai | Nguyen | 34 |
| Andre | Dumas | 35 |
| Radha | Palet | 34 |

7 wierszy zostało wybranych.

Komentarz:

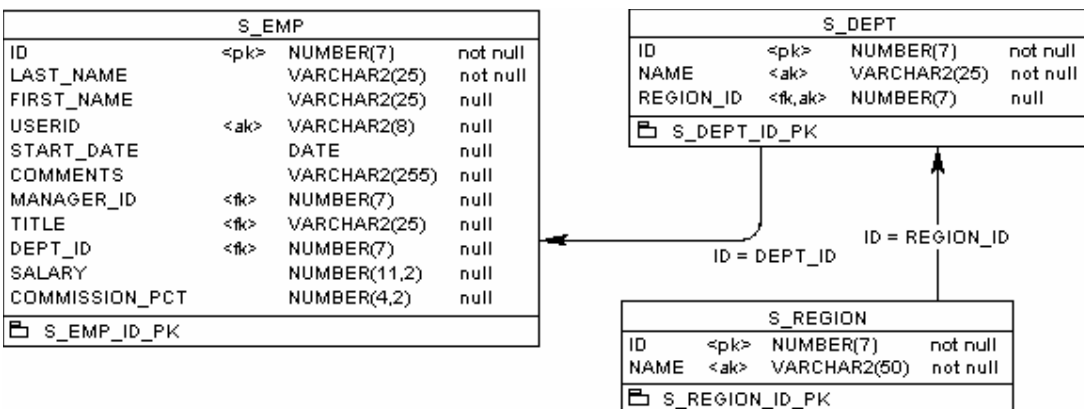
Wyświetlamy dane o pracownikach, którzy pracują w dziale o nazwie *Sales* (pamiętajmy, że w tabeli `DEPT` jest pięć działów o nazwie *Sales*, każdy zlokalizowany w innym regionie świata).

Tego typu zapytania wykonują się stosunkowo wolno. Podzapytań należy więc używać ostrożnie.

Często wymagany wynik można uzyskać bez potrzeby stosowania podzapytań. Jeżeli jest taka możliwość, powinniśmy ją wykorzystać. Po prostu złączenia wykonują się o wiele szybciej niż podzapytania. Porównajmy:

```
SELECT E.first_name, E.last_name, D.id
FROM emp E, dept D
WHERE E.dept_id = D.id AND
      D.name = 'Sales';
```

Przykład 71



```

SELECT first_name, last_name
FROM emp
WHERE dept_id IN
(
    SELECT id FROM dept
    WHERE region_id IN
    (
        SELECT id FROM region
        WHERE name = 'Europe'
    )
)
);
    
```

| FIRST_NAME | LAST_NAME |
|------------|-----------|
| Marta | Havel |
| Andre | Dumas |
| Bela | Dancs |
| Sylvie | Schwartz |

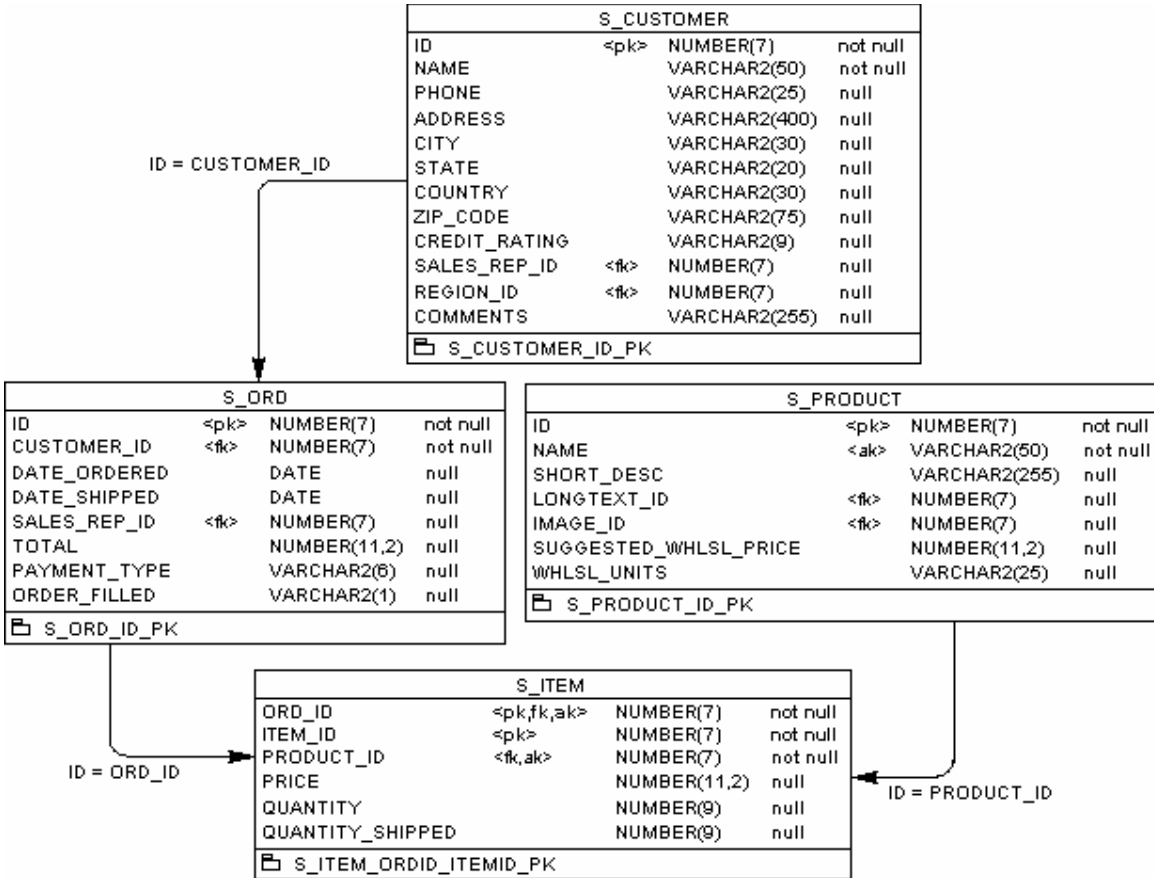
Komentarz:

Wyświetlamy dane o pracownikach, którzy pracują w jakimkolwiek dziale zlokalizowanym w Europie. Używamy **zagłębionych podzapytań**. Oczywiście dużo prościej jest to napisać z użyciem złączeń. A dodatkowo uzyskamy to, że takie zapytanie będzie działało zdecydowanie szybciej. Porównajmy:

```

SELECT E.first_name, E.last_name
FROM emp E, dept D, region R
WHERE
    E.dept_id = D.id AND
    D.region_id = R.id AND
    R.name = 'Europe';
    
```

Przykład 72



```

SELECT I.product_id, P.name, C.name
FROM item I, product P, customer C, ord O
WHERE
  I.product_id = P.id AND
  I.ord_id = O.id AND
  O.customer_id = C.id AND
  ord_id IN
  (
    SELECT id FROM ord
    WHERE customer_id IN
      (
        SELECT id FROM customer
        WHERE name = 'Unisports'
      )
  )
);
    
```

| PRODUCT_ID | NAME | NAME |
|------------|--------------------|-----------|
| 20106 | Junior Soccer Ball | Unisports |
| 30321 | Grand Prix Bicycle | Unisports |

Komentarz:

Wyświetlamy nazwy wszystkich produktów (z tabeli `PRODUCT`), które zostały zamówione przez klienta (tabela `CUSTOMER`) o nazwie *Unisports*. Użycie zagnieżdżonych podzapytań bardzo zagmatwało zapytanie. Zdecydowanie prościej uzyskamy ten sam wynik z użyciem złączeń. Porównajmy:

```

SELECT I.product_id, P.name, C.name
FROM item I, ord O, customer C, product P
WHERE
  I.product_id = P.id AND
  I.ord_id = O.id AND
  O.customer_id = C.id AND
  C.name = 'Unisports';

```

4.14.6. Podzapytania w klauzuli FROM

Przykład 73

```

SELECT 1 FROM (SELECT 1 FROM dual);

```

```

      1
-----
      1

```

```

-- Zwróćmy uwagę, że jest „gwiazdka” a wyświetla się tylko jedna kolumna.
-- Jest to kolumna z „widoku dynamicznego” zdefiniowanego w podzapytaniu.
SELECT * FROM
  (SELECT name FROM dept);

```

```

NAME
-----
Administration
Finance
Operations
Operations
Operations
Operations
Operations
Sales
Sales
Sales
Sales
Sales

```

```

SELECT moj_alias.*, ROWNUM rnum FROM
  (SELECT name FROM summit2.dept) moj_alias;

```

| NAME | ROWNUM |
|----------------|--------|
| Administration | 1 |
| Finance | 2 |
| Operations | 3 |
| Operations | 4 |
| Operations | 5 |
| Operations | 6 |
| Operations | 7 |
| Sales | 8 |
| Sales | 9 |
| Sales | 10 |
| Sales | 11 |
| Sales | 12 |

Komentarz:

Zwykle podzapytania używane są w klauzuli WHERE. Noszą one nazwę podzapytań zagnieżdżonych (ang. *nested subquery*). Istnieje również możliwość tworzenia podzapytań w klauzuli FROM.

Wówczas noszą one nazwę *inline views*. Polecenie w podzapytaniu można traktować jako swego rodzaju widok dynamiczny.

Zwróćmy uwagę, że w ostatnim przykładzie utworzono aliasy. Możliwość tworzenia takich aliasów jest bardzo przydatna w praktyce. Kolejne przykłady wyjaśniają to zagadnienie.

Przykład 74

```
-- Tu jest znak mniejszości

SELECT last_name FROM emp WHERE ROWNUM < 5;

LAST_NAME
-----
Velasquez
Ngao
Nagayama
Quick-To-See
```

```
-- Tu jest znak większości

SELECT last_name FROM summit2.emp WHERE ROWNUM > 5;
```

nie wybrano żadnych wierszy

Komentarz:

Tzw. pseudokolumna `ROWNUM` pokazuje kolejne numery **wyświetlanych** rekordów. Pierwszy wyświetlany rekord ma numer 1, drugi 2 itd. Używając tej pseudokolumny można ograniczać ilość wyświetlanych wierszy. O ile jednak pierwszy przykład zadziała prawidłowo, to już drugi **NIGDY** nie zwróci żadnych danych.

Dzieje się tak dlatego, że Oracle przypisuje numery rekordom już **po** uwzględnieniu wszystkich warunków w klauzuli `WHERE`. Pierwszy wybrany przez zapytanie wiersz ustawia wartość `ROWNUM` na 1 a ponieważ w warunki mamy `WHERE > 5`, więc warunek ten nie jest spełniony i wiersz nie zostaje wyświetlony (zostaje odrzucony). Kolejny pobierany wiersz ponownie ustawia wartość `ROWNUM` na wartość 1 i ponownie warunek `WHERE` nie jest spełniony. Schemat ten powtarza się dla każdego pobieranego wiersza i w efekcie nie zostanie wyświetlony żaden wiersz.

Przykład 75

```
-- Sprawdzamy czy i jakie indeksy są założone.
-- Jest tylko klucz główny.

SELECT index_name FROM user_indexes WHERE table_name = 'CUSTOMER';

INDEX_NAME
-----
CUSTOMER_ID_PK
```

```
-- Nie ma klauzuki WHERE. Rekordy wyświetlane są wówczas w kolejności,
-- w jakiej były rejestrowane.
```

```
SELECT name from customer;

NAME
-----
Unisports
OJ Atheletics
Delhi Sports
Womansport
```

```
Kam's Sporting Goods
Sportique
Sweet Rock Sports
Muench Sports
Beisbol Si!
Futbol Sonora
Kuhn's Sports
Hamada Sport
Big John's Sports Emporium
Ojibway Retail
Sporta Russia
15 wierszy zostało wybranych.
```

```
-- Rekordy są wyświetlane w takiej kolejności, w jakiej były rejestrowane.
-- W odróżnieniu od poprzedniego przykładu wyświetlamy tylko 5 pierwszych
-- rekordów.
```

```
SELECT name FROM customer WHERE ROWNUM < 5;
```

```
NAME
```

```
-----
Unisports
OJ Atheletics
Delhi Sports
Womansport
```

```
-- Kolejność wyświetlania rekordów jest „przypadkowa”
```

```
SELECT name from customer WHERE ROWNUM < 5 ORDER BY name;
```

```
NAME
```

```
-----
Delhi Sports
OJ Atheletics
Unisports
Womansport
```

```
-- Tworzymy indeks
```

```
CREATE INDEX customer_name ON customer (name ASC);
```

```
-- Potwierdzamy, że indeks się utworzył
```

```
SELECT index_name FROM user_indexes WHERE table_name = 'CUSTOMER';
```

```
INDEX_NAME
```

```
-----
CUSTOMER_NAME
CUSTOMER_ID_PK
```

```
-- Otrzymujemy INNY porządek wyświetlania !!!
```

```
SELECT name from customer WHERE ROWNUM < 5 ORDER BY name;
```

```
NAME
```

```
-----
Beisbol Si!
Big John's Sports Emporium
Delhi Sports
Futbol Sonora
```

Komentarz:

Używanie ROWNUM może nieść ze sobą pewne niebezpieczeństwo. Mianowicie możliwe jest uzyskanie różnych wyników w zależności od tego, czy na kolumnie jest założony indeks, czy też nie. W obu przypadkach Oracle pobiera rekordy w innej kolejności.

Przykład 76

```
SELECT * FROM (
  SELECT id, name FROM product ORDER BY id
)
WHERE ROWNUM <= 5
```

ID NAME

```
-----
10011 Bunny Boot
10012 Ace Ski Boot
10013 Pro Ski Boot
10021 Bunny Ski Pole
10022 Ace Ski Pole
```

-- To nie zadziała

```
SELECT * FROM (
  SELECT id, name FROM product ORDER BY id
)
WHERE ROWNUM > 10
```

nie wybrano żadnych wierszy

-- Wyświetlamy wiersze od 6 do 10

```
SELECT * FROM (
  SELECT wyniki.*, ROWNUM r FROM (
    SELECT id, name FROM summit2.product ORDER BY id
  ) wyniki
  WHERE ROWNUM <=10
)
WHERE r > 5
```

| ID NAME | R |
|-----------------------------|----|
| 10023 Pro Ski Pole | 6 |
| 20106 Junior Soccer Ball | 7 |
| 20108 World Cup Soccer Ball | 8 |
| 20201 World Cup Net | 9 |
| 20510 Black Hawk Knee Pads | 10 |

```
SELECT * FROM (
  SELECT wyniki.*, ROWNUM r FROM
  (
    SELECT id, name FROM product
    ORDER BY id
  ) wyniki
  WHERE ROWNUM <=10
)
WHERE r > 5
```

Wybiera wszystkie
rekordy z
PRODUCT

Tylko
pierwszych 10

Z tych 10. Wyświetl tylko 5 ostatnich

(na podstawie <http://asktom.oracle.com> „getting rows m through n”)

The trick is to order by the inner most subquery. AFTER the order by, assign rownum. Filter out rows that are past the end of your set (10 in your example). THEN after assigning the rownum in that subquery, filter out rows less then your minimum row.

Komentarz:

Przykład pokazuje jak można zrealizować (używając ROWNUM) pobieranie wierszy w określonych porcjach. Przydaje się to najczęściej w aplikacjach internetowych, gdzie chcemy zrealizować np. wyświetlanie rekordów porcjami po 10 rekordów, przykładowo:

←Poprzednie 5 6 7 8 9 10 Następne→

5. Polecenie INSERT

Przykład 77

| S_EMP | | | |
|----------------|------|---------------|----------|
| ID | <pk> | NUMBER(7) | not null |
| LAST_NAME | | VARCHAR2(25) | not null |
| FIRST_NAME | | VARCHAR2(25) | null |
| USERID | <ak> | VARCHAR2(8) | null |
| START_DATE | | DATE | null |
| COMMENTS | | VARCHAR2(255) | null |
| MANAGER_ID | <fk> | NUMBER(7) | null |
| TITLE | <fk> | VARCHAR2(25) | null |
| DEPT_ID | <fk> | NUMBER(7) | null |
| SALARY | | NUMBER(11,2) | null |
| COMMISSION_PCT | | NUMBER(4,2) | null |
| S_EMP_ID_PK | | | |

```
SQL> INSERT INTO emp VALUES (100, 'Gramacki', 'Artur', null, null, null, null, null, null, null, null);
```

1 wiersz został utworzony.

```
SQL> INSERT INTO emp VALUES (100, 'Gramacki', 'Artur', null, null, null, null, null, null, null, null);
INSERT INTO emp VALUES (100, 'Gramacki', 'Artur', null, null, null, null, null, null, null, null)
*
```

BŁĄD w linii 1:

```
ORA-00001: naruszono więzy unikatowe (SUMMIT2.EMP_ID_PK)
        (unique constraint (SUMMIT2.EMP_ID_PK) violated)
```

```
SQL> INSERT INTO emp VALUES (101, 'Gramacki', 'Jarosław', null, null, null, null, null, null, null, null);
```

1 wiersz został utworzony.

SQL>

```
SQL> INSERT INTO emp (id, last_name, start_date) VALUES (102, 'Kowalski', TO_DATE('01-05-2004', 'DD-MM-YYYY'));
```

1 wiersz został utworzony.

```
SQL> SELECT first_name, last_name FROM emp WHERE last_name LIKE 'Gra%';
```

| FIRST_NAME | LAST_NAME |
|------------|-----------|
| Artur | Gramacki |
| Jarosław | Gramacki |

SQL>

Komentarz:

Pokazano dwie wersje polecenia `INSERT`. Pierwsza wymaga podawania wartości dla wszystkich pól, nawet gdy są tam wartości `NULL`. Ponadto wartości te musimy podawać w ściśle określonej kolejności (tak, jak występują w tabeli).

Druga wersja jest wygodniejsza, gdyż wpisujemy wartości tylko do interesujących nas pól.

W obu przypadkach system odmówi wstawienia rekordu, gdy zostanie naruszone chociaż jedno z ograniczeń integralnościowych założonych na tabeli. W przykładzie powyżej próbujemy zarejestrować drugiego pracownika z tą samą wartością klucza głównego.

Przykład 78

| S_REGION | | | |
|----------------|------|--------------|----------|
| ID | <pk> | NUMBER(7) | not null |
| NAME | <ak> | VARCHAR2(50) | not null |
| S_REGION_ID_PK | | | |

```
INSERT INTO region (id, name)
  SELECT id*100, name||' --> kopia' FROM region;
```

```
SELECT * from region;
```

| ID | NAME |
|-----|--------------------------------|
| 1 | North America |
| 2 | South America |
| 3 | Africa / Middle East |
| 4 | Asia |
| 5 | Europe |
| 100 | North America --> kopia |
| 200 | South America --> kopia |
| 300 | Africa / Middle East --> kopia |
| 400 | Asia --> kopia |
| 500 | Europe --> kopia |

10 wierszy zostało wybranych.

Komentarz:

Ciekawą możliwością jest wstawianie rekordów w oparciu o wynik zwracany przez inne zapytanie a dokładniej mówiąc jest to w tym kontekście podzapytanie. Podzapytanie nie musi oczywiście pobierać danych z tej samej tabeli, do której dane wstawiamy. Porównajmy:

```
INSERT INTO region (id, name)
  SELECT id*1000, last_name||' --> z tabeli EMP'
  FROM emp
 WHERE salary > 1500;
```

```
SELECT * from region;
```

| ID | NAME |
|----|---------------|
| 1 | North America |
| 2 | South America |

```

3 Africa / Middle East
4 Asia
5 Europe
1000 Velasquez --> z tabeli EMP
5000 Ropeburn --> z tabeli EMP
13000 Sedeghi --> z tabeli EMP
14000 Nguyen --> z tabeli EMP

```

9 wierszy zostało wybranych.

Przykład 79

```

CREATE TABLE region_kopia (
  id    NUMBER(7),
  name  VARCHAR2(50)
);

```

```

INSERT INTO region_kopia (id, name)
  SELECT id, name FROM region;

```

5 wierszy zostało utworzonych.

```

SELECT * FROM region_kopia;

```

```

      ID NAME
-----
      1 North America
      2 South America
      3 Africa / Middle East
      4 Asia
      5 Europe

```

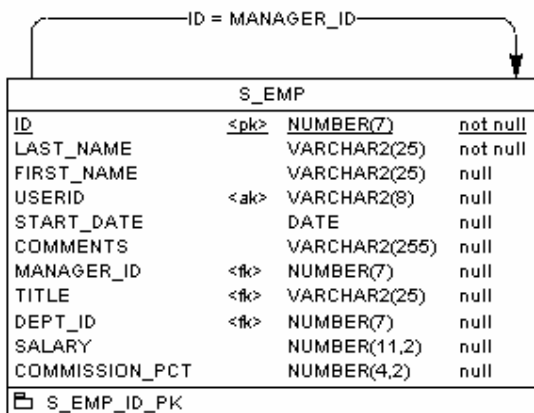
5 wierszy zostało wybranych.

Komentarz:

Tworzymy kopię tabeli REGION i przepisujemy do niej dane z oryginalnej tabeli.

6. Polecenie UPDATE

Przykład 80



```

UPDATE
  emp

```

```

SET
  userid = 'AG',
  salary = 2000,
  start_date = TO_DATE('01-01-2004', 'DD-MM-YYYY')
WHERE id = 100;

```

Komentarz:

Uaktualniamy jeden rekord. Zwróćmy uwagę, na użycie funkcji TO_DATE.

Przykład 81

```

UPDATE
  emp
SET
  comments = first_name||' - '||last_name||' - '||
  NVL(TO_CHAR(start_date), 'brak danych');

```

27 wierszy zostało zmodyfikowanych.

```

SELECT
  first_name, last_name, NVL(TO_CHAR(start_date), '?'), RPAD(comments,35)
FROM
  emp
WHERE
  id IN (100, 101);

```

| FIRST_NAME | LAST_NAME | START_DA | RPAD(COMMENTS,35) |
|------------|-----------|----------|-----------------------------------|
| Jarosław | Gramacki | ? | Jarosław - Gramacki - brak danych |
| Artur | Gramacki | 04/01/01 | Artur - Gramacki - 04/01/01 |

Komentarz:

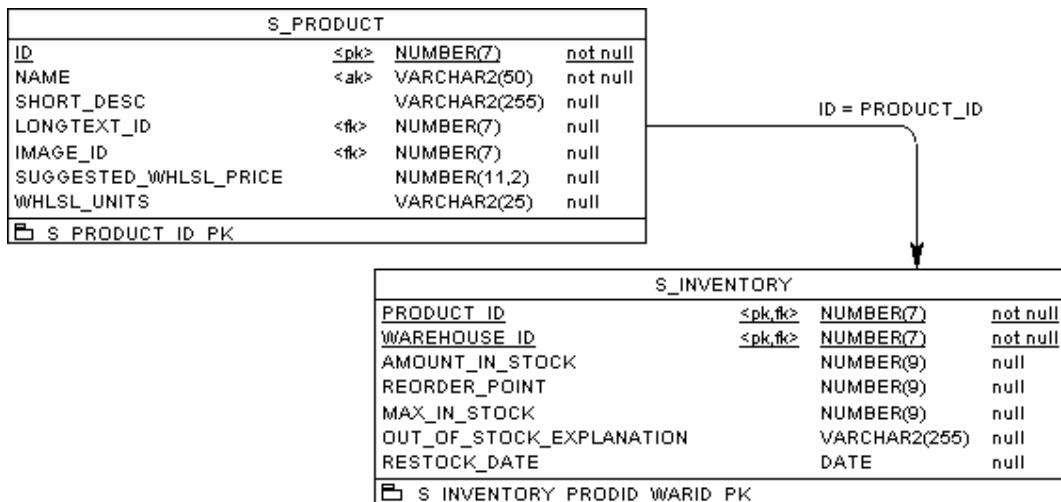
Uaktualniono wszystkie rekordy w tabeli EMP. W polu COMMENTS wpisano, w odpowiedni sposób sformatowane, dane z innych pól. Zwróćmy uwagę, że pole START_DATE zostało „owinięte” w funkcję TO_CHAR, aby funkcja NVL zadziałała prawidłowo. Porównajmy:

```

SQL> SELECT NVL(start_date, '?') FROM emp;
SELECT NVL(start_date, '?') FROM emp
          *
BŁĄD w linii 1:
ORA-01841: (pełny) rok musi być pomiędzy -4713 i +9999 i nie może być 0
          ((full) year must be between -4713 and +9999, and not be 0)
SQL>

```

Przykład 82



```
UPDATE
  product
SET
  suggested_whsls_price = suggested_whsls_price*0.9
WHERE
  id IN (
    SELECT product_id
    FROM inventory
    WHERE max_in_stock - amount_in_stock < 20 AND
    warehouse_id = (SELECT id FROM warehouse WHERE city = 'Sao Paolo')
  );
```

Komentarz:

Obniżamy (jednym zapytaniem SQL-owym – nie ręcznie) cenę tych produktów w bazie SUMMIT2, których sprzedano mniej niż 20 sztuk (różnica wartości w kolumnach MAX_IN_STOCK i AMOUNT_IN_STOCK). Obniżki cen dokonujemy tylko dla produktów z hurtowni w Sao Paolo.

Pisząc polecenie UPDATE warto, zanim je wykonamy, sprawdzić które rekordy zostaną uaktualnione. Najwygodniej jest to zrobić pisząc analogiczne polecenie SELECT. U nas warunek UPDATE-u sprawdzimy w następujący sposób:

```
SELECT product_id, max_in_stock, amount_in_stock, max_in_stock - amount_in_stock
FROM inventory
WHERE max_in_stock - amount_in_stock < 20 AND
warehouse_id = (SELECT id FROM warehouse WHERE city = 'Sao Paolo');
```

| PRODUCT_ID | MAX_IN_STOCK | AMOUNT_IN_STOCK | MAX_IN_STOCK-AMOUNT_IN_STOCK |
|------------|--------------|-----------------|------------------------------|
| 20510 | 175 | 175 | 0 |
| 20512 | 175 | 162 | 13 |
| 30426 | 210 | 200 | 10 |
| 32861 | 140 | 132 | 8 |
| 50417 | 100 | 82 | 18 |
| 50418 | 100 | 98 | 2 |
| 50536 | 100 | 97 | 3 |

7 wierszy zostało wybranych.

Z kolei wiersze, które zostaną uaktualnione sprawdzimy w następujący sposób:

```
SELECT id, name, suggested_whsls_price
```

```

FROM product
WHERE
  id IN (
    SELECT product_id
    FROM inventory
    WHERE max_in_stock - amount_in_stock < 20 AND
    warehouse_id = (SELECT id FROM warehouse WHERE city = 'Sao Paolo')
  );

```

| ID | NAME | SUGGESTED_WHLSL_PRICE |
|-------|-----------------------|-----------------------|
| 20510 | Black Hawk Knee Pads | 9 |
| 20512 | Black Hawk Elbow Pads | 8 |
| 30426 | Himalaya Tires | 8.25 |
| 32861 | Safe-T Helmet | 60 |
| 50417 | Griffey Glove | 80 |
| 50418 | Alomar Glove | 75 |
| 50536 | Winfield Bat | 50 |

7 wierszy zostało wybranych.

Teraz dopiero można „na spokojnie” uruchomić polecenie UPDATE.

Przykład 83

| ID | NAME | REGION_ID |
|----|----------------|-----------|
| 10 | Finance | 1 |
| 31 | Sales | 1 |
| 32 | Sales | 2 |
| 33 | Sales | 3 |
| 34 | Sales | 4 |
| 35 | Sales | 5 |
| 41 | Operations | 1 |
| 42 | Operations | 2 |
| 43 | Operations | 3 |
| 44 | Operations | 4 |
| 45 | Operations | 5 |
| 50 | Administration | 1 |

| ID | NAME |
|----|----------------------|
| 1 | North America |
| 2 | South America |
| 3 | Africa / Middle East |
| 4 | Asia |
| 5 | Europe |

| ID | LAST_NAME | FIRST_NAME | USERID | START_DATE | DEPT_ID | SALARY |
|----|--------------|------------|----------|----------------------------|------------|----------------|
| 4 | Quick-To-See | Mark | mquickto | 1990-04-07 | 10 | 1450 |
| 3 | Nagayama | Midori | mnagayam | 1991-06-17 | 31 | 1400 |
| 11 | Magee | Colin | cmagee | 1990-05-14 | 31 | 1400 |
| 12 | Giljum | Henry | hgiljum | 1992-01-18 | 32 | 1490 |
| 13 | Sedeghi | Yasmin | ysedeghi | 1991-02-18 (2004-03-01) | 33 (41) | 1515 (1666) |
| 14 | Nguyen | Mai | mnguyen | 1992-01-22 (2004-03-01) | 34 (41) | 1525 (1677) |
| 23 | Patel | Radha | rpatel | 1990-10-17 | 34 | 795 |
| 15 | Dumas | Andre | adumas | 1991-10-09 | 35 | 1450 |
| 2 | Ngao | LaDoris | lngao | 1990-03-08 | 41 | 1450 |
| 16 | Maduro | Elena | emaduro | 1992-02-07 | 41 | 1400 |
| .. | ... | ... | ... | ... | ... | ... |
| 19 | Patel | Vikram | vpatel | 1991-08-06 | 42 | 795 |

```

-- Nieładnie, gdyż trzeba podawać numerki id
UPDATE
  emp
SET
  dept_id = 41, start_date = SYSDATE, salary = salary * 1.1
WHERE
  salary > 1500 AND

```

```
dept_id IN (31, 32, 33, 34, 35);
```

```
UPDATE
  emp
SET
  -- jedna wartość !!!
  dept_id = (SELECT id FROM dept WHERE name = 'Operations' AND region_id = 1),
  start_date = sysdate,
  salary = salary * 1.1
WHERE
  -- kilka wartości: (31, 32, 33, 34, 35)
  dept_id IN (SELECT id FROM dept WHERE name = 'Sales') AND
  salary > 1500;
```

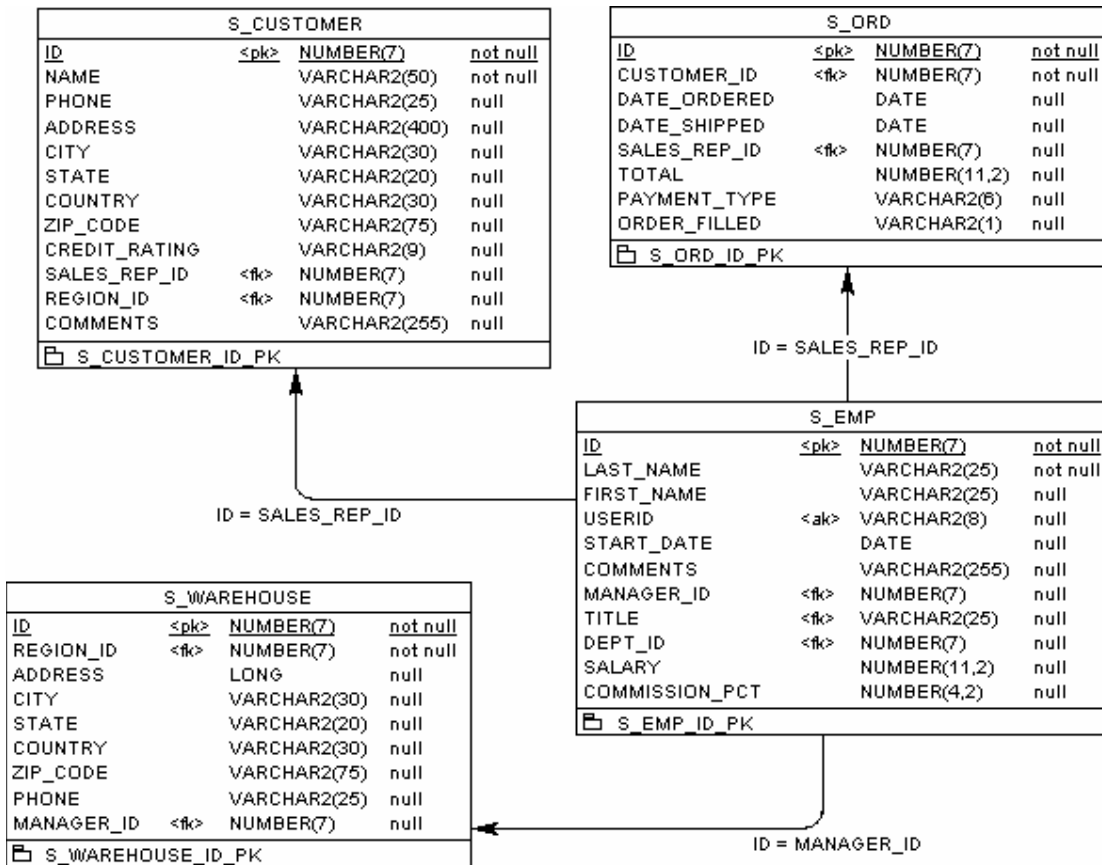
Komentarz:

W nawiasach podano wartości po zmianach.

Przepisano (jednym zapytaniem SQL-owym – nie ręcznie) wszystkich pracowników z wydziału *Sales* zarabiających ponad 1500 do wydziału *Operations* zlokalizowanym w regionie *North America*, zwiększając im jednocześnie płacę o 10% i modyfikując datę zatrudnienia (pole *START_DATE*) na bieżącą datę.

7. Polecenie DELETE

Przykład 84



```
SQL> DELETE FROM emp;
DELETE FROM emp
```



```
*
BŁĄD w linii 1:
ORA-02292: naruszono więzy integralności (SUMMIT2.WAREHOUSE_MANAGER_ID_FK) -
znaleziono rekord podrzędny

(integrity constraint (SUMMIT2.WAREHOUSE_MANAGER_ID_FK) violated - child record
found)

SQL>
```

Komentarz:

Próba wykasowania wszystkich rekordów w tabeli EMP nie powiodła się. System wykrył, że w innych tabelach istnieją powiązane rekordy (czyli te, które mają ograniczenie FOREIGN KEY. Na powyższym rysunku pokazano te tabele). Ponieważ polecenie (w tym przypadku DELETE, ale może to być też UPDATE lub INSERT) traktowane jest jako niepodzielna transakcja (tzn. musi ona być w całości wykonana lub w całości odrzucona) więc niemożność usunięcia choćby jednego rekordu anuluje całe polecenie. Gdy spróbujemy usunąć tylko „bezpieczne” rekordy, polecenie zakończy się sukcesem.

W systemie ORACLE możliwe jest takie zdefiniowanie tabeli, że w trakcie kasowania z niej rekordów zostaną również (w sposób automatyczny) wykasowane wszystkie ewentualnie istniejące rekordy powiązane. Chodzi tutaj o mechanizm tzw. kasowania **kaskadowego**. Realizowane jest to poprzez użycie klauzuli ON DELETE CASCADE w trakcie definiowania klucza obcego w tabeli podrzędnej (u nas chodzi o klucz warehouse_manager_id_fk).

Należy zauważyć, że tego typu rozwiązanie może być bardzo niebezpieczne w praktyce (można łatwo i nieświadomie utracić bardzo dużą liczbę danych). Powinno więc być używane z wielką uwagą. O kasowaniu kaskadowym będzie mowa również w rozdziale omawiającym polecenie DROP.

Przykład 85

```
SQL> SELECT count (*) FROM emp;

  COUNT (*)
-----
         27

SQL> DELETE FROM emp WHERE last_name LIKE 'Gramacki';

2 wierszy zostało usuniętych.

SQL> SELECT count (*) FROM emp;

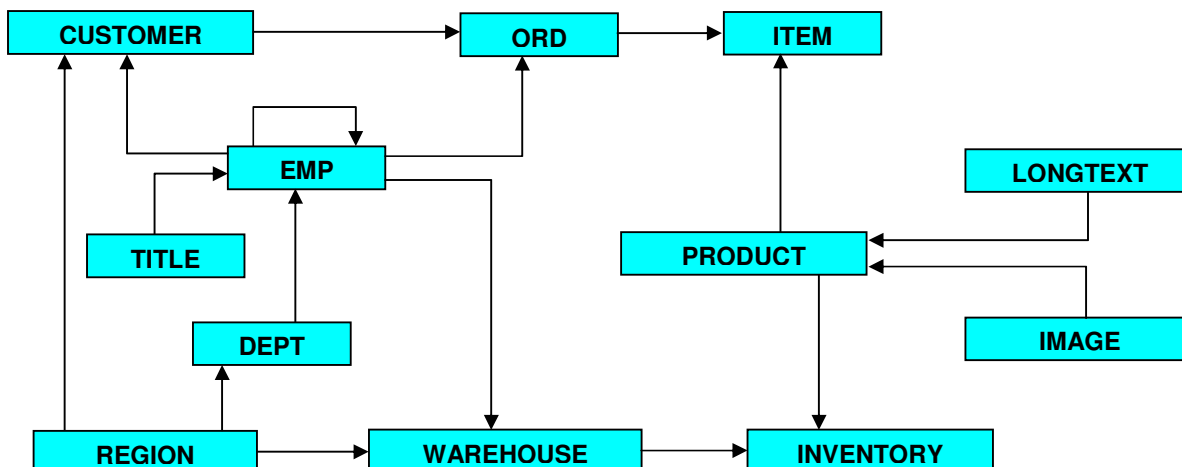
  COUNT (*)
-----
         25

SQL>
```

Komentarz:

Kasowanie rekordów udało się, gdyż nie posiadały one żadnych powiązanych rekordów.

Przykład 86



```
DELETE FROM item;
62 wierszy zostało usuniętych.
DELETE FROM inventory;
114 wierszy zostało usuniętych.
DELETE FROM product;
33 wierszy zostało usuniętych.
DELETE FROM longtext;
33 wierszy zostało usuniętych.
DELETE FROM image;
19 wierszy zostało usuniętych.
DELETE FROM warehouse;
5 wierszy zostało usuniętych.
DELETE FROM ord;
16 wierszy zostało usuniętych.
DELETE FROM customer;
15 wierszy zostało usuniętych.
DELETE FROM emp;
25 wierszy zostało usuniętych.
DELETE FROM title;
8 wierszy zostało usuniętych.
DELETE FROM dept;
12 wierszy zostało usuniętych.
DELETE FROM region;
5 wierszy zostało usuniętych.
```

Komentarz:

Wykasowanie danych z wszystkich tabel modelu SUMMIT2. Zwróćmy uwagę, że tabele musiały być kasowane w ściśle określonej kolejności. Kasowanie rozpoczynamy od tabel najbardziej „zagębnionych” w strukturze relacyjnej a kończymy na tych najbardziej „zewnętrznych”.

8. Wprowadzenie do mechanizmu transakcji

Przykład 87

```
INSERT INTO emp VALUES
(100, 'Gramacki', 'Artur', null, null, null, null, null, null, null);
```

1 wiersz został utworzony.

```
INSERT INTO emp VALUES
```

```

(101, 'Gramacki', 'Jarosław', null, null, null, null, null, null, null);

1 wiersz został utworzony.

-- Sprawdzamy ile jest rekordów w tabeli EMP.
SELECT count(*) FROM emp;

COUNT(*)
-----
      27

-- Kasujemy tylko „bezpieczne” rekordy.
DELETE FROM emp WHERE last_name LIKE 'Gramacki';

2 wierszy zostało usuniętych.

-- Potwierdzamy, że rekordy zostały wykasowane.
SELECT count(*) FROM emp;

COUNT(*)
-----
      25

-- Wycofujemy dokonane przed chwilą zmiany.
ROLLBACK;

Wycofywanie zostało zakończone.

-- W tabeli znów jest 27 rekordów.
SELECT count(*) FROM emp;

COUNT(*)
-----
      27

DELETE FROM emp WHERE last_name LIKE 'Gramacki';

2 wierszy zostało usuniętych.

-- Zatwierdzamy wszystkie wykonane do tej pory zmiany.
COMMIT;

Zatwierdzanie zostało ukończone.

-- Wcześniej wydaliśmy polecenie COMMIT, więc polecenie ROLLBACK nie ma już nic
-- do roboty.
ROLLBACK;

Wycofywanie zostało zakończone.

SELECT count(*) FROM emp;

COUNT(*)
-----
      25

```

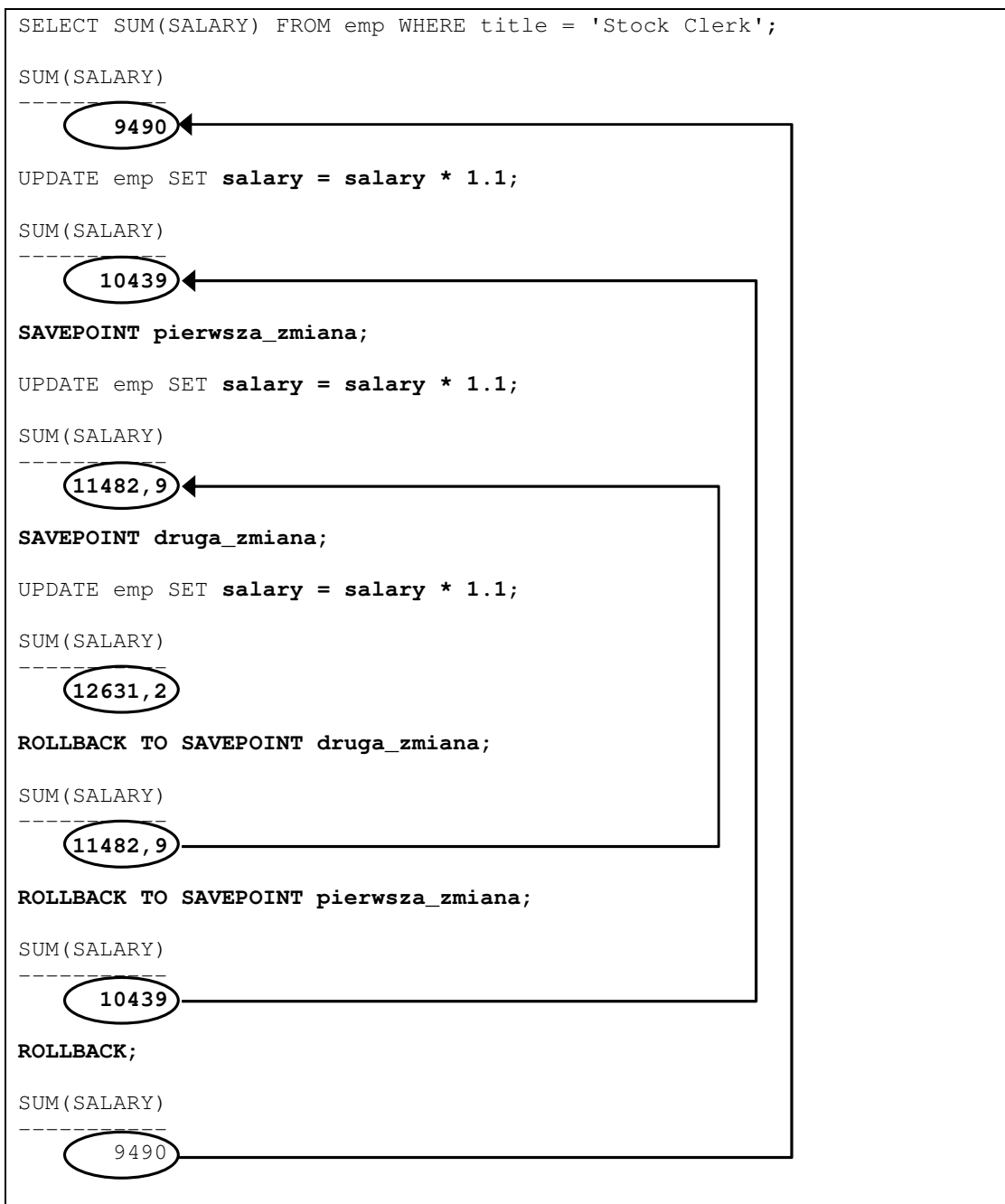
Komentarz:

Kasowanie rekordów udało się, gdyż nie posiadały one żadnych powiązanych rekordów. Dodatkowo zademonstrowano użycie poleceń COMMIT i ROLLBACK.

Każdą operację wykonaną za pomocą poleceń `INSERT`, `UPDATE` lub `DELETE` można wycofać, tzn. spowodować, aby dokonane zmiany zostały anulowane. Anulowanie (czyli wydanie polecenia `ROLLBACK`) jest możliwe tylko wówczas, gdy dokonane zmiany nie zostały jeszcze zatwierdzone poleceniem `COMMIT`. Po wydaniu polecenia `COMMIT` dokonane zmiany są trwale zapisywane w bazie danych. Ewentualne odzyskanie utraconych w ten sposób danych możliwe jest tylko z pomocą przygotowanych wcześniej kopii zapasowych (ang. *backup*).

Omawiany tu mechanizm nosi nazwę transakcji.

Przykład 88



Komentarz:

Zademonstrowano użycie polecenia SAVEPOINT. Dzięki niemu możemy ręcznie ustawiać tzw. punkty kontrolne, które mogą być następnie użyte do bardziej precyzyjnego wycofywania niezatwierdzonych zmian.

Przykład 89

```
SELECT id, first_name, last_name, salary
FROM emp
WHERE last_name LIKE 'Gramacki' ;
```

| ID | FIRST_NAME | LAST_NAME | SALARY |
|-----|------------|-----------|--------|
| 101 | Jarosław | Gramacki | |
| 100 | Artur | Gramacki | |

| Lp | Sesja SQL*Plus 1 | Sesja SQL*Plus 2 |
|----|---|--|
| 1 | UPDATE emp SET salary = 2000 WHERE id = 100; 1 wiersz został zmodyfikowany. | |
| 2 | | UPDATE emp SET salary = 2999 WHERE id = 101; 1 wiersz został zmodyfikowany. |
| 3 | UPDATE emp SET salary = 2000 WHERE id = 101; 1 wiersz został zmodyfikowany. | |
| 4 | Następuje wstrzymanie wykonywania polecenia ! | |
| 5 | | UPDATE emp SET salary = 2999 WHERE id = 100; 1 wiersz został zmodyfikowany. |
| 6 | | Następuje wstrzymanie wykonywania polecenia ! |
| 7 | ORA-00060: podczas oczekiwania na zasób wykryto zakleszczenie (deadlock detected while waiting for resource) | |
| 8 | COMMIT; Zatwierdzanie zostało ukończone. | W tym momencie zablokowana sesja „puszcza”. |
| 9 | | 1 wiersz został zmodyfikowany. |
| 10 | | COMMIT; Zatwierdzanie zostało ukończone. |
| 11 | SELECT id, first_name, last_name, salary FROM emp WHERE last_name LIKE 'Gramacki'; ID FIRST_NAME LAST_NAME SALARY ----- 101 Jarosław Gramacki 2999 100 Artur Gramacki 2999 | |

Komentarz:

Pokazano przykład tzw. zakleszczenia (ang. *deadlock*). Kiedy dwie sesje próbują modyfikować ten sam rekord (rekordy) dochodzi do sytuacji, w której obie sesje blokują sobie nawzajem zasoby. Efektem jest błąd ORA-00060.

9. Polecenie CREATE

9.1. Tworzenie tabel

Przykład 90

| Pracownicy | | |
|----------------|--------------|------|
| Pracownik_ID | NUMBER(10) | null |
| Imie | VARCHAR2(20) | null |
| Nazwisko | VARCHAR2(30) | null |
| Plec | CHAR | null |
| Data_urodzenia | DATE | null |
| Zarobki | NUMBER(8,2) | null |

```
DROP TABLE pracownicy CASCADE CONSTRAINTS;

CREATE TABLE pracownicy
(
  pracownik_id    NUMBER(10),
  imie            VARCHAR2(20),
  nazwisko        VARCHAR2(30),
  plec            CHAR,
  data_urodzenia  DATE,
  zarobki         NUMBER(8,2)
);
```

Komentarz:

Utworzona tabela posiada klucz główny (ang. *primary key*) a kolumny imie i nazwisko mają zdefiniowane ograniczenie NOT NULL. Po utworzeniu tabeli szczegóły jej budowy zapisane są w tzw. **słowniku bazy danych** (ang. *database dictionary*). Użytkownicy mają dostęp do słownika w trybie *read-only*. Mogą go jednak (w ramach posiadanych uprawnień) przeglądać. Wyczerpujący opis poszczególnych tabel słownika można znaleźć w dokumentacji: **Oracle® Reference**.

W ORACLE-u nie istnieje operacja nadpisania istniejącej już tabeli (uwaga ta w ogólności tyczy się wszystkich pozostałych obiektów, jak np. widoki, sekwencje – będzie o nich jeszcze mowa w dalszej części opracowania). Aby utworzyć nową tabelę, starą trzeba najpierw usunąć poleceniem DROP TABLE. Próba utworzenia tabeli o nazwie, która już istnieje, kończy się komunikatem o błędzie. Porównajmy:

```
SQL> CREATE TABLE pracownicy (id NUMBER);
CREATE TABLE pracownicy (id NUMBER)
      *
BŁĄD w linii 1:
ORA-00955: istniejący obiekt używa już tej nazwy
          (name is already used by an existing object)

SQL>
```

```
SELECT table_name FROM user_tables;

TABLE_NAME
-----
PRACOWNICY
```

```
CUSTOMER
DEPT
EMP
IMAGE
INVENTORY
ITEM
JOB
LONGTEXT
ORD
PRODUCT
REGION
TITLE
WAREHOUSE
```

14 wierszy zostało wybranych.

```
SELECT
  RPAD(table_name,11) "TABLE_NAME",
  RPAD(column_name,11) "COLUMN_NAME",
  RPAD(data_type,10) "DATA_TYPE",
  data_length,
  data_precision,
  nullable,
  column_id
FROM
  user_tab_columns
WHERE
  table_name = 'PRACOWNICY';
```

| TABLE_NAME | COLUMN_NAME | DATA_TYPE | DATA_LENGTH | DATA_PRECISION | N | COLUMN_ID |
|------------|-------------|-----------|-------------|----------------|---|-----------|
| PRACOWNICY | PRACOWNIK_I | NUMBER | 22 | 10 | Y | 1 |
| PRACOWNICY | IMIE | VARCHAR2 | 20 | | Y | 2 |
| PRACOWNICY | NAZWISKO | VARCHAR2 | 30 | | Y | 3 |
| PRACOWNICY | PLEC | CHAR | 1 | | Y | 4 |
| PRACOWNICY | DATA_URODZE | DATE | 7 | | Y | 5 |
| PRACOWNICY | ZAROBKI | NUMBER | 22 | 8 | Y | 6 |

6 wierszy zostało wybranych.

Przykład 91

```
COMMENT ON TABLE pracownicy IS 'To są pracownicy naszej firmy';
```

Komentarz został utworzony.

```
COMMENT ON COLUMN pracownicy.pracownik_id IS
'Numery pracowników muszą mieścić się w przedziale od 1000 do 10000';
```

Komentarz został utworzony.

```
SELECT table_name, RPAD(comments, 50)
FROM user_tab_comments WHERE comments IS NOT NULL;
```

| TABLE_NAME | RPAD(COMMENTS,50) |
|------------|-------------------------------|
| PRACOWNICY | To sa pracownicy naszej firmy |

```
SELECT table_name, RPAD(comments, 50)
FROM user_col_comments WHERE comments IS NOT NULL;
```

| | |
|------------|---|
| TABLE_NAME | RPAD (COMMENTS, 50) |
| PRACOWNICY | Numery pracowników muszą mieścić się w przedziale |

Komentarz:

Użytkownik ma możliwość wpisania dowolnego (długość do 4000 znaków) komentarza dla tabeli bądź kolumny w tabeli. Komentarze te zapisywane są w odpowiednich tabelach słownika bazy danych.

Przykład 92

```
DROP TABLE emp_temp;

CREATE TABLE emp_temp
  AS SELECT E.first_name, E.last_name, E.salary, D.name
  FROM emp E, dept D
  WHERE salary > 1500 AND E.dept_id = D.id;
```

```
SELECT * FROM emp_temp;
```

| FIRST_NAME | LAST_NAME | SALARY | NAME |
|------------|-----------|--------|----------------|
| Carmen | Velasquez | 2500 | Administration |
| Audry | Ropeburn | 1550 | Administration |
| Yasmin | Sedeghi | 1515 | Sales |
| Mai | Nguyen | 1525 | Sales |

```
DESC emp_temp;
```

| Nazwa | NULL? | Typ |
|-------------------|-----------------|-----------------------|
| FIRST_NAME | | VARCHAR2 (25) |
| LAST_NAME | NOT NULL | VARCHAR2 (25) |
| SALARY | | NUMBER (11, 2) |
| NAME | NOT NULL | VARCHAR2 (25) |

```
DESC emp
```

| Nazwa | NULL? | Typ |
|-------------------|-----------------|-----------------------|
| ID | NOT NULL | NUMBER (7) |
| LAST_NAME | NOT NULL | VARCHAR2 (25) |
| FIRST_NAME | | VARCHAR2 (25) |
| USERID | | VARCHAR2 (8) |
| START_DATE | | DATE |
| COMMENTS | | VARCHAR2 (255) |
| MANAGER_ID | | NUMBER (7) |
| TITLE | | VARCHAR2 (25) |
| DEPT_ID | | NUMBER (7) |
| SALARY | | NUMBER (11, 2) |
| COMMISSION_PCT | | NUMBER (4, 2) |

```
DESC dept;
```

| Nazwa | NULL? | Typ |
|-------------|-----------------|----------------------|
| ID | NOT NULL | NUMBER (7) |
| NAME | NOT NULL | VARCHAR2 (25) |
| REGION_ID | | NUMBER (7) |

Komentarz:

Ciekawą możliwością jest tworzenie tabel w oparciu o zapytanie. W ten sposób możemy bardzo szybko utworzyć tabelę zawierającą dane w interesującym nas układzie. Oczywiście nie należy „bez umiaru” tworzyć w ten sposób tabel. Każda tabela zajmuje określoną ilość miejsca na dysku i niepotrzebnie komplikuje zarządzanie całością.

Lepszym rozwiązaniem jest wykorzystywanie do tego celu widoków (ang. *view*). Jest o nich mowa w dalszych rozdziałach opracowania.

Przykład 93

```
DROP TABLE emp_temp;

CREATE TABLE emp_temp AS
  SELECT
    UPPER(E.first_name) "Imie",
    UPPER(E.last_name) "Nazwisko",
    'Zarobki: '||E.salary "Zarobki",
    D.name
  FROM emp E, dept D
 WHERE
  salary > 1500 AND
  E.dept_id = D.id;
```

```
SELECT * FROM emp_temp;
```

| Imie | Nazwisko | Zarobki | NAME |
|--------|-----------|---------------|----------------|
| CARMEN | VELASQUEZ | Zarobki: 2500 | Administration |
| AUDRY | ROPEBURN | Zarobki: 1550 | Administration |
| YASMIN | SEDEGHI | Zarobki: 1515 | Sales |
| MAI | NGUYEN | Zarobki: 1525 | Sales |

```
DESC emp_temp;
```

| Nazwa | NULL? | Typ |
|-----------------|----------|----------------------|
| Imie | | VARCHAR2 (25) |
| Nazwisko | | VARCHAR2 (25) |
| Zarobki | | VARCHAR2 (49) |
| NAME | NOT NULL | VARCHAR2 (25) |

Komentarz:

Przykład analogiczny do poprzedniego, tylko polecenie `SELECT` jest bardziej złożone. Zwróćmy uwagę, że muszą – a nie tylko mogą – być zdefiniowane aliasy. Utworzenie tych aliasów niesie za sobą odpowiednie konsekwencje, a mianowicie zmieniają się w stosunku do pierwowzorów typy oraz nazwy kolumn w nowozdefiniowanej tabeli. Ponieważ definiując aliasy używaliśmy cudzysłowów, nazwy kolumn zachowały wielkość liter (patrz też kolejny przykład). Porównajmy:

```
SQL> CREATE TABLE emp_temp AS
  2  SELECT
  3    UPPER(E.first_name),
  4    UPPER(E.last_name) "Nazwisko",
  5    'Zarobki: '||E.salary "Zarobki",
  6    D.name
  7  FROM emp E, dept D
  8  WHERE
  9    salary > 1500 AND
 10    E.dept_id = D.id;
    UPPER(E.first_name),
    *
```

BŁĄD w linii 3:

ORA-00998: wyrażenie to musi być nazwane z użyciem aliasa kolumny

(must name this expression with a column alias)

Przykład 94

```
CREATE TABLE test( id NUMBER(5), kolumna VARCHAR2(10));
CREATE TABLE "test" ( "id" NUMBER(5), "kolumna" VARCHAR2(10));
CREATE TABLE "TeSt" ( "iD" NUMBER(5), "KolumnA" VARCHAR2(10));
```

```
SELECT table_name, column_name
FROM user_tab_columns
WHERE UPPER(table_name) = UPPER('test');
```

| TABLE_NAME | COLUMN_NAME |
|------------|-------------|
| TEST | ID |
| TEST | KOLUMNA |
| Test | iD |
| Test | KolumnA |
| test | id |
| test | kolumna |

6 wierszy zostało wybranych.

```
SELECT table_name, column_name
FROM user_tab_columns
WHERE table_name = 'TEST';
```

| TABLE_NAME | COLUMN_NAME |
|------------|-------------|
| TEST | ID |
| TEST | KOLUMNA |

Komentarz:

W czasie tworzenia tabel, gdy nazwę tworzonej tabeli oraz nazwy kolumn ujmujemy w cudzysłowy zachowywana jest wielkość liter. Można więc utworzyć, jak w przykładzie powyżej, wiele tabel o tej samej nazwie (nie uwzględniając wielkości liter). Możliwości tej nie powinniśmy jednak nadużywać, gdyż może nam to po prostu utrudnić pracę z bazą danych (konieczność pamiętania o ujmowaniu nazw tabeli i lokum w cudzysłowy oraz pamiętanie wielkości liter!).

Ujmować w cudzysłowy można praktycznie nazwę każdego obiektu i jego atrybutu. W mocy pozostaje jednak podana wyżej uwaga, co do potencjalnych utrudnień w pracy z takim obiektami.

9.2. Tworzenie ograniczeń integralnościowych (ang. *constraints*)

| Ograniczenie | Opis |
|--------------|--|
| UNIQUE | Wymusza wpisywanie do kolumny (column) z tym ograniczeniem wartości unikalnych , przy czym wartości NULL są ignorowane przy sprawdzaniu unikalności. |
| NOT NULL | Wymusza wpisywanie do kolumny z tym ograniczeniem wartości niepustych . |
| CHECK | Wymusza wpisywanie do kolumny z tym ograniczeniem wartości występującej we wcześniej zdefiniowanym zbiornie wartości . |
| PRIMARY KEY | Wymusza wpisywanie do kolumny (column) z tym ograniczeniem wartości unikalnych i niepustych . |

| | |
|-------------|--|
| FOREIGN KEY | Wymusza wpisanie do kolumny (kolumn) z tym ograniczeniem tylko takich wartości, które występują w kolumnie referencyjnej (czyli tej z ograniczeniem PRIMARY KEY). |
|-------------|--|

Przykład 95

| Pracownicy2 | | | |
|----------------|------|--------------|----------|
| Pracownik_ID | <pk> | NUMBER(10) | not null |
| Imie | <ak> | VARCHAR2(20) | not null |
| Nazwisko | <ak> | VARCHAR2(30) | not null |
| Plec | | CHAR | not null |
| Data_urodzenia | | DATE | null |
| Zarobki | | NUMBER(8,2) | null |
| PESEL | <ak> | VARCHAR2(11) | null |

<ak> = UNIQUE

```
DROP TABLE pracownicy2 CASCADE CONSTRAINTS;

CREATE TABLE pracownicy2
(
  pracownik_id      NUMBER(10)          NOT NULL
    CONSTRAINT pracownicy2_pk PRIMARY KEY,
  imie              VARCHAR2(20)
    CONSTRAINT pracownicy2_imie_nn NOT NULL,
  nazwisko          VARCHAR2(30)          NOT NULL,
  plec              CHAR                 NOT NULL
    CONSTRAINT pracownicy2_plec_ck CHECK (plec IN ('M','K')),
  data_urodzenia    DATE                 NULL    ,
  zarobki           NUMBER(8,2)          NULL    ,
  pesel            VARCHAR2(11)          NULL
    CONSTRAINT pracownicy2_pesel_uq UNIQUE,
    CONSTRAINT pracownicy2_imie_nazw_uq UNIQUE (imie, nazwisko)
);
```

```
SELECT
  constraint_name,
  RPAD(constraint_type, 15) "CONSTRAINT_TYPE",
  search_condition
FROM
  user_constraints
WHERE
  table_name = 'PRACOWNICY2';
```

| CONSTRAINT_NAME | CONSTRAINT_TYPE | SEARCH_CONDITION |
|--------------------------|-----------------|----------------------------|
| SYS_C0017408 | C | "PRACOWNIK_ID" IS NOT NULL |
| PRACOWNICY2_IMIE_NN | C | "IMIE" IS NOT NULL |
| SYS_C0017410 | C | "NAZWISKO" IS NOT NULL |
| SYS_C0021854 | C | "PLEC" IS NOT NULL |
| PRACOWNICY2_PLEC_CK | C | plec IN ('M','K') |
| PRACOWNICY2_PK | P | |
| PRACOWNICY2_PESSEL_UQ | U | |
| PRACOWNICY2_IMIE_NAZW_UQ | U | |

8 wierszy zostało wybranych.

```
SELECT
  constraint_name,
  RPAD(column_name, 15) "COLUMN_NAME",
  position
```

```
FROM
  user_cons_columns
WHERE
  table_name = 'PRACOWNICY2';
```

| CONSTRAINT_NAME | COLUMN_NAME | POSITION |
|---------------------------------|-----------------|----------|
| PRACOWNICY2_IMIE_NAZW_UQ | IMIE | 1 |
| PRACOWNICY2_IMIE_NAZW_UQ | NAZWISKO | 2 |
| PRACOWNICY2_IMIE_NN | IMIE | |
| PRACOWNICY2_PESSEL_UQ | PESEL | 1 |
| PRACOWNICY2_PK | PRACOWNIK_ID | 1 |
| PRACOWNICY2_PLEC_CK | PLEC | |
| SYS_C0021851 | PRACOWNIK_ID | |
| SYS_C0021853 | NAZWISKO | |
| SYS_C0021854 | PLEC | |

9 wierszy zostało wybranych.

Komentarz:

Utworzono tabelę, w której zdefiniowano 8 różnych ograniczeń. Część z nich posiada nazwy zdefiniowane przez użytkownika a część nazwy „przydzielone” przez system ORACLE (te, których nazwy zaczynają się od frazy `sys_c`). Definiowanie własnych nazw ma tą zaletę, że w przypadku wystąpienia błędu naruszenia ograniczenia, łatwo jest zorientować się, które ograniczenie zostało naruszone. Gdy mamy do czynienia z nazwą systemową, odnalezienie odpowiedniej tabeli i kolumny może być dość uciążliwe (trzeba przeglądać tabele słownika bazy danych). Patrz też następny przykład.

Zwróćmy również uwagę, że jedno z ograniczeń typu `UNIQUE` jest wielokolumnowe (`pracownicy2_imie_nazw_uq` zdefiniowane dla kolumn `imie` oraz `nazwisko`). W takich przypadkach ograniczenie musi być zdefiniowane jako tablicowe (a nie kolumnowe – wyjaśnienie patrz kolejne przykłady).

Każda nowo definiowana kolumna jest domyślnie typu `NULL`, więc nie trzeba tego jawnie specyfikować. Gdy jednak tak zrobimy (jak w przykładzie powyżej) to nic złego się nie stanie.

Szczegóły definicji poszczególnych ograniczeń zapisane są w słowniku bazy danych, w tabelach `user_constraints` oraz `user_cons_columns`.

Przykład 96

| Pracownicy2 | | | |
|----------------|------|--------------|----------|
| Pracownik_ID | <pk> | NUMBER(10) | not null |
| Imie | <ak> | VARCHAR2(20) | not null |
| Nazwisko | <ak> | VARCHAR2(30) | not null |
| Plec | | CHAR | not null |
| Data_urodzenia | | DATE | null |
| Zarobki | | NUMBER(8,2) | null |
| PESEL | <ak> | VARCHAR2(11) | null |

```
DROP TABLE pracownicy2 CASCADE CONSTRAINTS;

CREATE TABLE pracownicy2
(
  pracownik_id      NUMBER(10)          NOT NULL PRIMARY KEY,
  imie              VARCHAR2 (20)       NOT NULL,
  nazwisko          VARCHAR2 (30)       NOT NULL,
  plec              CHAR                 NOT NULL CHECK (plec IN ('M','K')),
  data_urodzenia    DATE                 ,
  zarobki           NUMBER (8, 2)       ,
  pesel             VARCHAR2 (11)       UNIQUE,
```

```
UNIQUE (imie, nazwisko)
);
```

```
SELECT
  constraint_name,
  RPAD(constraint_type, 15) "CONSTRAINT_TYPE",
  search_condition
FROM
  user_constraints
WHERE
  table_name = 'PRACOWNICY2';
```

| CONSTRAINT_NAME | CONSTRAINT_TYPE | SEARCH_CONDITION |
|-----------------|-----------------|----------------------------|
| SYS_C0034078 | C | "PRACOWNIK_ID" IS NOT NULL |
| SYS_C0034079 | C | "IMIE" IS NOT NULL |
| SYS_C0034080 | C | "NAZWISKO" IS NOT NULL |
| SYS_C0034081 | C | "PLEC" IS NOT NULL |
| SYS_C0034082 | C | plec IN ('M','K') |
| SYS_C0034083 | P | |
| SYS_C0034084 | U | |
| SYS_C0034085 | U | |

8 wierszy zostało wybranych.

Komentarz:

Przykład analogiczny do poprzedniego z tą różnicą, że tym razem wszystkie ograniczenia otrzymują nazwy „systemowe” (czyli tworzone są przez system Oracle wg. schematu SYS_Cxxxxxxx).

Przykład 97

```
INSERT INTO pracownicy2 VALUES
(1, 'Artur', 'Gramacki', 'M', NULL, NULL, 1111111111);
```

1 wiersz został utworzony.

```
INSERT INTO pracownicy2 VALUES
(1, 'Jarosław', 'Gramacki', 'M', NULL, NULL, 1111111111);
```

```
ORA-00001: naruszono więzy unikatowe (SUMMIT2.PRACOWNICY2_PK) ←
(unique constraint (SUMMIT2.PRACOWNICY2_PK) violated)
```

Uwaga: Gdyby ograniczenie miało nazwę „systemową” to dużo trudniej byłoby nam zorientować się, gdzie jest błąd. Porównajmy:

```
ORA-00001: naruszono więzy unikatowe (SUMMIT2.SYS_C0034083) ←
```

```
INSERT INTO pracownicy2 VALUES
(2, 'Jarosław', 'Gramacki', 'm', NULL, NULL, 1111111111);
```

```
ORA-02290: naruszono więzy kontrolne (SUMMIT2.PRACOWNICY2_PLEC_CK)
(check constraint (SUMMIT2.PRACOWNICY2_PLEC_CK) violated)
```

```
INSERT INTO pracownicy2 VALUES
(2, 'Jarosław', 'Gramacki', 'M', NULL, NULL, 1111111111);
```

```
ORA-00001: naruszono więzy unikatowe (SUMMIT2.PRACOWNICY2_PESSEL_UQ)
(unique constraint (SUMMIT2.PRACOWNICY2_PESSEL_UQ) violated)
```

```
INSERT INTO pracownicy2 VALUES
(2, 'Jarosław', 'Gramacki', 'M', NULL, NULL, 2222222222);
```

1 wiersz został utworzony.

```
INSERT INTO pracownicy2 VALUES
(3, NULL, NULL, NULL, NULL, NULL, NULL);
```

```
ORA-01400: nie można wstawić wartości NULL do ("SUMMIT2"."PRACOWNICY2"."IMIE")
(cannot insert NULL into ("SUMMIT2"."PRACOWNICY2"."IMIE"))
```

Komentarz:

Pokazano w jaki sposób system ORACLE reaguje, gdy naruszane są zdefiniowane w tabeli ograniczenia. Zwróćmy uwagę, że tylko przy naruszeniu ograniczenia NOT NULL w tekście komunikatu pojawia się nazwa tabeli oraz kolumny. W pozostałych przypadkach pojawia się tylko nazwa naruszonego ograniczenia (poprzedzona nazwą użytkownika). Stąd takie istotne jest nadawanie tym ograniczeniom własnych nazw.

Przykład 98

```
CREATE TABLE unique_test
(
    kol1          NUMBER(2)          NULL    ,
    kol2          NUMBER(2)          NULL    ,
    CONSTRAINT unique_test_kol12_uq UNIQUE (kol1, kol2)
);

INSERT INTO unique_test VALUES (NULL, NULL);

1 wiersz został utworzony.

INSERT INTO unique_test VALUES (NULL, NULL);

1 wiersz został utworzony.

INSERT INTO unique_test VALUES (1, NULL);

1 wiersz został utworzony.

INSERT INTO unique_test VALUES (2, NULL);

1 wiersz został utworzony.

INSERT INTO unique_test VALUES (2, NULL)
ORA-00001: naruszono więzy unikatowe (SUMMIT2.UNIQUE_TEST_KOL12_UQ)
(unique constraint (SUMMIT2. UNIQUE_TEST_KOL12_UQ) violated)

INSERT INTO unique_test VALUES (NULL, 2);

1 wiersz został utworzony.

INSERT INTO unique_test VALUES (3, 4);

1 wiersz został utworzony.

INSERT INTO unique_test VALUES (3, 4);
ORA-00001: naruszono więzy unikatowe (SUMMIT2.UNIQUE_TEST_KOL12_UQ)
```

(unique constraint (SUMMIT2. UNIQUE_TEST_KOL12_UQ) violated)

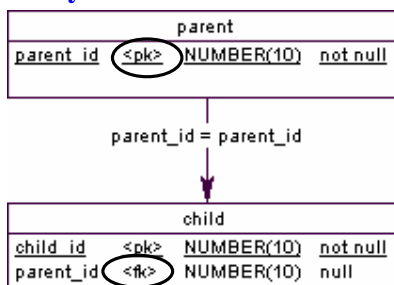
```
SQL> SELECT rownum, kol1, kol2 FROM unique_test;
```

| ROWNUM | KOL1 | KOL2 |
|--------|------|------|
| 1 | | |
| 2 | | |
| 3 | 1 | |
| 4 | 2 | |
| 5 | | 2 |
| 6 | 3 | 4 |

Komentarz:

Ograniczenie UNIQUE ignoruje wartości NULL. Innymi słowy unikalność sprawdzana jest dopiero wtedy, gdy w komórkach wpisano jakieś wartości. W powyższym przykładzie dwa pierwsze rekordy nie naruszają ograniczenia UNIQUE, gdyż oba zawierają wartości NULL.

Przykład 99



```
DROP TABLE child CASCADE CONSTRAINTS;  
DROP TABLE parent CASCADE CONSTRAINTS;
```

```
CREATE TABLE parent  
(  
    parent_id NUMBER(10) NOT NULL,  
    CONSTRAINT parent_pk PRIMARY KEY (parent_id)  
);  
  
CREATE TABLE child  
(  
    child_ID NUMBER(10) NOT NULL,  
    parent_ID NUMBER(10) NULL ,  
    CONSTRAINT child_pk PRIMARY KEY (child_id)  
);  
  
ALTER TABLE child  
ADD CONSTRAINT parent_child_fk  
FOREIGN KEY (parent_id)  
REFERENCES parent (parent_id);
```

Komentarz:

W tej metodzie najpierw tworzymy obie tabele a dopiero potem (za pomocą polecenia ALTER TABLE) dodajemy klucz obcy do tabeli CHILD. Tabele parent oraz child mogą być tworzone w dowolnej kolejności.

Przykład 100

```

CREATE TABLE parent
(
    parent_id      NUMBER(10)          NOT NULL,
    CONSTRAINT parent_pk PRIMARY KEY (parent_id)
);

CREATE TABLE child
(
    child_ID       NUMBER(10)          NOT NULL,
    parent_ID      NUMBER(10)          NULL
    CONSTRAINT parent_child_fk
    REFERENCES parent (parent_id),
    CONSTRAINT child_pk PRIMARY KEY (child_id)
);

```

Komentarz:

W tej metodzie w trakcie tworzenia tabeli `CHILD` tworzymy klucz obcy do tabeli `PARENT`. Zwróćmy uwagę, że aby polecenie to wykonało się bezbłędnie musi już być utworzona tabela `PARENT`. Kolejność tworzenia tabel jest tutaj bardzo ważna. Najpierw musi być utworzona tabela `parent` a dopiero potem `child`.

Ponieważ ograniczenie `FOREIGN KEY` tworzymy w momencie tworzenia **kolumny** `PARENT_ID`, nosi ono nazwę **ograniczenia kolumnowego**.

Przykład 101

```

CREATE TABLE parent
(
    parent_id      NUMBER(10)          NOT NULL,
    CONSTRAINT parent_pk PRIMARY KEY (parent_id)
);

CREATE TABLE child
(
    child_ID       NUMBER(10)          NOT NULL,
    parent_ID      NUMBER(10)          NULL
    CONSTRAINT child_pk PRIMARY KEY (child_id),
    CONSTRAINT parent_child_fk
    FOREIGN KEY (parent_id)
    REFERENCES parent (parent_id),
);

```

Komentarz:

W tej metodzie w trakcie tworzenia tabeli `child` tworzymy klucz obcy do tabeli `parent`. Zwróćmy uwagę, że aby polecenie to wykonało się bezbłędnie musi już być utworzona tabela `parent`. Kolejność tworzenia tabel jest tutaj bardzo ważna. Najpierw musi być utworzona tabela `parent` a dopiero potem `child`.

Ponieważ ograniczenie `FOREIGN KEY` tworzymy **po utworzeniu wszystkich kolumn**, nosi ono nazwę **ograniczenia tablicowego**.

Przykład 102

```

-- metoda 1: ALTER TABLE

ALTER TABLE child
ADD CONSTRAINT parent_child_fk
FOREIGN KEY (parent_id)

```

```

-- metoda 2: ograniczenie kolumnowe

```



```

CREATE TABLE child
(
  child_ID      NUMBER(10)          NOT NULL,
  parent_ID     NUMBER(10)          NULL
  CONSTRAINT parent_child_fk
  REFERENCES parent (parent_id),
  CONSTRAINT child_pk PRIMARY KEY (child_id)
);

```

```

-- metoda 3: ograniczenie tablicowe

CREATE TABLE child
(
  child_ID      NUMBER(10)          NOT NULL,
  parent_ID     NUMBER(10)          NULL
  CONSTRAINT child_pk PRIMARY KEY (child_id),
  CONSTRAINT parent_child_fk
  FOREIGN KEY (parent_id)
  REFERENCES parent (parent_id),
);

```

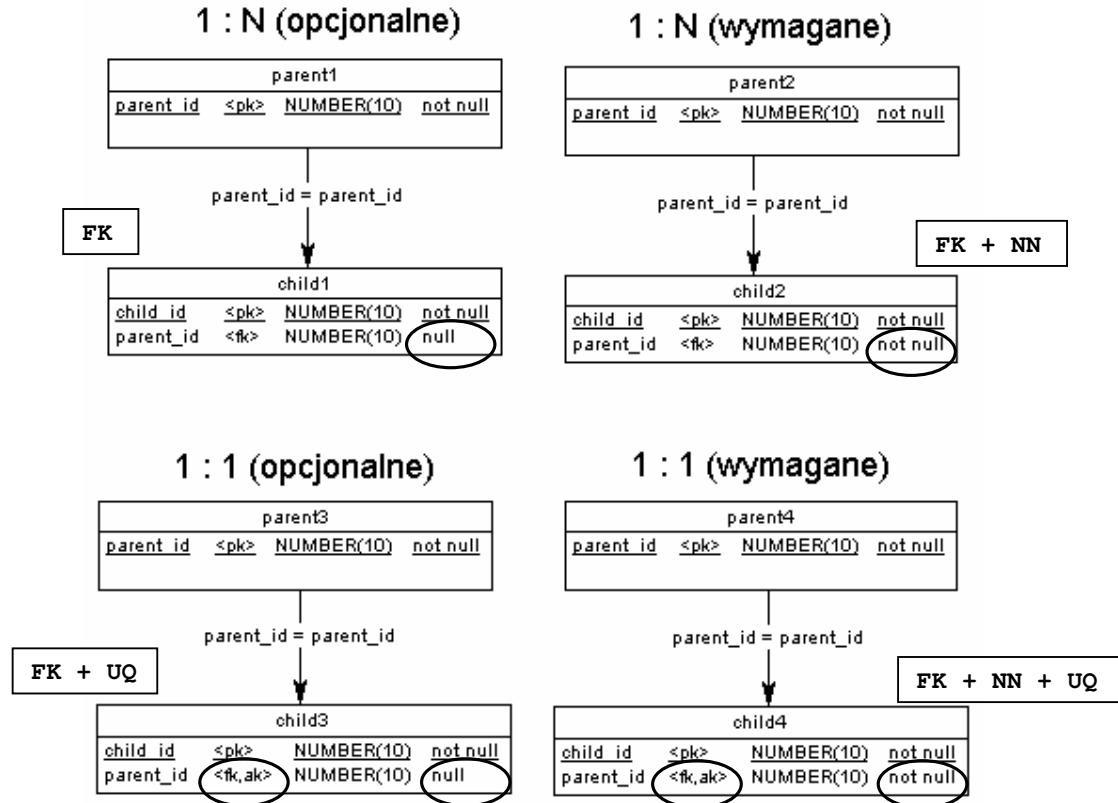
Komentarz:

W tym przykładzie zestawiono wszystkie trzy sposoby tworzenia ograniczeń FOREIGN KEY.

Ograniczenie tablicowe, kolumnowe oraz tworzone za pomocą ALTER TABLE różnią się od siebie tylko składnią. Efekt końcowy (czyli to co zapisywane jest w słowniku bazy danych – patrz tabela user_constraints) jest za każdym razem identyczny.

Będziemy raczej preferować metodę ALTER TABLE, gdyż uwalnia nas ona od potrzeby tworzenia tabel w ściśle określonej kolejności a ponadto zapewnia większą przejrzystość zapisu niż pozostałe metody.

Przykład 103

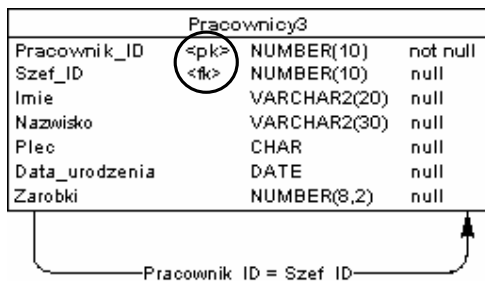


Komentarz:

Na powyższym rysunku pokazano wszystkie 4 możliwe przypadki zdefiniowania ograniczenia FOREIGN KEY. Zwróćmy uwagę, że dla jednej kolumny można zdefiniować więcej niż jedno ograniczenie (np. FOREIGN KEY oraz UNIQUE oraz NOT NULL w ostatnim przypadku).

Symbolem <ak> na rysunku oznaczone jest ograniczenie UNIQUE.

Przykład 104



```
CREATE TABLE Pracownicy3
(
    pracownik_ID      NUMBER(10)          NOT NULL,
    szef_ID           NUMBER(10)          NULL,
    imie              VARCHAR2(20)        NULL,
    nazwisko          VARCHAR2(30)        NULL,
    plec              CHAR                NULL,
    data_urodzenia    DATE                NULL,
    zarobki           NUMBER(8,2)         NULL,
    CONSTRAINT pracownicy3_pk PRIMARY KEY (pracownik_ID)
```

```
);  
  
ALTER TABLE Pracownicy3  
ADD CONSTRAINT pracownicy3_szef_id_fk  
FOREIGN KEY (Szef_ID)  
REFERENCES Pracownicy3 (Pracownik_ID);
```

Komentarz:

Zdefiniowano ograniczenie FOREIGN KEY typu „same do siebie” (ang. *self join*). Definicja takiego ograniczenia niczym w istocie nie różni się od definicji FOREIGN KEY w oddzielnej tabeli.

Przykład 105

```
ALTER TABLE dept  
  ADD CONSTRAINT dept_region_id_fk  
  FOREIGN KEY (region_id) REFERENCES region (id);  
  
ALTER TABLE emp  
  ADD CONSTRAINT emp_manager_id_fk  
  FOREIGN KEY (manager_id) REFERENCES emp (id);  
  
ALTER TABLE emp  
  ADD CONSTRAINT emp_dept_id_fk  
  FOREIGN KEY (dept_id) REFERENCES dept (id);  
  
ALTER TABLE emp  
  ADD CONSTRAINT emp_title_fk  
  FOREIGN KEY (title) REFERENCES title (title);  
  
ALTER TABLE customer  
  ADD CONSTRAINT sales_rep_id_fk  
  FOREIGN KEY (sales_rep_id) REFERENCES emp (id);  
  
ALTER TABLE customer  
  ADD CONSTRAINT customer_region_id_fk  
  FOREIGN KEY (region_id) REFERENCES region (id);  
  
ALTER TABLE ord  
  ADD CONSTRAINT ord_customer_id_fk  
  FOREIGN KEY (customer_id) REFERENCES customer (id);  
  
ALTER TABLE ord  
  ADD CONSTRAINT ord_sales_rep_id_fk  
  FOREIGN KEY (sales_rep_id) REFERENCES emp (id);  
  
ALTER TABLE product  
  ADD CONSTRAINT product_image_id_fk  
  FOREIGN KEY (image_id) REFERENCES image (id);  
  
ALTER TABLE product  
  ADD CONSTRAINT product_longtext_id_fk  
  FOREIGN KEY (longtext_id) REFERENCES longtext (id);  
  
ALTER TABLE item  
  ADD CONSTRAINT item_ord_id_fk  
  FOREIGN KEY (ord_id) REFERENCES ord (id);  
  
ALTER TABLE item  
  ADD CONSTRAINT item_product_id_fk  
  FOREIGN KEY (product_id) REFERENCES product (id);  
  
ALTER TABLE warehouse
```

```

ADD CONSTRAINT warehouse_manager_id_fk
FOREIGN KEY (manager_id) REFERENCES emp (id);

ALTER TABLE warehouse
ADD CONSTRAINT warehouse_region_id_fk
FOREIGN KEY (region_id) REFERENCES region (id);

ALTER TABLE inventory
ADD CONSTRAINT inventory_product_id_fk
FOREIGN KEY (product_id) REFERENCES product (id);

ALTER TABLE inventory
ADD CONSTRAINT inventory_warehouse_id_fk
FOREIGN KEY (warehouse_id) REFERENCES warehouse (id);

```

Komentarz:

Pokazano końcową część skryptu definiującego model SUMMIT2. Widać, że wszystkie ograniczenia FOREIGN KEY definiowane są w jednym miejscu z pomocą polecenia ALTER TABLE. Takie rozwiązanie zdecydowanie zwiększa czytelność skryptu.

9.3. Tworzenie i wykorzystywanie sekwencji

Przykład 106

```
SELECT * FROM user_sequences;
```

| SEQUENCE_NAME | MIN_VALUE | MAX_VALUE | INCREMENT_BY | C | O | CACHE_SIZE | LAST_NUMBER |
|---------------|-----------|-----------|--------------|---|---|------------|-------------|
| CUSTOMER_ID | 1 | 9999999 | | 1 | N | N | 0 |
| DEPT_ID | 1 | 9999999 | | 1 | N | N | 0 |
| EMP_ID | 1 | 9999999 | | 1 | N | N | 0 |
| IMAGE_ID | 1 | 9999999 | | 1 | N | N | 0 |
| LONGTEXT_ID | 1 | 9999999 | | 1 | N | N | 0 |
| ORD_ID | 1 | 9999999 | | 1 | N | N | 0 |
| PRODUCT_ID | 1 | 9999999 | | 1 | N | N | 0 |
| REGION_ID | 1 | 9999999 | | 1 | N | N | 0 |
| WAREHOUSE_ID | 1 | 9999999 | | 1 | N | N | 0 |

9 wierszy zostało wybranych.

```
DESC user_sequences;
```

| Nazwa | NULL? | Typ |
|---------------|----------|--------------|
| SEQUENCE_NAME | NOT NULL | VARCHAR2(30) |
| MIN_VALUE | | NUMBER |
| MAX_VALUE | | NUMBER |
| INCREMENT_BY | NOT NULL | NUMBER |
| CYCLE_FLAG | | VARCHAR2(1) |
| ORDER_FLAG | | VARCHAR2(1) |
| CACHE_SIZE | NOT NULL | NUMBER |
| LAST_NUMBER | NOT NULL | NUMBER |

Komentarz:

Oglądając zawartość tabeli słownikowej user_sequences zapoznajemy się ze zdefiniowanymi sekwencjami. Sekwencje mogą generować wyłącznie liczby całkowite.

Przykład 107

```
DROP SEQUENCE my_seq_1;
DROP SEQUENCE my_seq_2;
DROP SEQUENCE my_seq_3;
```

```
CREATE SEQUENCE my_seq_1;
```

```
CREATE SEQUENCE my_seq_2
  MINVALUE 0
  MAXVALUE 299
  INCREMENT BY 100
  START WITH 0
  NOCACHE
  NOORDER
  CYCLE;
```

```
CREATE SEQUENCE my_seq_3
  MINVALUE 0
  MAXVALUE 3
  INCREMENT BY 1
  START WITH 1
  NOCYCLE;
```

```
SELECT * FROM user_sequences WHERE sequence_name LIKE 'MY_SEQ%';
```

| SEQUENCE_NAME | MIN_VALUE | MAX_VALUE | INCREMENT_BY | C | O | CACHE_SIZE | LAST_NUMBER |
|---------------|-----------|------------|--------------|---|---|------------|-------------|
| MY_SEQ_1 | 1 | 1,0000E+27 | 1 | N | N | 20 | 1 |
| MY_SEQ_2 | 0 | 299 | 100 | Y | N | 0 | 0 |
| MY_SEQ_3 | 0 | 3 | 1 | N | N | 20 | 1 |

Komentarz:

Utworzono trzy sekwencje. Pierwsza ma wszystkie parametry domyślne. W drugiej i trzeciej wyspecyfikowaliśmy te parametry jawnie. W tabeli słownikowej `user_sequences` zapisane są parametry tych sekwencji.

Użycie opcji `CYCLE` będzie powodowało, że po osiągnięciu przez sekwencję wartości `MAXVALUE` jej licznik zostanie wyzerowany i zacznie generować ponownie zadaną sekwencję liczb.

Przykład 108

```
SQL> SELECT my_seq_2.NEXTVAL FROM DUAL;
```

```
  NEXTVAL
-----
         0
```

```
SQL> SELECT my_seq_2.NEXTVAL FROM DUAL;
```

```
  NEXTVAL
-----
        100
```

```
SQL> SELECT my_seq_2.NEXTVAL FROM DUAL;
```

```
  NEXTVAL
-----
        200
```

```
SQL> SELECT my_seq_2.NEXTVAL FROM DUAL;
```

```
  NEXTVAL  
-----  
         0
```

```
SQL> SELECT my_seq_3.NEXTVAL FROM DUAL;
```

```
  NEXTVAL  
-----  
         1
```

```
SQL> SELECT my_seq_3.NEXTVAL FROM DUAL;
```

```
  NEXTVAL  
-----  
         2
```

```
SQL> SELECT my_seq_3.NEXTVAL FROM DUAL;
```

```
  NEXTVAL  
-----  
         3
```

```
SQL> SELECT my_seq_3.NEXTVAL FROM DUAL;
```

```
SELECT my_seq_3.NEXTVAL FROM DUAL
```

```
*
```

```
BŁĄD w linii 1:
```

```
ORA-08004:sekwencja MY_SEQ_3.NEXTVAL exceeds MAXVALUE i nie może być przyjęte  
(sequence MY_SEQ_3.NEXTVAL exceeds MAXVALUE and cannot be instantiated)
```

```
SQL>
```

Komentarz:

Użyto utworzone wcześniej sekwencje. Zwróćmy uwagę, że sekwencja `my_seq_2` „zawinęła” się i zaczęła ponownie generować te same liczby. Natomiast sekwencja `my_seq_3` po osiągnięciu wartości maksymalnej zakończyła działanie.

Do uruchomienia sekwencji użyto tabeli o nazwie `DUAL`. Jest to w zasadzie normalna tabela, tyle, że ma tylko jedną kolumnę oraz jeden rekord. Jest ona tworzona automatycznie w czasie instalowania systemu ORACLE. Można w zasadzie użyć dowolnej innej tabeli. Porównajmy:

```
SQL> DESC dual;
```

```
Nazwa          NULL? Typ  
-----  
DUMMY          VARCHAR2(1)
```

```
SQL> SELECT * FROM DUAL;
```

```
D  
-  
X
```

```
SQL>
```

```
SQL> SELECT my_seq_1.NEXTVAL FROM region;
```

```
  NEXTVAL  
-----  
         1
```

```

2
3
4
5

SQL> SELECT my_seq_1.NEXTVAL FROM region;

NEXTVAL
-----
        6
        7
        8
        9
       10

```

Przykład 109

```

SQL> INSERT INTO parent VALUES (my_seq_1.NEXTVAL);

1 wiersz został utworzony.

SQL> INSERT INTO parent VALUES (my_seq_1.NEXTVAL);

1 wiersz został utworzony.

SQL> INSERT INTO parent VALUES (my_seq_1.CURRVAL);
INSERT INTO parent VALUES (my_seq_1.CURRVAL)
*
BŁĄD w linii 1:
ORA-00001: naruszono więzy unikatowe (SUMMIT2.PARENT_PK)
          (unique constraint (SUMMIT2.PARENT_PK) violated)

SQL> SELECT * FROM parent;

PARENT_ID
-----
        11
        12

SQL>

```

Komentarz:

Użyto zdefiniowanych wcześniej sekwencji do wstawiania wartości w komórkach tabeli. Użycie klauzuli NEXTVAL pobiera z sekwencji kolejną liczbę, CURRVAL pobiera liczbę poprzednio wygenerowaną. W naszym przypadku generowany jest błąd, gdyż wstawiamy wartości dla pól typu PRIMARY KEY, które muszą być oczywiście unikalne. Porównajmy:

```

SQL> SELECT my_seq_1.CURRVAL FROM region;

CURRVAL
-----
       12
       12
       12
       12
       12

SQL>

```

Przykład 110

```
DROP SEQUENCE parent_id_seq;
DROP SEQUENCE child_id_seq;
```

```
CREATE SEQUENCE parent_id_seq;
CREATE SEQUENCE child_id_seq;
```

```
INSERT INTO parent VALUES (parent_id_seq.NEXTVAL);
INSERT INTO child VALUES (child_id_seq.NEXTVAL, parent_id_seq.CURRVAL);
INSERT INTO child VALUES (child_id_seq.NEXTVAL, parent_id_seq.CURRVAL);
INSERT INTO child VALUES (child_id_seq.NEXTVAL, parent_id_seq.CURRVAL);
```

```
INSERT INTO parent VALUES (parent_id_seq.NEXTVAL);
INSERT INTO child VALUES (child_id_seq.NEXTVAL, parent_id_seq.CURRVAL);
INSERT INTO child VALUES (child_id_seq.NEXTVAL, parent_id_seq.CURRVAL);
INSERT INTO child VALUES (child_id_seq.NEXTVAL, parent_id_seq.CURRVAL);
```

```
SELECT * FROM parent;
```

```
  PARENT_ID
-----
         2
         1
```

```
SELECT * FROM child ORDER BY parent_id, child_id;
```

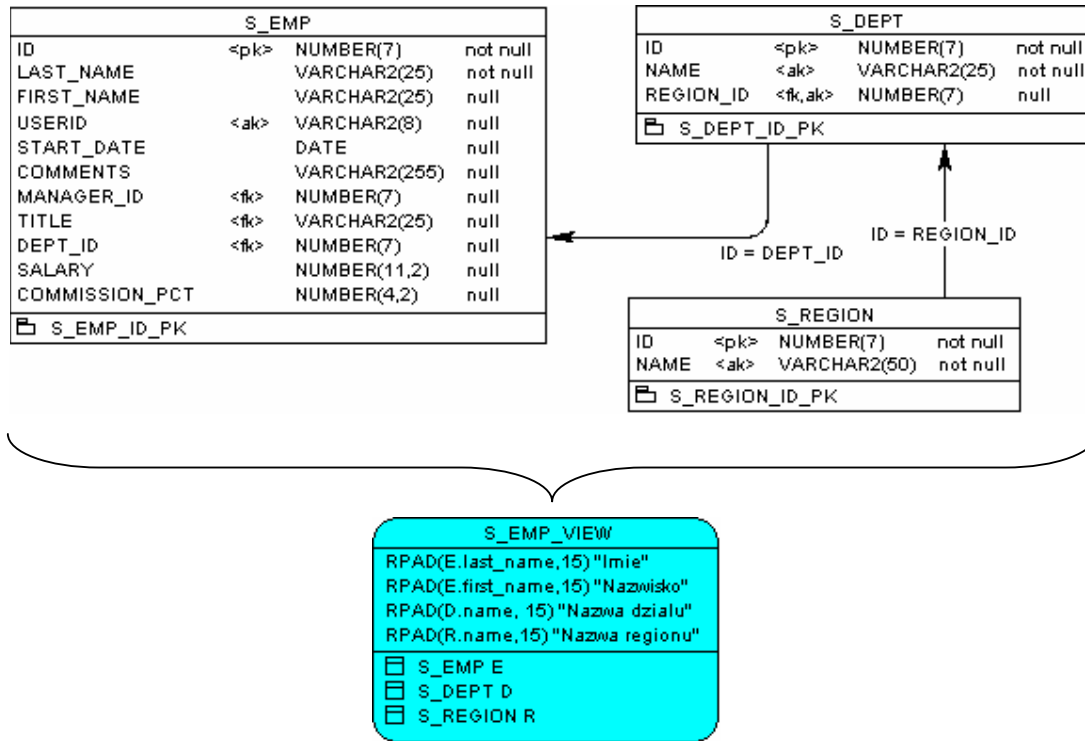
```
  CHILD_ID  PARENT_ID
-----
         1         1
         2         1
         3         1
         4         2
         5         2
         6         2
```

Komentarz:

Odpowiednio manipulując klauzulami NEXTVAL oraz CURRVAL wstawiamy rekordy do tabel PARENT oraz CHILD.

9.4. Tworzenie i wykorzystywanie widoków (ang. view)

Przykład 111



```
CREATE OR REPLACE VIEW emp_view_1 AS SELECT
```

```
  RPAD(E.last_name,15) "Imie",
  RPAD(E.first_name,15) "Nazwisko",
  RPAD(D.name, 15) "Nazwa dzialu",
  RPAD(R.name,15) "Nazwa regionu"
FROM emp E, dept D, region R
WHERE E.dept_id = D.id AND
      D.region_id = R.id AND
      E.salary > 1500;
```

```
SELECT * FROM emp_view_1;
```

| Imie | Nazwisko | Nazwa dzialu | Nazwa regionu |
|-----------|----------|----------------|-----------------|
| Velasquez | Carmen | Administration | North America |
| Ropeburn | Audry | Administration | North America |
| Sedeghi | Yasmin | Sales | Africa / Middle |
| Nguyen | Mai | Sales | Asia |

```
DESC emp_view_1
```

| Nazwa | NULL? | Typ |
|---------------|-------|---------------|
| Imie | | VARCHAR2 (15) |
| Nazwisko | | VARCHAR2 (15) |
| Nazwa dzialu | | VARCHAR2 (15) |
| Nazwa regionu | | VARCHAR2 (15) |

DESC user_views

| Nazwa | NULL? | Typ |
|------------------|----------|----------------|
| VIEW_NAME | NOT NULL | VARCHAR2(30) |
| TEXT_LENGTH | | NUMBER |
| TEXT | | LONG |
| TYPE_TEXT_LENGTH | | NUMBER |
| TYPE_TEXT | | VARCHAR2(4000) |
| OID_TEXT_LENGTH | | NUMBER |
| OID_TEXT | | VARCHAR2(4000) |
| VIEW_TYPE_OWNER | | VARCHAR2(30) |
| VIEW_TYPE | | VARCHAR2(30) |

```
SELECT view_name, text_length, text FROM user_views;
```

| VIEW_NAME | TEXT_LENGTH | TEXT |
|------------|-------------|--|
| EMP_VIEW_1 | 265 | SELECT RPAD(E.last_name,15) "Imie", |

SET LONG 300

```
SELECT RPAD(view_name, 10), RPAD(text_length, 4), text FROM user_views;
```

| RPAD(VIEW_ | RPAD | TEXT |
|------------|------|---|
| EMP_VIEW | 265 | SELECT RPAD(E.last_name,15) "Imie", RPAD(E.first_name,15) "Nazwisko", RPAD(D.name, 15) "Nazwa dzialu", RPAD(R.name,15) "Nazwa regionu" FROM emp E, dept D, region R WHERE E.dept_id = D.id AND D.region_id = R.id AND E.salary > 1500 |

Komentarz:

Utworzyliśmy widok oparty o dane z trzech różnych tabel. Zwróćmy uwagę na nazwy i typy danych poszczególnych kolumn w widoku. Różnią się one od pierwowzorów. Ponadto są pisane małymi literami, co będzie skutkowało tym, że aby powołać się na tą kolumnę, trzeba będzie jej nazwę ujmować w cudzysłowy.

Warto też zauważyć, że użyliśmy polecenia `CREATE OR REPLACE`, co oznacza, że aby zmienić definicję widoku nie musimy go wcześniej kasować. Mamy tutaj do czynienia ze swego rodzaju nadpisywaniem obiektu. Pamiętamy, że w przypadku innych obiektów (jak np. tabele, sekwencje), ich zmiana wymagała *de facto* wykasowania obiektu i utworzenia go od nowa.

Definicje widoków użytkownika przechowywane są w tabeli słownikowej `user_views`. Zwróćmy uwagę na kolumnę `TEXT`, która jest typu `LONG`. Wyświetlanie w programie `SQL*Plus` danych zapisanych w kolumnach tego typu (też `LOB` oraz `NCLOB`) wymaga odpowiedniego ustawienia parametru `LONG` (u nas `SET LONG 300`, gdyż wielkość widoku wynosi 265 a 300 to lekki zapas). Gdy wielkość tego parametru będzie zbyt mała nie wyświetli się po prostu cała definicja widoku.

Przykład 112

```
SQL> UPDATE emp_view_1 SET Nazwisko = 'Maia' WHERE Nazwisko LIKE 'Mai';
UPDATE emp_view_1 SET Nazwisko = 'Maia' WHERE Nazwisko LIKE 'Mai'
```

*

BŁĄD w linii 1:

```
ORA-00904: nieprawidłowa nazwa kolumny
          (invalid column name)
```

```
SQL>
```

```
SQL> UPDATE emp_view_1 SET "Nazwisko" = 'Maia' WHERE "Nazwisko" LIKE 'Mai';
UPDATE emp_view_1 SET "Nazwisko" = 'Maia' WHERE "Nazwisko" LIKE 'Mai'
          *
```

```
BŁĄD w linii 1:
```

```
ORA-01733: w tym miejscu kolumna wirtualna nie jest dozwolona
          (virtual column not allowed here)
```

```
SQL>
```

Komentarz:

Próba wykonania polecenia UPDATE na zdefiniowanym widoku nie powiodła się. W pierwszym przypadku wpisaliśmy w niewłaściwy sposób nazwę kolumny. Powinno być w cudzysłowach, gdyż w tym przypadku małe i duże litery są rozróżnialne (porównajmy wynik polecenia DESC emp_view_1. W drugim przypadku (gdy poprawiliśmy już niewłaściwe wywołanie nazwy kolumny) system odmówił uaktualnienia danych. Okazuje się bowiem, że polecenia INSERT, UPDATE oraz DELETE użyte w odniesieniu do widoków podlegają pewnym ograniczeniom.

Polecenie DELETE możemy używać, gdy:

- widok utworzony jest na bazie tylko jednej tabeli,
- definicja widoku nie zawiera klauzuli GROUP BY, DISTINCT, funkcji agregujących oraz odwołań do pseudokolumn (w ORACLE np. ROWNUM).

Polecenie UPDATE możemy używać, gdy:

- spełnione są warunki jak dla DELETE,

oraz

- definicja widoku nie określa żadnej z aktualizowanych kolumn za pomocą wyrażeń ani konkatenacji.

Polecenie INSERT możemy używać, gdy:

- spełnione są warunki jak dla DELETE oraz UPDATE,

oraz

- wszystkie kolumny tabeli bazowej ze zdefiniowanym ograniczeniem NOT NULL są uwzględnione w perspektywie.

Z powyższego widać, że ograniczeń w używaniu widoków jest bardzo dużo i w praktyce używane są one raczej tylko w trybie do odczytu.

10. Polecenie DROP

Przykład 113

```
SELECT object_type, RPAD(object_name, 30) "OBJECT_NAME", created
FROM user_objects
ORDER BY object_type, object_name;
```

| OBJECT_TYPE | OBJECT_NAME | CREATED |
|-------------|----------------|----------|
| INDEX | CUSTOMER_ID_PK | 04/03/31 |

| | | |
|----------|---------------------------|----------|
| INDEX | DEPT_ID_PK | 04/03/31 |
| INDEX | DEPT_NAME_REGION_ID_UK | 04/03/31 |
| INDEX | EMP_ID_PK | 04/03/31 |
| INDEX | EMP_USERID_UK | 04/03/31 |
| INDEX | IMAGE_ID_PK | 04/03/31 |
| INDEX | INVENTORY_PRODID_WARID_PK | 04/03/31 |
| INDEX | ITEM_ORDID_ITEMID_PK | 04/03/31 |
| INDEX | ITEM_ORDID_PRODID_UK | 04/03/31 |
| INDEX | LONGTEXT_ID_PK | 04/03/31 |
| INDEX | ORD_ID_PK | 04/03/31 |
| INDEX | PRODUCT_ID_PK | 04/03/31 |
| INDEX | PRODUCT_NAME_UK | 04/03/31 |
| INDEX | REGION_ID_PK | 04/03/31 |
| INDEX | REGION_NAME_UK | 04/03/31 |
| INDEX | TITLE_TITLE_PK | 04/03/31 |
| INDEX | WAREHOUSE_ID_PK | 04/03/31 |
| SEQUENCE | CUSTOMER_ID | 04/03/31 |
| SEQUENCE | DEPT_ID | 04/03/31 |
| SEQUENCE | EMP_ID | 04/03/31 |
| SEQUENCE | IMAGE_ID | 04/03/31 |
| SEQUENCE | LONGTEXT_ID | 04/03/31 |
| SEQUENCE | ORD_ID | 04/03/31 |
| SEQUENCE | PRODUCT_ID | 04/03/31 |
| SEQUENCE | REGION_ID | 04/03/31 |
| SEQUENCE | WAREHOUSE_ID | 04/03/31 |
| TABLE | CUSTOMER | 04/03/31 |
| TABLE | DEPT | 04/03/31 |
| TABLE | EMP | 04/03/31 |
| TABLE | IMAGE | 04/03/31 |
| TABLE | INVENTORY | 04/03/31 |
| TABLE | ITEM | 04/03/31 |
| TABLE | LONGTEXT | 04/03/31 |
| TABLE | ORD | 04/03/31 |
| TABLE | PRODUCT | 04/03/31 |
| TABLE | REGION | 04/03/31 |
| TABLE | TITLE | 04/03/31 |
| TABLE | WAREHOUSE | 04/03/31 |

38 wierszy zostało wybranych.

Komentarz:

Wyświetlono zawartość tabeli słownikowej `USER_OBJECTS`, która zawiera zbiorcze zestawienie wszystkich obiektów bieżącego użytkownika. Mamy tutaj wszystkie obiekty, które są tworzone po uruchomieniu skryptu `SUMMIT2.SQL`.

Widzimy trzy rodzaje obiektów: indeksy, sekwencje oraz tabele. W ORACLE-u istnieją jeszcze inne rodzaje obiektów, takie jak np. widoki (ang. *view*), procedury, funkcje, pakiety, migawki (ang. *snapshot*), wyzwalacze (ang. *triggers*).

Każdy z tych obiektów można zmienić (polecenie `ALTER`) lub usunąć (polecenie `DROP`). Obiekty te zostały utworzone za pomocą polecenia `CREATE`.

Przykład 114

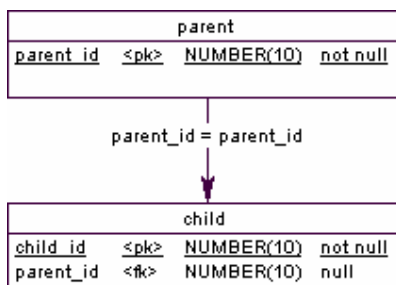
```
SQL> DROP TABLE emp;
DROP TABLE emp
      *
BŁĄD w linii 1:
ORA-02449: tabela ma unikatowe/główne klucze, do których odwołują się obce
klucze
      (unique/primary keys in table referenced by foreign keys)
```

SQL>

Komentarz:

Próbujemy wykasować tabelę, do której odwołują się klucze obce (z pola ID tabeli EMP korzysta kilka innych tabel). System odmówi wykonania takiej operacji. Aby móc wykasować tabelę EMP musimy najpierw usunąć wszystkie powiązane rekordy z tabel ORD, CUSTOMER, WAREHOUSE. Alternatywą jest zdefiniowanie tabeli z opcją ON DELETE CASCADE. Wówczas system bez żadnego pytania się usunie wszystkie ew. istniejące powiązane rekordy. Oczywiście opcję tą należy używać niezwykle ostrożnie!

Przykład 115



```
DROP TABLE child CASCADE CONSTRAINTS;
DROP TABLE parent CASCADE CONSTRAINTS;

CREATE TABLE parent (
    parent_id NUMBER(10) NOT NULL,
    CONSTRAINT parent_pk PRIMARY KEY (parent_id));

CREATE TABLE child (
    child_ID NUMBER(10) NOT NULL,
    parent_ID NUMBER(10) NULL,
    CONSTRAINT child_pk PRIMARY KEY (child_id));

ALTER TABLE child
ADD CONSTRAINT parent_child_fk
FOREIGN KEY (parent_id)
REFERENCES parent (parent_id)
ON DELETE CASCADE;

INSERT INTO parent VALUES (1);
INSERT INTO parent VALUES (2);

INSERT INTO child VALUES (1, 1);
INSERT INTO child VALUES (2, 1);
INSERT INTO child VALUES (3, 1);
INSERT INTO child VALUES (4, 2);
INSERT INTO child VALUES (5, 2);
INSERT INTO child VALUES (6, 2);
```

```
SELECT * FROM child;
```

| CHILD_ID | PARENT_ID |
|----------|-----------|
| 1 | 1 |
| 2 | 1 |
| 3 | 1 |
| 4 | 2 |

```
5      2
6      2
```

6 wierszy zostało wybranych.

```
DELETE FROM parent WHERE parent_id = 1;
```

1 wiersz został usunięty.

```
SELECT * FROM child;
```

| CHILD_ID | PARENT_ID |
|----------|-----------|
| 4 | 2 |
| 5 | 2 |
| 6 | 2 |

Komentarz:

Utworzyliśmy ograniczenie `FOREIGN KEY` z użyciem klauzuli `ON DELETE CASCADE`. Kasując więc jeden rekord z tabeli `PARENT` zostały również wykasowane (niejako w tle, bez jawnego informowania nas o tym!) tryz wiersze w tabeli `CHILD`.

Łatwo jest więc sobie wyobrazić sytuację, że usunięcie jednego rekordu pociąga za sobą utratę wielu tysięcy, często bezcennych, danych z innych tabel. Klauzulę `ON DELETE CASCADE` powinniśmy więc stosować bardzo rozsądnie i z umiarem.

11. Polecenie ALTER

Przykład 116

```
ALTER TABLE region ADD (nazwa VARCHAR2(50));
```

Komentarz:

Modyfikujemy definicję tabeli. Dodajemy dodatkową kolumnę. Do wersji `ORACLE 8.1.7` włącznie nie było możliwe kasowanie kolumn (możliwość tą wprowadzono dopiero w wersji 9). Trzeba było robić to „na piechotę”, wg. następującego schematu:

- przemianowanie oryginalnej tabeli,
- usunięcie wszystkich ograniczeń `FOREIGN KEY` z tabel podrzędnych,
- z przemianowanej tabeli usunięcie wszystkich ograniczeń o zdefiniowanych przez użytkownika nazwach,
- utworzenie nowej tabeli ze zmianami, które chcemy wprowadzić,
- wstawienie do nowoutworzonej tabeli rekordów z przemianowanej wcześniej tabeli,
- odtworzenie w nowej tabeli wszystkich istniejących w starej tabeli ograniczeń.

Porównajmy:

```
ALTER TABLE parent ADD ( nazwisko VARCHAR2(30) );
```

Teraz chcemy wykasować utworzoną kolumnę `nazwisko`. Jak widzimy poniżej nie jest to takie proste i łatwo o pomyłkę (są oczywiście narzędzia wspomagające ten proces i poniższy kod został wygenerowany automatycznie z użyciem jednego z takich narzędzi):

```
-- Lock original table before rename
LOCK TABLE parent IN EXCLUSIVE MODE ;

-- Make backup copy of original table
RENAME parent TO parent_x ;
```

```

-- drop fKey constraint from SUMMIT2.CHILD
ALTER TABLE child DROP CONSTRAINT parent_child_fk ;

-- Remove all other NAMED Table Constraints because
-- they will cause errors when re-creating the table

-- Remove original Primary Key now that FKeys are dropped
ALTER TABLE parent_x DROP CONSTRAINT parent_pk ;

-- Recreate original table
CREATE TABLE parent ( parent_id NUMBER (10) NOT NULL ) ;

-- Copy the data from the renamed table
INSERT INTO parent ( parent_id )
  SELECT parent_id FROM parent_x ;

COMMIT ;

-- Recreate indexes EXCLUDING those created via Unique Constraints
-- Recreate indexes EXCLUDING those created via Unique Constraints

-- Recreate the PKey Constraint
ALTER TABLE parent
ADD CONSTRAINT parent_pk PRIMARY KEY ( parent_id ) ;

-- Recreate the FKey Constraints from the NEW table
-- Grant any privs associated with the old table

-- Recreate the FKey Constraints that reference the NEW table

ALTER TABLE child
ADD CONSTRAINT parent_child_fk FOREIGN KEY (parent_id)
REFERENCES parent (parent_id) ON DELETE CASCADE ;

```

Przykład 117

```

SQL> ALTER TABLE region ADD ( nazwa2 VARCHAR2(50) NOT NULL );
ALTER TABLE region ADD ( nazwa2 VARCHAR2(50) NOT NULL )
      *
BŁĄD w linii 1:
ORA-01758: aby dodać obowiązkową kolumnę (NOT NULL) tabela musi być pusta
          (table must be empty to add mandatory (NOT NULL) column)
SQL>

```

Komentarz:

Nie zawsze jest możliwe dodanie kolumny do istniejącej tabeli. W powyższym przykładzie chcemy dodać kolumnę z ograniczeniem NOT NULL, gdy w tabeli są już jakieś rekordy. Nie jest to możliwe. To co ew. można zrobić, to dodać kolumnę z użyciem opcji DEFAULT, jak poniżej:

```

ALTER TABLE region ADD ( NAZWA2 VARCHAR2(50) DEFAULT '???' NOT NULL );

-- Pozbywamy się wartości domyślnej
ALTER TABLE region MODIFY ( nazwa2 DEFAULT NULL );

```

Poniżej podsumowano jakim restrykcjom podlega modyfikowanie istniejących w tabeli kolumn oraz dodawanie nowych kolumn:

| Ograniczenie, które | Gdy modyfikujemy kolumnę | Gdy tworzymy nową kolumnę |
|---------------------|--------------------------|---------------------------|
|---------------------|--------------------------|---------------------------|

| zamierzamy otworzyć | | |
|---------------------|---|---|
| NOT NULL | Nie może być zdefiniowane, gdy chociaż jednak komórka zawiera wartość NULL | Nie może być zdefiniowane, gdy tabela zawiera choć jeden wiersz |
| UNIQUE | Nie może być zdefiniowane, gdy komórki zawierają duplikaty | Zawsze wykonalne |
| PRIMARY KEY | Nie może być zdefiniowane, gdy komórki zawierają duplikaty lub wartości NULL | Nie może być zdefiniowane, gdy tabela zawiera choć jeden wiersz |
| FOREIGN KEY | Nie może być zdefiniowane, gdy komórki zawierają wartości nie mające swojego odpowiednika w tabeli referencyjnej (macierzystej) | Zawsze wykonalne |
| CHECK | Nie może być zdefiniowane, gdy komórki zawierają wartości nie spełniające warunku CHECK. | Zawsze wykonalne |

Przykład 118

```
ALTER TABLE item DROP CONSTRAINT item_product_id_fk;

ALTER TABLE item
  ADD CONSTRAINT item_product_id_fk
  FOREIGN KEY (product_id) REFERENCES product (id)
  ON DELETE CASCADE;
```

Komentarz:

Zmieniamy definicję ograniczenia typu FOREIGN KEY (dodajemy możliwość kasowania kaskadowego). Od tej pory wykasowanie rekordu (rekordów) z tabeli ORD pociągnie za sobą automatyczne usunięcie wszystkich powiązanych rekordów z tabeli ITEM.

Parametry utworzonego ograniczenia odczytujemy oczywiście z tabeli słownikowej. Zwróćmy uwagę na kolumny r_constraint_name oraz delete_rule. Pierwsza podaje nazwę ograniczenia „macierzystego” a druga, czy włączona jest opcja kaskadowego kasowania.

```
SELECT
  constraint_name,
  RPAD(constraint_type, 15) "CONSTRAINT_TYPE",
  RPAD(r_constraint_name, 15) "R_CONSTRAINT_NAME",
  delete_rule,
  status
FROM
  user_constraints
WHERE
  constraint_name = 'ITEM_PRODUCT_ID_FK';

CONSTRAINT_NAME          CONSTRAINT_TYPE  R_CONSTRAINT_NAME  DELETE_RULE  STATUS
-----
```


12. Słownik bazy danych

Każda baza danych ORACLE posiada tzw. słownik bazy danych (ang. *database dictionary*). Jest to zbiór tabel oraz widoków, dostępnych dla użytkowników w trybie tylko do odczytu, zawierający komplet informacji na temat bieżącego stanu oraz bieżącej konfiguracji bazy (np. nazwy utworzonych użytkowników, ich uprawnienia, nazwy wszystkich istniejących w bazie tabel, szczególności fizycznej budowy bazy danych, itd.).

Słownik tworzony jest w momencie zakładania nowej bazy danych. Jego zawartość jest na bieżąco uaktualniana przez mechanizmy systemowe systemu Oracle.

Tabele tworzące słownik nie są bezpośrednio dostępne dla użytkownika, są natomiast dostępne za pośrednictwem widoków. Listę dostępnych dla bieżącego użytkownika widoków słownika zawiera widok o nazwie `DICTIONARY`. Jego zawartość można poznać wydając po prostu polecenie:

```
SELECT * FROM DICTIONARY;
```

lub

```
SELECT * FROM DICT_COLUMNS;
```

aby zapoznać się ze szczegółowym opisem wszystkich kolumn wchodzących w skład poszczególnych widoków.

Wiele widoków słownika występuje w trzech różnych wersjach, zawierających różne porcje informacji z bazowych tabel słownika. Są to:

- widoki z przedrostkiem `ALL_`
Zawierają informacje o obiektach bieżącego użytkownika oraz obiektach innych użytkowników, do których bieżący użytkownik ma dostęp.
- widoki z przedrostkiem `DBA_`
Zawierają informacje o wszystkich obiektach bazy danych. Dostęp do nich ma tylko administrator oraz ew. inni użytkownicy, którym nadano uprawnienie `SELECT_ANY_TABLE`.
- widoki z przedrostkiem `USER_`
Zawierają informacje o obiektach bieżącego użytkownika. Każdy użytkownik ma z definicji pełen dostęp do swoich obiektów (aby być właścicielem obiektu nie trzeba go samemu utworzyć. Może to zrobić inny użytkownik, który posiada takowe uprawnienia).

Z powyższego wynika, że widoki `ALL_` oraz `USER_` są podzbiorami widoków z grupy `DBA_`.

W systemie ORACLE istnieją jeszcze jedna grupa widoków. Są to tzw. widoki `VS`. W dokumentacji można przeczytać, że widoki z grupy `DBA_` określane są wspólną nazwą *Słownik danych statycznych* (ang. *Static Data Dictionary*), natomiast widoki `VS` są przedstawiane jako *Widoki wydajności dynamicznej* (ang. *Dynamic Performance Views*). Nazwa ta jest adekwatna do spełnianej funkcji, gdyż przedstawiają one w czasie rzeczywistym dane, które prezentują bieżący stan bazy danych.

Innymi słowy można powiedzieć, że widoki z grupy `DBA_` pokazują z jakich obiektów składa się baza danych, natomiast widoki `VS` pokazują funkcjonowanie tej bazy. Większość widoków `VS` dostępnych jest tylko dla administratora bazy.

Wyczerpujące informacje na temat słownika bazy danych można znaleźć w dokumentacji: [Oracle8i Reference Release 2 \(8.1.6\)](#)

13. Użytkownicy bazy danych

13.1. Tworzenie, modyfikowanie i kasowanie użytkowników

Uwaga: składnia podana poniżej jest uproszczona. Pełen opis znajdziesz oczywiście w dokumentacji do systemu Oracle. Warto się z nim zapoznać!

```
-- Tworzenie użytkowników
CREATE USER uzytkownik IDENTIFIED BY haslo;

-- Zmiana hasla
ALTER USER uzytkownik IDENTIFIED BY haslo;

-- Nadawanie uprawnień
GRANT uprawnienie [, uprawnienie ...] TO uzytkownik;
GRANT connect, resource TO scott;

-- Kasowanie użytkowników
DROP USER uzytkownik [CASCADE];

-- Informacje o uzytkownikach
dba_users
```

13.2. Uprawnienia

Uwaga: składnia podana poniżej jest uproszczona. Pełen opis znajdziesz oczywiście w dokumentacji do systemu Oracle. Warto się z nim zapoznać!

```
-- Uprawnienia systemowe

GRANT uprawnienie [, uprawnienie ... ]
TO uzytkownik [, uzytkownik ...]
[WITH ADMIN OPTION];

REVOKE uprawnienie [, uprawnienie ... ]
FROM uzytkownik [, uzytkownik ...];

-- Przykłady
GRANT select any table TO summit2;
REVOKE select ant table FROM summit2;

-- Informacje o uprawnieniach systemowych
dba_sys_privs
```

Komentarz:

Uprawnienia systemowe może nadawać tylko administrator bazy danych. Zezwala on wybranym użytkownikom na wykonywanie określonych operacji na obiektach określonego typu. Przykładem takiego uprawnienia jest `SELECT ANT TABLE`, które umożliwia oglądanie zawartości **każdej tabeli u każdego użytkownika**. Jest to więc bardzo „mocne” uprawnienie. Wiele innych uprawnień z tej grupy też nie należy do słabych. Administrator powinien więc bardzo ostrożnie nimi dysponować.

Nadanie uprawnień z opcją administracyjną (`WITH ADMIN OPTION`) pozwala użytkownikowi otrzymującemu dane uprawnienie przekazywać je dalej innym użytkownikom (bezpieczeństwo!).

```

-- Uprawnienia do obiektów

GRANT uprawnienie [, uprawnienie ... ]
ON obiekt
TO użytkownik [, użytkownik ...]
[WITH ADMIN OPTION];

REVOKE uprawnienie [, uprawnienie ... ]
ON obiekt
FROM użytkownik [, użytkownik ...];

-- Przykłady
GRANT select, update, delete ON emp TO scott;
REVOKE select ON emp FROM scott;

-- Informacje o uprawnieniach do obiektów
dba_tab_privs
dba_col_privs

```

Komentarz:

W zasadzie każdy użytkownik może zezwalać innym użytkownikom na dostęp do swoich obiektów. Lista uprawnień jest oczywiście zależna od typu obiektu. Przykładowo uprawnienie EXECUTE można nadawać tylko do procedur lub funkcji PL/SQL ale nie do tabel.

Istnieją następujące rodzaje praw do obiektów: SELECT, UPDATE, INSERT, ALTER, DELETE, EXECUTE, INDEX, REFERENCES.

Tutaj też istnieje możliwość nadanie uprawnień z opcją administracyjną (WITH ADMIN OPTION).

W przypadku tabel uprawnienia INSERT, UPDATE oraz REFERENCES mogą zostać zawężone do wybranych kolumn, przykładowo:

```
GRANT select, update(first_name, last_name) ON emp TO scott;
```

13.3. Role bazodanowe

Uwaga: składnia podana poniżej jest uproszczona. Pełen opis znajdziesz oczywiście w dokumentacji do systemu Oracle. Warto się z nim zapoznać!

```

-- Tworzenie roli
CREATE ROLE nazwa;

```

```

-- Przypisanie i odbieranie roli uprawnień systemowych
GRANT uprawnienie [, uprawnienie ...]
TO rola [rola ...];

REVOKE uprawnienie [, uprawnienie ...]
FROM rola [rola ...];

```

```

-- Przypisanie i odbieranie roli uprawnień do obiektów
GRANT uprawnienie [, uprawnienie ...]
ON obiekt [obiekt ...]
TO rola [rola ...];

REVOKE uprawnienie [, uprawnienie ...]
ON obiekt [obiekt ...]
FROM rola [rola ...];

```

```
-- Przekazywanie i odbieranie ról użytkownikom
```

```
GRANT rola [, rola ... ]  
TO użytkownik [, użytkownik ...]  
  
REVOKE rola [, rola ... ]  
FROM użytkownik [, użytkownik ...]
```

```
-- Usuwanie roli
```

```
DROP ROLE nazwa;
```

```
-- Informacje o zdefiniowanych rolach
```

```
dba_roles  
dba_role_privs  
role_sys_privs  
role_tab_privs  
role_role_privs
```

13.4. Role predefiniowane

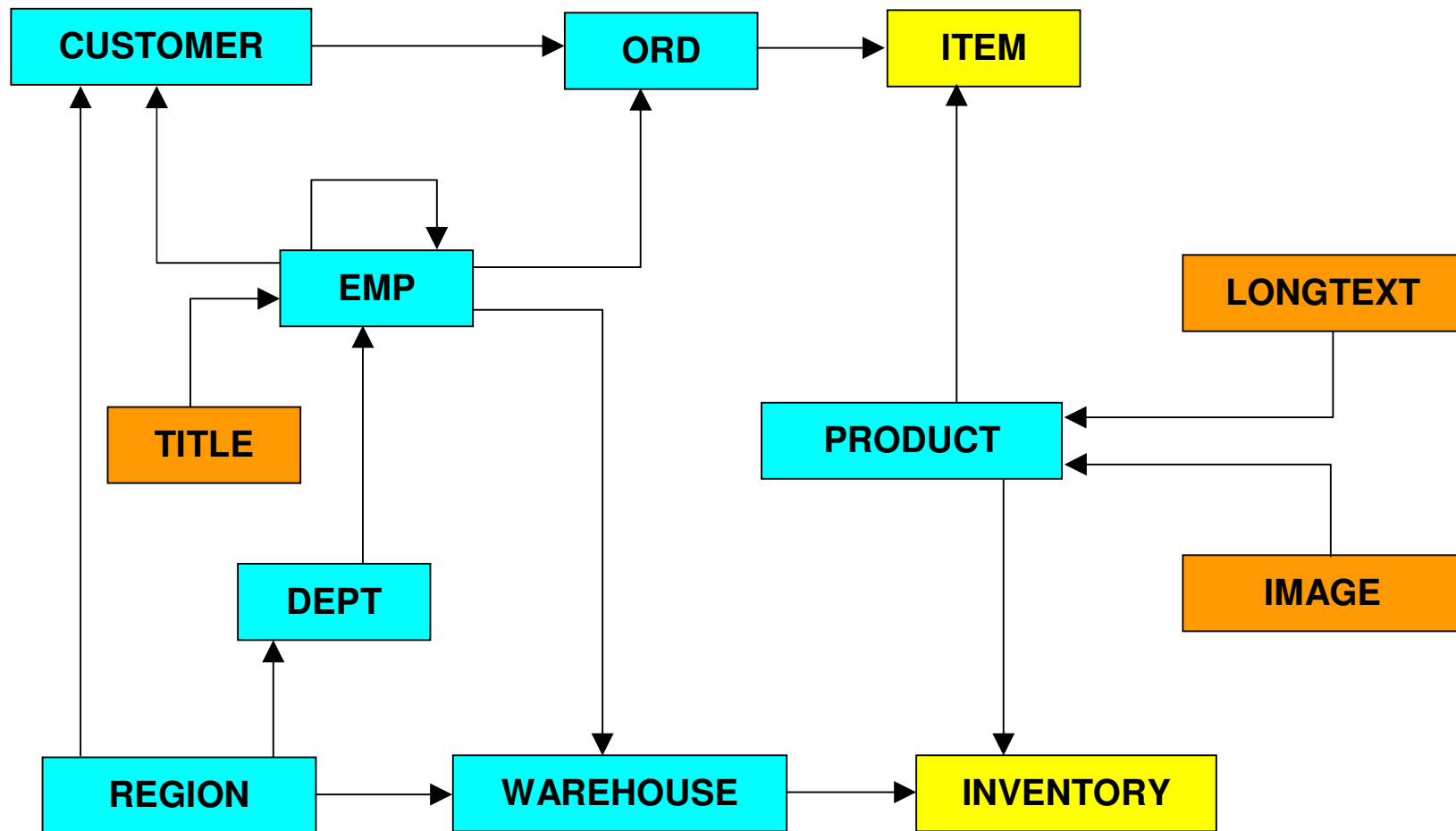
Zaraz po utworzeniu nowej bazy danych poleceniem `CREATE DATABASE` tworzona jest pewna liczba predefiniowanych ról. Ich ilość różni się w zależności od wersji bazy danych. Ich pełna lista znajduje się w widoku `dba_roles`. Najczęściej używane to: `CONNECT`, `RESOURCE`, `DBA`, `IMP_FULL_DATABASE`, `EXP_FULL_DATABASE`, `SELECT_CATALOG_ROLE`. Analizując widok `role_sys_privs` otrzymujemy informację o uprawnieniach przypisanych do poszczególnych ról, przykładowo dla roli `CONNECT` mamy:

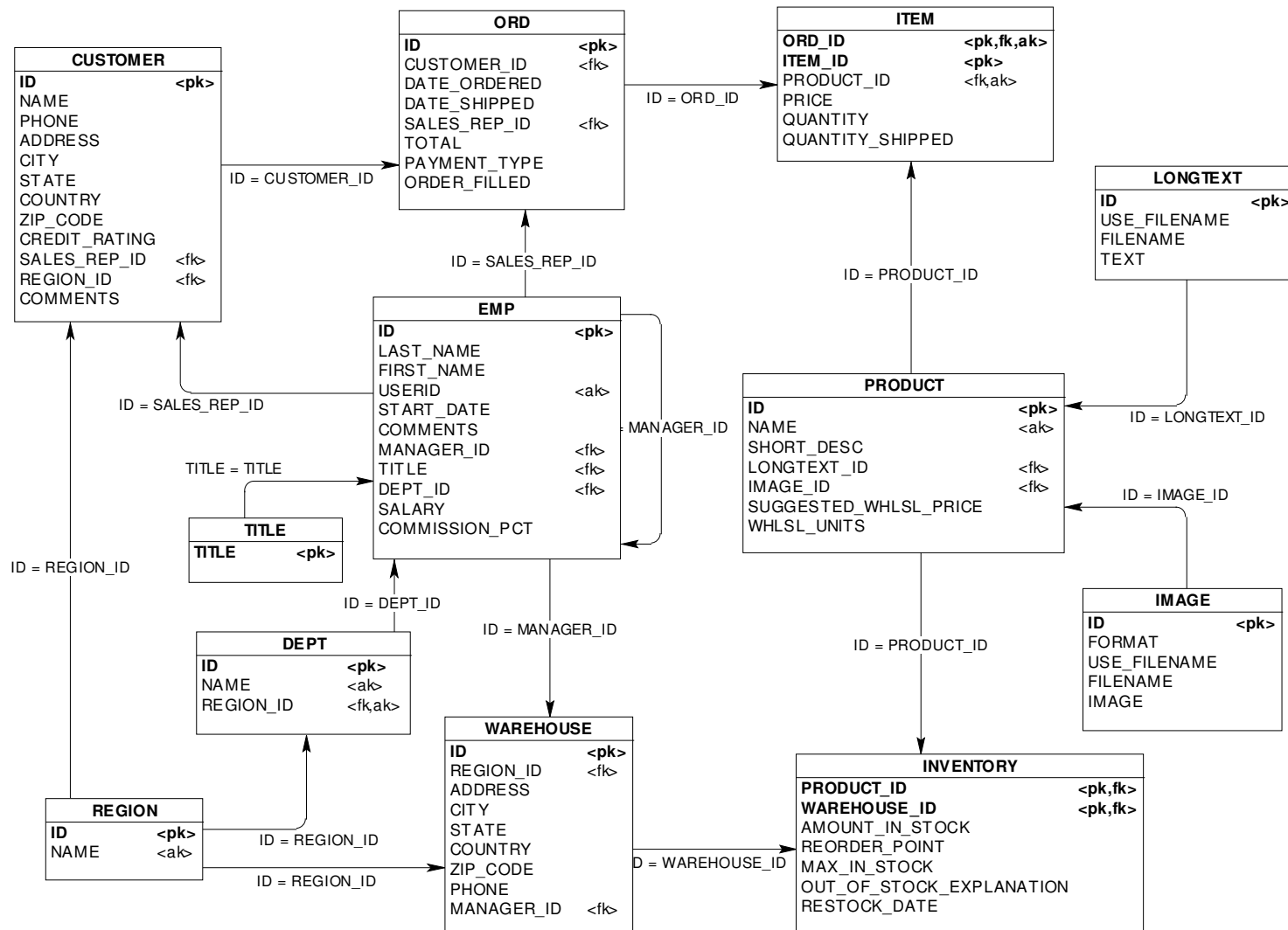
| ROLE | PRIVILEGE | ADM |
|---------|----------------------|-----|
| CONNECT | CREATE VIEW | NO |
| CONNECT | CREATE TABLE | NO |
| CONNECT | ALTER SESSION | NO |
| CONNECT | CREATE CLUSTER | NO |
| CONNECT | CREATE SESSION | NO |
| CONNECT | CREATE SYNONYM | NO |
| CONNECT | CREATE SEQUENCE | NO |
| CONNECT | CREATE DATABASE LINK | NO |

8 rows selected.

I

14. Model SUMMIT2





| ID | NAME | PHONE | ADDRESS | CITY | STATE | COUNTRY | ZIP_CODE | CREDIT_RATING | SALES_REP_ID | REGION_ID | COMMENTS |
|-----|----------------------------|----------------|--------------------|----------------------|------------|--------------------|----------|---------------|--------------|-----------|----------|
| 201 | Unisports | 55-2066101 | 72 Via Bahia | Sao Paolo | | Brazil | | EXCELLENT | 12 | 2 | |
| 202 | OJ Athletics | 81-20101 | 6741 Takashi Blvd. | Osaka | | Japan | | POOR | 14 | 4 | |
| 203 | Delhi Sports | 91-10351 | 11368 Chanakya | New Delhi | | India | | GOOD | 14 | 4 | |
| 204 | Womansport | 1-206-104-0103 | 281 King Street | Seattle | Washington | USA | | EXCELLENT | 11 | 1 | |
| 205 | Kam's Sporting Goods | 852-3692888 | 15 Henessey Road | Hong Kong | | | | EXCELLENT | 15 | 4 | |
| 206 | Sportique | 33-2257201 | 172 Rue de Rivoli | Cannes | | France | | EXCELLENT | 15 | 5 | |
| 207 | Sweet Rock Sports | 234-6036201 | 6 Saint Antoine | Lagos | | Nigeria | | GOOD | | 3 | |
| 208 | Muench Sports | 49-527454 | 435 Gruenstrasse | Stuttgart | | Germany | | GOOD | 15 | 5 | |
| 209 | Beisbol Si! | 809-352689 | 792 Playa Del Mar | San Pedro de Macon's | | Dominican Republic | | EXCELLENT | 11 | 1 | |
| 210 | Futbol Sonora | 52-404562 | 3 Via Saguaro | Nogales | | Mexico | | EXCELLENT | 12 | 2 | |
| 211 | Kuhn's Sports | 42-111292 | 7 Modrany | Prague | | Czechoslovakia | | EXCELLENT | 15 | 5 | |
| 212 | Hamada Sport | 20-1209211 | 57A Corniche | Alexandria | | Egypt | | EXCELLENT | 13 | 3 | |
| 213 | Big John's Sports Emporium | 1-415-555-6281 | 4783 18th Street | San Francisco | CA | USA | | EXCELLENT | 11 | 1 | |
| 214 | Ojibway Retail | 1-716-555-7171 | 415 Main Street | Buffalo | NY | USA | | POOR | 11 | 1 | |
| 215 | Sporta Russia | 7-3892456 | 6000 Yekatamina | Saint Petersburg | | Russia | | POOR | 15 | 5 | |

CUSTOMER

| ORD_ID | ITEM_ID | PRODUCT_ID | PRICE | QUANTITY | QUANTITY_SHIPPED |
|--------|---------|------------|-------|----------|------------------|
| 97 | 1 | 20106 | 9 | 1000 | 1000 |
| 100 | 1 | 10011 | 135 | 500 | 500 |
| 100 | 2 | 10013 | 380 | 400 | 400 |
| 100 | 3 | 10021 | 14 | 500 | 500 |
| 100 | 4 | 10023 | 36 | 400 | 400 |
| 102 | 1 | 20108 | 28 | 100 | 100 |
| 106 | 1 | 20108 | 28 | 46 | 46 |
| 107 | 1 | 20106 | 11 | 50 | 50 |
| 107 | 2 | 20108 | 28 | 22 | 22 |
| 109 | 1 | 10011 | 140 | 150 | 150 |
| 109 | 2 | 10012 | 175 | 600 | 600 |
| 109 | 3 | 10022 | 21,95 | 300 | 300 |
| 112 | 1 | 20106 | 11 | 50 | 50 |

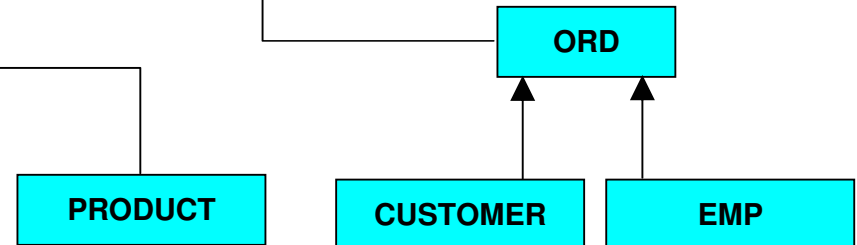
Uwaga: pokazano tylko pozycje zawierające produkty, gdzie **PRODUCT_ID** jest od 10011 do 20108

ITEM

| ID | CUSTOMER_ID | DATE_ORDERED | DATE_SHIPPED | SALES_REP_ID | TOTAL | PAYMENT_TYPE | ORDER_FILLED |
|-----|-------------|--------------|--------------|--------------|---------|--------------|--------------|
| 100 | 204 | 1992-08-31 | 1992-09-10 | 11 | 601100 | CREDIT | Y |
| 101 | 205 | 1992-08-31 | 1992-09-15 | 14 | 8056,6 | CREDIT | Y |
| 102 | 206 | 1992-09-01 | 1992-09-08 | 15 | 8335 | CREDIT | Y |
| 103 | 208 | 1992-09-02 | 1992-09-22 | 15 | 377 | CASH | Y |
| 104 | 208 | 1992-09-03 | 1992-09-23 | 15 | 32430 | CREDIT | Y |
| 105 | 209 | 1992-09-04 | 1992-09-18 | 11 | 2722,24 | CREDIT | Y |
| 106 | 210 | 1992-09-07 | 1992-09-15 | 12 | 15634 | CREDIT | Y |
| 107 | 211 | 1992-09-07 | 1992-09-21 | 15 | 142171 | CREDIT | Y |
| 108 | 212 | 1992-09-07 | 1992-09-10 | 13 | 149570 | CREDIT | Y |
| 109 | 213 | 1992-09-08 | 1992-09-28 | 11 | 1020935 | CREDIT | Y |
| 110 | 214 | 1992-09-09 | 1992-09-21 | 11 | 1539,13 | CASH | Y |
| 111 | 204 | 1992-09-09 | 1992-09-21 | 11 | 2770 | CASH | Y |
| 97 | 201 | 1992-08-28 | 1992-09-17 | 12 | 84000 | CREDIT | Y |
| 98 | 202 | 1992-08-31 | 1992-09-10 | 14 | 595 | CASH | Y |
| 99 | 203 | 1992-08-31 | 1992-09-18 | 14 | 7707 | CREDIT | Y |
| 112 | 210 | 1992-08-31 | 1992-09-10 | 12 | 550 | CREDIT | Y |

| ID | NAME | SHORT_DESC | LONGTEXT_ID | IMAGE_ID | SUGGESTED_WHLSL_PRICE | WHLSL_UNITS |
|-------|-----------------------|-----------------------|-------------|----------|-----------------------|-------------|
| 10011 | Bunny Boot | Beginner's ski boot | 518 | 1001 | 150 | |
| 10012 | Ace Ski Boot | Intermediate ski boot | 519 | 1002 | 200 | |
| 10013 | Pro Ski Boot | Advanced ski boot | 520 | 1003 | 410 | |
| 10021 | Bunny Ski Pole | Beginner's ski pole | 528 | 1011 | 16,25 | |
| 10022 | Ace Ski Pole | Intermediate ski pole | 529 | 1012 | 21,95 | |
| 10023 | Pro Ski Pole | Advanced ski pole | 530 | 1013 | 40,95 | |
| 20106 | Junior Soccer Ball | Junior soccer ball | 613 | | 11 | |
| 20108 | World Cup Soccer Ball | World cup soccer ball | 615 | | 28 | |

Uwaga: pokazano tylko produkty, gdzie **ID** jest od 10011 do 20108



| ID | REGION_ID | CITY | STATE | COUNTRY | ZIP_CODE | PHONE | MANAGER_ID |
|-------|-----------|------------|-------|----------------|----------|-------|------------|
| 101 | 1 | Seattle | WA | USA | | | 6 |
| 10501 | 5 | Bratislava | | Czechoslovakia | | | 10 |
| 201 | 2 | Sao Paolo | | Brazil | | | 7 |
| 301 | 3 | Lagos | | Nigeria | | | 8 |
| 401 | 4 | Hong Kong | | | | | 9 |

WAREHOUSE

PRODUCT

INVENTORY

Uwaga: pokazano tylko pozycje zawierające produkty, gdzie **PRODUCT_ID** jest od 10011 do 20108

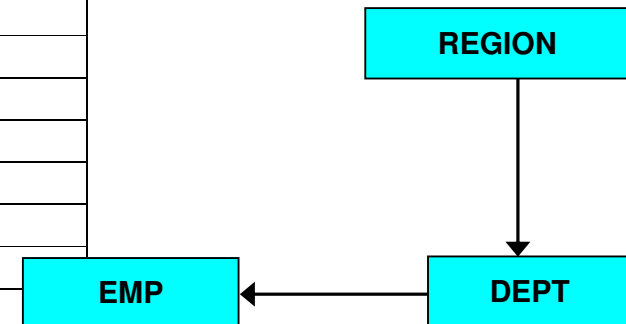
Uwaga: pokazano tylko produkty, gdzie **ID** jest od 10011 do 20108

| ID | NAME | SHORT_DESC | LONGTEXT_ID | IMAGE_ID | SUGGESTED_WHLSL_PRICE | WHLSL_UNITS |
|-------|-----------------------|-----------------------|-------------|----------|-----------------------|-------------|
| 10011 | Bunny Boot | Beginner's ski boot | 518 | 1001 | 150 | |
| 10012 | Ace Ski Boot | Intermediate ski boot | 519 | 1002 | 200 | |
| 10013 | Pro Ski Boot | Advanced ski boot | 520 | 1003 | 410 | |
| 10021 | Bunny Ski Pole | Beginner's ski pole | 528 | 1011 | 16,25 | |
| 10022 | Ace Ski Pole | Intermediate ski pole | 529 | 1012 | 21,95 | |
| 10023 | Pro Ski Pole | Advanced ski pole | 530 | 1013 | 40,95 | |
| 20106 | Junior Soccer Ball | Junior soccer ball | 613 | | 11 | |
| 20108 | World Cup Soccer Ball | World cup soccer ball | 615 | | 28 | |

| PRODUCT_ID | WAREHOUSE_ID | AMOUNT_IN_STOCK | REORDER_POINT | MAX_IN_STOCK | OUT_OF_STOCK_EXPLANATION | RESTOCK_DATE |
|------------|--------------|-----------------|---------------|--------------|--------------------------|--------------|
| 10011 | 101 | 650 | 625 | 1100 | | |
| 10012 | 101 | 600 | 560 | 1000 | | |
| 10012 | 10501 | 300 | 300 | 525 | | |
| 10013 | 101 | 400 | 400 | 700 | | |
| 10013 | 10501 | 314 | 300 | 525 | | |
| 10021 | 101 | 500 | 425 | 740 | | |
| 10022 | 101 | 300 | 200 | 350 | | |
| 10022 | 10501 | 502 | 300 | 525 | | |
| 10023 | 101 | 400 | 300 | 525 | | |
| 10023 | 10501 | 500 | 300 | 525 | | |
| 20106 | 101 | 993 | 625 | 1000 | | |
| 20106 | 201 | 220 | 150 | 260 | | |
| 20106 | 10501 | 150 | 100 | 175 | | |
| 20108 | 101 | 700 | 700 | 1225 | | |
| 20108 | 201 | 166 | 150 | 260 | | |
| 20108 | 10501 | 222 | 200 | 350 | | |

| ID | LAST_NAME | FIRST_NAME | USERID | START_DATE | COMMENTS | MANAGER_ID | TITLE | DEPT_ID | SALARY | COMMISSION_PCT |
|----|--------------|------------|----------|------------|----------|------------|----------------------|---------|--------|----------------|
| 1 | Velasquez | Carmen | cvelasqu | 1990-03-03 | | | President | 50 | 2500 | |
| 2 | Ngao | LaDoris | lngao | 1990-03-08 | | 1 | VP, Operations | 41 | 1450 | |
| 3 | Nagayama | Midori | mnagayam | 1991-06-17 | | 1 | VP, Sales | 31 | 1400 | |
| 4 | Quick-To-See | Mark | mquickto | 1990-04-07 | | 1 | VP, Finance | 10 | 1450 | |
| 5 | Ropeburn | Audry | aropebur | 1990-03-04 | | 1 | VP, Administration | 50 | 1550 | |
| 6 | Urguhart | Molly | murguhar | 1991-01-18 | | 2 | Warehouse Manager | 41 | 1200 | |
| 7 | Menchu | Roberta | rmenchu | 1990-05-14 | | 2 | Warehouse Manager | 42 | 1250 | |
| 8 | Biri | Ben | bbiri | 1990-04-07 | | 2 | Warehouse Manager | 43 | 1100 | |
| 9 | Catchpole | Antoinette | acatchpo | 1992-02-09 | | 2 | Warehouse Manager | 44 | 1300 | |
| 10 | Havel | Marta | mhavel | 1991-02-27 | | 2 | Warehouse Manager | 45 | 1307 | |
| 11 | Magee | Colin | cmagee | 1990-05-14 | | 3 | Sales Representative | 31 | 1400 | 10 |
| 12 | Giljum | Henry | hgiljum | 1992-01-18 | | 3 | Sales Representative | 32 | 1490 | 12,5 |
| 13 | Sedeghi | Yasmin | ysedeghi | 1991-02-18 | | 3 | Sales Representative | 33 | 1515 | 10 |
| 14 | Nguyen | Mai | mnguyen | 1992-01-22 | | 3 | Sales Representative | 34 | 1525 | 15 |
| 15 | Dumas | Andre | adumas | 1991-10-09 | | 3 | Sales Representative | 35 | 1450 | 17,5 |
| 16 | Maduro | Elena | emaduro | 1992-02-07 | | 6 | Stock Clerk | 41 | 1400 | |
| 17 | Smith | George | gsmith | 1990-03-08 | | 6 | Stock Clerk | 41 | 940 | |
| 18 | Nozaki | Akira | anozaki | 1991-02-09 | | 7 | Stock Clerk | 42 | 1200 | |
| 19 | Patel | Vikram | vpatel | 1991-08-06 | | 7 | Stock Clerk | 42 | 795 | |
| 20 | Newman | Chad | cnewman | 1991-07-21 | | 8 | Stock Clerk | 43 | 750 | |
| 21 | Markarian | Alexander | amarkari | 1991-05-26 | | 8 | Stock Clerk | 43 | 850 | |
| 22 | Chang | Eddie | echang | 1990-11-30 | | 9 | Stock Clerk | 44 | 800 | |
| 23 | Patel | Radha | rpatel | 1990-10-17 | | 9 | Stock Clerk | 34 | 795 | |
| 24 | Dancs | Bela | bdancs | 1991-03-17 | | 10 | Stock Clerk | 45 | 860 | |
| 25 | Schwartz | Sylvie | sschwart | 1991-05-09 | | 10 | Stock Clerk | 45 | 1100 | |

| ID | NAME |
|----|----------------------|
| 1 | North America |
| 2 | South America |
| 3 | Africa / Middle East |
| 4 | Asia |
| 5 | Europe |



| ID | NAME | REGION_ID |
|----|----------------|-----------|
| 10 | Finance | 1 |
| 31 | Sales | 1 |
| 32 | Sales | 2 |
| 33 | Sales | 3 |
| 34 | Sales | 4 |
| 35 | Sales | 5 |
| 41 | Operations | 1 |
| 42 | Operations | 2 |
| 43 | Operations | 3 |
| 44 | Operations | 4 |
| 45 | Operations | 5 |
| 50 | Administration | 1 |

A. Literatura

1. Oryginalna dokumentacja firmy Oracle: *Oracle8i SQL Reference*
2. Oryginalna dokumentacja firmy Oracle: *Oracle8i Reference Release 2 (8.1.6)*