

Robert Chwastek

Bazy danych

Niniejszy tekst zawiera fragmenty wykładu pod nazwą "Bazy Danych", który prowadzę dla studentów 3-go roku specjalności Telekomunikacja w Akademii Górniczo-Hutniczej w Krakowie.

Wszystkie nazwy producentów i produktów są własnością odpowiednich firm i zostały użyte jedynie w celach informacyjnych.

Copyright (c) Robert Chwastek, Kraków 1996.

Spis treści:

1.	POJĘCIA PODSTAWOWE	5
1.1.	DEFINICJA BAZY DANYCH.....	5
1.2.	SYSTEM ZARZĄDZANIA BAZĄ DANYCH	6
1.3.	TRANSAKCJE	6
1.4.	JĘZYKI STOSOWANE W BAZACH DANYCH.....	7
2.	TYPY DANYCH	8
2.1.	SPIS TYPÓW DANYCH.....	8
2.2.	TYP NUMERYCZNY	8
2.3.	KONWERSJE TYPÓW	10
2.4.	WARTOŚCI PUSTE	10
3.	MODEL RELACYJNY	12
3.1.	STRUKTURY DANYCH W MODELU RELACYJNYM	12
3.2.	ROZKAZY JĘZYKA SQL	13
3.3.	SKŁADNIA ROZKAZÓW SQL	14
3.3.1.	Definicje podstawowe	14
3.3.2.	Rozkaz CREATE TABLE.....	15
3.3.3.	Rozkaz DROP	16
3.3.4.	Rozkaz INSERT.....	17
3.3.5.	Rozkaz DELETE	18
3.3.6.	Rozkaz CREATE SEQUENCE	19
3.3.7.	Rozkaz SELECT.....	21
3.3.8.	Rozkaz UPDATE.....	22
3.3.9.	Rozkaz RENAME	23
3.3.10.	Rozkaz ALTER TABLE	24
3.3.11.	Rozkaz CREATE INDEX	25
3.3.12.	Rozkaz CREATE VIEW	27
3.3.13.	Rozkaz COMMIT.....	28
3.3.14.	Rozkaz ROLLBACK.....	28
3.3.15.	Rozkaz SAVEPOINT.....	28
3.3.16.	Rozkaz SET TRANSACTION	29
3.4.	OPERACJE RELACYJNE.....	30
3.4.1.	Selekcja.....	30
3.4.2.	Projekcja.....	31
3.4.3.	Produkt	32
3.4.4.	Połączenie.....	32
3.4.5.	Operacje mnogościowe.....	33
3.4.6.	Grupowanie	34
3.4.7.	Kolejność klauzul w rozkazie SELECT	35
3.5.	PODZAPYTANIA	35
3.6.	WIDOKI (PERSPEKTYWY).....	37
3.7.	TRANSAKCJE	38
3.8.	NORMALIZACJA RELACJI	38
3.8.1.	Cele normalizacji.....	38
3.8.2.	Pierwsza postać normalna.....	39
3.8.3.	Definicje pomocnicze.....	40
3.8.4.	Druga postać normalna.....	44
3.8.5.	Trzecia postać normalna	46
3.8.6.	Czwarta postać normalna.....	47
3.8.7.	Piąta postać normalna.....	48
3.8.8.	Podsumowanie	48
4.	WARUNKI I WYRAŻENIA.....	50
4.1.	OPERATORY	50
4.1.1.	Operatory arytmetyczne.....	50
4.1.2.	Operatory znakowe	50
4.1.3.	Operatory porównania	51
4.1.4.	Operatory logiczne	52
4.1.5.	Operatory mnogościowe.....	53

4.2.	WYRAŻENIA.....	53
4.3.	WARUNKI.....	55
5.	STANDARDOWE FUNKCJE JĘZYKA SQL.....	57
5.1.	FUNKCJE NUMERYCZNE	57
5.2.	FUNKCJE ZNAKOWE.....	57
5.3.	FUNKCJE GRUPOWE.....	59
5.4.	FUNKCJE KONWERSJI.....	60
5.5.	FUNKCJE OPERACJI NA DATACH	61
5.6.	INNE FUNKCJE	63
5.7.	FORMATY ZAPISU DANYCH.....	64
5.7.1.	<i>Formaty numeryczne</i>	<i>64</i>
5.7.2.	<i>Formaty dat</i>	<i>64</i>
6.	PROGRAMOWANIE PROCEDURALNE - PL/SQL.....	66
6.1.	WPROWADZENIE	66
6.2.	STRUKTURA BLOKU.....	67
6.3.	PROCEDURY I FUNKCJE	67
6.4.	KURSORY.....	68
6.5.	REKORDY.....	70
6.6.	OBSŁUGA BŁĘDÓW.....	71
6.6.1.	<i>Informacje podstawowe</i>	<i>71</i>
6.6.2.	<i>Wyjątki predefiniowane</i>	<i>73</i>
6.6.3.	<i>Obsługa wyjątków.....</i>	<i>73</i>
6.6.4.	<i>Wyjątki zdefiniowane przez użytkownika.....</i>	<i>74</i>
6.7.	ROZKAZY JĘZYKA PL/SQL	75
6.7.1.	<i>Rozkaz OPEN</i>	<i>75</i>
6.7.2.	<i>Rozkaz CLOSE.....</i>	<i>76</i>
6.7.3.	<i>Rozkaz FETCH</i>	<i>76</i>
6.7.4.	<i>Rozkaz SELECT ... INTO.....</i>	<i>77</i>
6.7.5.	<i>Rozkaz IF.....</i>	<i>77</i>
6.7.6.	<i>Rozkaz LOOP</i>	<i>78</i>
6.7.7.	<i>Rozkaz EXIT</i>	<i>80</i>
6.7.8.	<i>Rozkaz GOTO.....</i>	<i>80</i>
7.	LITERATURA.....	81

1. Pojęcia podstawowe

1.1. Definicja bazy danych

W pewnym uproszczeniu przez **bazę danych** rozumiemy uporządkowany zbiór danych, a przez **system bazy danych** - bazę danych wraz z oprogramowaniem umożliwiającym operowanie na niej. Baza danych jest przechowywana na nośnikach komputerowych. Precyzując definicję bazy danych można powiedzieć, że baza danych jest abstrakcyjnym, informatycznym modelem wybranego fragmentu rzeczywistości (ten fragment rzeczywistości bywa nazywany miniświatem).

Fragment rzeczywistości może być rozumiany jako:

- rzeczywistość fizyczna - taka, którą postrzegamy naszymi organami percepcji
- rzeczywistość konceptualna - istniejąca najczęściej w wyobraźni pewnych osób; przykładem tej rzeczywistości może być projekt nowego samolotu firmy Boeing, który istnieje tylko w wyobraźni konstruktorów.

Poprawne (z punktu widzenia człowieka) operowanie na bazie danych wiąże się z właściwą interpretacją danych, które zostały w niej zapisane. W związku z tym konieczny jest opis semantyki (znaczenia) danych, przechowywanych w bazie. System bazy danych służy więc do modelowania rzeczywistości (fragmentu). W systemach baz danych rzeczywistość opisuje się za pomocą **modelu danych**. Przez model danych rozumiemy zbiór abstrakcyjnych pojęć umożliwiających reprezentację określonych własności tego świata. Zbiór pojęć użyty do opisu własności konkretnego fragmentu świata rzeczywistego, istotnych z punktu widzenia danego zastosowania tworzy **schemat bazy danych**. Baza danych jest modelem **logicznie spójnym** służącym określonej **celowi**. W związku z tym baza danych nie może (nie powinna) przyjąć stanu, który nie jest nigdy osiągalny w modelowanej rzeczywistości.

Z bazy danych korzysta pewna ściśle określona grupa użytkowników. Szczególnymi użytkownikami są projektanci bazy danych, którzy definiują jej strukturę i przygotowują niezbędne programy zwane **aplikacjami**. Baza danych jest wypełniania danymi i przetwarzana. Grupy osób wypełniające i przetwarzające bazę danych mogą być rozłączne. Do wypełniania i przetwarzania bazy danych służą najczęściej wykonane w tym celu aplikacje.

Można więc powiedzieć, że każda baza danych posiada:

- źródło danych
- użytkowników
- związki z reprezentowaną rzeczywistością

Baza danych to dane i tzw. schemat bazy danych. Dane opisują cechy (własności) modelowanych obiektów. Nie jest jednak możliwa ich interpretacja bez użycia schematu. Schemat jest opisem struktury (formatu) przechowywanych danych oraz wzajemnych powiązań między nimi.

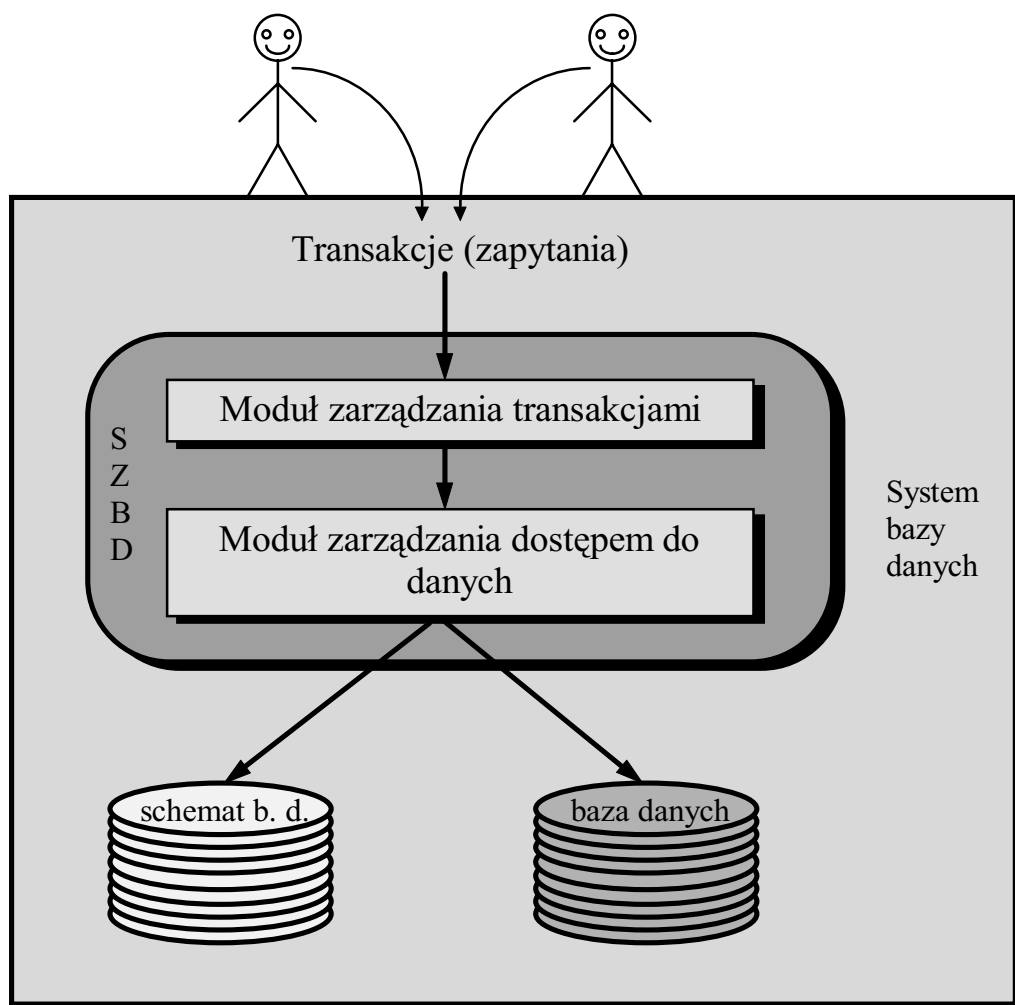
1.2. System zarządzania bazą danych

System zarządzania bazą danych (SZBD) jest to zestaw programów umożliwiających tworzenie i eksploatację bazy danych. System zarządzania bazą danych jest oprogramowaniem ogólnego przeznaczenia. **System bazy danych** składa się z bazy danych, systemu zarządzania bazą danych i ewentualnie z zestawu aplikacji wspomagających pracę poszczególnych grup użytkowników.

1.3. Transakcje

Transakcja jest sekwencją instrukcji po wykonaniu której spójna baza danych nadal zachowuje swą spójność (zgodność z modelowaną rzeczywistością). Transakcja jest operacją atomową tzn. system zarządzania bazą danych może wykonać wszystkie instrukcje wchodzące w skład transakcji albo żadnej. W rzeczywistości system zarządzania bazą danych wykonuje po kolei instrukcje wchodzące w skład transakcji i w przypadku niepowodzenia którejkolwiek z nich wycofuje instrukcje uprzednio wykonane.

Podsumowując podane wcześniej informacje, system bazy danych przedstawia następujący rysunek:



1.4. Języki stosowane w bazach danych

Języki, które stosuje się do projektowania i wypełniania bazy danych można podzielić na cztery różne grupy:

- język definiowania danych (Data Definition Language - DDL), który umożliwia definiowanie struktury danych przechowywanych w bazie, czyli tworzenie schematu implementacyjnego
- język manipulowania danymi (Data Manipulation Language - DML), który umożliwia wypełnianie, modyfikowanie i usuwanie informacji z bazy danych.
- język sterowania danymi (Data Control Language - DCL), który umożliwia sterowanie transakcjami (np. zatwierdzanie lub wycofywanie)
- język zapytań (Query Language), który umożliwia pobieranie z bazy informacji zgodnych z podanymi warunkami

2. Typy danych

2.1. Spis typów danych

<i>Typ</i>	<i>Opis</i>
char(size)	Ciąg znaków o zmiennej długości nie większej niż podany rozmiar. Dla tego typu maksymalny rozmiar może wynosić 255. W przypadku nie podania rozmiaru domyślnie przyjmowana jest wartość 1
character	Synonim do char
varchar(size)	W aktualnej wersji ORACLE'a jest to synonim do char, konieczne jest jednak podanie rozmiaru. W przyszłych wersjach zakłada się, że char będzie ciągiem znaków o stałej długości, natomiast varchar o zmiennej.
date	Poprawne daty z zakresu 1 stycznia 4712 p.n.e. do 31 grudnia 4712 n.e. Domyślny format wprowadzania to "DD-MON-YY np.: '01-JAN-89"
long	Ciąg znaków o zmiennej długości nie większej niż 65535 znaków. Można zdefiniować tylko jedną kolumnę typu long w jednej tabeli.
long varchar	synonim do long
raw(size)	Ciąg bajtów o podanej długości. Specyfikacja rozmiaru jest konieczna. Rozmiar maksymalny dla tego typu to 255. Wartości do pół tego typu są wstawiane jako ciągi znaków w notacji szesnastkowej.
long raw	Ciąg bajtów o zmiennej długości. Pozostałe własności jak dla typu long. Wartości do pół tego typu są wstawiane jako ciągi znaków w notacji szesnastkowej.
rowid	Unikalna wartość identyfikująca wiersz. Podany typ jest pseudotypem, tzn. kolumna tego typu nie może być utworzona w tabeli i nie jest w niej przechowywana, ale obliczana na podstawie informacji o fizycznym położeniu wiersza na dysku, w pliku itp. Wartość typu rowid może być przekonwertowana do typu znakowego za pomocą funkcji ROWIDTOCHAR.
number	Typ numeryczny. Jego wartości mogą się zmieniać w zakresie od $1.0 * 10^{-129}$ do $9.99 * 10^{124}$. Możliwe jest ograniczenie podanego zakresu przez specyfikację precyzji i skali w sposób opisany poniżej.

2.2. Typ numeryczny

Typ numeryczny jest używany do przechowywania liczb zarówno zmiennie jak i stałoprzecinkowych. Dla kolumn numerycznych typ można wyspecyfikować na jeden z trzech sposobów:

- number
- number (precyzja)
- number (precyzja, skala)

Precyzja określa całkowitą liczbę cyfr znaczących i może się zmieniać od 1 do 38. Skala określa liczbę cyfr po prawej stronie kropki dziesiętnej i może się zmieniać w zakresie od -84 do 127.

W momencie definiowania kolumny numerycznej dobrym zwyczajem jest podawanie zarówno precyzji jak i skali, ponieważ wymusza to automatyczną kontrolę wprowadzanych wartości, a więc zwiększa szanse na zachowanie spójności bazy danych. Jeśli wartość przekracza maksymalną precyzję, to generowany jest błąd. Jeśli wartość przekracza skalę, to jest zaokrąglana. Jeśli skala jest ujemna, to wartość jest zaokrąglana do podanej liczby miejsc po lewej stronie kropki dziesiętnej; np. specyfikacja (10, -2) oznacza zaokrąglenie do setek. Czasami specyfikuje się również skalę większą niż precyzję. Oznacza to wtedy, że wprowadzane liczby muszą mieć po kropce dziesiętnej taką liczbę zer jaka jest różnica między skalą a precyzją; np: number(4, 5) będzie wymagał jednego zera po kropce dziesiętnej.

Liczby można również zapisywać w formacie zmiennoprzecinkowym. Składa się ona wtedy z ułamka dziesiętnego, bezpośrednio po którym znajduje się litera E i wykładnik potęgi liczby 10 przez jaki trzeba pomnożyć ten ułamek. Dla przykładu zapis 9.87E-2 oznacza $9.87 * 10^{-2}$.

Inne systemy baz danych posiadają kilka różnych typów numerycznych, które w ORACLE'u implementowane są jako number. Możliwe jest stosowanie nazw tych typów. Nazwy te oraz sposób implementacji za pomocą typu number przedstawia tabela:

<i>Specyfikacja</i>	<i>Typ</i>	<i>Precyzja</i>	<i>Skala</i>
number	number	38	null
number(*)	number	38	null
number(*, s)	number	38	s
number(p)	number	p	0
number(p,s)	number	p	s
decimal	number	38	0
decimal(*)	number	38	0
decimal(*, s)	number	38	s
decimal(p)	number	p	0
decimal(p, s)	number	p	s
integer	number	38	0
smallint	number	38	0
float	number	38	null
float(*)	number	38	null
float(b)	number	b	null
real	number	63 binary (18 decimal)	null
double precision	number	38	null

2.3. Konwersje typów

W systemie zarządzania bazą danych możliwe są konwersje danych jednego typu do danych innego typu. Dane te muszą spełniać pewne warunki, aby konwersja taka była możliwa, np. chcąc przekonwertować ciąg znaków do liczby, ciąg ten powinien składać się z cyfr. Poniższa tabela przedstawia funkcje służące do wykonywania konwersji pomiędzy poszczególnymi typami w systemie ORACLE:

Z typu	Do typu		
	char	number	date
char	zbędna	TO_NUMBER	TO_DATE
number	TO_CHAR	zbędna	TO_DATE
date	TO_CHAR	niemożliwa	zbędna

2.4. Wartości puste

Pola tabeli mogą przyjmować wartości puste, pod warunkiem, że nie zostało to zabronione przez projektanta bazy danych. Wartość pusta (NULL) nie jest równa wartości 0 i w wyniku obliczenia dowolnego wyrażenia, którego argumentem jest NULL otrzymuje się również wartość pustą (NULL).

Funkcja NVL pozwala dokonać konwersji wartości aktualnej (do niej samej) lub wartości pustej do wartości domyślnej. Działanie funkcji NVL ilustruje przykład:

NVL(COMM, 0) zwróci wartość COMM, jeśli nie jest to wartość pusta lub 0 jeśli COMM ma wartość NULL.

Większość funkcji grupujących ignoruje wartość NULL. Np. zapytanie, którego zadaniem jest obliczenie średniej z pięciu następujących wartości: 1000, NULL, NULL, NULL i 2000 zwróci 1500 ponieważ $(1000 + 2000)/2 = 1500$.

Jedyne operatory porównania, które można użyć do wartości pustej to IS NULL i IS NOT NULL. Jeśli zostanie użyty jakikolwiek inny operator porównania do wartości pustej, to wynik jest nieokreślony. Ponieważ NULL reprezentuje brak wartości, więc nie może on być równy ani nierówny jakiegokolwiek innej wartości, również innemu NULL.

ORACLE traktuje warunki, których wynik jest nieznany jako fałszywe. Tak więc warunek COMM = NULL jest nieznany, w związku z czym rozkaz SELECT z takim warunkiem nie zwróci nigdy żadnego wiersza. Jednak w takiej sytuacji ORACLE nie zgłosi informacji o wystąpieniu błędu.

3. Model relacyjny

Zgodnie z teorią model danych w relacyjnych bazach danych składa się z trzech podstawowych elementów:

- relacyjnych struktur danych
- operatorów relacyjnych umożliwiających tworzenie, przeszukiwanie i modyfikację bazy danych
- więzów integralności jawnie lub niejawnie określających wartości danych

3.1. Struktury danych w modelu relacyjnym

Podstawową strukturą danych jest relacja będąca podzbiorem iloczynu kartezyjskiego dwóch wybranych zbiorów reprezentujących dopuszczalne wartości. W bazach danych relacja przedstawiana jest w postaci tabeli. Relacja jest zbiorem krotek posiadających taką samą strukturę, lecz różne wartości. Każda krotka odpowiada jednemu wierszowi tablicy. Każda krotka posiada co najmniej jeden atrybut odpowiadający pojedynczej kolumnie tablicy. Każda relacja (tablica) posiada następujące własności:

- krotki (wiersze) są unikalne
- atrybuty (kolumny) są unikalne
- kolejność krotek (wierszy) nie ma znaczenia
- kolejność atrybutów (kolumn) nie ma znaczenia
- wartości atrybutów (pól) są atomowe

Przykładową tabelę wraz z jej elementami przedstawia rysunek:

Pesel	Imię	Nazwisko	Wykształcenie
72030403987	Małgorzata	Albinos	WT
65081002987	Damian	Jędrzejek	SO
44101202034	Barbara	Bibicka	P
70010101231	Piotr	Burzyński	WT
55121201223	Mateusz	Manicki	ST

Dokumentacja systemów zarządzania bazami danych posługuje się najczęściej terminologią tabela, wiersz i kolumna, a nie terminologią relacyjną. Wynika to z tego, że

operacje na relacjach są opisywane za pomocą matematycznych operacji na zbiorach i relacjach, które są ścisłe, ale trudno zrozumiałe dla przeciętnego użytkownika. Natomiast posługiwanie się tabelami, wierszami i kolumnami jest mniej formalne i ścisłe, ale bardziej przejrzyste. W dalszej części tego wykładu będzie stosowana zarówno jedna jak i druga terminologia.

Tabela może reprezentować:

- zbiór encji wraz z atrybutami
- zbiór powiązań pomiędzy encjami wraz z ich atrybutami
- zbiór encji wraz z atrybutami i ich powiązania z innymi encjami (wraz z atrybutami)

Każdy wiersz w tabeli reprezentuje pojedynczą encję, powiązanie lub encję wraz z powiązaniami. W tabeli nie powinny powtarzać się dwa identyczne wiersze - zabezpieczenie przed tym powtórzeniem jest realizowane poprzez pola kluczowe. Wiersze w odróżnieniu od kolumn są dynamiczne - działanie bazy danych polega na dopisywaniu, modyfikacji i usuwaniu wierszy. W raz utworzonej tabeli rzadko dopisuje się lub kasuje kolumny - ponieważ każda z kolumn reprezentuje pewną własność modelowanej rzeczywistości.

W przypadku projektowania tabeli w bazie danych należy stosować się do następujących wskazówek:

- Używaj nazw opisowych do nazwania kolumn tabeli. Kolumny nie powinny mieć znaczenia ukrytego, ani reprezentować kilku atrybutów (złożonych w pojedynczą wartość).
- Bądź konsekwentny w stosowaniu liczby pojedynczej lub mnogiej przy nazywaniu tabeli.
- Twórz tylko te kolumny, które są niezbędne do opisanie modelowanej encji lub powiązania - tabele z mniejszą ilością kolumn są łatwiejsze w użyciu.
- Utwórz kolumnę pól kluczowych dla każdej tabeli.
- Unikaj powtarzania informacji w bazie danych (normalizacja).

3.2. Rozkazy języka SQL

Poniższa tabela zawiera spis podstawowych rozkazów języka SQL wraz z krótkim opisem. Operacje relacyjne będą wyjaśnione dokładniej w dalszej części wykładu.

<i>Rozkaz</i>	<i>Typ</i>	<i>Opis</i>
ALTER TABLE	DDL	Dodaje kolumnę do tabeli, redefiniuje kolumnę w istniejącej tabeli lub redefiniuje ilość miejsca zarezerwowaną dla danych
CREATE INDEX	DDL	Tworzy indeks dla tabeli
CREATE SEQUENCE	DDL	Tworzy obiekt służący do generowania kolejnych liczb - sekwencję. Sekwencji można użyć do generowania unikalnych identyfikatorów w tabelach
CREATE TABLE	DDL	Tworzy tabelę i definiuje jej kolumny oraz alokację przestrzeni dla danych

Bazy danych		Robert Chwastek
CREATE VIEW	DDL	Definiuje widok dla jednej lub większej ilości tabel lub innych widoków
DELETE	DML	Usuwa wszystkie lub wyróżnione wiersze z tabeli
DROP obiekt	DDL	Usuwa indeks, sekwencje, tablicę, widok lub inny obiekt
INSERT	DML	Dodaje nowy wiersz (lub wiersze) do tabeli lub widoku
RENAME	DDL	Zmienia nazwę tabeli, widoku lub innego obiektu
SELECT	DML	Wykonuje zapytanie. Wybiera wiersze i kolumny z jednej lub kilku tabel
UPDATE	DML	Zmienia dane w tabeli
COMMIT	DML	Kończy transakcję i na stałe zapisuje zmiany
ROLLBACK	DML	Wycofuje zmiany od początku transakcji lub zaznaczonego punktu.
SAVEPOINT	DML	Zaznacza punkt, do którego możliwe jest wykonanie rozkazu ROLLBACK
SET TRANSACTION	DDL	Zaznacza aktualną transakcję jako read-only (tylko do odczytu).

3.3. Składnia rozkazów SQL

3.3.1. Definicje podstawowe

- Identyfikator (nazwa) - ciąg liter, cyfr i znaków podkreślenia rozpoczynający się literą lub znakiem podkreślenia. Różne systemy baz danych umożliwiają stosowanie innych znaków wewnątrz identyfikatorów (np. znak '\$', lub '!'). Stosowanie tych znaków nie jest jednak zalecane ze względu na późniejsze problemy związane z przenośnością napisanych w ten sposób aplikacji.
- Słowa zarezerwowane - identyfikatory zastrzeżone posiadające specjalne znaczenie w języku SQL. Spis wszystkich słów zarezerwowanych w języku SQL przez twórców ORACLE'a przedstawia tabela:

access	add	all	alter	and	any
as	asc	audit	between	by	char
check	cluster	column	comment	compress	connect
create	current	date	dba	decimal	default
delete	desc	distinct	drop	else	exclusive
exists	file	float	for	from	grant
graphic	group	having	identified	if	immediate
in	increment	index	install	initial	insert
integer	intersect	into	is	level	like
lock	long	maxextents	minus	mode	modify
noaudit	nocompress	not	nowait	null	number
of	offline	on	online	option	or
order	pctfree	prior	privileges	public	raw

Bazy danych				Robert Chwastek	
rename	resource	revoke	row	rowid	rownum
rows	select	session	set	share	size
smallint	start	successful	synonym	sysdate	table
then	to	trigger	uid	union	unique
update	user	validate	values	varchar	vargraphic
view	whenever	where	with		

- Liczby - mogą być całkowite lub rzeczywiste. Liczba całkowita nie posiada kropki dziesiętnej. W systemie ORACLE liczby można zapisywać w formacie zwykłym lub wykładniczym. Format wykładniczy składa się z liczby oraz wykładnika liczby 10, przez który należy pomnożyć tę liczbę oddzielonego literą 'e' lub 'E'. Przykłady:

$$7E2 = 7 * 10^2$$

$$25e-03 = 25 * 10^{-3}$$

Dodatkowo w systemie ORACLE liczbę całkowitą można zakończyć literą 'K' lub literą 'M'. Litera 'K' oznacza, że cała liczba ma być pomnożona przez 1024 (1 KB), natomiast litera 'M', że liczbę należy pomnożyć przez 1048576 (1 MB). Przykłady:

$$256K = 256 * 1024$$

$$1M = 1 * 1048576$$

- Rozkazy języka SQL kończą się średnikiem

3.3.2. Rozkaz CREATE TABLE

Rozkaz CREATE TABLE służy do tworzenia struktury tabeli (bez danych) i posiada dodatkowe opcje umożliwiające:

- określenie sposobu alokacji przestrzeni do przechowywania danych
- określenie rozmiaru tabeli
- przydzielenie tabeli do określonego klastra
- załadowanie danych będących wynikiem podanego zapytania, do tabeli

Rozkaz CREATE TABLE posiada następującą składnię:

```
CREATE TABLE [user.]table
( {column_element | table_constraint}
[, {column_element | table_constraint} ] ... )
[ PCTFREE n ] [ PCTUSED n ]
[ INITTRANS n ] [ MAXTRANS n ]
[ TABLESPACE tablespace ]
[ STORAGE storage ]
[ CLUSTER cluster (column [, column] ...) ]
[ AS query ]
```

Parametry:

- user - właściciel tabeli, jeśli nie zostanie podany, to właścicielem staje się osoba tworząca tabelę. Tabele dla innych użytkowników może tworzyć tylko administrator systemu zarządzania bazą danych (DBA)

- `table` - nazwa tabeli, powinna być prawidłowym identyfikatorem. Wszystkie obiekty danego użytkownika powinny mieć unikalne nazwy
- `column_element` - definiuje kolumnę i opcjonalne ograniczenia na wartości w tej kolumnie. Tabela musi zawierać co najmniej jedną kolumnę (jak to wynika ze składni)
- `table_constraints` - określa ograniczenia jakie musi spełniać cała tabela
- `tablespace` - określa obszar, w którym należy umieścić tabelę
- `storage` - określa przyszły sposób alokacji pamięci
- `cluster` - określa klastr (którego właścicielem musi być właściciel tabeli), do którego należy przydzielić tabelę
- `query` - jest poprawnym zapytaniem takim samym jak zdefiniowane w rozkazie `SELECT`. Jeśli podane jest zapytanie, to można podać tylko nazwy kolumn - typy i rozmiary są kopiowane z odpowiednich kolumn określonych w zapytaniu. Możliwe jest również pominięcie nazw kolumn, ale tylko wtedy, gdy nazwy te są unikalne i dobrze zdefiniowane w zapytaniu. Liczba wyspecyfikowanych kolumn musi być taka sama jak liczba kolumn w zapytaniu.

Przykłady:

```
CREATE TABLE pracownicy(  
    nr_pracownika NUMBER NOT NULL PRIMARY KEY,  
    imie CHAR(15) NOT NULL CHECK (imie = UPPER(imie)),  
    nazwisko CHAR(25) NOT NULL  
        CHECK (nazwisko = UPPER(nazwisko)),  
    nr_wydzialu NUMBER (3) NOT NULL  
);
```

3.3.3. Rozkaz DROP

Rozkaz `drop` służy do kasowania obiektów różnego rodzaju. Ogólna postać tego rozkazu jest następująca:

```
DROP object_type [user.]object
```

Poniżej przedstawione są różne postacie rozkazu `drop` służące do kasowania poszczególnych typów obiektów:

- `DROP CLUSTER [user.]cluster [INCLUDING TABLES]` - kasowanie klastra. W przypadku podania klauzuli `INCLUDING TABLES` zostaną skasowane wszystkie tabele przydzielone uprzednio do kasowanego klastra. Jeśli klauzula `INCLUDING TABLES` nie zostanie podana, to przed skasowaniem klastra muszą być skasowane wszystkie należące do niego tabele. Jest to zabezpieczenie przed omyłkowym skasowaniem klastra zawierającego tabele, które są potrzebne.
- `DROP [PUBLIC] DATABASE LINK link` - usuwanie połączenia. Jeśli połączenie jest publiczne to skasować je może tylko administrator (DBA).
- `DROP INDEX [user.]index` - kasowanie indeksu.
- `DROP [PUBLIC] ROLLBACK SEGMENT segment` - kasowanie segmentu wycofywania (rollback). Można usunąć tylko te segmenty wycofywania, które nie są używane w danym momencie. Kasowanie segmentów wycofywania może wykonywać tylko administrator bazy danych.

- DROP SEQUENCE [user.]sequence - kasowanie sekwencji.
- DROP [PUBLIC] SYNONYM [user.]synonym - usuwanie synonimu. Synonim publiczny może zostać usunięty tylko przez administratora (DBA). Poszczególni użytkownicy mogą usuwać tylko te segmenty, których są właścicielami.
- DROP TABLE [user.]table - usuwanie tabeli. W momencie usunięcia tabeli automatycznie kasowane są skojarzone z nią indeksy zarówno utworzone przez właściciela tabeli jak i przez innych użytkowników. Widoki i synonimy wskazujące na tabelę nie są kasowane automatycznie, ale stają się nieprawidłowe.
- DROP TABLESPACE tablespace [INCLUDING CONTENTS] - usuwanie obszaru danych. Rozkaz ten może być wykonany tylko przez administratora (DBA). W przypadku podania klauzuli INCLUDING CONTENTS obszar danych zostanie skasowany nawet wtedy, gdy zawiera dane. Jeśli klauzula INCLUDING CONTENTS nie została podana, a obszar zawiera dane, to nie zostanie skasowany.
- DROP VIEW [user.]view - usuwanie widoku. Po usunięciu widoku, inne widoki lub synonimy, które odwoływały się do widoku skasowanego, nie zostaną skasowane, ale stają się nieprawidłowe.

3.3.4. Rozkaz INSERT

Rozkaz insert dodaje nowe wiersze do tabeli lub do tabel przynależących do widoku. Aby dodać wiersze do tabeli należy być właścicielem tabeli, administratorem (DBA) lub posiadać uprawnienia dopisywania do tej tabeli.

Składnia rozkazu:

```
INSERT INTO [user.]table [ (column [, column] ...) ]  
    { VALUES (value [, value] ...) | query }
```

Parametry:

- user - nazwa właściciela tabeli
- table - nazwa tabeli, do której dopisywane są wiersze
- column - nazwa kolumny wewnątrz tabeli lub widoku
- value - pojedyncza wartość odpowiadająca odpowiedniej pozycji na liście kolumn. Wartość może być dowolnym wyrażeniem. Jeśli wprowadzana wartość nie jest równa NULL to musi być zgodna z typem wartości kolumny, do której zostanie dopisana.
- query - prawidłowy rozkaz SELECT, który zwraca taką ilość wartości jak podana w liście określającej kolumny. Zapytanie nie może mieć klauzuli ORDER FOR ani FOR UPDATE.

Opis:

Rozkaz INSERT użyty z klauzulą VALUES zawsze dodaje dokładnie jeden wiersz. Do pól wyspecyfikowanych w liście kolumn (lub do wszystkich kolumn) wstawiane są podane wartości. Kolumny nie wyspecyfikowane na liście kolumn przyjmują wartości puste NULL (w związku z tym nie mogą być uprzednio zadeklarowane jako NOT NULL).

Jeśli użyje się rozkazu SELECT zamiast klauzuli VALUES, to możliwe jest dodanie większej ilości wierszy (wszystkich zwróconych przez zapytanie). Po wykonaniu

zapytania kolumny będące jego rezultatem są dopasowywane i wpisywane do kolumn podanych na liście kolumn (lub do wszystkich kolumn, jeśli ich nie wyspecyfikowano). Zapytanie może odwoływać się również do tabeli, do której dopisywane są wiersze.

W przypadku, gdy lista kolumn nie jest podana, to wartości są dopasowywane do poszczególnych kolumn na podstawie ich wewnętrznego porządku. Porządek ten nie musi być taki sam jak kolejność kolumn przy tworzeniu tabeli.

Żaden wiersz nie zostanie dopisany, jeśli zapytanie nie zwróci żadnych wierszy.

Przykłady:

```
INSERT INTO pracownicy VALUES  
  (50, 'JAN', 'KOWALSKI', 3);
```

```
INSERT INTO ksiazki (tytul, autor, miejsce)  
  SELECT 'Pan Tadeusz', autor_nr, miejsce_nr  
  FROM autorzy, miejsca  
  WHERE nazwisko = 'Mickiewicz' AND  
        miejsce = 'lewa polka'  
;
```

3.3.5. Rozkaz DELETE

Rozkaz DELETE służy do usuwania wierszy z tabeli.

Składnia:

```
DELETE [FROM] [user.]table [alias] [WHERE condition]
```

Parametry:

user - nazwa użytkownika

table - nazwa tabeli lub widoku, z którego należy usunąć wiersze

alias - nazwa aliasu odnoszącego się do tabeli, który jest używany w rozkazie DELETE z powiązanymi zapytaniem

condition - warunek jaki muszą spełniać wiersze, które należy usunąć. Warunek ten może odwoływać się do tabeli, na której przeprowadza się operację i zawierać powiązane z nim zapytania. Konieczne jest jednak, by warunek, dla każdego z wiersza podanej tabeli, był obliczany do wartości TRUE lub FALSE.

Opis:

Cała przestrzeń zwolniona przez skasowane wiersze i elementy indeksów jest zatrzymywana przez tę tabelę i indeks.

Przykłady:

Skasowanie wszystkich wierszy w tabeli pracownicy:

```
DELETE FROM pracownicy ;
```

Skasowanie wszystkich wierszy zawierających książki, których autor oznaczony jest numerem 2:

```
DELETE FROM ksiazki WHERE autor = 2 ;
```

3.3.6. Rozkaz CREATE SEQUENCE

Tworzy obiekt (nazywany sekwencją), za pomocą którego wielu użytkowników może generować unikalne liczby całkowite. Sekwencję mogą być użyte do generacji kluczy pierwotnych w sposób automatyczny. Do utworzenia sekwencji konieczne są przynajmniej uprawnienia RESOURCE w conajmniej jednej przestrzeni tabel.

Składnia:

```
CREATE SEQUENCE [user.]sequence
  [INCREMENT BY n]
  [START WITH n]
  [MAXVALUE n | NOMAXVALUE]
  [MINVALUE n | NOMINVALUE]
  [CYCLE | NOCYCLE]
  [CACHE n | NOCACHE]
  [ORDER | NOORDER]
```

Parametry:

- user - nazwa użytkownika
- sequence - nazwa tworzonej sekwencji, musi być poprawnym identyfikatorem i być unikalna w obrębie danego użytkownika.
- INCREMENT BY - określa różnicę między kolejno generowanymi liczbami. Jeśli liczba ta jest ujemna, to będą generowane liczby w porządku malejącym, w przeciwnym wypadku - w porządku rosnącym. Domyślnie przyjmowana jest wartość 1. Dozwolona jest każda liczba różna od 0.
- START WITH - pierwsza liczba, która powinna być wygenerowana przez sekwencję. Domyślną wartością jest MINVALUE dla sekwencji rosnących i MAXVALUE dla sekwencji malejących. Utworzona sekwencja nie jest zainicjalizowana i pierwszą wartość otrzymuje się po jednokrotnym odczytaniu pseudokolumny NEXTVAL.
- MINVALUE - określa minimalną wartość jaką może wygenerować sekwencja. Domyślnie dla sekwencji rosnących jest to 1, natomiast dla malejących wartość ta wynosi $-10e27 + 1$. Podanie NOMINVALUE powoduje, że sekwencja nie będzie sprawdzać wartości minimalnej.
- MAXVALUE - określenie maksymalnej wartości, jaką może wygenerować sekwencja. Wartościami domyślnymi są -1 i $10e27 - 1$ odpowiednio dla sekwencji malejącej i rosnącej. Wyprecyzowanie NOMAXVALUE powoduje, że sekwencja nie będzie sprawdzać wartości maksymalnej.
- CYCLE, NOCYCLE - domyślną wartością jest NOCYCLE, które powoduje, że żadne dodatkowe numery nie zostaną wygenerowane po osiągnięciu końca sekwencji. W tym wypadku każda próba generacji kolejnego numeru spowoduje zgłoszenie błędu. W przypadku podania klauzuli CYCLE po osiągnięciu wartości maksymalnej sekwencja powróci do wartości minimalnej (dla sekwencji rosnących) lub po osiągnięciu wartości minimalnej powróci do maksymalnej (dla sekwencji malejących) rozpoczynając kolejny cykl generacji numerów.

- **CACHE, NOCACHE** - klauzula **CACHE** włącza wykonywanie pre-alokacji numerów sekwencji i przechowywanie ich w pamięci, co skutkuje zwiększeniem szybkości generacji kolejnych liczb. Klauzula **NOCACHE** wyłącza tę możliwość. Domyślnie przyjmowane jest **CACHE 20**. Wartość podana w **CACHE** musi być mniejsza niż **MAXVALUE - MINVALUE**.
- **ORDER, NOORDER** - klauzula **ORDER** gwarantuje, że kolejne liczby będą generowane w porządku jakim otrzymane zostały przez system polecenia ich generacji. Klauzula **NOORDER** wyłącza tę własność. Kolejność generacji numerów w sekwencji jest ważna w aplikacjach, w których ważna jest kolejność (czasowa) wykonywanych operacji. Zwykle nie jest ona ważna w aplikacjach, które wykorzystują sekwencje tylko do generacji kluczy pierwotnych.

Opis:

Sekwencje mogą być używane do generacji kluczy pierwotnych dla jednej tabeli lub wielu tabel i wielu użytkowników. Aby mieć dostęp do sekwencji, której właścicielem jest inny użytkownik, należy mieć uprawnienia **SELECT** do tej sekwencji. Sekwencja może posiadać synonim.

Numerzy w sekwencjach są generowane niezależnie od tabel, dlatego mogą być używane jako liczby unikalne dla kilku różnych tabel i użytkowników. Jest jednak możliwe, że niektóre numery z sekwencji zostaną pominięte, ponieważ zostały one wygenerowane i użyte w transakcji, która następnie została wycofana. Dodatkowo jeden użytkownik może nie zdawać sobie sprawy, że inni użytkownicy korzystają z tej samej sekwencji (co również skutkuje pominięciem numerów dla tego użytkownika).

Dostęp do sekwencji zapewniają dwie pseudokolumny: **NEXTVAL** i **CURRVAL**. Pseudokolumna **NEXTVAL** jest używana do generacji następnej wartości z podanej sekwencji. Składnia jest następująca:

`sequence.NEXTVAL`

gdzie `sequence` jest nazwą sekwencji.

Pseudokolumna **CURRVAL** pozwala na odczytanie aktualnej wartości sekwencji. Aby możliwe było użycie **CURRVAL** konieczne jest wcześniejsze użycie **NEXTVAL** w aktualnej sesji dla danej sekwencji. Składnia tego rozkazu jest następująca:

`sequence.CURRVAL`

gdzie `sequence` jest nazwą sekwencji.

Pseudokolumny **NEXTVAL** i **CURRVAL** mogą być używane w:

- w klauzuli **SELECT** i rozkazie **SELECT** (z wyjątkiem widoków)
- liście wartości rozkazu **INSERT**
- wyrażeniu **SET** w rozkazie **UPDATE**

Pseudokolumny **NEXTVAL** i **CURRVAL** nie można używać w:

podzapytaniach

w liście **select** dla widoków

- ze słowem kluczowym **DISTINCT**
- z klauzulami **ORDER BY**, **GROUP BY** i **HAVING** w rozkazie **SELECT**
- z operatorem ustawienia (**UNION**, **INTERSECT**, **MINUS**)

Przykłady:

```
CREATE SEQUENCE eseq INCREMENT BY 10 ;
INSERT INTO pracownicy
VALUES (eseq.NEXTVAL, 'Jan', 'Kowalski', 3) ;
```

3.3.7. Rozkaz SELECT

Rozkaz SELECT służy do wyświetlania wierszy i kolumn z jednej lub kilku tabel. Może być używany jako osobny rozkaz lub (z pewnymi ograniczeniami) jako zapytanie lub podzapytanie w innych poleceniach. Aby odczytać dane z określonej tabeli trzeba być jej właścicielem, mieć uprawnienia SELECT dla tej tabeli lub być administratorem bazy (DBA).

Składnia:

```
SELECT [ALL | DISTINCT]
      { * | table.* | expr [c_alias] }
      [, { table.* | expr [c_alias] } ] ...
FROM [user.]table [t_alias]
     [, [user.]table [t_alias]] ...
[ WHERE condition ]
[ CONNECT BY condition [START WITH condition] ]
[ GROUP BY expr [. Expr] ... [HAVING condition] ]
[ {UNION | INTERSECT | MINUS} SELECT ...]
[ ORDER BY {expr | position} [ASC | DESC]
           [, {expr | position} [ASC | DESC]] ] ...
[ FOR UPDATE OF column [, column] ... [NOWAIT] ]
```

Parametry:

- ALL - ustawiane domyślnie, oznacza, że wszystkie wiersze, które spełniają warunki rozkazu SELECT powinny zostać pokazane.
- DISTINCT - określa, że wiersze powtarzające się powinny zostać usunięte przed zwróceniem ich na zewnątrz. Dwa wiersze traktuje się jako równe jeśli wszystkie wartości dla każdej z kolumn zwracanych rozkazem SELECT są sobie równe.
- * - oznacza, że wszystkie kolumny ze wszystkich wymienionych tabel powinny zostać pokazane.
- table.* - oznacza, że wszystkie kolumny z podanej tabeli powinny zostać pokazane
- expr - wyrażenie, zostanie opisane w dalszej części wykładu
- c_alias - jest inną nazwą dla kolumny (aliasem) i powoduje, że nazwa ta zostanie użyta jako nagłówek kolumny podczas wyświetlania. W żaden sposób nie jest zmieniana rzeczywista nazwa kolumny. Aliasy kolumn nie mogą być używane w dowolnym miejscu zapytania.
- [user.]table - określa które tabele i widoki należy pokazać. Jeśli użytkownik nie jest podany to domyślnie przyjmowany jest użytkownik aktualny (wykonujący rozkaz SELECT).

- `t_alias` - pozwala określić inną nazwę dla tabeli w celu obliczenia zapytania. Najczęściej jest używane w zapytaniach powiązanych. W tym wypadku inne odwołania do tabeli wewnątrz zapytania muszą posługiwać się wyspecyfikowanym aliasem.
- `condition` - warunek, jaki muszą spełniać wiersze, aby zostały zwrócone przez zapytanie. Warunki zostaną opisane dokładniej w dalszej części wykładu.
- `position` - identyfikuje kolumnę bazując na jej tymczasowym położeniu w rozkazie `SELECT`, a nie na nazwie.
- `ASC`, `DESC` - określa, że zwracane wiersze powinny być posortowane w kolejności rosnącej lub malejącej (odpowiednio).
- `column` - nazwa kolumny należąca do jednej z tabel podanych w klauzuli `FROM`.
- `NOWAIT` - określa, że `ORACLE` powinien zwrócić sterowanie do użytkownika, zamiast czekać na możliwość zablokowania wiersza, który został uprzednio zablokowany przez innego użytkownika.

Opis:

Użycie nazwy tabeli przed nazwą kolumny i nazwy użytkownika przed nazwą tabeli jest najczęściej opcjonalne, to jednak dobrym zwyczajem jest podawanie nazw w pełni kwalifikowanych z dwóch powodów:

- jeśli dwie tabele mają kolumny o tej samej nazwie, to nie wiadomo, która powinna być użyta w rozkazie `SELECT`
- `ORACLE` wykonuje znacznie mniej obliczeń, jeśli nazwy te są podane i nie trzeba ich szukać.

Pozostałe operacje wykonywane przez rozkaz `SELECT` zostaną opisane w dalszej części wykładu.

Przykłady:

```
SELECT imie, nazwisko FROM pracownicy ;
```

```
SELECT tytuł, autorzy.imie, autorzy.nazwisko,
       miejsca.miejsce
FROM ksiazki, autorzy, miejsca
WHERE ksiazki.autor = autorzy.autor_nr AND
      ksiazki.miejsce = miejsca.miejsce_nr
;
```

3.3.8. Rozkaz UPDATE

Rozkaz `UPDATE` służy do zmiany danych zapisanych w tabeli. Warunkiem wykonania tego polecenia jest bycie właścicielem tabeli, administratorem (DBA) lub posiadanie uprawnień `UPDATE` dla tej tabeli.

Składnia:

```
UPDATE [user.]table [alias]
  SET column = expr [, column = expr] ...
  [ WHERE condition ]
```

lub

```
UPDATE [user.]table [alias]
  SET (column [, column] ...) = (query)
    [, column [, column] ...) = (query) ] ...
  [ WHERE condition ]
```

Parametry:

- user - nazwa właściciela tabeli.
- table - nazwa istniejącej tabeli.
- alias - dodatkowa nazwa używana do dostępu do tabeli w pozostałych klauzulach rozkazu.
- column - kolumna wewnątrz tabeli. Nawiasy nie są potrzebne jeśli lista kolumn zawiera tylko jedną kolumnę.
- expr - wyrażenie - zostanie opisane w dalszej części wykładu
- query - rozkaz SELECT bez klauzul ORDER BY i FOR UPDATE, często skorelowany ze zmienianą tabelą.
- condition - poprawny warunek. Warunek musi zwracać wartość TRUE lub FALSE. Warunki będą opisane w dalszej części wykładu

Opis:

Klauzula SET określa, które kolumny zostaną zmienione i jakie nowe wartości mają być w nich zapisane. Klauzula WHERE określa warunki jakie muszą spełniać wiersze, w których należy wymienić wartości podanych wcześniej kolumn. Jeśli klauzula WHERE nie jest podana, to zmieniane są wszystkie wiersze w tabeli.

Rozkaz UPDATE dla każdego wiersza, który spełnia warunki klauzuli WHERE oblicza wartości wyrażeń znajdujących się po prawej stronie operatora '=' i przypisuje te wartości do pola określanego przez nazwę kolumny z lewej strony.

Jeśli klauzula SET posiada podzapytanie, to musi ono zwrócić dokładnie jeden wiersz dla każdego ze zmienianych wierszy. Każda wartość jest przypisywana zgodnie z kolejnością na liście kolumn. Jeśli zapytanie (w przypadku klauzuli postaci SET value = query) nie zwróci wierszy to odpowiednie pola są ustawiane na NULL.

Zapytanie może odwoływać się do zmienianej tabeli. Jest ono obliczane oddzielnie dla każdego zmienianego wiersza a nie dla całego rozkazu UPDATE.

Przykłady:

```
UPDATE pracownicy
  SET nr_wydziału = 4
  WHERE nr_wydziału = 3
;
```

3.3.9. Rozkaz RENAME

Rozkaz RENAME zmienia nazwę tabeli, widoku lub synonimu. Zmiany może dokonać właściciel tabeli, widoku lub synonimu.

Składnia:

```
RENAME old TO new
```

Parametry:

old - aktualna nazwa tabeli, widoku lub synonimu

new - żądana nazwa tabeli, widoku lub synonimu

Opis:

Wszystkie pozwolenia, które posiadał obiekt o starej nazwie, przechodzą na obiekt o nowej nazwie. Za pomocą tego rozkazu nie można zmieniać nazw kolumn. Zmiana nazwy kolumny może być dokonana za pomocą trzech rozkazów: CREATE TABLE, DROP TABLE i RENAME w następujący sposób:

```
CREATE TABLE temporary (new_column_name)
  AS SELECT old_column_name FROM table ;
DROP TABLE table ;
RENAME temporary TO table ;
```

Przykłady:

```
RENAME wydzialy TO jednostki ;
```

3.3.10. Rozkaz ALTER TABLE

Rozkaz służący do zmieniania tabeli. Wykonuje następujące operacje:

- dodaje kolumny i warunki
- modyfikuje definicje kolumn jak typy i warunki
- usuwa warunki
- modyfikuje przyszły sposób alokacji przestrzeni
- zapisuje, że operacja BACKUP została wykonana dla tej tabeli.

Aby wykonać tę operację trzeba być właścicielem tabeli, mieć uprawnienia ALTER dla tabeli lub być administratorem (DBA).

Składnia:

```
ALTER TABLE [user.]table
  [ADD ( {column_element | table_constraint}
    [, {column_element | table_constraint}] ... ) ]
  [MODIFY (column_element [,column_element] ...)]
  [DROP CONSTRAINT constraint] ...
  [PCTFREE integer] [PCTUSED integer]
  [INITTRANS integer] [MAXTRANS integer]
  [STORAGE storage]
  [BACKUP]
```

Parametry:

- [user.]table - właściciel i tabela, którą trzeba zmienić. Jeśli nazwa użytkownika nie jest podana, to domyślnie przyjmowany jest użytkownik, który wywołał rozkaz.

- **ADD/MODIFY column_element** - dodaje lub modyfikuje definicję kolumny, ograniczenia kolumny lub wartości domyślne określonej kolumny.
- **ADD table_constraint** - dodaje ograniczenia na wartości w tabeli.
- **DROP constraint** - usuwa podaną kolumnę lub ograniczenie.
- **BACKUP** - zmienia zawartość słownika danych (Data Dictionary) tak, że zostaje zapisana informacja o wykonaniu backup'u tabeli, który nastąpił w czasie wykonywania rozkazu **ALTER TABLE**.

Opis:

Jeśli użyta zostanie klauzula **ADD** w celu dodania nowej kolumny do istniejącej tabeli, to wartość każdego pola w tej kolumnie będzie równa **NULL**. W związku z tym możliwe jest dodanie kolumny z warunkiem **NOT NULL** tylko do kolumn, które nie mają wierszy.

Klauzula **MODIFY** może zostać użyta do zmiany następujących atrybutów kolumny:

- rozmiar
- typ danych
- **NOT NULL**

Zmiana typu lub zmniejszenie rozmiaru możliwe jest tylko wtedy, gdy wszystkie wartości w kolumnie są równe **NULL**. Możliwe jest nałożenie ograniczenia **NOT NULL** na istniejącą kolumnę tylko wtedy, gdy nie zawiera ona wartości pustych. Jeśli zmieniany jest rozmiar kolumny zadeklarowanej jako **NOT NULL** i w klauzuli **MODIFY** nie poda się **NOT NULL**, to kolumna nadal pozostaje z warunkiem **NOT NULL**.

W przypadku widoków z zapytaniem wybierającym wszystkie kolumny tabeli (**SELECT * FROM ...**), widok może nie pracować poprawnie, jeśli do tabeli, z którą jest powiązany została dodana nowa kolumna.

Przykłady:

```
ALTER TABLE pracownicy  
  ADD (placa NUMBER(7, 2))  
;
```

```
ALTER TABLE pracownicy  
  MODIFY (placa NUMBER(9, 2))  
;
```

3.3.11. Rozkaz **CREATE INDEX**

Rozkaz tworzy nowy indeks dla tabeli lub klastra. Indeks zapewnia bezpośredni dostęp do wierszy w tabeli w celu zredukowania czasu wykonywania operacji. Indeks zawiera informację o każdej wartości, która jest zapisana w indeksowanej kolumnie. Indeks może utworzyć właściciel tabeli, użytkownik posiadający uprawnienia **INDEX** dla danej tabeli lub administrator (**DBA**).

Składnia:

```
CREATE [UNIQUE] INDEX index ON  
{table(column [ASC|DESC][, column [ASC|DESC]]...)} |
```

```
CLUSTER cluster}  
[INITTRANS n] [MAXTRANS n]  
[TABLESPACE tablespace]  
[STORAGE storage]  
[PCTFREE n]  
[NOSORT]
```

Parametry:

- **UNIQUE** - zakłada, że tabela nie ma nigdzie dwóch wierszy zawierających te same wartości we wszystkich indeksowanych kolumnach. W aktualnej wersji ORACLE'a jeśli indeks typu UNIQUE nie zostanie utworzony dla tabeli, to tabela może zawierać powtarzające się wiersze.
- **indeks** - nazwa tworzonego indeksu. Nazwa ta musi być inna od każdego innego obiektu bazy danych danego użytkownika.
- **table** - nazwa istniejącej tabeli, dla której tworzy się indeks.
- **column** - nazwa kolumny w tabeli.
- **ASC, DESC** - zostały dodane w systemie ORACLE w celu zapewnienia kompatybilności z systemem DB2, ale zawsze są tworzone w porządku rosnącym.
- **CLUSTER cluster** - określa klaster, dla którego tworzony jest indeks
- **NOSORT** - wskazuje ORACLE'owi, że wiersze przechowywane w bazie są już posortowane, w związku z czym nie jest konieczne sortowanie podczas tworzenia indeksu.

Opis:

Indeksy są tworzone w celu przyspieszenia operacji:

- dostępu do danych w posortowanych według kolumn indeksowanych
- wyszukiwania wierszy, zawierających dane z indeksowanych kolumn.

Należy jednak zwrócić uwagę, że indeks spowalnia wstawianie, usuwanie i zmiany wartości w indeksowanych kolumnach, ponieważ jego zawartość musi ulec zmianie w momencie zmiany zawartości tabeli.

Do jednego indeksu wstawionych może być co najwyżej 16 kolumn. Jeden element indeksu jest konkatenacją wartości tych kolumn w poszczególnych wierszach. W momencie wyszukiwania może być użyty cały element indeksu lub pewna jego część początkowa. Dlatego kolejność kolumn w indeksie jest ważna. Jeśli więc indeks zostanie utworzony na podstawie trzech kolumn A, B, C w takiej kolejności, to zostanie on użyty do wyszukiwania konkatenacji kolumn A, B, C, kolumn A i B lub tylko kolumny A. Nie będzie natomiast używany w przypadku wyszukiwania połączenia kolumn B i C lub pojedynczej kolumny B lub C.

Możliwe jest utworzenie dowolnej ilości indeksów dla jednej lub kilku tabel. Należy jednak pamiętać, że oprócz spowolnienia operacji modyfikacji tabeli, indeksy zajmują również dość dużą ilość miejsca na dysku.

Przykłady:

```
CREATE INDEX i_prac_imie ON pracownicy (imie) ;
```

3.3.12. Rozkaz CREATE VIEW

Rozkaz służący do tworzenia widoku, czyli logicznej tabeli bazującej na jednej lub wielu tabelach. Utworzyć widok może właściciel tabel, użytkownik posiadający do nich co najmniej uprawnienia SELECT lub administrator.

Składnia:

```
CREATE VIEW [user.]view [(alias [, alias] ...)]  
    AS query  
    [ WITH CHECK OPTION [CONSTRAINT constraint] ]
```

Parametry:

- user - właściciel tworzonego widoku
- view - nazwa tworzonego widoku
- query - identyfikuje kolumny i wiersze tabel, na których bazuje widok. Zapytanie może być dowolnym poprawnym rozkazem SELECT nie zawierającym kluzul ORDER BY ani FOR UPDATE.
- WITH CHECK OPTION - informuje, że wstawienia i zmiany wykonywane poprzez widok, są niedozwolone jeśli spowodują wygenerowanie wierszy, które będą niedostępne dla widoku. Klauzula WITH CHECK OPTION może być użyta w widoku bazującym na innym widoku.
- CONSTRAINT - nazwa dołączona do warunku WITH CHECK OPTION.

Opis:

Widok jest logicznym oknem dla jednej lub kilku tabel. Widok ma następujące właściwości:

- widok nie przechowuje danych - jest on przeznaczony do pokazywania danych zawartych w innych tabelach.
- widok może być użyty w rozkazie SQL w dowolnym miejscu, w którym możliwe jest użycie tabeli z zastrzeżeniem, że można wykonywać selekcję z widoku tylko wtedy, gdy zapytanie na którym bazuje widok zawiera:
 - ♦ połączenie
 - ♦ klauzule GROUP BY, CONNECT BY lub START WITH
 - ♦ klauzulę DISTINCT, pseudokolumny lub wyrażenia na liście kolumn

Możliwa jest zmiana danych zawartych w widoku, który posiada pseudokolumny lub wyrażenia dotąd dopóki rozkaz UPDATE nie odwołuje się do pseudokolumny lub wyrażenia.

Widoki są używane do:

- utworzenia dodatkowego poziomu zabezpieczenia tabeli poprzez ograniczenie dostępu do określonych kolumn lub wierszy tabeli bazowej
- ukrycia złożoności danych - na przykład widok może być użyty do operacji na wielu tabelach tak, by wydawało się, że operacje wykonywane są na jednej tabeli.

- pokazywania danych z innej perspektywy - dla przykładu widok może zostać użyty do zmiany nazwy kolumny bez zmiany rzeczywistych danych zapisanych w tabeli.
- zapewnienia poziomu integralności.

Przykłady:

```
CREATE VIEW bibl
AS SELECT ksiazki.tytul, autorzy.imie,
        autorzy.nazwisko, miejsca.miejsce
FROM ksiazki, autorzy, miejsca
WHERE ksiazki.autor = autorzy.autor_nr
AND ksiazki.miejsce = miejsca.miejsce_nr
WITH CHECK OPTION CONSTRAINT chkopt
;
```

3.3.13. Rozkaz COMMIT

Składnia:

```
COMMIT [WORK]
```

Opis:

Rozkaz COMMIT i COMMIT WORK wykonują tę samą operację polegającą na zakończeniu aktualnej transakcji i stałym zapisaniu wszystkich dokonanych zmian w bazie danych.

3.3.14. Rozkaz ROLLBACK

Składnia:

```
ROLLBACK [ WORK ] [TO [ SAVEPOINT ] savepoint ]
```

Parametry:

- WORK - opcjonalne, wprowadzone tylko dla kompatybilności ze standardem ANSI
- SAVEPOINT - opcjonalne, nie zmienia działania rozkazu ROLLBACK
- savepoint - nazwa punktu zaznaczonego podczas wykonywania aktualnej transakcji.

Opis:

Rozkaz ROLLBACK wycofuje wszystkie zmiany aż do podanego punktu (w przypadku klauzuli TO) lub początku transakcji (bez klauzuli TO).

Przykłady:

```
ROLLBACK ;
```

```
ROLLBACK TO SAVEPOINT SP5 ;
```

3.3.15. Rozkaz SAVEPOINT

Składnia:

```
SAVEPOINT savepoint
```

Parametry:

- savepoint - nazwa punktu w aktualnej transakcji zaznaczanego przez wykonywany rozkaz

Opis:

Rozkaz SAVEPOINT jest używany w połączeniu z ROLLBACK do wycofywania fragmentów wykonywanej transakcji. Nazwy punktów muszą być unikalne w jednej transakcji. Systemy zarządzania bazami danych wprowadzają najczęściej ograniczenia na liczbę punktów, które można zaznaczyć w jednej transakcji.

Przykłady:

```
UPDATE pracownicy
  SET placa_podstawowa = 2000
  WHERE nazwisko = 'Kowalski'
;
SAVEPOINT Kow_plac;
```

```
UPDATE pracownicy
  SET placa_podstawowa = 1500
  WHERE nazwisko = 'Nowak'
;
SAVEPOINT Now_plac;
```

```
SELECT SUM(placa_podstawowa) FROM pracownicy;
ROLLBACK TO SAVEPOINT Kow_plac;
```

```
UPDATE pracownicy
  SET placa_podstawowa = 1300
  WHERE nazwisko = 'Nowak'
;
```

```
COMMIT;
```

3.3.16. Rozkaz SET TRANSACTION

Składnia:

```
SET TRANSACTION { READ ONLY }
```

Parametry:

- READ ONLY - klauzula, która musi wystąpić

Opis:

Rozkaz informuje system, że wykonywana transakcja będzie składać się tylko z zapytań. Nie jest możliwe używanie w takiej transakcji rozkazów INSERT, UPDATE lub DELETE. Rozkaz SET TRANSACTION musi wystąpić jako pierwszy w transakcji, w przeciwnym razie zgłoszony zostanie błąd.

3.4. Operacje relacyjne

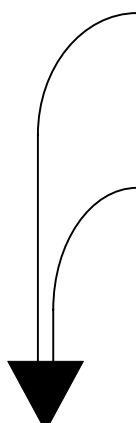
Operacje algebry relacji działają na jednej lub wielu relacjach, a ich wynikiem są inne relacje. Wyróżnia się następujące operacje relacyjne:

- selekcja - umożliwiająca wybór krotek (wierszy) relacji spełniających określone warunki;
- projekcja - umożliwiająca okrojenie relacji do wybranych atrybutów (kolumn)
- połączenie - umożliwiająca łączenie krotek (wierszy) wielu relacji
- operacje teorii mnogości, jak suma mnogościowa, produkt kartezjański, itp.

3.4.1. Selekcja

Operacja selekcji umożliwia pobranie krotek (wierszy) spełniających określony warunek. Operacja ta nazywana jest również **podzbiorem poziomym**.

Pesel	Imię	Nazwisko	Wykształcenie
72030403987	Małgorzata	Albinos	WT
65081002987	Damian	Jędrzejek	SO
44101202034	Barbara	Bibicka	P
70010101231	Piotr	Burzyński	SO
55121201223	Mateusz	Manicki	ST



W języku SQL wykonanie selekcji umożliwia rozkaz SELECT z klauzulą WHERE. Przykładowo polecenie:

```
SELECT * FROM osoby;
```

spowoduje wybranie wszystkich krotek (wierszy) z relacji (tabeli) ludzie.

W celu pobrania wierszy, dla których pole w kolumnie 'Wykształcenie' jest równe 'SO' (średnie ogólne) należy napisać:

```
SELECT * FROM osoby
WHERE Wykształcenie = 'SO'
;
```

Warunki selekcji mogą być złożone. Przykładowo, aby wybrać wszystkie osoby, które mają wykształcenie średnie (średnie techniczne - ST lub średnie ogólne - SO) można odpowiednie warunki połączyć spójnikiem logicznym OR, czyli zapisać w następujący sposób:

```
SELECT * FROM osoby
WHERE Wykształcenie = 'ST' OR Wykształcenie = 'SO'
;
```

Budowa wyrażeń i warunków zostanie opisana dokładniej w dalszej części wykładu.

3.4.2. Projekcja

Projekcja umożliwia pobranie wartości wybranych atrybutów, wymienionych po słowie kluczowym SELECT z wszystkich krotek relacji. Operacja ta jest nazywana także **podzbiorem pionowym**.

Pesel	Imię	Nazwisko	Wykształcenie
72030403987	Małgorzata	Albinos	WT
65081002987	Damian	Jędrzejek	SO
44101202034	Barbara	Bibicka	P
70010101231	Piotr	Burzyński	SO
55121201223	Mateusz	Manicki	ST

Wyrażenia i funkcje

72030403987	WT
65081002987	SO
44101202034	P
70010101231	SO
55121201223	ST

Przykładową operację projekcji pokazaną na rysunku można wykonać za pomocą następującego rozkazu SELECT:

```
SELECT Pesel, Wykształcenie FROM osoby ;
```

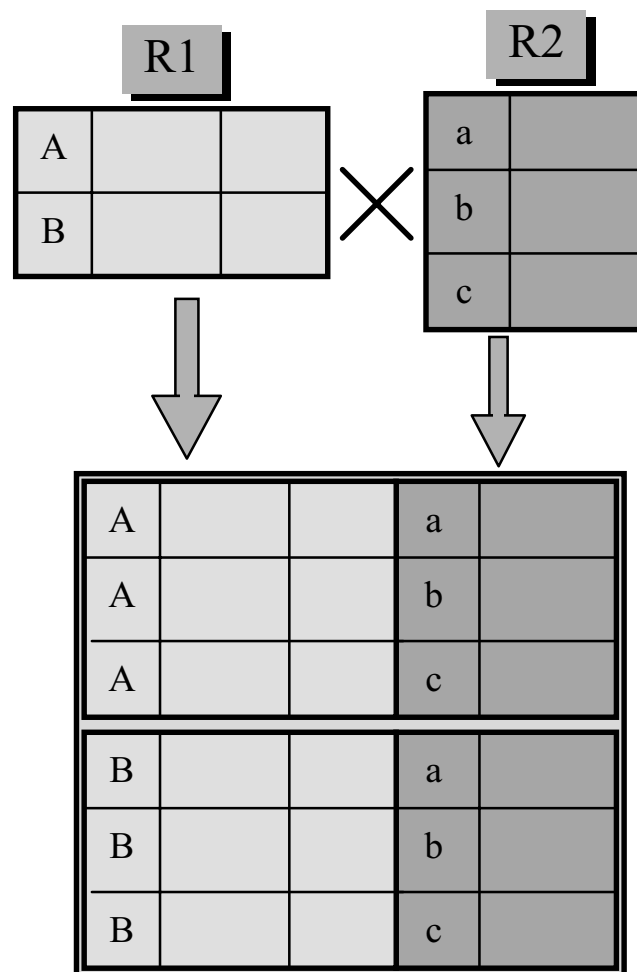
Operacje selekcji i projekcji mogą być łączone w jednym rozkazie SELECT. I tak chcąc otrzymać kolumny zawierające Pesel i Nazwisko osób mających średnie wykształcenie należy napisać:

```
SELECT Pesel, Nazwisko FROM osoby  
WHERE Wykształcenie = 'ST' OR Wykształcenie = 'SO'
```

;

3.4.3. Produkt

Produkt (iloczyn kartezjański) jest operacją teorii zbiorów. Operacja ta umożliwia łączenie dwóch lub więcej relacji w taki sposób, że każda krotka pierwszej relacji, jest łączona z każdą krotką drugiej relacji. W przypadku większej ilości relacji, operacja ta jest wykonywana, na pierwszych dwóch, a następnie na otrzymanym wyniku i relacji następnej, aż do wyczerpania wszystkich argumentów. Przykładowe wykonanie iloczynu kartezjańskiego przedstawia rysunek.



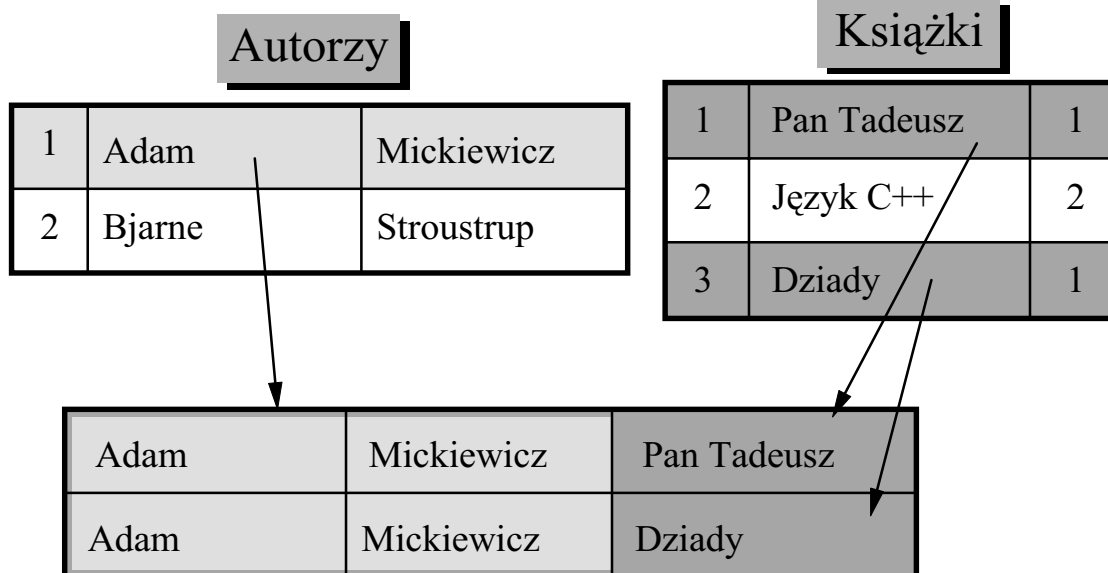
Znajdowanie iloczynu kartezjańskiego dwóch relacji (tabel) jest również wykonywane przez rozkaz SELECT. Przedstawioną na rysunku operację można wykonać za pomocą następującego rozkazu:

```
SELECT * FROM R1, R2;
```

Operacja znajdowania iloczynu kartezjańskiego może być łączona zarówno z operacją selekcji, jak również projekcji lub oboma równocześnie.

3.4.4. Połączenie

Operacja ta polega na łączeniu krotek dwóch lub więcej relacji z zastosowaniem określonego warunku łączenia. Wynikiem połączenia jest podzbiór produktu relacji.



Operację pokazaną na rysunku można wykonać następującym poleceniem SELECT.

```
SELECT imie, nazwisko, tytul
FROM autorzy, ksiazki
WHERE autorzy.nazwisko = 'Mickiewicz' and
      autorzy.nr = ksiazki.autor
;
```

3.4.5. Operacje mnogościowe

Operacje mnogościowe są operacjami teorii zbiorów. W języku SQL operacje te możemy stosować do relacji (tabel), zarówno istniejących w systemie, jak również będących wynikiem działania innych rozkazów.

3.4.5.1. Unia

Unia pozwala na zsumowanie zbiorów krotek dwóch lub więcej relacji (bez powtórzeń - zgodnie z teorią mnogości). Warunkiem poprawności tej operacji jest zgodność liczby i typów atrybutów (kolumn) sumowanych relacji. Przykład przedstawiony poniżej sumuje zbiory pracowników i właścicieli okrojone do imienia i nazwiska (za pomocą projekcji), w celu uzyskania informacji o wszystkich ludziach powiązanych z firmą:

```
SELECT imie, nazwisko FROM pracownicy
UNION
SELECT imie, nazwisko FROM wlasciciele ;
```

3.4.5.2. Przekrój

Przekrój pozwala znaleźć iloczyn dwóch lub więcej zbiorów krotek tzn. takich, które występują zarówno w jednej jak i w drugiej relacji. Podobnie jak w przypadku unii, warunkiem poprawności tej operacji jest zgodność liczby i typów atrybutów relacji bazowych.

Poniższy przykład znajduje wszystkie nazwiska (np. stosunek pracy, powiązania rodzinne), które występują zarówno w relacji pracownicy jak i w relacji właściciele.

```
SELECT nazwisko FROM pracownicy  
INTERSECT  
SELECT nazwisko FROM wlascciele ;
```

3.4.5.3. Różnica

Operacja obliczania różnicy dwóch relacji polega na znalezieniu wszystkich krotek, które występują w pierwszej relacji, ale nie występują w drugiej.

Przykład znajduje wszystkie osoby, które są współwłaścicielami spółki, ale nie są w niej zatrudnieni:

```
SELECT imie, nazwisko FROM wlascciele  
MINUS  
SELECT imie, nazwisko FROM pracownicy ;
```

3.4.6. Grupowanie

Klauzule GROUP BY i HAVING występujące w rozkazie SELECT pozwalają dzielić relację wynikową na grupy, wybierać niektóre z tych grup i na każdej z nich z osobna wykonywać pewne (dozwolone przez system) operacje. Operacje te działają na wszystkich wierszach wchodzących w skład grupy. Na samym końcu zwracana jest tylko zbiorcza informacja o wybranych grupach (nie zwraca się wszystkich wierszy wchodzących w skład grupy).

Klauzula GROUP BY służy do dzielenia krotek relacji na mniejsze grupy. Sposób takiego podziału ilustruje przykład:

```
SELECT stanowisko, avg(placa_podstawowa)  
FROM pracownicy  
GROUP BY stanowisko ;
```

Istnieje możliwość odrzucenia pewnych krotek przed podziałem na grupy. Dokonuje się tego za pomocą klauzuli WHERE:

```
SELECT stanowisko, avg(placa_podstawowa)  
FROM pracownicy  
WHERE stanowisko != 'KIEROWCA'  
GROUP BY stanowisko ;
```

Dzielenie na grupy może być zagnieżdżane, co umożliwia wydzielanie podgrup w uprzednio znalezionych podgrupach. W przykładzie poniżej wszyscy pracownicy są dzieleni na wydziały, w których pracują, a w ramach każdego wydziału grupowani według stanowiska:

```
SELECT wydzial, stanowisko, avg(placa_podstawowa)  
FROM pracownicy  
GROUP BY nr_wydzialu, stanowisko ;
```

Klauzula HAVING ogranicza wyświetlanie grup do tych, które spełniają określony warunek. Chcąc wyświetlić tylko te grupy, w których płaca podstawowa przynajmniej jednego pracownika jest większa niż 3 000 należy zastosować następujące zapytanie:

```
SELECT stanowisko, max(placa_podstawowa)  
FROM pracownicy
```

```
GROUP BY stanowisko  
HAVING max(placa_podstawowa) > 3000 ;
```

3.4.7. Kolejność klauzul w rozkazie SELECT

Klauzule mające wpływ na realizację rozkazu SELECT uwzględniane są w następującej kolejności:

1. SELECT i WHERE
2. GROUP BY
3. HAVING
4. ORDER BY

3.5. Podzapytania

Zapytania języka SQL mogą być zagnieżdżane, tzn. wynik jednego zapytania może być użyty np. jako warunek selekcji innego zapytania. Podzapytania można podzielić na dwa rodzaje:

- podzapytania proste (nazywane po prostu podzapytaniem) - podzapytanie jest wykonywane **przed** wykonaniem zapytania głównego;
- podzapytania skorelowane - podzapytanie jest wykonywane dla każdej krotki podzapytania głównego.

Najczęściej podzapytania używane są w klauzuli WHERE rozkazu SELECT. Jeśli wiadomo, że wynikiem podzapytania będzie pojedyncza wartość, to wartość tą można użyć bezpośrednio w warunku klauzuli WHERE w następujący sposób:

```
SELECT nazwisko  
FROM pracownicy  
WHERE placa_podstawowa =  
      (SELECT min(placa_podstawowa)  
        FROM pracownicy)  
;
```

Powyższy przykład znajduje nazwiska pracowników zarabiających najmniej.

Jeśli jednak w wyniku podzapytania będzie kilka wartości, to konieczne jest użycie operatora IN w zapytaniu głównym. Na przykład, w celu znalezienia pracowników zarabiających najmniej w swoich grupach, należy posłużyć się następującym rozkazem:

```
SELECT nazwisko, nr_wydzialu  
FROM pracownicy  
WHERE placa_podstawowa IN  
      (SELECT min(placa_podstawowa)  
        FROM pracownicy  
        GROUP BY nr_wydzialu)  
;
```

Liczba wartości i ich typy muszą być zgodne z listą znajdującą się po prawej stronie operatora IN.

W podzapytaniach stosować można również operatory ANY i ALL. Operator ANY pozwala sprawdzić, czy chociaż jeden element z listy spełnia podany warunek. Operator ALL umożliwia sprawdzenie czy warunek jest spełniony dla wszystkich elementów listy. Składnia operatorów ANY i ALL oraz przykłady użycia zostaną podane w dalszej części wykładu.

Podzapytania można umieszczać również w klauzuli HAVING. Poniższy przykład wyświetla zespoły, w których średnie zarobki są większe niż w zespole 30:

```
SELECT nr_wydzialu, avg(placa_podstawowa)
FROM pracownicy
GROUP BY nr_wydzialu
HAVING avg(placa_podstawowa) >
      (SELECT avg(placa_podstawowa)
       FROM pracownicy
       WHERE nr_wydzialu = 30)
;
```

Zapytania mogą być zagnieżdżane na dowolną ilość poziomów. Przy zagnieżdżaniu zapytań należy jednak pamiętać o:

- w podzapytaniach nie może występować klauzula ORDER BY (może ona wystąpić jako ostatnia tylko w zapytaniu głównym);
- zapytania zagnieżdżone są wykonywane w kolejności od najbardziej zagnieżdżonego do zapytania głównego, chyba, że mamy do czynienia z podzapytaniem skorelowanym.

Podzapytanie skorelowane jest zapytaniem zagnieżdżonym, które jest wykonywane dla **każdej** krotki analizowanej przez zapytanie zewnętrzne. Podstawowa własność podzapytania skorelowanego polega na tym, że operuje ono na informacji przekazanej przez zapytanie główne.

Następujący przykład pokazuje użycie podzapytania skorelowanego w celu znalezienia pracowników zarabiających więcej niż wynosi średnia płaca w ich działach:

```
SELECT imie, nazwisko, placa_podstawowa, nr_wydzialu
FROM pracownicy pracownik
WHERE placa_podstawowa >
      (SELECT avg(placa_podstawowa)
       FROM pracownicy
       WHERE nr_wydzialu = pracownik.nr_wydzialu)
;
```

W podzapytaniach skorelowanych, w przeciwieństwie do podzapytań prostych, występuje odwołanie do atrybutu krotki analizowanej aktualnie przez zapytanie zewnętrzne. Jeśli więc zapytania skorelowane operują na tej samej relacji, to konieczne jest użycie aliasu w celu odwołania się do atrybutu krotki analizowanej przez zapytanie główne (w tym wypadku jest to realizowane za pomocą aliasu „pracownik”).

3.6. Widoki (perspektywy)

Widoki są traktowane przez system zarządzania bazą danych podobnie jak tabele m. in. posiadają kolumny i wiersze służące do przechowywania informacji. Widok nie posiada jednak własnych danych. Wszystkie dane, udostępniane przez widok są danymi zawartymi w jednej lub kilku tabelach (albo widokach). Widoki stosuje się w celu:

- ograniczenia dostępu do tabel w bazie danych
- uproszczenia zapytań kierowanych do systemu
- zapewnienia niezależności danych wewnątrz aplikacji

Ze względu na ilość tabel, na których zdefiniowany został widok, widoki można podzielić na:

- proste
- złożone

Widok definiuje się przy pomocy polecenia `SELECT`, które jest zapamiętywane przez system i wykonywane automatycznie w momencie otrzymania żądania dostępu do danych zawartych w widoku.

Widok prosty udostępnia dane tylko z jednej tabeli i w jego definicji nie stosuje się poleceń języka SQL ani też grupowania wierszy.

Widok złożony udostępnia dane zawarte w kilku tabelach i może zawierać operacje relacyjne oraz grupowanie wierszy. Ceną płaconą za możliwość zdefiniowania widoku złożonego jest najczęściej brak możliwości zapisu danych w tym widoku (ale nie we wszystkich systemach).

Do tworzenia widoków służy opisany wcześniej rozkaz `CREATE VIEW`. Poniższy przykład pokazuje tworzenie widoku na bazie tabeli `pracownicy` udostępniającego tylko imię i nazwisko pracownika (bez możliwości dostępu do płacy czy numeru wydziału):

```
CREATE VIEW personalia
AS
  SELECT imie, nazwisko
     FROM pracownicy
;
```

Tworzenie widoku złożonego przedstawia przykład podobny do znajdującego się w opisie rozkazu `CREATE VIEW`:

```
CREATE VIEW bibl
AS SELECT ksiazki.tytul, autorzy.imie,
        autorzy.nazwisko, miejsca.miejsce
   FROM ksiazki, autorzy, miejsca
  WHERE ksiazki.autor = autorzy.autor_nr
        AND ksiazki.miejsce = miejsca.miejsce_nr
;
```

3.7. Transakcje

Transakcje są wykonywane za pomocą dwóch rozkazów: COMMIT i ROLLBACK. Pierwszy z tych rozkazów jest używany do zapisywania w bazie wszystkich dokonanych zmian, drugi do wycofywania zmian uprzednio wprowadzonych. Przykładem transakcji jest dokonywanie przelewu pomiędzy jednym bankiem a drugim. Operacja ta wymaga wykonania dwóch rozkazów UPDATE - zmniejszenia stanu konta w banku dokonującym przelewu i zwiększenia odpowiedniego stanu konta w banku otrzymującym przelew. Z charakteru operacji przelewu wynika, że muszą być wykonane oba rozkazy albo żaden. Jeśli bowiem wykonano by tylko jeden z nich to pieniądze albo by „znikły” albo się „rozmnóżyły”.

Rozkaz COMMIT służy do zapisywania na stałe wykonanych uprzednio operacji. Przed wykonaniem tego rozkazu żadne zmiany w bazie danych nie są widoczne dla innych użytkowników. W podanym uprzednio przykładzie po wykonaniu dwóch rozkazów UPDATE, należy wykonać COMMIT w celu trwałego zapisania dokonanych zmian. Od tego momentu zmienione stany kont będą widoczne dla innych użytkowników.

Rozkaz ROLLBACK jest odwrotnością COMMIT. Jest to również rozkaz kończący transakcję, jednak powoduje wycofanie wszystkich zmian w bazie od poprzedniego rozkazu COMMIT lub ROLLBACK.

Rozkaz SAVEPOINT pozwala na zaznaczenie i nazwanie pewnego punktu wewnątrz transakcji. W ten sposób za pomocą rozkazu ROLLBACK TO możliwe jest częściowe wycofanie transakcji (tzn. do podanego punktu wewnątrz aktualnie wykonywanej transakcji). Wykonanie rozkazu ROLLBACK TO do określonego punktu powoduje skasowanie wszystkich zaznaczeń poniżej. Wykonanie ROLLBACK lub COMMIT powoduje skasowanie wszystkich uprzednio zaznaczonych punktów.

Jest dobrym zwyczajem, by każdą transakcję kończyć rozkazem COMMIT lub ROLLBACK. Jeśli nie jest to zrobione, a kończy się skrypt lub blok, to system sam podejmuje decyzję, czy rozpoczętą transakcję wykonać czy wycofać. Może to w pewnych przypadkach prowadzić do rezultatów, które nie były oczekiwane przez osobę tworzącą skrypt.

3.8. Normalizacja relacji

3.8.1. Cele normalizacji

Normalizacja relacji ma na celu takie jej przekształcenie, by nie posiadała ona cech niepożądanych. Jako główną cechę niepożądaną wymienić należy redundancję (powtarzanie się) danych. Jeśli system bazy danych wymaga kilkakrotnego wprowadzania tej samej informacji (np. tego samego imienia i nazwiska), to prawie na pewno jest źle zaprojektowany (informację powinno wprowadzać się raz a następnie wybierać ją z dostępnego w systemie menu lub tabeli). Inne cechy niepożądane związane są z możliwościami operowania na bazie danych (np. wyszukiwania wg zadanych warunków). Weźmy dla przykładu relację opisującą zajęcia odbywające się na uczelni w jednym semestrze. Relacja ta może zawierać następujące atrybuty:

- nazwa przedmiotu,
- imię,
- nazwisko,
- adres prowadzącego

Przykładowa krotka takiej relacji mogłaby mieć postać:

(język angielski, Lucyna, Nowak,
ul. Królewska 30/3 Kraków)

Jednak z tak zaprojektowaną relacją związane są następujące problemy:

- adres składający się z kilku części nie został podzielony w związku z tym nie jest możliwe sprawdzenie ile osób mieszka w Krakowie i nie potrzebuje hotelu;
- jeden prowadzący może mieć zajęcia z kilku przedmiotów, w związku z tym występuje redundancja danych;
- zmiana jednej z informacji o prowadzącym (np. adresu) powoduje konieczność zmiany wszystkich krotek zawierających te dane w celu zachowania integralności;
- nie jest możliwe wprowadzenie informacji o prowadzącym, który w aktualnym semestrze nie ma żadnych zajęć;
- usunięcie przedmiotu może spowodować również usunięcie wszelkich informacji o osobie, która go prowadziła.

Utrzymanie integralności takiej bazy nie jest więc proste. Jednak opisaną relację można zamienić na dwie inne, które nie będą posiadały tych wad:

- relacja Prowadzący:

identyfikator, imię, nazwisko, kod pocztowy, miejscowość,
ulica, nr_domu, nr_mieszkania

- relacja Zajęcia

nazwa, id_prowadzącego

Każda krotka relacji „Zajęcia”, jest powiązana z krotką relacji „Prowadzący” za pomocą klucza obcego o nazwie „id_prowadzącego”. Te dwie relacje nie posiadają opisanych wcześniej cech niepożądanych ponieważ:

- adres jest zdekomponowany na części składowe, w związku z czym możliwe jest wyszukiwanie danych np. według miejscowości zamieszkania prowadzącego;
- zmiana informacji o prowadzącym (np. adresu) nie powoduje konieczności zmian danych w relacji „Zajęcia”. Zmiana ta odbywa się tylko w jednym miejscu;
- możliwe jest wprowadzenie informacji o osobie, która nie ma zajęć w aktualnym semestrze, ale być może będzie je miała w semestrze następnym;
- usunięcie przedmiotu nie powoduje usunięcia informacji o osobie, która go prowadziła.

Jednak taka reprezentacja danych posiada wady podobne do opisanych wcześniej, ale dotyczące przedmiotów. Dlatego w dobrze zaprojektowanej bazie danych konieczne jest wydzielenie trzeciej tabeli, która będzie zawierała spis przedmiotów.

3.8.2. Pierwsza postać normalna

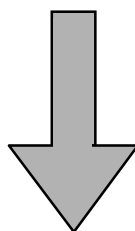
Relacja jest w pierwszej postaci normalnej, jeśli wartości atrybutów są elementarne tzn. są to pojedyncze wartości określonego typu, a nie zbiory wartości.

Pierwsza postać normalna jest konieczna aby, tabelę można było nazwać relacją. Większość systemów baz danych nie ma możliwości zbudowania tabel nie będących w

pierwszej postaci normalnej. Przekształcenie z postaci nie znormalizowanej do pierwszej postaci normalnej ilustruje rysunek:

Zamówienia

Nr zamówienia	Id dostawcy	Nazwa dostawcy	Adres dostawcy	Id części	Nazwa części	Ilość
001	010	Seagate	Borsucza 8	054	Dysk twardy	30
				055	Sterownik I/O	50
002	020	Toshiba	Wilcza 3	070	Napęd CD	10
003	010	Seagate	Borsucza 8	054	Dysk twardy	40
				070	Napęd CD	15



Zamówienia

Nr zamówienia	Id dostawcy	Nazwa dostawcy	Adres dostawcy	Id części	Nazwa części	Ilość
001	010	Seagate	Borsucza 8	054	Dysk twardy	30
001	010	Seagate	Borsucza 8	055	Sterownik I/O	50
002	020	Toshiba	Wilcza 3	070	Napęd CD	10
003	010	Seagate	Borsucza 8	054	Dysk twardy	40
003	010	Seagate	Borsucza 8	070	Napęd CD	15

3.8.3. Definicje pomocnicze

Aby ułatwić przekształcanie relacji do postaci optymalnej wprowadzono pojęcie postaci normalnej. Przed omówieniem procesu normalizacji konieczne jest jednak wprowadzenie kilku pojęć:

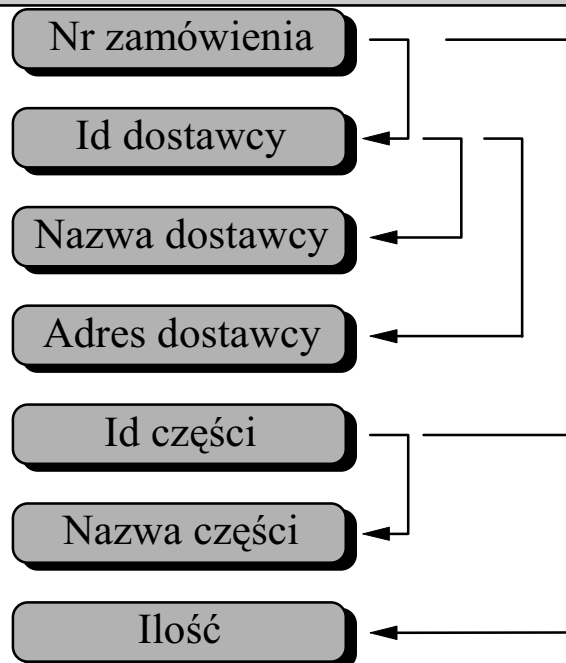
- **Uniwersalny schemat relacji** $R = \{A_1, A_2, \dots, A_n\}$ jest zbiorem atrybutów tworzących relację.
- **Zbiorem identyfikującym relacji** $R = \{A_1, A_2, \dots, A_n\}$ nazywamy zbiór atrybutów $S \subseteq R$, który jednoznacznie identyfikuje wszystkie krotki relacji o schemacie R . Inaczej mówiąc w żadnej relacji o schemacie R nie mogą istnieć dwie krotki t_1 i t_2 takie, że $t_1[S] = t_2[S]$.
- **Kluczem** K schematu relacji R nazywamy minimalny zbiór identyfikujący, tzn. taki, że nie istnieje $K' \subset K$ będące zbiorem identyfikującym schematu R . Klucze dzielą się na klucze proste i złożone.
- Klucz nazywamy **kluczem prostym**, jeżeli zbiór atrybutów wchodzących w jego skład jest zbiorem jednoelementowym; w przeciwnym wypadku mamy do czynienia z **kluczem złożonym**. Najczęściej w relacji można wyróżnić wiele kluczy, które nazywamy **kluczami potencjalnymi**. Jeden (wybrany) klucz spośród kluczy potencjalnych nazywamy **kluczem głównym** (primary key), natomiast pozostałe kluczami drugorzędnymi (secondary key).

Dla przykładu w relacji „Zamówienia” jedynym kluczem potencjalnym jest para atrybutów (Nr zamówienia, Id części). Należy zauważyć, że sam numer zamówienia nie jest kluczem, ponieważ jedno zamówienie może dotyczyć wielu części.

- **Atrybut** relacji nazywamy **podstawowym**, jeżeli należy do dowolnego z kluczy tej relacji.
- **Atrybut** relacji nazywamy **wtórny**, jeżeli nie należy do żadnego z kluczy tej relacji.
- Atrybut B relacji R jest **funkcjonalnie zależny** od atrybutu A tej relacji (co określa się również słowami, że A **identyfikuje** B i oznacza $A \rightarrow B$), jeśli dowolnej wartości a atrybutu A odpowiada nie więcej niż jedna wartość b atrybutu B .

Zależność funkcjonalna między dwoma atrybutami A i B nie jest związana z przypadkowym układem ich wartości, ale wynika z charakteru zależności między tymi atrybutami w modelowanej rzeczywistości.

Informację, że atrybut A identyfikuje B (tzn. że B jest funkcjonalnie zależny od A) zaznacza się na rysunkach strzałką biegnącą od A do B . W ten sposób schemat zależności funkcjonalnych dla przykładowej relacji „Zamówienia” można narysować następująco:



- Atrybut B jest w pełni funkcjonalnie zależny od zbioru atrybutów X w schemacie R, jeżeli $X \rightarrow Y$ i nie istnieje podzbiór $X' \subset X$ taki, że $X' \rightarrow Y$.

Należy zwrócić uwagę, że wszystkie zależności funkcjonalne przedstawione w poprzednim przykładzie są pełnymi zależnościami funkcjonalnymi.

- Zbiór atrybutów Y jest częściowo funkcjonalnie zależny od zbioru atrybutów X w schemacie R, jeżeli $X \rightarrow Y$ i istnieje podzbiór $X' \subset X$ taki, że $X' \rightarrow Y$.
- Niech X, Y i Z będą trzema rozłącznymi podzbiorami atrybutów danej relacji. Podzbiór atrybutów Z jest **przechodnio funkcjonalnie zależny** od podzbioru atrybutów X, jeśli podzbiór atrybutów Z jest funkcjonalnie zależny od podzbioru atrybutów Y i podzbiór atrybutów Y jest funkcjonalnie zależny od podzbioru atrybutów X, natomiast podzbiór atrybutów X nie jest funkcjonalnie zależny od Y i podzbiór atrybutów Y nie jest funkcjonalnie zależny od Z.

Przykładowo w relacji „Zamówienia” atrybuty „Nazwa dostawcy” i „Adres dostawcy” są przechodnio funkcjonalnie zależne od atrybutu „Nr zamówienia”, ponieważ atrybut „Id dostawcy” jest funkcjonalnie zależny od atrybutu „Nr zamówienia” i atrybuty „Nazwa dostawcy” oraz „Adres dostawcy” są funkcjonalnie zależne od „Id dostawcy”. Równocześnie można zauważyć, że „Id dostawcy” nie jest funkcjonalnie zależne od „Nazwy dostawcy” (mogą być dostawcy o tej samej nazwie) ani od „Adresu dostawcy” (ponieważ dwóch dostawców może mieć ten sam adres).

- Mówimy, że podzbiór atrybutów Y jest wielowartościowo funkcjonalnie zależny od podzbioru atrybutów X w schemacie R, jeżeli dla dowolnej relacji r w schemacie R i dla dowolnej pary krotek t_1 i t_2 z relacji r takich, że $t_1[X] = t_2[X]$, istnieje taka para krotek s_1 i s_2 w relacji r, że

$$s_1[X] = s_2[X] = t_1[X] = t_2[X] \text{ oraz}$$

$$s_1[Y] = t_1[Y] \text{ i } s_1[R-X-Y] = t_2[R-X-Y] \text{ oraz}$$

$$s_2[Y] = t_2[Y] \text{ i } s_2[R-X-Y] = t_1[R-X-Y].$$

Mówiąc inaczej, jeżeli dla dowolnej pary krotek t_1 i t_2 z relacji r takich, że wartości tych krotek dla atrybutów z podzbioru X są sobie równe (co zapisujemy $t_1[X] = t_2[X]$), zamienimy w tych krotkach wartości atrybutów z podzbioru Y , to otrzymane w ten sposób krotki s_1 i s_2 również należą do relacji r .

Przykładowo chcąc zapisać, że „Kowalski” ma dzieci o imionach „Agnieszka” i „Magda” oraz, że wykłada przedmioty „Język C” i „Systemy operacyjne”, natomiast „Nowak” ma dzieci „Jarosław”, „Jan” i „Aleksander” oraz wykłada przedmioty „Bazy danych” i „Teoria kompilatorów” można zdefiniować następującą tabelę:

Pracownicy

Nazwisko	Dziecko	Wykład
Kowalski	Agnieszka	Język C
Kowalski	Agnieszka	Systemy operacyjne
Kowalski	Magda	Język C
Kowalski	Magda	Systemy operacyjne
Nowak	Jarosław	Bazy danych
Nowak	Jarosław	Teoria kompilatorów
Nowak	Jan	Bazy danych
Nowak	Jan	Teoria kompilatorów
Nowak	Aleksander	Bazy danych
Nowak	Aleksander	Teoria kompilatorów

Określając podzbiór $X = (\text{Nazwisko})$ i $Y = (\text{Dziecko})$ otrzymujemy, że $R-X-Y = (\text{Wykład})$. Parze krotek:

$t_1 = (\text{Kowalski}, \text{Agnieszka}, \text{Język C})$

$t_2 = (\text{Kowalski}, \text{Magda}, \text{Systemy operacyjne})$

odpowiada para krotek

$s_1 = (\text{Kowalski}, \text{Agnieszka}, \text{Systemy operacyjne})$

$s_2 = (\text{Kowalski}, \text{Magda}, \text{Język C})$

Należy zwrócić uwagę, że wystąpienie krotek spełniających wymienione wyżej warunki, wiąże się najczęściej z niezależnością jednego zbioru atrybutów od drugiego (w tym wypadku jednoelementowego zbioru atrybutów (Dziecko) od jednoelementowego zbioru atrybutów (Wykład)).

- Z definicji wielowartościowej zależności funkcjonalnej wynika, że:
 - ◊ podzbiór pusty jest zawsze wielowartościowo funkcjonalnie zależny od dowolnego zbioru atrybutów X

◇ jeśli $X \cup Y = R$, to X jest wielowartościowo funkcjonalnie zależny od Y i na odwrót. Zależności te nazywamy trywialnymi wielowartościowymi zależnościami funkcjonalnymi.

- Dekompozycją schematu $R = (A_1, A_2, \dots, A_n)$ nazywamy zastąpienie go zbiorem (niekoniecznie rozłącznych) schematów relacji (R_1, R_2, \dots, R_m) takich, że każdy schemat R_i z tego zbioru stanowi podzbiór zbioru atrybutów (A_1, \dots, A_n) i

$$\bigcup_i R_i = R = (A_1, \dots, A_n)$$

- Mówimy, że w schemacie relacji $R = (A_1, \dots, A_n)$ występuje połączeniowa zależność funkcjonalna (co zapisuje się: $*R[R_1, \dots, R_n]$) wtedy i tylko wtedy, gdy możliwa jest dekompozycja relacji r (o schemacie R) na relacje r_1, \dots, r_n taka, że relację pierwotną r można zrekonstruować przez wykonanie sekwencji operacji połączenia relacji r_1, \dots, r_n . Można pokazać, że wielowartościowa zależność funkcjonalna stanowi szczególny przypadek połączeniowej zależności funkcjonalnej dla $m = 2$.
- Mówimy, że połączeniowa zależność funkcjonalna $*R[R_1, \dots, R_m]$ wynika z zależności atrybutów schematu R od klucza wtedy i tylko wtedy, gdy w dowolnej sekwencji połączeń relacji składowych wykonywanych w celu rekonstrukcji relacji r , każda operacja połączenia jest wykonywana względem zbioru identyfikującego schematu R .

3.8.4. Druga postać normalna

Relacja jest w drugiej postaci normalnej, jeżeli każdy atrybut wtórny (tzn. nie wchodzący w skład żadnego klucza potencjalnego) tej relacji jest w pełni funkcjonalnie zależny od wszystkich kluczy potencjalnych tej relacji.

Można zauważyć, że relacja „Zamówienia” nie jest w drugiej postaci normalnej, ponieważ atrybuty „Id dostawcy”, „Nazwa dostawcy”, „Adres dostawcy” i „Nazwa części” nie są w pełni funkcjonalnie zależne od jedyne go klucza potencjalnego - pary („Nr zamówienia”, „Id części”).

W celu sprowadzenia relacji do drugiej postaci normalnej, należy podzielić ją na takie relacje, których wszystkie atrybuty będą w pełni funkcjonalnie zależne od kluczy. W tym celu przykładową relację „Zamówienia” należy podzielić na trzy relacje: „Dostawca na zamówieniu”, „Zamówione dostawy”, „Części” w następujący sposób:

Dostawca na zamówieniu

Nr zamówienia	Id dostawcy	Nazwa dostawcy	Adres dostawcy
001	010	Seagate	Borsucza 8
002	020	Toshiba	Wilcza 3
003	010	Seagate	Borsucza 8

Zamówione dostawy

Nr zamówienia	Id części	Ilość
001	054	30
001	055	50
002	070	10
003	054	40
003	070	15

Części

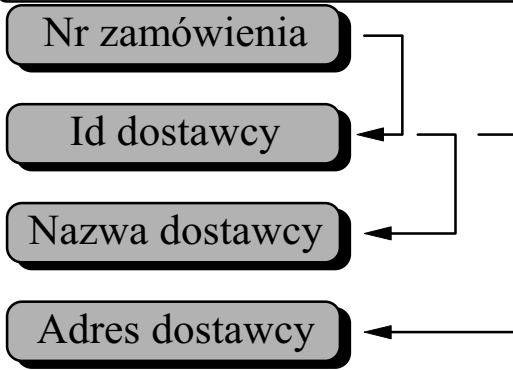
Id części	Nazwa części
054	Dysk twardy
055	Sterownik I/O
070	Napęd CD

Jak widać wszystkie te trzy relacje są w drugiej postaci normalnej, ponieważ klucze relacji „Dostawca na zamówieniu” oraz „Części” są kluczami prostymi, natomiast atrybut „Ilość” w relacji „Zamówione dostawy” jest w pełni funkcjonalnie zależny od klucza złożonego („Nr zamówienia”, „Id części”).

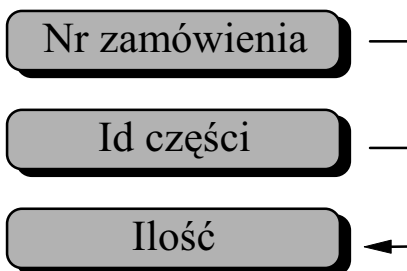
Należy zauważyć, że relacja będąca w pierwszej postaci normalnej, jest równocześnie w drugiej postaci normalnej, jeśli wszystkie jej klucze potencjalne są kluczami prostymi.

Po przekształceniu relacji „Zamówienia” do drugiej postaci normalnej otrzymujemy następujące zależności funkcjonalne:

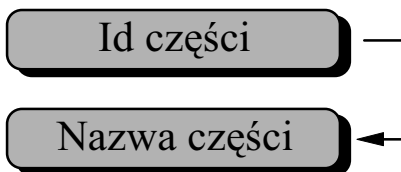
- Dostawca na zamówieniu



- Zamówione dostawy



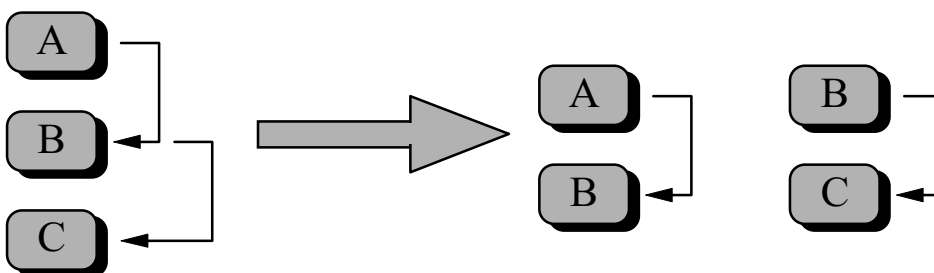
- Części



3.8.5. Trzecia postać normalna

Dana relacja jest w trzeciej postaci normalnej, jeśli jest ona w drugiej postaci normalnej i każdy jej atrybut nie входяcy w skład żadnego klucza potencjalnego nie jest przechodnio funkcjonalnie zależny od żadnego klucza potencjalnego tej relacji.

Aby doprowadzić relację, której atrybuty pozostają w przechodniej zależności funkcjonalnej, należy podzielić ją na relacje zawierające tylko zależność funkcjonalną. Podział relacji ilustruje rysunek:



W opisywanym przykładzie przechodnia zależność funkcjonalna występuje pomiędzy atrybutami „Nazwa dostawcy” i „Adres dostawcy” a atrybutem „Nr

zamówienia” w relacji „Dostawca na zamówieniu”. W związku z tym konieczne jest dokonanie podziału relacji „Dostawca na zamówieniu” na dwie relacje: „Zamówienia” i „Dostawcy” w następujący sposób:

Zamówienia

Nr Id
zamówienia dostawcy

001	010
002	020
003	010

Dostawcy

Id Nazwa
dostawcy dostawcy Adres dostawcy

010	Seagate	Borsucza 8
020	Toshiba	Wilcza 3
030	Sony	Ptasia 15

3.8.6. Czwarta postać normalna

Dana jest relacja o schemacie R oraz trzy parami rozłączne i niepuste podzbiory X , Y , Z atrybutów z R takie, że $X \cup Y \cup Z = R$ i podzbiór Y jest nietrywialnie wielowartościowo zależny od X .

Dana relacja R jest w czwartej postaci normalnej wtedy i tylko wtedy, gdy jest w trzeciej postaci normalnej i wielowartościowa zależność zbioru Y od X pociąga za sobą funkcjonalną zależność wszystkich atrybutów tej relacji od X .

Łatwo zauważyć, że tabela „Pracownicy” z definicji wielowartościowej zależności funkcjonalnej jest w trzeciej postaci normalnej, ale nie jest w czwartej postaci normalnej, ponieważ atrybuty „Dziecko” i „Wykład” nie są funkcjonalnie zależne od atrybutu „Nazwisko”, tzn. w tej relacji nie występuje żadna funkcjonalna zależność pomiędzy atrybutami.

Jak wynika z definicji relacja, która zawiera trywialną wielowartościową zależność funkcjonalną jest w czwartej postaci normalnej. Stąd wniosek, że relację zawierającą nietrywialną wielowartościową zależność funkcjonalną należy podzielić na takie relacje, które będą zawierać tylko zależności trywialne.

W opisywanym przykładzie relację „Pracownicy” można podzielić na dwie relacje: „Dzieci” i „Wykłady”, które będą zawierać tylko trywialną wielowartościową zależność funkcjonalną:

Dzieci

Nazwisko	Dziecko
Kowalski	Agnieszka
Kowalski	Magda
Nowak	Jarosław
Nowak	Jan
Nowak	Aleksander

Wykłady

Nazwisko	Wykład
Kowalski	Język C
Kowalski	Systemy operacyjne
Nowak	Bazy danych
Nowak	Teoria kompilatorów

3.8.7. Piąta postać normalna

Dana relacja r o schemacie R jest w piątej postaci normalnej wtedy i tylko wtedy, gdy jest w czwartej postaci normalnej i w przypadku występowania w niej połączeniowej zależności funkcjonalnej $*R[R_1, \dots, R_m]$ zależność ta wynika z zależności atrybutów od klucza.

Definicja ta mówi, że jeśli relacja posiada klucz i możliwe jest jej podzielenie na na dwie lub więcej relacji w taki sposób by można ją było odtworzyć (odtworzenie musi być jednoznaczne) i którakolwiek z relacji powstałych w wyniku podziału nie zawiera klucza relacji pierwotnej lub odtworzenia można dokonać bez potrzeby użycia klucza, to relacja pierwotna nie jest w piątej postaci normalnej.

Wynika z tego, że w celu doprowadzenia pewnej relacji do piątej postaci normalnej konieczne jest podzielenie jej na takie relacje, które spełniać będą podany wyżej warunek.

3.8.8. Podsumowanie

Przekształcenie relacji do kolejnych postaci normalnych wiąże się najczęściej ze zmniejszeniem ilości pamięci potrzebnej do przechowania informacji.

Proces normalizacji ma na celu takie przekształcenie relacji, by uniknąć dublowania informacji. Unikanie powtórzeń pozwala na łatwiejsze i w wielu przypadkach szybsze posługiwanie się bazą danych. Mechanizmy języków stosowanych w bazach danych pozwalają ukryć złożoność struktury bazy przed użytkownikiem i operować na danych w sposób efektywny zapewniając spójność informacji.

4. Warunki i wyrażenia

Warunki i wyrażenia składają się z operatorów, funkcji oraz danych, na których działają.

4.1. Operatory

Spis wszystkich operatorów stosowanych w języku SQL podzielonych ze względu na zastosowanie przedstawiają kolejne podrozdziały.

4.1.1. Operatory arytmetyczne

Operatory arytmetyczne działają zasadniczo na danych typu numerycznego. Jednak niektóre z tych operatorów mogą być użyte do danych typu DATE. Spis operatorów arytmetycznych podzielonych według priorytetu przedstawia tabela:

Operator	Opis	Przykład
()	Zmienia normalną kolejność wykonywania działań. Wszystkie działania wewnątrz nawiasów są wykonywane przed działaniami poza nawiasami.	SELECT (X+Y)/(Y+Z) ...
+, -	Operatory jednoargumentowe zachowania i zmiany znaku.	... WHERE NR = -1 ... WHERE -PLACA < 0
*, /	Mnożenie, dzielenie	SELECT 2*X+1 ... WHERE X > Y/2
+, -	Dodawanie, odejmowanie	SELECT 2*X+1 ... WHERE X > Y-Z

4.1.2. Operatory znakowe

Jedynym operatorem działającym na ciągach znaków jest operator konkatencji. Rezultatem działania tego operatora jest ciąg znaków będący połączeniem operandów. Należy pamiętać, że ciągi znaków typu CHAR nie mogą być dłuższe niż 255 znaków. Ograniczenie to dotyczy również ciągu znakowego będącego wynikiem działania operatora konkatencji.

Operator	Opis	Przykład
	Konkatenacja ciągów znaków	SELECT 'Nazwa: ' ENAME ...

4.1.3. Operatory porównania

Operatory porównania są wykorzystywane w wyrażeniach i warunkach do porównywania dwóch wyrażeń. Wynikiem działania operatorów porównania jest zawsze wartość logiczna (TRUE lub FALSE).

Operator	Opis	Przykład
()	Zmienia normalną kolejność wykonywania działań	... NOT (A=1 OR B=1)
=	Sprawdza, czy dwa wyrażenia są równe	... WHERE PLACA = 1000
!=, ^=, <>	Sprawdza, czy dwa wyrażenia są różne	... WHERE PLACA != 1000
>	Większe niż	... WHERE PLACA > 1000
<	Mniejsze niż	... WHERE PLACA < 1000
>=	Większe lub równe niż	... WHERE PLACA >= 1000
<=	Mniejsze lub równe niż	... WHERE PLACA <= 1000
IN	Równy dowolnemu elementowi. Synonim do „= ANY”	... WHERE ZAWOD IN ('URZEDNIK', 'INFORMATYK') ... WHERE PLACA IN (SELECT PLACA FROM PRAC WHERE WYDZIAL=30)
NOT IN	Różny od każdego z elementów. Wynikiem jest FALSE jeśli dowolny element zbioru jest równy NULL Synonim do „!= ALL”	... WHERE PLACA NOT IN (SELECT PLACA FROM PRAC WHERE WYDZIAL=30)
ANY	Porównuje wartość z każdą wartością ze zbioru po prawej stronie. Musi być poprzedzony jednym z operatorów: =, !=, >, <, <=, >=. Zwraca TRUE, jeśli przynajmniej jeden z elementów spełnia podany warunek.	... WHERE PLACA = ANY (SELECT PLACA FROM PRAC WHERE WYDZIAL=30)
ALL	Porównuje wartość z każdą wartością ze zbioru po prawej stronie. Musi być poprzedzony jednym z operatorów: =, !=, >, <, <=, >=. Zwraca TRUE, jeśli każdy z elementów spełnia podany warunek.	... WHERE (PLACA, PREMIA) >= ALL ((14900, 300), (3000, 0))
[NOT] BETWEEN	[Nie] większy lub równy x i mniejszy lub równy y.	... WHERE A BETWEEN 1 AND 9

Bazy danych		Robert Chwastek
x AND y		
[NOT] EXISTS	Zwraca TRUE jeśli zapytanie [nie] zwraca przynajmniej jeden wiersz.	... WHERE EXISTS (SELECT PLACA FROM PRAC WHERE WYDZIAL = 30)
[NOT] LIKE	[Nie] spełnia podany wzorzec. Litera '%' jest używana do zapisywania dowolnego ciągu znaków (0 lub więcej), który nie jest równy NULL. Litera '_' zastępuje dowolną pojedynczą literę.	... WHERE STAN LIKE 'T%'
IS [NOT] NULL	[Nie] jest równe NULL.	... WHERE ZAWOD IS NULL

Operator NOT IN zwróci FALSE (co w przypadku klauzuli WHERE spowoduje, że żadne wiersze nie zostaną zwrócone), jeśli choć jeden z elementów listy jest równy NULL. Np. rozkaz:

```
SELECT 'TRUE'
FROM prac
WHERE wydzial NOT IN (5, 15, NULL) ;
```

nie zwróci żadnych wierszy, ponieważ
wydzial NOT IN (5, 15, NULL)

zostanie rozwinięty do

wydzial != 5 AND wydzial != 15 AND wydzial != NULL

Wynikiem działania operatorów porównania i logicznych dla wartości NULL jest wartość NULL. Dlatego też wynikiem całego opisywanego rozkazu będzie wartość NULL.

4.1.4. Operatory logiczne

Operatory logiczne służą do wykonywania obliczeń na wartościach typu logicznego (w szczególności będących wynikiem obliczania warunków).

Operator	Opis	Przykład
()	Zmienia normalną kolejność wykonywania działań	SELECT ... WHERE x = y AND (a = b OR p = q)
NOT	Zaprzeczenie wyrażenia logicznego	... WHERE NOT (zawod IS NULL) WHERE NOT (A=1)
AND	Logiczne 'i'. Wynik jest równy TRUE, jeśli wartości obu operandów są równe TRUE	... WHERE A = 1 AND B = 2
OR	Logiczne 'lub'. Wynik jest równy TRUE, jeśli wartość przynajmniej	... WHERE A = 1 OR B = 3

jednego operandu jest równa
TRUE

Poniższe tabele przedstawiają wynik działania operatora AND i OR dla różnych wartości:

AND	true	false	null
true	true	false	null
false	false	false	false
null	null	false	null

OR	true	false	null
true	true	true	true
false	true	false	null
null	true	null	null

4.1.5. Operatory mnogościowe

Operatory zbiorowe działają na wynikach zapytań lub listach wartości.

Operator	Opis	Przykład
UNION	Unia dwóch zbiorów. Łączy dwa zbiory, powtarzające się elementy występują tylko raz.	... SELECT ... UNION SELECT ...
INTERSECT	Część wspólna dwóch zbiorów. Powtarzające się elementy występują tylko raz	... SELECT ... INTERSECT SELECT ...
MINUS	Oblicza różnicę dwóch zbiorów. W wyniku umieszczane są tylko te elementy, które występują w pierwszym zbiorze i nie występują w drugim. Elementy powtarzające się występują tylko raz	... SELECT ... MINUS SELECT ...

4.2. Wyrażenia

Wyrażenie jest ciągiem jednej lub więcej wartości, operatorów lub funkcji. Wynik obliczania wyrażenia musi być wartością. W ogólności typ wyniku zależy od typów operandów.

Następujące przykłady pokazują wyrażenia różnych typów:

- numeryczny: $2 * 2$
- znakowy: `TO_CHAR (TRUNC (SYSDATE + 7))`

Wyrażenie może być użyte wszędzie tam, gdzie możliwe jest użycie wartości stałej, np.:
SET Nazwisko = LOWER(Nazwisko)

Istnieje pięć form wyrażień:

- kolumna, stała lub wartość specjalna

Składnia:

```
[table.] { column | ROWID }  
text  
number  
sequence.CURRVAL  
sequence.NEXTVAL  
NULL  
ROWNUM  
LEVEL  
SYSDATE  
UID  
USER
```

Przykłady:

```
pracownicy.nazwisko  
'to jest ciąg znaków'  
10  
SYSDATE
```

- zmienna łączona z opcjonalną zmienną indykatorową

Składnia:

```
: { n | variable } [ :ind_variable ]
```

Przykłady:

```
:nazwisko_pracownika:nazwisko_pracownika_indykator  
:położenie_wydziału
```

- wywołanie funkcji

Składnia:

```
function_name( [DISTINCT | ALL] expr [, expr] ... )
```

Przykłady:

```
LENGTH('Kowalski')  
ROUND(1234.567*82)
```

- kombinacja wyrażień wymienionych w poprzednich punktach

Składnia:

```
(expr)  
+expr, -expr, PRIOR expr  
expr * expr, expr / expr  
expr + expr, expr - expr, expr || expr
```

Przykłady:

```
('Kowalski: ' || 'Nauczyciel')  
LENGTH('Nowak') * 57  
SQRT(144) + 72
```

- lista wyrażień w nawiasach

Składnia:

```
(expr [, expr], ...)
```

Przykłady:

```
( 'Kowalski', 'Nowak', 'Burzynski' )
(10, 20, 40)
(LENGTH('Kowalski') * 5, -SQRT(144) + 77, 59)
```

Wyrażenia są używane w:

- liście kolumn w rozkazie SELECT
- jako warunek w klauzulach WHERE i HAVING
- w klauzulach CONNECT BY, START WITH, ORDER BY
- klauzuli VALUE w rozkazie INSERT
- w klauzuli SET rozkazu UPDATE

4.3. Warunki

Warunkiem nazywamy ciąg jednego lub więcej wyrażeń i operatorów logicznych. Warunek jest zawsze obliczany do wartości TRUE lub FALSE. Warunki mogą mieć siedem różnych postaci:

- porównanie z wyrażeniem lub wynikiem zapytania
`<expr> <comparison operator> <expr>`
`<expr> <comparison operator> <query>`
`<expr-list> <equal-or-not> <expr-list>`
`<expr-list> <equal-or-not> <query>`
- porównanie z dowolnym lub ze wszystkimi elementami listy lub zapytania
`<expr> <comparison> { ANY | ALL }`
`(<expr> [, <expr>] ...)`

`<expr> <comparison> { ANY | ALL } <query>`

`<expr-list> <equal-or-not> { ANY | ALL }`
`(<expr-list> [, <expr-list>] ...)`

`<expr-list> <equal-or-not> { ANY | ALL } <query>`

- sprawdzenie przynależności do listy lub zapytania
`<expr> [NOT] IN (<expr> [, <expr>] ...)`

`<expr> [NOT] IN <query>`

`<expr-list> [NOT] IN`
`(<expr-list> [, <expr-list>] ...)`

`<expr-list> [NOT] IN <query>`

- sprawdzenie przynależności do zakresu wartości
`<expr> [NOT] BETWEEN <expr> AND <expr>`

- sprawdzenie czy wartość jest równa NULL
`<expr> IS [NOT] NULL`

- sprawdzenie czy zapytanie zwróciło jakiegokolwiek wiersze

EXISTS <query>

- kombinacja innych warunków (podana zgodnie z priorytetami)

(<condition>)

NOT <condition>

<condition> AND <condition>

<condition> OR <condition>

Przykłady:

Nazwisko = 'Kowalski'

pracownicy.Wydzial = Wydzialy.Wydzial

Data_urodzenia > '01-JAN-67'

Zawod IN ('Dyrektor', 'Urzednik', 'Informatyk')

Placa BETWEEN 500 AND 1500

5. Standardowe funkcje języka SQL

5.1. Funkcje numeryczne

Składnia	Przeznaczenie	Przykład
ABS(n)	Zwraca wartość absolutną liczby n	ABS(-15) Wynik: 15
CEIL(n)	Zwraca najmniejszą liczbę całkowitą większą lub równą n	CEIL(15.7) Wynik: 16
FLOOR(n)	Zwraca największą liczbę całkowitą mniejszą lub równą n	FLOOR(15.7) Wynik: 15
MOD(m, n)	Zwraca resztę z dzielenia liczby m przez n	MOD(7, 5) Wynik: 2
POWER(m, n)	Zwraca liczbę m podniesioną do potęgi n. Liczba n musi być całkowita; w przeciwnym wypadku wystąpi błąd.	POWER(2, 3) Wynik: 8
ROUND(n[, m])	Zwraca liczbę n zaokrągloną do m miejsc po przecinku. Jeśli m jest pominięte, to przyjmuje się 0. Liczba m może być dodatnia lub ujemna (zaokrąglenie do odpowiedniej liczby cyfr przed przecinkiem)	ROUND(16.167, 1) Wynik: 16.2 ROUND(16, 167, -1) Wynik: 20
SIGN(n)	Zwraca 0, jeśli n jest równe 0, -1 jeśli n jest mniejsze od 0, 1 jeśli n jest większe od 0	SIGN(-15) Wynik: -1
SQRT(n)	Zwraca pierwiastek kwadratowy liczby n. Jeśli $n < 0$ to wystąpi błąd	SQRT(25) Wynik: 5
TRUNC(m[, n])	Zwraca m obcięte do n miejsc po przecinku. Jeśli n nie jest podane, to przyjmuje się 0. Jeśli n jest ujemne to obcinane są cyfry przed przecinkiem.	TRUNC(15.79, 1) Wynik: 15.7 TRUNC(15.79, -1) Wynik: 10

5.2. Funkcje znakowe

Składnia	Przeznaczenie	Przykład
CHR(n)	Zwraca znak o podanym kodzie	CHR(65) Wynik: „A”

Bazy danych		Robert Chwastek
INITCAP(string)	Zwraca string, w którym każde słowo ma dużą pierwszą literę, a pozostałe są małe.	INITCAP('PAN JAN NOWAK') Wynik: „Pan Jan Nowak”
LOWER(string)	Zamienia wszystkie litery w podanym stringu na małe.	LOWER('PAN JAN NOWAK') Wynik: „pan jan nowak”
LPAD(string1, n [, string2])	Zwraca string 1 uzupełniony do długości n lewostronnie ciągami znaków ze stringu 2. Jeśli string2 nie jest podany to przyjmowana jest spacja. Jeśli n jest mniejsze od długości string1, to zwracane jest n pierwszych znaków z tekstu string1.	LPAD('Ala ma ', kota*, 17) Wynik: „kota*kota*Ala ma ”
LTRIM(string [, zbiór])	Usuwa litery z tekstu string od lewej strony aż do napotkania litery nie należącej do tekstu zbiór. Jeśli zbiór nie jest podany to przyjmowany jest ciąg pusty.	LTRIM('xxxXxxOstatnie słowo', 'x') Wynik: „XxxOstatnie słowo”
REPLACE(string, search [, replace])	Zwraca string z zamienionym każdym wystąpieniem tekstu search na tekst replace.	REPLACE('Jack & Jue', 'J', Bl') Wynik: „Black & Blue”
RPAD(string1, n [, string2])	Zwraca string 1 uzupełniony prawostronnie do długości n ciągami string2. Jeśli string2 nie jest podany, to przyjmuje się spację. Jeśli n jest mniejsze od długości string1, to zwracane jest n pierwszych znaków z tekstu string1.	RPAD('Ala ma ', 17, 'kota*') Wynik: „Ala ma kota*kota*”
RTRIM(string [, zbiór])	Zwraca string1 z usuniętymi ostatnimi literami, które znajdują się w stringu zbiór. Jeśli zbiór nie jest podany to przyjmowany jest ciąg pusty	RTRIM('Ostatnie słowxxxXxxx', 'x') Wynik: „Ostatnie słowxxxX”
SOUNDEX(string)	Zwraca ciąg znaków reprezentujący wymowę słów wchodzących w skład string1. Funkcja SOUNDEX może być użyta do porównywania słów zapisywanych w różny sposób, ale wymawianych tak samo.	SELECT nazwisko FROM bibl WHERE SOUNDEX (nazwisko) = SOUNDEX (‘Mickiewicz’);
SUBSTR(string, m [, n])	Zwraca podciąg z ciągu znaków string zaczynający się na znaku m i o długości n. Jeśli n nie jest podane, to zwracany	SUBSTR('ABCDE',2, 3) Wynik:

Bazy danych		Robert Chwastek
	jest podciąg od znaku m do ostatniego w string. Pierwszy znak w ciągu ma numer 1.	„BCD”
TRANSLATE(string, from, to)	Zwraca string powstały po zamianie wszystkich znaków from na znak to.	TRANSLATE('HELLO! THERE!', '!', '-') Wynik: „HELLO- THERE-”
UPPER(string)	Zamienia wszystkie znaki z ciągu string na duże litery.	UPPER('Jan Nowak') Wynik: „JAN NOWAK”
ASCII(string)	Zwraca kod ASCII pierwszej litery w podanym ciągu znaków	ASCII('A') Wynik: 65
INSTR(string1, string2 [, n [, m]])	Zwraca pozycję m-tego wystąpienia string2 w string1, jeśli szukanie rozpoczęto od pozycji n. Jeżeli m jest pominięte, to przyjmowana jest wartość 1. Jeśli n jest pominięte, przyjmowana jest wartość 1.	INSTR('MISSISSIPPI', 'S', 5, 2) Wynik: 7
LENGTH(string)	Zwraca długość podanego ciągu znaków.	LENGTH('Nowak') Wynik: 5

5.3. Funkcje grupowe

Funkcje grupowe zwracają swoje rezultaty na podstawie grupy wierszy a nie pojedynczych wartości. Domyślnie cały wynik jest traktowany jako jedna grupa. Klauzula GROUP BY z rozkazu SELECT może jednak podzielić wiersze wynikowe na grupy. Klauzula DISTINCT wybiera z grupy tylko pojedyncze wartości (drugie i następne są pomijane). Klauzula ALL powoduje wybranie wszystkich wierszy wynikowych do obliczenia wyniku. Wszystkie wymienione w tym podrozdziale funkcje opuszczają wartości NULL z wyjątkiem COUNT(*). Wyrażenia będące argumentami funkcji mogą być typu CHAR, NUMBER lub DATE.

Składnia	Przeznaczenie	Przykład
AVG([DISTINCT <u>ALL</u>] num)	Zwraca wartość średnią ignorując wartości puste	SELECT AVG(placa) ”Srednia” FROM pracownicy
COUNT([DISTINCT <u>ALL</u>] expr)	Zwraca liczbę wierszy, w których expr nie jest równe NULL	SELECT COUNT(nazwisko) ”Liczba” FROM pracownicy
COUNT(*)	Zwraca liczbę wierszy w tabeli włączając powtarzające się i równe	SELECT COUNT(*) ”Wszystko” FROM pracownicy

Bazy danych		Robert Chwastek
	NULL	
MAX([DISTINCT <u>ALL</u>] expr)	Zwraca maksymalną wartość wyrażenia	SELECT MAX(Placa) "Max" FROM pracownicy
MIN([DISTINCT <u>ALL</u>] expr)	Zwraca minimalną wartość wyrażenia	SELECT MIN(Placa) "Min" FROM pracownicy
STDDEV([DISTINCT <u>ALL</u>] num)	Zwraca odchylenie standardowe wartości num ignorując wartości NULL.	SELECT STDDEV(Placa) "Odchylenie" FROM pracownicy
SUM([DISTINCT <u>ALL</u>] num)	Zwraca sumę wartości num.	SELECT SUM(Placa) "Koszty osobowe" FROM pracownicy
VARIANCE([DISTINCT <u>ALL</u>] num)	Zwraca wariancję wartości num ignorując wartości NULL	SELECT VARIANCE(Placa) "Wariancja" FROM pracownicy

5.4. Funkcje konwersji

Funkcje konwersji służą do zamiany wartości jednego typu na wartość innego typu. Ogólnie nazwy funkcji konwersji tworzone są według następującego schematu: *typ*TO*typ*. Pierwszy typ jest typem, z którego wykonywana jest konwersja, drugi jest typem wynikowym.

Składnia	Przeznaczenie	Przykład
CHARTOROWID (string)	Wykonuje konwersję ciągu znaków na ROWID	SELECT nazwisko FROM pracownicy WHERE ROWID = CHARTOROWID (‘0000000F.0003.0002’)
CONVERT(string [,dest_char_set [,source_char_set]])	Wykonuje konwersję pomiędzy dwoma różnymi implementacjami zestawu znaków. Zestawem domyślnym jest US7ASCII.	SELECT CONVERT (‘New WORD’, ‘US7ASCII’, ‘WE8HP’) "Conversion" FROM DUAL
HEXTORAW (string)	Konwertuje ciąg znaków zawierający cyfry szesnastkowe na wartość binarną, którą można umieścić w polu typu RAW	INSERT INTO GRAPHICS (RAW_COLUMN) SELECT HEXTORAW (‘7D’) FROM DUAL
ROWTOHEX(raw)	Przekształca wartość typu raw na tekst zawierający cyfry szesnastkowe odpowiadające podanej liczbie.	SELECT RAWTOHEX (RAW_COLUMN) "Graphics" FROM GRAPHICS
ROWIDTOCHAR	Przekształca identyfikator	SELECT ROWID FROM

Bazy danych		Robert Chwastek
	wiersza na tekst. Wynik konwersji ma zawsze długość 18 znaków.	GRAPHICS WHERE ROWIDTOCHAR(ROWID) LIKE '%F38%'
TO_CHAR(n [, fmt]) (konwersja numeryczna)	Konwertuje wartość numeryczną na znakową używając opcjonalnego ciągu formatującego. Jeśli ciąg formatujący nie jest podany, to wartość jest konwertowana tak, by zawrzeć wszystkie cyfry znaczące.	SELECT TO_CHAR(17145, '\$099,999') "Char" FROM DUAL
TO_CHAR(d [, fmt]) (konwersja daty)	Konwertuje datę na tekst, używając podanego formatu.	SELECT TO_CHAR(HIREDATE, 'Month DD, YYYY') "New date format" FROM EMP WHERE ENAME = 'SMITH'
TO_DATE(string [, fmt])	Przekształca ciąg znaków w datę. Używa danych aktualnych, jeśli nie mogą być one odczytane z podanego tekstu. Do konwersji używany jest podany ciąg formatujący lub wartość domyślna postaci „DD-MON-YY”	INSERT INTO BONUS (BONUS_DATE) SELECT TO_DATE ('January 15, 1989', 'Month dd, YYYY') FROM DUAL
TO_NUMBER (string)	Przekształca tekst zawierający zapis liczby na liczbę	UPDATE EMP SET SAL = SAL + TO_NUMBER(SUBSTR('\$100 raise', 2, 3)) WHERE ENAME = 'BLAKE'

5.5. Funkcje operacji na datach

Składnia	Przeznaczenie	Przykład
ADD_MONTHS (date, n)	Zwraca padaną datę powiększoną o podaną liczbę miesięcy n. Liczba ta może być ujemna	SELECT ADD_MONTHS(HIREDATE, 12) "Next year" FROM EMP WHERE ENAME = 'SMITH'
LAST_DAY(date)	Zwraca datę będącą ostatnim dniem w miesiącu zawartym w	SELECT LAST_DAY(SYSDATE) "Last"

Bazy danych		Robert Chwastek
	podanej dacie.	FROM DUAL
MONTHS_BETWEEN (date1, date2)	Zwraca liczbę miesięcy pomiędzy datami date1 i date2. Wynik może być dodatni lub ujemny. Część ułamkowa jest częścią miesiąca zawierającego 31 dni.	SELECT MONTHS_BETWEEN ('02-feb-86', '01-jan-86')) „Months” FROM DUAL
NEW_TIME(date, a, b)	Zwraca datę i czas w strefie czasowej b, jeśli data i czas w strefie a są równe date. Parametry a i b są wyrażeniami znakowymi i mogą być jednym z: AST, ADT - Atlantic Standard or Daylight Time BST, BDT - Bering Standard or Daylight Time CST, CDT - Central Standard or Daylight Time EST, EDT - Eastern Standard or Daylight Time GMT - Greenwich Mean Time HST, HDT - Alaska-Hawaii Standard or Daylight Time MST, MDT - Mountain Standard or Daylight Time NST - Newfoundland Standard Time PST, PDT - Pacific Standard or Daylight Time YST, YDT - Yukon Standard or Daylight Time	SELECT TO_CHAR(NEW_TIME(TO_DATE('17:47', 'hh24:mi'), 'PST', 'GMT'), 'hh24:mi') "GREENWICH TIME" FROM DUAL
NEXT_DAY(date, string)	Zwraca datę pierwszego dnia tygodnia podanego w string, który jest późniejszy niż data date. Parametr string musi być poprawną nazwą dnia.	SELECT NEXT_DAY('17-MAR-89', 'TUESDAY') "NEXT DAY" FROM DUAL
ROUND(date [, fmt])	Zwraca datę zaokrągloną do jednostki zaokrąglania podanej w fmt. Domyślnie jest to najbliższy dzień.	SELECT ROUND (TO_DATE('27-OCT-88'), 'YEAR') "FIRST OF THE YEAR" FROM DUAL
SYSDATE	Zwraca aktualny czas i datę. Nie wymaga podania argumentów.	SELECT SYSDATE FROM DUAL

Bazy danych		Robert Chwastek
TRUNC(date [, fmt])	Zwraca datę obciętą do jednostki podanej w fmt. Domyślnie jest to dzień, tzn. usuwana jest informacja o czasie.	SELECT TRUNC(TO_DATE('28-OCT-88', 'YEAR') 'First Of The Year' FROM DUAL

W funkcjach ROUND i TRUNC można używać następujących tekstów do identyfikacji jednostki zaokrąglenia lub obcięcia:

CC, SCC	wiek
SYYY, YYYY, YEAR, SYEAR, YY, Y	rok (zaokrąglenie w zwyż od 1.07)
Q	kwartał (zaokrąglenie w górę od 16-tego drugiego miesiąca)
MONTH, MON, MM	miesiąc (zaokrąglenie w górę od 16)
WW	pierwszy tydzień roku
W	pierwszy tydzień miesiąca
DDD, DD, J	dzień
DAY, DY, D	najbliższa niedziela
HH, HH12, HH24	godzina
MI	minuta

5.6. Inne funkcje

Składnia	Przeznaczenie	Przykład
GREATEST(expr [, expr] ...)	Znajduje największą z listy wartości. Wszystkie wyrażenia począwszy od drugiego są konwertowane do typu pierwszego wyrażenia przed wykonaniem porównania.	SELECT GREATEST ('Harry', 'Harriot', 'Harold') "GREATEST" FROM DUAL
LEAST(expr [, expr] ...)	Zwraca najmniejszą z listy wartości. Wszystkie wyrażenia począwszy od drugiego są konwertowane do typu pierwszego wyrażenia przed wykonaniem porównania.	SELECT LEAST ('Harry', 'Harriot', 'Harold') "LEAST" FROM DUAL
NVL (expr1, expr2)	Jeśli expr1 jest równe NULL, to zwraca expr2, w przeciwnym wypadku zwraca expr1.	SELECT ENAME NVL(TO_CHAR(COMM), 'NOT APPLICABLE') "COMMISION" FROM EMP WHERE DEPTNO = 30

Bazy danych		Robert Chwastek
UID	Zwraca unikalny identyfikator użytkownika wywołującego funkcję.	SELECT USER, UID FROM DUAL
USER	Zwraca nazwę użytkownika	SELECT USER, UID FROM DUAL

5.7. Formaty zapisu danych

Formaty zapisu danych używane są w dwóch podstawowych celach:

- zmiany sposobu wyświetlania informacji w kolumnie;
- wprowadzanie danej zapisanej inaczej niż domyślnie.

Formaty zapisu używane są w funkcjach TO_CHAR i TO_DATE.

5.7.1. Formaty numeryczne

Formaty numeryczne są używane w połączeniu z funkcją TO_CHAR do przekształcenia wartości numerycznej na wartość znakową.

Użycie formatu numerycznego powoduje zaokrąglenie do podanej w nim liczby cyfr znaczących.

Jesli wartość numeryczna ma więcej cyfr z lewej strony niż to zostało przewidziane, to wartość ta zastępowana jest gwiazdką '*'.

Poniższa tabela przedstawia elementy, które może zawierać specyfikacja formatu numerycznego:

Element	Przykład	Opis
9	9999	Liczba '9' określa szerokość wyświetlania
0	0999	Pokazuje wiodące zera
\$	\$9999	Poprzedza wartość znakiem '\$'
B	B9999	Wyświetla zera jako spacje (nie jako zera)
MI	9999MI	Wyświetla '-' po wartości ujemnej
PR	9999PR	Wyświetla wartość ujemną w nawiasach kątowych '<', '>'
, (przecinek)	9,999	Wyświetla przecinek na podanej pozycji
. (kropka)	99.99	Wyświetla kropkę na podanej pozycji
V	999V99	Mnoży wartość przez 10 ⁿ , gdzie n jest liczbą dziewiątek po 'V'
E	9.999EEEE	Wyświetla liczbę w notacji wykładniczej (format musi zawierać dokładnie cztery litery E)
DATE	DATE	Dla dat przechowywanych w postaci numerycznej. Wyświetla datę w formacie 'MM/DD/YY'

5.7.2. Formaty dat

Formaty dat są używane w funkcji TO_CHAR w celu wyświetlenia daty. Mogą być również użyte w funkcji TO_DATE w celu wprowadzenia daty w określonym formacie. Format standardowy, to 'DD-MON-YY'.

Elementy formatu dat przedstawia tabela:

<i>Element</i>	<i>Opis</i>
SCC lub CC	Wiek; 'S' poprzedza daty przed naszą erą znakiem '-'
YYYY lub SYYYY	Czterocyfrowy rok, 'S' poprzedza daty przed naszą erą znakiem '-'
YYY, YY lub Y	Ostatnie 3, 2 lub 1 cyfra roku
Y,YYY	Rok z przecinkiem na podanej pozycji
SYEAR lub YEAR	Rok przeliterowany. 'S' powoduje poprzedzenie daty przed naszą erą znakiem '-'
BC lub AD	Znak BC/AD (przed naszą erą/naszej ery)
B.C. lub A.D	Znak BC/AD z kropkami
Q	Kwartał roku (1, 2, 3 lub 4)
MM	Miesiąc (01-12)
MONTH	Nazwa miesiąca wyrównana do 9 znaków za pomocą spacji
MON	Trzyliterowy skrót nazwy miesiąca
WW	Tydzień roku (1-52) (tydzień zaczyna się w pierwszy dniu roku i trwa 7 dni)
W	Tydzień miesiąca (1-5) (tydzień zaczyna się w pierwszym dniu miesiąca i trwa 7 dni)
DDD	Dzień roku (1-366)
DD	Dzień miesiąca (1-31)
D	Dzień tygodnia (1-7)
DAY	Nazwa dnia wyrównana do 9 znaków za pomocą spacji
DY	Trzyliterowy skrót nazwy dnia
AM lub PM	Wskaźnik pory dnia
A.M. lub P.M.	Wskaźnik pory dnia z kropkami
HH lub HH12	Godzina (1-12)
HH24	Godzina (1-24)
MI	Minuta (0-59)
SS	Sekunda (0-59)
SSSS	Sekundy po północy (0-86399)
/,.	Znaki przestankowe umieszczane w wyniku
"..."	Ciąg znaków umieszczany w wyniku

Dodatkowo w ciągu znaków określających format można użyć:

- FM - „Fill Mode” przełącznik włączający/wyłączający wypełnianie tekstów spacjami i liczb zerami;
- TH - dodany po kodzie pola powoduje wyświetlanie liczby porządkowej np. 4TH dla liczby 4;
- SP - dodany po kodzie pola powoduje, że jest ono literowane
- SPTH lub THSP - połączenie SP i TH.

6. Programowanie proceduralne - PL/SQL

6.1. Wprowadzenie

Rozkazy języka SQL są niewystarczające do tworzenia efektywnych systemów baz danych, a w szczególności do kontroli warunków integralności bazy w momencie wprowadzania do niej danych. Dlatego firma Oracle wprowadziła rozszerzenia proceduralne do swojej implementacji języka SQL i tak powstał język nazwany PL/SQL.

PL/SQL pozwala wykorzystywać prawie wszystkie operacje standardowego SQL. Wyjątkiem są tu operacje definiowania danych (ALTER, CREATE i RENAME) oraz niektóre rozkazy kontroli danych jak CONNECT, GRANT i REVOKE.

Kod napisany w PL/SQL składa się z rozkazów standardowego SQL oraz rozszerzeń proceduralnych. Możliwe jest stosowanie wszystkich standardowych funkcji SQL w rozkazach SQL oraz prawie wszystkich (tj. z wyłączeniem funkcji grupowych) w rozszerzeniach. Każdy rozkaz PL/SQL kończy się średnikiem.

PL/SQL pozwala na definiowanie zmiennych. Zmienne służą do przechowywania wyników zapytań i obliczeń w celu ich późniejszego wykorzystania. Jednak wszystkie zmienne muszą być zadeklarowane przed ich użyciem. Każda zmienna posiada typ. Typy zmiennych są takie same jak typy stosowane w SQL'u. Zmienne deklaruje się pisząc nazwę zmiennej a następnie jej typ, np.:

```
 premia NUMBER(7, 2);
```

Wartość do zmiennej przypisuje się za pomocą operatora przypisania „:=”, np.:

```
 podatek := cena * stopa;
```

Druga możliwość nadania wartości zmiennej to użycie rozkazu SELECT lub FETCH do wpisania wartości do zmiennej:

```
 SELECT placa INTO placa_aktualna FROM pracownicy  
 WHERE nazwisko = 'Nowak' ;
```

PL/SQL posiada zmienne strukturalne nazywane rekordami, które podzielone są na pola.

Istnieje możliwość deklarowania stałych. Deklaracja taka jest podobna do deklaracji zmiennej, ale konieczne jest dodatkowo użycie słowa CONSTANT i natychmiastowe przypisanie wartości do zmiennej. Deklarację stałej pokazuje następujący przykład:

```
 stopa_premii CONSTANT NUMBER(3, 2) := 0.10;
```

Wszystkie obiekty posiadają atrybuty. Jednym z nich jest typ (zarówno zmiennej jak i kolumny). Możliwe jest użycie zapisu %TYPE w celu zapisania typu np. w deklaracji zmiennej. Zapis taki pozwala zadeklarować zmienną o takim samym typie jak inna zmienna lub kolumna (należy przy tym zauważyć, że typ ten nie jest znany osobie piszącej program):

```
 tytul books.tytul%TYPE
```

Możliwe jest również zadeklarowanie rekordu odpowiadającego jednemu wierszowi tabeli. W tym celu należy użyć konstrukcji %ROWTYPE.

W PL/SQL można stosować następujące operatory porównań: =, !=, <, >, >=, <=. Mogą one działać na operandach różnych typów: numerycznym, daty i ciągach znaków (wykonują wtedy porównanie leksykalne).

6.2. Struktura bloku

Kod języka PL/SQL jest podzielony na bloki. Blok ma następującą strukturę:

```
DECLARE
    deklaracje
BEGIN
    rozkazy wykonywalne
EXCEPTION
    obsługa sytuacji wyjątkowych
END;
```

Każdy blok może zawierać inne bloki, tzn. bloki mogą być zagnieżdżone. W PL/SQL identyfikator jest nazwą dowolnego obiektu (tj. stałej, zmiennej, rekordu, kursora lub wyjątku). Nie jest możliwa dwukrotna deklaracja tego samego identyfikatora w jednym bloku. Można jednak zadeklarować te same identyfikatory w dwóch różnych blokach. Oba takie obiekty są różne i jakakolwiek zmiana w jednym z nich nie powoduje zmiany w drugim. Zakres obowiązywania identyfikatora określa, w którym bloku mogą wystąpić do niego odwołania. Blok ma dostęp tylko do obiektów lokalnych i globalnych. Identyfikatory zadeklarowane w bloku są lokalne dla tego bloku i globalne dla wszystkich bloków w nim zawartych (podbloków). Identyfikatory globalne mogą zostać zredefiniowane w podblokach, co powoduje, że obiekt lokalny ma pierwszeństwo przed globalnym. Dostęp do obiektu globalnego jest możliwy w tym przypadku tylko wtedy, gdy użyta zostanie nazwa odpowiedniego bloku. Blok nie ma dostępu do obiektów zadeklarowanych w innych blokach na tym samym poziomie zagnieżdżenia, ponieważ nie są one ani lokalne, ani globalne w tym bloku.

6.3. Procedury i funkcje

PL/SQL w wersji 2.0 pozwala na definiowanie funkcji i procedur.

Składnia definicji procedury jest następująca:

```
PROCEDURE name [ (parameter [, parameter] ... ) ] IS
    [local declarations]
BEGIN
    executable statements
[EXCEPTION
    exception handlers]
END [name];
```

Składnia definicji parametrów jest następująca:

```
var_name [IN | OUT | IN OUT] datatype
    [{ := | DEFAULT } value]
```

Określenie typu dla danego parametru nie może zawierać ograniczeń, tzn. nie jest możliwe zapisanie np. INT(5).

Procedura składa się z dwóch części - nagłówka i ciała. Nagłówek rozpoczyna się słowem PROCEDURE i kończy na nazwie procedury lub liście parametrów. Deklaracja parametrów jest opcjonalna. Procedury bezparametrowe zapisuje się bez nawiasów. Ciało procedury rozpoczyna się słowem IS a kończy słowem END (po którym może opcjonalnie wystąpić nazwa procedury). Część deklaracyjna (pomiędzy słowem IS i BEGIN) zawiera deklaracje obiektów lokalnych. W tym przypadku nie używa się słowa DECLARE. Część wykonywalna (pomiędzy BEGIN a EXCEPTION lub END) zawiera rozkazy języka PL/SQL. W tej części musi wystąpić conajmniej jeden rozkaz. Definicję procedury zwiększającą płacę wybranego pracownika pokazuje przykład:

```
PROCEDURE zwieksz (prac_id INTEGER, kwota REAL) IS
    placa_aktualna REAL;
BEGIN
    UPDATE pracownicy SET placa_podstawowa =
        placa_podstawowa + kwota
    WHERE prac_id = id_pracownika;
END zwieksz;
```

Zadaniem funkcji jest obliczenie wartości. Definicja funkcji jest taka sama jak procedury z tym wyjątkiem, że funkcja posiada klauzulę RETURN.

```
FUNCTION name [ (argument [, argument] ... ) ]
    RETURN datatype IS
    [local declarations]
BEGIN
    executable statements
[EXCEPTION
    exception handlers]
END [name];
```

Wewnątrz funkcji musi pojawić się przynajmniej jeden rozkaz RETURN. Rozkaz RETURN natychmiast kończy wykonanie funkcji i zwraca sterowanie do modułu wywołującego. W przypadku funkcji musi wystąpić przynajmniej jeden rozkaz RETURN, w którym musi być wyrażenie. Wynik tego wyrażenia musi mieć taki typ jak podany w nagłówku. Jeśli funkcja nie kończy się rozkazem RETURN, to PL/SQL zgłosi odpowiedni wyjątek. W przypadku procedury rozkaz RETURN nie może zawierać żadnego wyrażenia.

6.4. Kursory

W celu wykonania rozkazu SQL system tworzy pewien obszar roboczy nazywany przestrzenią kontekstu. W przestrzeni tej przechowywane są informacje konieczne do wykonania rozkazu. PL/SQL pozwala nazwać przestrzeń kontekstu i odwoływać się do zawartych w niej danych za pomocą mechanizmu nazywanego kursorem. PL/SQL używa dwóch typów kursorów:

- jawnych - użytkownik może w sposób jawny utworzyć kursor dla zapytań, których wynikiem jest wiele wierszy i wykonywać operacje na tych wierszach (najczęściej za pomocą rozkazu FOR);
- niejawnych - PL/SQL automatycznie tworzy kursor dla wszystkich pozostałych operacji.

Jeśli zapytanie zwraca wiele wierszy, to możliwe jest utworzenie kursora, który będzie umożliwiał dostęp do pojedynczych wierszy ze zwracanej listy. Kursor definiuje się w części deklaracji bloku PL/SQL przez nazwanie go i specyfikację zapytania. Sposób deklaracji kursora pokazuje przykład:

```
DECLARE
  CURSOR prac_kursor IS SELECT nazwisko, wydział
    FROM pracownicy
    WHERE płaca > 2000 ;
  ...
BEGIN
  ...
```

Sama deklaracja nie powoduje wykonania występującego w niej zapytania. Do tego konieczne jest otwarcie kursora, którego można dokonać instrukcją OPEN w następujący sposób:

```
OPEN prac_kursor;
```

W celu odczytania pojedynczego wiersza z kursora konieczne jest następnie użycie rozkazu FETCH. Każdy kolejny rozkaz FETCH dla danego kursora powoduje odczytanie kolejnego wiersza. Rozkaz FETCH może być użyty w następujący sposób:

```
FETCH prac_kursor INTO prac_nazw, prac_wydz;
```

Po zakończeniu pracy z kursorem konieczne jest poinformowanie o tym systemu w celu zwolnienia zasobów. Dokonuje się tego rozkazem CLOSE, np.:

```
CLOSE prac_kursor;
```

Każdy kursor posiada pewne atrybuty, których wartości informują o jego stanie. Nazwy atrybutów poprzedzone są znakiem '%' i wpisywane bezpośrednio po nazwie kursora. Zdefiniowano następujące atrybuty kursorów:

- %NOTFOUND - TRUE, jeśli ostatni rozkaz FETCH zakończył się niepowodzeniem z powodu braku wierszy
- %FOUND - TRUE, jeśli ostatni rozkaz FETCH zakończył się sukcesem
- %ROWCOUNT - liczba wierszy w kursorze (po otwarciu kursora)
- %ISOPEN - TRUE, jeśli kursor jest otwarty

Następujące przykłady pokazują sposób użycia atrybutów kursora:

```
LOOP
  FETCH prac_kursor INTO prac_nazw, prac_wydz;
  IF prac_kursor%ROWCOUNT > 10 THEN
    -- więcej niż 10
    ...
  EXIT WHEN prac_kursor%NOTFOUND;
  ...
END LOOP;
```

6.5. Rekordy

W języku PL/SQL rekordem nazywana jest zmienna złożona, będąca grupą zmiennych elementarnych. Każda zmienna elementarna w rekordzie nazywana jest polem. Rekordy są używane najczęściej do przechowywania zawartości pojedynczego wiersza w bazie danych. PL/SQL pozwala definiować rekordy odpowiadające pojedynczym wierszom w tabeli, widoku lub kursorze, nie pozwala jednak na definiowanie typów poszczególnych pól.

Deklarowanie rekordu najlepiej jest wyjaśnić na przykładzie. Jeśli wystąpiła deklaracja kursora w tabeli pracownicy, który zwraca nazwisko, wydział i datę zatrudnienia, to możliwe jest zadeklarowanie odpowiedniego rekordu za pomocą atrybutu %ROWTYPE:

```
DECLARE
    CURSOR prac_kursor IS
        SELECT nazwisko, wydzial, data_zatrudnienia
            FROM pracownicy;
    prac_rek prac_kursor%ROWTYPE
...
    FETCH prac_kursor INTO prac_rek;
```

Rekord może być tworzony nie tylko za pomocą kursora, ale również za pomocą nazwy tabeli w następujący sposób:

```
nazwa_rekordu nazwa_tabeli%ROWTYPE;
```

Rzeczywistą deklarację pokazuje przykład:

```
DECLARE
    prac_rek pracownicy%ROWTYPE;
...
BEGIN
    ...
END;
```

Dostęp do pola rekordu możliwy jest za pomocą nazwy tego rekordu i poprzedzonej kropką nazwy pola:

```
nazwa_rekordu.nazwa_pola;
```

Aby więc dodać pojedyncze wynagrodzenie do sumy (przy użyciu zdefiniowanego wcześniej rekordu prac_rek) można napisać:

```
suma := suma + prac_rek.placa;
```

Możliwe jest przypisanie wartości do pola rekordu lub rekordu jako całości. Należy jednak pamiętać, że rekord jest zmienną i zmiana wartości jego pól nie powoduje zmiany wartości odpowiedniego wiersza w bazie danych.

Przypisanie wartości do pola rekordu można dokonać za pomocą operatora przypisania ':= ' w następujący sposób:

```
nazwa_rekordu.nazwa_pola := plsql_wyrazenie;
```

Użycie tej konstrukcji ilustruje przykład:

```
prac_rek.nazwisko := UPPER(prac_rek.nazwisko);
```

Przypisanie zawartości całego rekordu możliwe jest na dwa sposoby:

- przypisanie zawartości jednego rekordu do drugiego (deklaracje obu rekordów muszą odwoływać się do tego samego kursora lub tabeli;

- wstawienie wartości do rekordu rozkazem SELECT ... INTO lub FETCH ... INTO
- Użycie tych dwóch operacji ilustruje przykład:

```
DECLARE
    prac_rek1 pracownicy%ROWTYPE;
    prac_rek2 pracownicy%ROWTYPE;
BEGIN
    SELECT nazwisko, imie, wydzial, placa_podstawowa
        INTO prac_rek1 FROM pracownicy
        WHERE wydzial = 30;
    prac_rek2 := prac_rek1;
    ...
END;
```

PL/SQL niejawnie deklaruje rekord w pętli FOR dla kursora. Sytuację tę ilustruje przykład:

```
DECLARE
    CURSOR prac_kursor IS
        SELECT nazwisko, imie, wydzial, placa_podstawowa
            FROM pracownicy;
BEGIN
    FOR pracownik IN prac_kursor LOOP
        suma := suma + pracownik.placa_podstawowa;
        ...
    END LOOP;
```

Niejawnie deklarowanym rekordem jest tu zmienna o nazwie pracownik. Zmienna ta jest automatycznie deklarowana tak, jakby wystąpiła jawna deklaracja postaci nazwa_kursora%ROWTYPE.

6.6. Obsługa błędów

6.6.1. Informacje podstawowe

Błędy podczas wykonania programu powodowane są wieloma różnymi przyczynami. Wśród nich można wymienić następujące: błędy projektowe, błędy kodowania, uszkodzenia sprzętu, niewłaściwe dane itp. Nie jest możliwe przewidzenie wszystkich możliwych błędów, można jednak zaplanować obsługę niektórych z nich. W starszych językach programowania błąd taki jak „Przepełnienie stosu” powodował zgłoszenie komunikatu i zakończenie wykonania programu. W nowoczesnych językach (C++, Java, PL/SQL) zmieniło się podejście do obsługi błędów. Języki te udostępniają bowiem mechanizm nazywany obsługą wyjątków, który pozwala zdefiniować akcję wywoływaną w momencie wystąpienia błędu i dalej kontynuować wykonanie programu.

Wyjątkiem nazywamy spełnienie warunków wystąpienia błędów. Wyjątki dzielą się na predefiniowane (przez twórców języka) i definiowane przez użytkownika. Przykładami wyjątków predefiniowanych mogą być: „Out of memory”, „Division by zero”. W języku PL/SQL użytkownik może definiować wyjątki w części deklaracyjnej bloku PL/SQL.

Przykładowo możliwe jest zdefiniowanie wyjątku „Płaca poniżej minimalnej”, aby wskazać, że proponowana płaca jest zbyt niska. Gdy zachodzi błąd, to wyjątek jest wywoływany (raise), tzn. wykonywanie programu zostaje przerwane i sterowanie jest przekazywane do odpowiedniego fragmentu programu, którego zadaniem jest obsługa danego wyjątku (funkcji obsługi wyjątku). Wyjątki predefiniowane są wykrywane i wywoływane automatycznie. Wyjątki użytkownika muszą być wywołane jawnie za pomocą rozkazu RAISE. W momencie zaistnienia wyjątku wykonanie aktualnego bloku kończy się, następnie wywołuje się funkcję obsługi tego wyjątku i sterowanie jest zwracane do następnej instrukcji w bloku **zawierającym** blok, w którym wystąpił wyjątek. Jeśli taki blok nie istnieje to sterowanie jest zwracane do systemu.

Przykładowy fragment programu zawierający wyjątek „dzielenie przez zero” (ZERO_DIVIDE) oraz jego obsługę obliczający wskaźnik giełdowy C/Z:

```
DECLARE
  cz_wsk NUMBER(3,1);
BEGIN
  ...
  SELECT cena / zysk FROM akcje
    WHERE nazwa = 'ABC'; -- może wywołać wyjątek
                        -- ZERO_DIVIDE
  INSERT INTO informacje (nazwa, c_z)
    VALUES ('ABC', cz_wsk);
  COMMIT

EXCEPTION
  WHEN ZERO_DIVIDE THEN
    INSERT INTO informacje (nazwa, c_z)
      VALUES ('ABC', NULL);
    COMMIT;
  ...
  WHEN OTHERS THEN
    ROLLBACK;

END;
```

Posługiwanie się wyjątkami ma wiele zalet. Za pomocą uprzednio stosowanych technik kontrola wystąpienia błędów była bardzo złożona i prowadziła do znacznego skomplikowania kodu programu. W szczególności konieczne mogło być sprawdzanie poprawności wykonania każdego rozkazu:

```
BEGIN
  SELECT ...
  -- kontrola błędu „brak danych”
  SELECT ...
  -- kontrola błędu „brak danych”
  SELECT ...
  -- kontrola błędu „brak danych”
END;
```


Ponadto kod obsługi błędu nie był odseparowany od kodu wykonywanego normalnie, co zmniejszało przejrzystość programu i powodowało, że algorytmy stawały się nieczytelne.

Ten sam problem można znacznie prościej i łatwiej rozwiązać za pomocą wyjątków:

```
BEGIN
    SELECT ...
    SELECT ...
    SELECT ...
EXCEPTION
    WHEN NO_DATA_FOUND THEN
        ... -- obsługa wszystkich błędów „brak danych”
END;
```

Wyjątki nie tylko zwiększają czytelność programu i upraszczają jego konstrukcję. Zapewniają również, że wszystkie błędy zostaną obsłużone. Jeśli odpowiedniej funkcji nie ma w aktualnym bloku, to wyjątek jest przekazywany do bloku nadrzędnego, aż do znalezienia funkcji obsługi lub powrotu do systemu.

6.6.2. Wyjątki predefiniowane

Twórcy języka PL/SQL zdefiniowali zestaw wyjątków, związanych z systemem zarządzania bazą danych i językiem. Wyjątki te wywoływane są automatycznie w momencie zajścia odpowiednich warunków. Wybrane z nich przedstawione są poniżej:

- **CURSOR_ALREADY_OPEN** - wywoływany w przypadku próby otwarcia kursora już otwartego;
- **DUP_VAL_ON_INDEX** - wywoływany w przypadku próby wykonania rozkazu INSERT lub UPDATE, który spowodowałby utworzenie dwóch takich samych wierszy w indeksie zadeklarowanym jako UNIQUE;
- **INVALID_CURSOR** - wywoływany w przypadku próby dostępu do nieprawidłowego kursora (np. nie otwartego);
- **INVALID_NUMBER** - wywoływany w przypadku próby wykonania konwersji do typu numerycznego z tekstu, który nie reprezentuje liczby;
- **NO_DATA_FOUND** - wywoływany wtedy, gdy rozkaz SELECT powinien zwrócić jeden wiersz a nie zwraca żadnego (np. SELECT ... INTO);
- **STORAGE_ERROR** - wywoływany w przypadku braku wolnej pamięci lub uszkodzenia zawartości pamięci;
- **TOO_MANY_ROWS** - wywoływany w przypadku, gdy rozkaz SELECT zwraca więcej niż jeden wiersz, a oczekiwany jest tylko jeden (np. SELECT ... INTO);
- **VALUE_ERROR** - wywoływany w przypadku przypisania złej wartości do zmiennej lub pola;
- **ZERO_DIVIDE** - próba dzielenia przez zero;

6.6.3. Obsługa wyjątków

Aby obsłużyć („złapać”) wyjątek konieczne jest napisanie własnej funkcji obsługi tego wyjątku. Dokonuje się tego w części bloku PL/SQL rozpoczynającej się słowem kluczowym EXCEPTION, które występuje zawsze na końcu bloku. Każda funkcja

obsługi wyjątku składa się ze słowa WHEN, po którym podaje się nazwę wyjątku oraz słowa THEN, po którym występuje ciąg instrukcji wykonywanych w momencie zajścia podanego błędu. Funkcja obsługi wyjątku kończy wykonanie bloku, w związku z czym nie jest możliwy powrót do miejsca, w którym błąd wystąpił. Opcjonalne słowo OTHERS (zamiast nazwy wyjątku) pozwala zdefiniować funkcję obsługi wszystkich pozostałych wyjątków (tzn. nie wymienionych wcześniej). Ostatecznie część EXCEPTION wygląda następująco:

```
EXCEPTION
  WHEN ... THEN
    -- obsługa wyjątku
  WHEN ... THEN
    -- obsługa wyjątku
  WHEN ... THEN
    -- obsługa wyjątku
  WHEN OTHERS THEN
    -- obsługa pozostałych wyjątków
END;
```

Jeśli zachodzi potrzeba przypisania tej samej akcji różnym wyjątkom, to można nazwy tych wyjątków wypisać w klauzuli WHEN oddzielając słowem OR:

```
WHEN over_limit OR under_limit OR VALUE_ERROR THEN
...
```

Nie można jednak użyć słowa OTHERS w takiej liście. Słowo OTHERS zawsze musi wystąpić oddzielnie. Należy pamiętać również, że dla jednego wyjątku może być zdefiniowana tylko jedna funkcja obsługi w danym bloku.

W funkcjach obsługi wyjątków mają zastosowanie normalne reguły przesłaniania tzn. widoczne są tylko zmienne globalne lub lokalne.

6.6.4. Wyjątki zdefiniowane przez użytkownika

Język PL/SQL pozwala użytkownikowi na definiowanie swoich własnych wyjątków. Wyjątki takie muszą być jawnie zadeklarowane i w przypadku zajścia odpowiednich warunków, jawnie wywołane za pomocą rozkazu RAISE.

Wyjątki deklaruje się podobnie jak zmienne, z tą różnicą, że zamiast nazwy typu występuje słowo EXCEPTION. Deklarowanie wyjątku ilustruje następujący przykład:

```
DECLARE
  overflow EXCEPTION;
  result NUMBER(5);
BEGIN
  ...
END;
```

Należy zwrócić uwagę, że wyjątek w języku PL/SQL nie jest obiektem (w przeciwieństwie do zmiennych), ale informacją o spełnieniu pewnych określonych warunków. W związku z tym do wyjątku nie jest możliwe przypisanie żadnej wartości, ani skojarzenie z nim dodatkowej informacji. Wyjątek nie może być również używany w rozkazach SQL.

Nie jest możliwa deklaracja tego samego wyjątku dwa razy w tym samym bloku. Można jednak zadeklarować ten sam wyjątek w różnych blokach.

Jak podano wcześniej wyjątki predefiniowane wywoływane są przez system automatycznie. Wyjątki zdefiniowane przez użytkownika, muszą być przez niego wywołane. Służy do tego rozkaz RAISE. Użycie tego rozkazu ilustruje przykład:

```
DECLARE
    brak_czesci EXCEPTION;
    liczba_czesci NUMBER(4);
BEGIN
    ...
    IF liczba_czesci < 1 THEN
        RAISE brak_czesci;
    END IF;
    ...
EXCEPTION
    WHEN brak_czesci THEN
        -- obsługa błędu „brak części”
END;
```

Możliwe jest również jawne (za pomocą rozkazu RAISE) wywołanie predefiniowanych wyjątków:

```
RAISE INVALID_NUMBER;
```

Czasami istnieje konieczność powtórnego wywołania wyjątku z funkcji, która go obsługuje, w celu przekazania go do bloku nadrzędnego. Przykładem może tu być wycofanie transakcji w bloku zagnieżdżonym i zgłoszenie informacji o błędzie w bloku nadrzędnym. W związku z tym możliwe jest użycie rozkazu RAISE w funkcji obsługi wyjątku. Należy pamiętać, że wyjątki zgłoszone w funkcji obsługi innego wyjątku są zawsze przekazywane do bloku nadrzędnego i tam wyszukiwana jest odpowiednia funkcja obsługi zgodnie z zasadami opisanymi wcześniej. Podobnie wyjątki zgłaszane w części deklaracyjnej przekazywane są do bloku nadrzędnego i tam podlegają przetwarzaniu.

6.7. Rozkazy języka PL/SQL

6.7.1. Rozkaz OPEN

Rozkaz OPEN wykonuje zapytanie skojarzone z jawnie zadeklarowanym kursorem i alokuje niezbędne zasoby potrzebne do wykonywania dalszych operacji. Cursor ustawiany tuż przed pierwszym wierszem wyniku zapytania.

Składnia:

```
OPEN cursor_name
    [(input_parameter [, input_parameter] ... )] ;
```

Parametry:

- `cursor_name` - nazwa kursora uprzednio zadeklarowanego, który nie jest aktualnie otwarty.
- `input_parameter` - wyrażenie języka PL/SQL, które przekazywane jest do kursora. Jest ono najczęściej używane do wykonania zapytania (najczęściej stosowane jest w klauzuli WHERE).

Parametry w rozkazie OPEN mogą być użyte tylko wtedy, gdy odpowiednia ilość parametrów została podana w deklaracji kursora. Ilość parametrów w instrukcji OPEN musi być równa ilości parametrów w deklaracji kursora. Parametry instrukcji OPEN służą tylko i wyłącznie do wczytywania danych do kursora i nie mogą być stosowane w celu pobrania ich z kursora.

Przyporządkowanie parametrów aktualnych (w instrukcji OPEN) do parametrów formalnych (w deklaracji kursora) może odbywać się na dwa sposoby:

- ♦ przyporządkowanie przez pozycję
- ♦ przyporządkowanie przez nazwę

W pierwszym przypadku wartość wyrażenia na odpowiedniej pozycji w instrukcji OPEN jest przyporządkowywana parametrowi znajdującemu się na tej samej pozycji w deklaracji kursora. W drugim przypadku parametry mogą być podane w dowolnej kolejności, ale każde wyrażenie musi być poprzedzone nazwą parametru formalnego i znakami '=>'. Sposób użycia obu możliwości ilustrują przykłady:

```
DECLARE
  CURSOR prac_kur(nazw CHAR, wydz NUMBER) IS ...
BEGIN
  OPEN prac_kur('Kowalski', 10);
  ...
  OPEN prac_kur(wydz => 15, nazw => 'Nowak');
END;
```

Można używać równocześnie przyporządkowania przez pozycję i przez nazwę, należy wtedy jednak pamiętać, że parametry przyporządkowywanej przez pozycję muszą wystąpić przed parametrami przyporządkowywanymi przez nazwę.

6.7.2. Rozkaz CLOSE

Rozkaz CLOSE służy do zamknięcia aktualnie otwartego kursora. Każdy kursor przed ponownym otwarciem musi zostać zamknięty. Rozkaz CLOSE zwalnia wszystkie zasoby przydzielone do obsługi kursora.

Składnia:

```
CLOSE cursor_name ;
```

Parametry:

- cursor_name - nazwa aktualnie otwartego kursora.

6.7.3. Rozkaz FETCH

Rozkaz FETCH zwraca następny wiersz danych z aktywnego zbioru (danych spełniających warunek rozkazu SELECT w kursorze). Odczytane informacje przechowywane są w zmiennych. Zwrócone dane odpowiadają zawartości kolejnych kolumn w aktualnym wierszu.

Składnia:

```
FETCH cursor_name INTO
  { record_name |
    variable_name [, variable_name] ... } ;
```

Parametry:

- cursor_name - nazwa aktualnie otwartego kursora.
- variable_name - prosta zmienna, do której zostaną zapisane dane. Wszystkie zmienne na liście muszą być uprzednio zadeklarowane. Dla każdej kolumny w kursorze, musi wystąpić odpowiadająca jej zmienna. W dodatku typy kolumn muszą być takie same jak odpowiadające im typy zmiennych lub konwertowalne do nich.
- record_name - nazwa zmiennej będącej rekordem (deklarowanej z użyciem atrybutu %ROWTYPE).

Przykład:

```
...
OPEN prac_kursor;
...
LOOP
    FETCH prac_kursor INTO prac_rek;
    EXIT WHEN prac_kursor%NOTFOUND;
    ...
END LOOP;
```

6.7.4. Rozkaz SELECT ... INTO

Rozkaz SELECT ... INTO odczytuje informacje z bazy danych i zapisuje je do zmiennych. W języku PL/SQL standardowy (z SQL) rozkaz SELECT został rozszerzony o klauzulę INTO. Aby rozkaz ten działał poprawnie konieczne jest by SELECT zwracał tylko jeden wiersz (w przypadku wielu wierszy należy zadeklarować kursor i za jego pomocą odczytywać dane).

Składnia rozkazu SELECT z klauzulą INTO:

```
SELECT select_list_item [, select_list_item] ... INTO
    { record_name |
      variable_name [, variable_name] ... }
    rest_of_select_statement ;
```

Parametry:

Zobacz opis rozkazu FETCH.

Przykład:

```
SELECT nazwisko, placa*12 INTO pnazw, plac_sum
    FROM pracownicy
    WHERE pracownik_nr = 12345;
```

6.7.5. Rozkaz IF

Rozkaz IF pozwala na warunkowe wykonywanie rozkazów.

Składnia:

```
IF plsql_condition THEN seq_of_statements
    [ELSEIF plsql_condition THEN seq_of_statements]
    ...
    [ELSE seq_of_statements]
END IF;
```

Parametry:

- plsql_condition - warunek (wyrażenie obliczane do wartości logicznej)

- seq_of_statements - ciąg instrukcji, które mają być wykonane w razie spełnienia (bądź nie spełnienia) podanego warunku

Opis:

Rozkaz IF pozwala uzależnić wykonanie rozkazów od wyników obliczania warunku (lub warunków). Jeśli pierwszy z warunków jest prawdziwy, to wykonywany jest ciąg instrukcji po THEN, aż do napotkania odpowiedniego ELSEIF, ELSE lub END IF, a następnie sterowanie przekazywane jest do najbliższego rozkazu po odpowiednim END IF. W przypadku, gdy warunek nie jest spełniony, to sprawdzany jest warunek w pierwszym ELSEIF. Jeśli ten warunek jest spełniony, to wykonywany jest ciąg instrukcji po odpowiadającym mu THEN i wykonanie instrukcji IF się kończy. Jeśli jednak ten warunek również nie jest spełniony, to sprawdzany jest warunek w następnym ELSEIF. Jeśli żaden z warunków nie jest prawdziwy, to wykonywany jest ciąg instrukcji po ELSE (jeśli istnieje). Wynik obliczania warunku równy NULL jest traktowany jako niespełnienie tego warunku.

Przykład:

```
IF liczba_czesci < 20 THEN
    ilosc_zamawianych := 50;
ELSEIF liczba_czesci < 30 THEN
    ilosc_zamawianych := 20;
ELSE
    ilosc_zamawianych := 5;
END IF;
INSERT INTO zamowienia
VALUES(typ_czesci, ilosc_zamawianych);
```

6.7.6. Rozkaz LOOP

Rozkaz LOOP umożliwia tworzenie pętli w języku PL./SQL. Dopuszczalne są cztery rodzaje pętli:

- pętle podstawowe
- pętle WHILE
- pętle FOR numeryczne
- pętle FOR dla kursorów

Składnia:

```
[<<label_name>>]
[ { WHILE plsql_condition } |
  { FOR {numeric_loop_param | cursor_loop_param } } ]
LOOP seq_of_statements END LOOP [ label_name ] ;
```

Składnia numeric_loop_param:

```
index IN [REVERSE] integer_expr .. integer_expr
```

Składnia cursor_loop_param:

```
record_name IN
    { cursor_name [(parameter [, parameter] ...)] |
      ( select_statement ) }
```

Parametry:

- `label_name` - ten parametr pozwala na opcjonalne nazwanie pętli. Możliwe jest wtedy użycie nazwy pętli w rozkazie EXIT w celu określenie, z której pętli powinno nastąpić wyjście. Ponadto możliwy jest dostęp do indeksu pętli zewnętrznej w pętli wewnętrznej, jeśli indeksy te mają tę samą nazwę za pomocą konstrukcji:

`label_name.index`

- `seq_of_statements` - ciąg rozkazów, które będą powtarzane w pętli
- `plsql_condition` - warunek języka PL./SQL. W pętli WHILE warunek ten jest obliczany przed każdą iteracją. Odpowiedni ciąg instrukcji wykonuje się tylko wtedy, gdy warunek ten ma wartość TRUE. W przeciwnym wypadku sterowanie przechodzi do pierwszej instrukcji za pętlą.
- `index` - zmienna sterująca pętli FOR. Nie jest konieczna wcześniejsza jej deklaracja.
- `integer_expr` - wyrażenie, którego wynikiem jest liczba całkowita. Wyrażenie to jest obliczane tylko przy pierwszym wejściu do pętli FOR.
- `REVERSE` - klauzula, nakazująca zmniejszać `index` (zamiast zwiększania).
- `cursor_name` - nazwa uprzednio zadeklarowanego kursora. W momencie wejścia do pętli FOR kursor jest automatycznie otwierany.
- `parameter` - jeden z parametrów otwarcia kursora (jeśli kursor został zadeklarowany z parametrami).
- `select_statement` - zapytanie związane z wewnętrznym kursorem, niedostępnym dla użytkownika. PL./SQL automatycznie tworzy, otwiera i pobiera dane z kursora (a następnie zamyka go).

Opis:

Instrukcje w pętli podstawowej są powtarzane bez sprawdzania żadnych warunków. Twórca programu jest odpowiedzialny za zakończenie pętli instrukcją EXIT. Przykład pętli podstawowej:

```
<<loop1>> LOOP
    ...
    IF (x > 10) THEN EXIT loop1;
    ...
END LOOP loop1;
```

Pętla WHILE pozwala powtarzać ciąg instrukcji, dotąd dopóki podany warunek jest prawdziwy. Warunek jest obliczany przed każdym powtórzeniem ciągu instrukcji w pętli. W związku z tym ciąg instrukcji może nie wykonać się ani raz. Przykład:

```
WHILE x < 10 LOOP
    ...
    x := x - y;
    ...
END LOOP;
```

Pętla FOR pozwala powtarzać podany ciąg instrukcji określoną ilość razy. Do stwierdzenia, które powtórzenie jest aktualnie wykonywane służy zmienna sterująca nazywana indeksem. Indeks może być zwiększany lub zmniejszany. Przykład:

```
FOR i IN 1 .. n LOOP
    silnia := silnia * n;
```

```
END LOOP;
```

Pętlę FOR można stosować również w celu odczytywania kolejnych wierszy z kursora (lub zapytania). Ciąg instrukcji w pętli wykonywany jest wtedy, dla każdego wiersza. Przykład:

```
DECLARE
  CURSOR prac_kursor IS select * FROM pracownicy;
  prac_rek prac_kursor%ROWTYPE;

BEGIN
  ...
  FOR prac_rek IN prac_kursor LOOP
    suma := suma + prac_rek.placa_podstawowa;
  END LOOP;
  ...
END;
```

6.7.7. Rozkaz EXIT

Rozkaz EXIT służy do wyjścia z pętli. Rozkaz ten ma dwie formy: bezwarunkową i warunkową.

Składnia:

```
EXIT [label_name] [WHEN plsql_condition] ;
```

Parametry:

- `label_name` - opcjonalna nazwa pętli, z której ma nastąpić wyjście. Jeśli nazwa nie jest podana, to rozkaz EXIT powoduje wyjście z najbardziej zagnieżdżonej pętli aktualnie wykonywanej.
- `plsql_condition` - używany w instrukcji EXIT warunkowej. Musi być poprawnym warunkiem języka PL/SQL. Wyjście następuje tylko wtedy, gdy wynikiem obliczenia warunku jest wartość TRUE.

6.7.8. Rozkaz GOTO

Rozkaz GOTO służy do natychmiastowego przekazania sterowania od rozkazu aktualnego do pierwszego rozkazu występującego po podanej etykiecie.

Składnia deklaracji etykiety:

```
<< label_name >>
```

Składnia rozkazu GOTO:

```
GOTO label_name ;
```

Opis:

Rozkaz GOTO umożliwia przeniesienie sterowania do innego miejsca w tym samym bloku lub bloku nadrzędnym, ale nie do funkcji obsługi wyjątku. Z funkcji obsługi wyjątku możliwy jest skok do bloku nadrzędnego, ale nie do bloku aktualnego. Nie jest możliwe również przeniesienie sterowania do pętli z zewnątrz. Jeśli rozkaz GOTO używany jest w celu opuszczenia pętli FOR dla kursora, to kursor zamykany jest automatycznie.

7. Literatura

1. Wojciech Cellary, Zbyszko Królikowski “Wprowadzenie do projektowania baz danych dBase III”, Wydawnictwa Naukowo -Techniczne, Warszawa 1988
2. ORACLE SQL Language Reference Manual
3. ORACLE PL/SQL User’s Guide and Reference
4. ORACLE SQL*Plus User’s Guide and Reference