Projekt jest finansowany
ze środków Unii Europejskiej, Europejskiego Funduszu
Społecznego i budżetu państwa

Studia Podyplomowe dla Nauczycieli

Bazy danych

SQL

Języki baz danych

Interfejs DBMS składa się z podjęzyka bazy danych. Jest to język programowania przeznaczony specjalnie do inicjowania funkcji DBMS. Składa się on z trzech części:

- **Język definiowania danych (DDL)**

  Język definiowania danych jest stosowany do tworzenia i usuwania struktur danych oraz do uzupełniania istniejących struktur. DDL aktualizuje metadane przechowywane w słowniku danych.
- **Język operowania danymi (DML)**

  Język operowania danymi jest używany do określania poleceń, które realizują działania CRUD na bazie danych. DML jest podstawowym mechanizmem stosowanym przy określaniu transakcji wykonywanych na bazie danych.
- **Język kontroli danych (DCL)**

  Język kontroli danych jest przeznaczony do wykorzystania przez administratora bazy. Jest on stosowany zwłaszcza do definiowania użytkowników bazy danych oraz przyznawania im uprawnień.
- Głównym przykładem takiego podjęzyka jest strukturalny **język zapytań SQL**.

Język zapytań SQL - historia

- SQL = **S**tructured **Q**uery **L**anguage (Strukturalny Język Zapytań)
- Opracowany w latach siedemdziesiątych przez firmę IBM.
- Po raz pierwszy zaimplementowany przez IBM w prototypowym, relacyjnym produkcie nazwanym System R (1973).
- Pierwszy komercyjny relacyjny system zarządzania bazami danych wprowadziła na rynek w 1977 firma ORACLE.
- W 1986 powstał pierwszy standard języka. Drugi został opracowany w 1989 a trzeci w 1992 (SQL 92).
- Komercyjne implementacje wprowadzają często własne konstrukcje, jak również czasami nie zachowują pełnej zgodności z wybranym standardem.

SQL - DML

- Najczęściej używaną częścią języka SQL jest część dotycząca manipulowania danymi.
- Najważniejszą instrukcją jest instrukcja służąca do wydobywania informacji z bazy danych – instrukcja SELECT.

Przykładowa tabela

OSOBY(id, imie, nazwisko, nr, wiek, plec)

	id	imie	nazwisko	nr	wiek	plec
	1	jan	kowalski	7654	34	1
	2	piotr	nowak	4567	92	1
	3	anna	dymna	5674	19	0
	4	ala	okos	9618	48	0
	5	pawel	nijaki	3456	14	1
	6	jan	kowalski	2432	15	1
	7	piotr	babel	1358	16	1
	8	iwona	wolna	2435	77	0
	9	ala	nowak	9879	27	0
	10	ola	nowak	7659	45	0
	11	anna	okos	1111	34	0

SELECT

```
SELECT *
FROM Osoby
```

id	imie	nazwisko	nr	wiek	plec
1	jan	kowalski	7654	34	1
2	piotr	nowak	4567	92	1
3	anna	dymna	5674	19	0
4	ala	okos	9618	48	0
5	pawel	nijaki	3456	14	1
6	jan	kowalski	2432	15	1
7	piotr	babel	1358	16	1
8	iwona	wolna	2435	77	0
9	ala	nowak	9879	27	0
10	ola	nowak	7659	45	0
11	anna	okos	1111	34	0

Przykład. Wyrażenie wybiera z tabeli OSOBY wszystkie wiersze i kolumny.

SELECT

Przykład. Instrukcja wybiera z tabeli OSOBY wszystkie wiersze i jedną kolumnę (*nazwisko*)

```
SELECT nazwisko
FROM Osoby
```

nazwisko
kowalski
nowak
dymna
okos
nijaki
kowalski
babel
wolna
nowak
nowak
okos
*

SELECT

Przykład. Wyrażenie wybiera z tabeli OSOBY wszystkie wiersze i dwie kolumny (*imię* oraz *nazwisko*).

```
SELECT imie, nazwisko
FROM Osoby
```

imie	nazwisko
jan	kowalski
piotr	nowak
anna	dymna
ala	okos
pawel	nijaki
jan	kowalski
piotr	babel
iwona	wolna
ala	nowak
ola	nowak
anna	okos

SELECT a operacja rzutu

- Instrukcja postaci

  SELECT nazwa_kolumny_1, ..., nazwa_kolumny_k

  FROM nazwa_tabeli

  jest częściowym odpowiednikiem operacji rzutu z algebry relacyjnej.
- Częściowym, ponieważ wynikiem rzutu relacji jest nowa relacja, a relacja jest zbiorem krotek, więc nie może w nim wystąpić kilka razy ten sam element.
- Wynikiem instrukcji SELECT może być natomiast tabela, w której znajduje się kilka takich samych wierszy.
- Aby otrzymać prawdziwie relacyjny wynik (czyli tabelę bez powtarzających się wierszy), należy użyć słowa kluczowego DISTINCT (zaraz po klauzuli SELECT).

SELECT - DISTINCT

```
SELECT plec
FROM Osoby
```

plec
1
1
0
0
1
1
1
0
0
0
0

```
SELECT DISTINCT plec
FROM Osoby
```

plec
0
1

SELECT - aliasy

- Domyślne nagłówki kolumn możemy zastąpić innymi nazwami, które będą bardziej znaczące.
- Alias podaje się bezpośrednio po nazwie kolumny, której nazwę chcemy zmienić. Pomiędzy nazwami można użyć słowa kluczowego AS

```
SELECT nr AS polisa
FROM Osoby
```

polisa
7654
4567
5674
9618
3456
2432
1358
2435
9879
7659
1111

```
SELECT nazwisko, nr polisa
FROM Osoby
```

nazwisko	polisa
kowalski	7654
nowak	4567
dymna	5674
okos	9618
nijaki	3456
kowalski	2432
babel	1358
wolna	2435
nowak	9879
nowak	7659
okos	1111

SELECT – ORDER BY

- Aby określić kolejność, w jakiej będą zwracane wyniki, należy użyć klauzuli ORDER BY (uporządkuj wg). Klauzula ORDER BY musi być ostatnią klauzulą polecenia SELECT.
- Kolumnę sortującą można wyznaczyć przez nazwę:

```
SELECT *
FROM Osoby
ORDER BY wiek
```

id	imie	nazwisko	nr	wiek	plec
5	pawel	nijaki	3456	14	1
6	jan	kowalski	2432	15	1
7	piotr	babel	1358	16	1
3	anna	dymna	5674	19	0
9	ala	nowak	9879	27	0
1	jan	kowalski	7654	34	1
11	anna	okos	1111	34	0
10	ola	nowak	7659	45	0
4	ala	okos	9618	48	0
8	iwona	wolna	2435	77	0
2	piotr	nowak	4567	92	1

SELECT – ORDER BY

- Kolumnę sortującą można również wyznaczyć przez liczbę. Wskazuje ona numer kolumny sortującej w tabeli wynikowej.

```
SELECT *
FROM Osoby
ORDER BY 3
```

id	imie	nazwisko	nr	wiek	plec
7	piotr	babel	1358	16	1
3	anna	dymna	5674	19	0
1	jan	kowalski	7654	34	1
6	jan	kowalski	2432	15	1
5	pawel	nijaki	3456	14	1
2	piotr	nowak	4567	92	1
9	ala	nowak	9879	27	0
10	ola	nowak	7659	45	0
4	ala	okos	9618	48	0
11	anna	okos	1111	34	0
8	iwona	wolna	2435	77	0

```
SELECT id, nazwisko, imie
FROM Osoby
ORDER BY 2
```

id	nazwisko	imie
7	babel	piotr
3	dymna	anna
1	kowalski	jan
6	kowalski	jan
5	nijaki	pawel
2	nowak	piotr
9	nowak	ala
10	nowak	ola
4	okos	ala
11	okos	anna
8	wolna	iwona

W obydwu przypadkach sortujemy wg kolumny *nazwisko*.

SELECT – ORDER BY

- Liczby trzeba użyć wtedy, gdy kolumna sortująca jest wyrażeniem i nie posiada aliasu.

```
SELECT id, nazwisko, nr, 2 * nr
FROM Osoby
ORDER BY 4
```

id	nazwisko	nr	
11	okos	1111	2222
7	babel	1358	2716
6	kowalski	2432	4864
8	wolna	2435	4870
5	nijaki	3456	6912
2	nowak	4567	9134
3	dymna	5674	11348
1	kowalski	7654	15308
10	nowak	7659	15318
4	okos	9618	19236
9	nowak	9879	19758

SELECT – ORDER BY

- Domyślnie dane są sortowane w porządku rosnącym (ASCENDING).
- Aby odwrócić kolejność sortowania należy użyć słowa DESC (DESCENDING) użytego bezpośrednio po nazwie kolumny wyspecyfikowanej w klauzuli ORDER BY.

```
SELECT id, imie, nazwisko
FROM Osoby
ORDER BY nazwisko ASC
```

id	imie	nazwisko
7	piotr	babel
3	anna	dymna
1	jan	kowalski
6	jan	kowalski
5	pawel	nijaki
2	piotr	nowak
9	ala	nowak
10	ola	nowak
4	ala	okos
11	anna	okos
8	iwona	wolna

```
SELECT id, imie, nazwisko
FROM Osoby
ORDER BY nazwisko DESC
```

id	imie	nazwisko
8	iwona	wolna
4	ala	okos
11	anna	okos
2	piotr	nowak
9	ala	nowak
10	ola	nowak
5	pawel	nijaki
1	jan	kowalski
6	jan	kowalski
3	anna	dymna
7	piotr	babel

Słowa kluczowego ASC nie trzeba używać, gdyż domyślnie porządek sortowania jest rosnący.

SELECT

- Można sortować według kilku kolumn, wtedy po słowie kluczowym ORDER BY należy podać nazwy kolumn, po których chcemy sortować, przy czym pierwsza kolumna ma najwyższy priorytet.

```
SELECT *
FROM Osoby
ORDER BY nazwisko DESC, imie ASC
```

id	imie	nazwisko	nr	wiek	plec
8	iwona	wolna	2435	77	0
4	ala	okos	9618	48	0
11	anna	okos	1111	34	0
9	ala	nowak	9879	27	0
10	ola	nowak	7659	45	0
2	piotr	nowak	4567	92	1
5	pawel	nijaki	3456	14	1
1	jan	kowalski	7654	34	1
6	jan	kowalski	2432	15	1
3	anna	dymna	5674	19	0
7	piotr	babel	1358	16	1

SELECT - WHERE

- Klauzula WHERE odpowiada operacji selekcji.
- Klauzula ta specyfikuje kryteria doboru wierszy. Klauzula WHERE, o ile jest, musi występować bezpośrednio po klauzuli FROM.

```
SELECT kolumna_1, ..., kolumna_k
FROM tabela
WHERE warunek;
```

- Operatory w klauzuli WHERE mogą być dwojakiego rodzaju:
  - operatory logiczne (=, <, >, <=, >=, <>, AND, OR, NOT)
  - operatory SQL.
- W klauzuli WHERE można również porównywać dla każdego wiersza wartości dwóch kolumn.
- Klauzula WHERE określa, czy dany wiersz ma znaleźć się w tabeli wynikowej, przed jakimkolwiek przetwarzaniem.

SELECT - WHERE

- Przykład. Instrukcja wyszukuje wszystkie osoby z tabeli OSOBY mające więcej niż 20 lat.

```
SELECT imie, nazwisko, wiek
FROM osoby
WHERE wiek>20;
```

Query 1

	imie	nazwisko	wiek
1	jan	kowalski	34
2	piotr	nowak	92
3	ala	okos	48
4	iwona	wolna	77
5	ala	nowak	27
6	ola	nowak	45
7	anna	okos	34

- Przykład. Instrukcja wyszukuje wszystkie kobiety z tabeli OSOBY.
- Można w warunku używać atrybutów, których później nie wyświetlamy.

```
SELECT imie, nazwisko, wiek
FROM osoby
WHERE plec=0;
```

Query 1

	imie	nazwisko	wiek
1	anna	dymna	19
2	ala	okos	48
3	iwona	wolna	77
4	ala	nowak	27
5	ola	nowak	45
6	anna	okos	34

Operatory SQL

Wyróżniamy 4 operatory SQL, działające na wszystkich typach danych:

- BETWEEN ... AND,
- IN (lista),
- LIKE,
- IS NULL.

BETWEEN ... AND

- Operator BETWEEN...AND służy do sprawdzenia, czy wartość znajduje się w podanym przedziale (wliczając w to krańce przedziału). Górna granica musi następować po dolnej.
- Przykład. Instrukcje wybierają osoby z przedziału wieku <20;40>.

```
SELECT imie, nazwisko, wiek
FROM osoby
WHERE wiek BETWEEN 20 AND 40;
```

Query 1

	imie	nazwisko	wiek
1	jan	kowalski	34
2	ala	nowak	27
3	anna	okos	34

```
SELECT imie, nazwisko, wiek
FROM osoby
WHERE wiek>=20 AND wiek<=40;
```

Query 1

	imie	nazwisko	wiek
1	jan	kowalski	34
2	ala	nowak	27
3	anna	okos	34

IN

- Operator IN służy do sprawdzania, czy dana wartość znajduje się na wyspecyfikowanej liście.
- Przykład. Instrukcje wybierają osoby z nazwiskiem *Nowak* lub *Kowalski*.

```
SELECT imie, nazwisko, wiek
FROM osoby
WHERE nazwisko IN ('Nowak', 'Kowalski');
```

Query 1

	imie	nazwisko	wiek
1	jan	kowalski	34
2	piotr	nowak	92
3	jan	kowalski	15
4	ala	nowak	27
5	ola	nowak	45

```
SELECT imie, nazwisko, wiek
FROM osoby
WHERE nazwisko ='Nowak' OR nazwisko= 'Kowalski';
```

Query 1

	imie	nazwisko	wiek
1	jan	kowalski	34
2	piotr	nowak	92
3	jan	kowalski	15
4	ala	nowak	27
5	ola	nowak	45

LIKE

- Operator LIKE służy do wybierania wartości odpowiadających podanemu wzorcowi. Wzorzec tworzą dwa specjalne symbole:
  - % (znak procent) — odpowiada dowolnemu ciągowi znaków (również pustemu),
  - _ (znak podkreślenia) — odpowiada dokładnie jednemu dowolnemu znakowi.
- Przykład. Instrukcje wybierają osoby
  - z nazwiskiem 5-literowym zawierającym „*owa*”,
  - z nazwiskiem co najmniej 4-literowym zawierającym „*owa*”.

```
SELECT imie, nazwisko, wiek
FROM osoby
WHERE nazwisko LIKE '_owa_';
```

Query 1

	imie	nazwisko	wiek
1	piotr	nowak	92
2	ala	nowak	27
3	ola	nowak	45

```
SELECT imie, nazwisko, wiek
FROM osoby
WHERE nazwisko LIKE '_owa%';
```

Query 1

	imie	nazwisko	wiek
1	jan	kowalski	34
2	piotr	nowak	92
3	jan	kowalski	15
4	ala	nowak	27
5	ola	nowak	45

IS NULL

- Operator **`IS NULL`** służy do wyszukiwania wartości NULL.
  - x IS NULL – zwraca TRUE, gdy dana wartość jest NULL
  - x IS NOT NULL – zwraca TRUE, gdy dana wartość nie jest NULL.
- Przykład. Instrukcja wybiera osoby bez podanego numeru.

```
SELECT imie, nazwisko, nr, wiek
FROM osoby
WHERE nr IS NULL;
```

Query 1

	imie	nazwisko	nr	wiek
1	anna	okos	[NULL]	34

Dobrze!

```
SELECT imie, nazwisko, nr, wiek
FROM osoby
WHERE nr = NULL;
```

Query 1

imie	nazwisko	nr	wiek

Źle!!!

Funkcje grupujące

- Funkcje grupujące służą do działania na grupach wierszy. Wynikiem funkcji grupującej jest pojedyncza wartość dla całej grupy.
- Przykładowe funkcje grupujące, to:

Funkcja	Wynik funkcji
COUNT ([**DISTINCT** | **ALL**] wyrażenie)	ilość wystąpień wartości wyrażeń różnych od NULL, gwiazdka (*) użyta w miejscu wyrażenia powoduje obliczenia ilości wszystkich wierszy łącznie z duplikatami i wartościami NULL
AVG ([**DISTINCT** | **ALL**] wyrażenie)	wartość średnia wyrażeń, NULL nie jest uwzględniane
MAX ([**DISTINCT** | **ALL**] wyrażenie)	maksymalna wartość wyrażenia
MIN ([**DISTINCT** | **ALL**] wyrażenie)	minimalna wartość wyrażenia
SUM ([**DISTINCT** | **ALL**] wyrażenie)	suma wartości wyrażeń, bez uwzględniania wartości NULL

Funkcje grupujące

- Kwalifikator DISTINCT ogranicza działanie funkcji grupujących do różnych wartości argumentów.
- Kwalifikator ALL jest domyślny — funkcje grupujące nie eliminują duplikatów.
- Argumentami funkcji grupujących są liczby, a w przypadku funkcji MAX, MIN i COUNT także daty, znaki i ciągi znaków.
- Wszystkie funkcje grupujące, za wyjątkiem COUNT(*) ignorują wartości NULL.

Przykładowa tabela

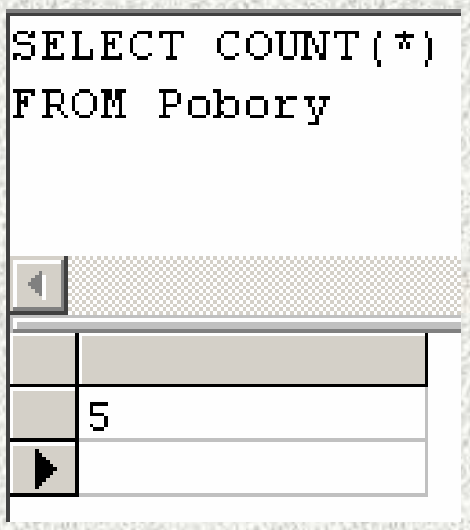
POBORY(id, imie, nazwisko, plec, dzial, pensja)

nr	imie	nazwisko	plec	wiek	dzial	pensja
1	jan	nowak	m	23	2	1500
2	iwona	smyk	k	54	1	5000
3	adam	sztur	m	<NULL>	0	<NULL>
4	anna	sosnowska	k	25	2	2500
5	tomasz	wolny	m	33	1	5000

Funkcje grupujące - przykłady

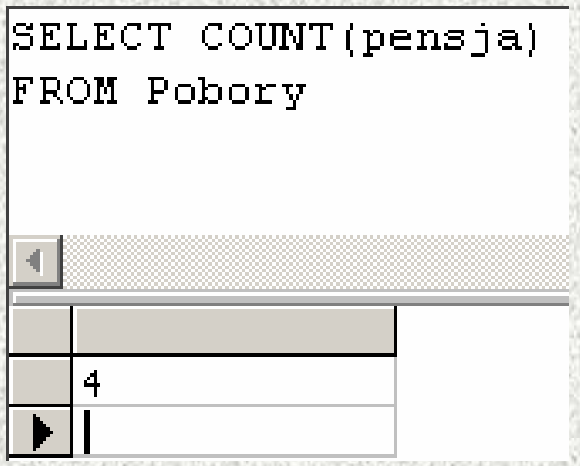
- Przykład. Zliczanie wszystkich wierszy zwracanych przez zapytanie.

```
SELECT COUNT(*)
FROM Pobory
```



- Przykład. Zliczanie tych wierszy zwracanych przez zapytanie, w których wartością atrybutu *Pensja* nie jest NULL.

```
SELECT COUNT(pensja)
FROM Pobory
```



Funkcje grupujące - przykłady

nr	imie	nazwisko	plec	wiek	dzial	pensja
1	jan	nowak	m	23	2	1500
2	iwona	smyk	k	54	1	5000
3	adam	sztur	m	<NULL>	0	<NULL>
4	anna	sosnowska	k	25	2	2500
5	tomasz	wolny	m	33	1	5000

```
SELECT MIN(pensja)
FROM Pobory
```

1500

```
SELECT MAX(nazwisko)
FROM Pobory
```

wolny

```
SELECT SUM(pensja)
FROM Pobory
```

14000

```
SELECT AVG(pensja)
FROM Pobory
```

3500

Funkcje grupujące – GROUP BY

- Znacznie częściej funkcji grupujących używa się do grup wartości.
- Pojedynczą grupę stanowią wszystkie wiersze, dla których wartości dla danych atrybutów są identyczne.
- Do podzielenia wierszy tablicy na grupy używamy klauzuli GROUP BY.
- Wiersze w tabeli wynikowej uporządkowane są wg kolumny wymienionej w klauzuli GROUP BY.
- Na liście wyboru klauzuli SELECT mogą występować tylko i wyłącznie nazwy kolumn, które są przedmiotem działania klauzuli GROUP BY, chyba, że występują one jako argument funkcji grupującej.

GROUP BY - przykłady

nr	imie	nazwisko	plec	wiek	dzial	pensja
1	jan	nowak	m	23	2	1500
2	iwona	smyk	k	54	1	5000
3	adam	sztur	m	<NULL>	0	<NULL>
4	anna	sosnowska	k	25	2	2500
5	tomasz	wolny	m	33	1	5000

```
SELECT COUNT(*) AS Liczba, dzial
FROM Pobory
GROUP BY dzial
```

Liczba	dzial
1	0
2	1
2	2

```
SELECT COUNT(pensja) AS Liczba, dzial
FROM Pobory
GROUP BY dzial
```

Liczba	dzial
0	0
2	1
2	2

GROUP BY

nr	imie	nazwisko	plec	wiek	dzial	pensja
1	jan	nowak	m	23	2	1500
2	iwona	smyk	k	54	1	5000
3	adam	sztur	m	<NULL>	0	<NULL>
4	anna	sosnowska	k	25	2	2500
5	tomasz	wolny	m	33	1	5000

W klauzuli GROUP BY można użyć nazw kilku kolumn. Wówczas najpierw wiersze tabeli są grupowane wg wartości w pierwszej wymienionej kolumnie. Następnie w obrębie każdej grupy wiersze są grupowane wg drugiej kolumny itd..

```
SELECT COUNT(nr) AS Liczba, dzial, pensja
FROM Pobory
GROUP BY dzial, pensja
```

Liczba	dzial	pensja
1	0	<NULL>
1	2	1500
1	2	2500
2	1	5000

HAVING

- Klauzula WHERE określa, czy dany wiersz ma znaleźć się w tabeli wynikowej, przed jakimkolwiek przetwarzaniem.
- Jeżeli chcemy najpierw wykonać obliczenia i pogrupować dane, a dopiero później wybrać niektóre już pogrupowane wiersze, to należy użyć klauzuli HAVING.
- Klauzula HAVING określa, czy dany wiersz ma znaleźć się w tabeli wynikowej, po przetwarzaniu danych.

```
SELECT nazwy_grupowanych_kolumn
FROM nazwa_tabeli
GROUP BY nazwy_kolumn
HAVING warunek;
```

HAVING - przykład

nr	imie	nazwisko	plec	wiek	dzial	pensja
1	jan	nowak	m	23	2	1500
2	iwona	smyk	k	54	1	5000
3	adam	sztur	m	<NULL>	0	7000
4	anna	sosnowska	k	25	2	2500
5	tomasz	wolny	m	33	1	5000

```
SELECT dzial, AVG(pensja)
FROM Pobory
GROUP BY dzial
```

dzial	
0	7000
1	5000
2	2000

```
SELECT dzial, AVG(pensja)
FROM Pobory
GROUP BY dzial
HAVING AVG(pensja) < 4000
```

dzial	
2	2000

Klauzule w instrukcji SELECT

Kolejność wykonywania poszczególnych klauzul w instrukcji SELECT jest następująca:

1. FROM
2. WHERE
3. GROUP BY
4. HAVING
5. SELECT
6. ORDER BY

Pobieranie danych z wielu tabel

- W celu wybrania danych z wielu tabel dokonujemy złączeń.
- Tabelom biorącym udział w złączeniu można nadawać aliasy (inne nazwy).
- Jeżeli nazwy kolumn w tabelach nie są unikalne, trzeba poprzedzić je nazwa tabeli lub jej aliasem.

Iloczyn kartezjański

- W wyniku otrzymujemy złączenia każdego wiersza z jednej tabeli z każdym wierszem z drugiej tabeli.
- Składnia (starsza)

  SELECT kolumny_z_tab_1, kolumny_z_tab_2

  FROM tab_1, tab_2;
- Składnia (nowsza)

  SELECT kolumny_z_tab_1, kolumny_z_tab_2

  FROM tab_1 CROSS JOIN tab_2;

Iloczyn kartezjański - przykład

t1

	nr	nazwisko
1	12	Nowak
2	18	Kowalski
3	35	Brown

t2

	id	nazwisko	telefon
1	1	Nowak	12-15
2	2	Kowalski	13-25
3	3	Kowalski	24-13
4	4	Brown	36-28

```
SELECT t1.*, t2.*
FROM t1, t2;
```

Query 1

	nr	nazwisko	id	nazwisko	telefon
1	12	Nowak	1	Nowak	12-15
2	18	Kowalski	1	Nowak	12-15
3	35	Brown	1	Nowak	12-15
4	12	Nowak	2	Kowalski	13-25
5	18	Kowalski	2	Kowalski	13-25
6	35	Brown	2	Kowalski	13-25
7	12	Nowak	3	Kowalski	24-13
8	18	Kowalski	3	Kowalski	24-13
9	35	Brown	3	Kowalski	24-13
10	12	Nowak	4	Brown	36-28
11	18	Kowalski	4	Brown	36-28
12	35	Brown	4	Brown	36-28

```
SELECT t1.*, t2.*
FROM t1 CROSS JOIN t2;
```

Query 1

	nr	nazwisko	id	nazwisko	telefon
1	12	Nowak	1	Nowak	12-15
2	18	Kowalski	1	Nowak	12-15
3	35	Brown	1	Nowak	12-15
4	12	Nowak	2	Kowalski	13-25
5	18	Kowalski	2	Kowalski	13-25
6	35	Brown	2	Kowalski	13-25
7	12	Nowak	3	Kowalski	24-13
8	18	Kowalski	3	Kowalski	24-13
9	35	Brown	3	Kowalski	24-13
10	12	Nowak	4	Brown	36-28
11	18	Kowalski	4	Brown	36-28
12	35	Brown	4	Brown	36-28

Złączenie wewnętrzne

- W tabeli wynikowej pojawią się tylko pasujące wiersze, czyli wiersze spełniające zadany warunek.
- Przykład. Niech tab_1(A1, A2, ..., An) oraz tab_2(B1, B2, ..., Bm) będą dwiema tabelami. Szukamy w tych tabelach wierszy, które majĄ takie same wartości dla atrybutów A1 i B3.
- Wyrażenie wybierające takie wiersze ma postać (starsza składnia):

```
SELECT tab_1.*, tab_2.*
FROM tab_1, tab_2
WHERE tab_1.A1=tab_2.B3
```

- Nowsza składnia ma postać:

```
SELECT tab_1.*, tab_2.*
FROM tab_1 INNER JOIN tab_2 ON tab_1.A1=tab_2.B3
```

Złączenie wewnętrzne

- Jeżeli tabele mają wspólne kolumny (atrybuty o takich samych nazwach) to można użyć jeszcze dwóch innych postaci.
- Przykład. Niech tab_1(A1, A2, ..., An, C1, C2, C3) oraz tab_2(B1, B2, ..., Bm, C1, C2, C3) będą dwiema tabelami mającymi wspólne atrybuty C1, C2 i C3.
- Jeżeli w warunku nie chcemy użyć wszystkich wspólnych atrybutów, to zapisujemy instrukcje wyboru z użyciem słowa USING, np..:

```
SELECT tab_1.*, tab_2.*
FROM tab_1 INNER JOIN tab_2 USING (C1, C2)
```

- Jeżeli natomiast chcemy użyc wszystkich wspólnych atrybutów, to wystarczy napisać:

```
SELECT tab_1.*, tab_2.*
FROM tab_1 NATURAL JOIN tab_2
```

Złączenie wewnętrzne - przykłady

t1

	nr	nazwisko
1	12	Nowak
2	18	Kowalski
3	35	Brown

t2

	id	nazwisko	telefon
1	1	Nowak	12-15
2	2	Kowalski	13-25
3	3	Kowalski	24-13
4	4	Brown	36-28

```
SELECT t1.*, t2.*
FROM t1, t2
WHERE t1.nazwisko=t2.nazwisko;
```

Query 1

	nr	nazwisko	id	nazwisko	telefon
1	12	Nowak	1	Nowak	12-15
2	18	Kowalski	2	Kowalski	13-25
3	18	Kowalski	3	Kowalski	24-13
4	35	Brown	4	Brown	36-28

```
SELECT t1.*, t2.*
FROM t1 INNER JOIN t2 ON t1.nazwisko=t2.nazwisko;
```

Query 1

	nr	nazwisko	id	nazwisko	telefon
1	12	Nowak	1	Nowak	12-15
2	18	Kowalski	2	Kowalski	13-25
3	18	Kowalski	3	Kowalski	24-13
4	35	Brown	4	Brown	36-28

Złączenie wewnętrzne - przykłady

t1

	nr	nazwisko
1	12	Nowak
2	18	Kowalski
3	35	Brown

t2

	id	nazwisko	telefon
1	1	Nowak	12-15
2	2	Kowalski	13-25
3	3	Kowalski	24-13
4	4	Brown	36-28

```
SELECT t1.*, t2.*
FROM t1 INNER JOIN t2 USING (nazwisko);
```

Query 1

	nr	nazwisko	id	nazwisko	telefon
1	12	Nowak	1	Nowak	12-15
2	18	Kowalski	2	Kowalski	13-25
3	18	Kowalski	3	Kowalski	24-13
4	35	Brown	4	Brown	36-28

```
SELECT t1.*, t2.*
FROM t1 NATURAL JOIN t2 ;
```

Query 1

	nr	nazwisko	id	nazwisko	telefon
1	12	Nowak	1	Nowak	12-15
2	18	Kowalski	2	Kowalski	13-25
3	18	Kowalski	3	Kowalski	24-13
4	35	Brown	4	Brown	36-28

```
SELECT t1.*, t2.*
FROM t1 NATURAL JOIN t2
WHERE nr>15;
```

Query 1

	nr	nazwisko	id	nazwisko	telefon
1	18	Kowalski	2	Kowalski	13-25
2	18	Kowalski	3	Kowalski	24-13
3	35	Brown	4	Brown	36-28

Złączenia zewnętrzne

- Za pomocą złączenia zewnętrznego można ograniczyć w tabeli wynikowej wiersze z jednej tabeli.
- Możemy wyróżnić:
  - złączenie lewostronnie zewnętrzne,
  - złączenie prawostronnie zewnętrzne.

Złączenie lewostronnie zewnętrzne

- Składa się ze wszystkich wierszy z tabeli pierwszej i wierszy z pasującymi wartościami z tabeli drugiej. Gdy w tabeli drugiej nie ma pasujących wierszy, w kolumnach wybranych z tej tabeli będą wartości NULL.

t1

	nr	nazwisko
1	12	Nowak
2	18	Kowalski
3	35	Brown

t2

	id	nazwisko	telefon
1	1	Nowak	12-15
2	2	Kowalski	13-25
3	3	Kowalski	24-13
4	4	Green	36-28

```
SELECT t1.*, t2.*
FROM t1 LEFT OUTER JOIN t2 ON (t1.nazwisko=t2.nazwisko);
```

Query 1

	nr	nazwisko	id	nazwisko	telefon
1	12	Nowak	1	Nowak	12-15
2	18	Kowalski	2	Kowalski	13-25
3	18	Kowalski	3	Kowalski	24-13
4	35	Brown	[NULL]	[NULL]	[NULL]

Złączenie prawostronnie zewnętrzne

- Składa się ze wszystkich wierszy z tabeli drugiej i wierszy z pasującymi wartościami z tabeli pierwszej. Gdy w tabeli pierwszej nie ma pasujących wierszy, w kolumnach wybranych z tej tabeli będą wartości NULL.

t1

	nr	nazwisko
1	12	Nowak
2	18	Kowalski
3	35	Brown

t2

	id	nazwisko	telefon
1	1	Nowak	12-15
2	2	Kowalski	13-25
3	3	Kowalski	24-13
4	4	Green	36-28

```
SELECT t1.*, t2.*
FROM t1 RIGHT OUTER JOIN t2 ON (t1.nazwisko=t2.nazwisko);
```

Query 1

	nr	nazwisko	id	nazwisko	telefon
1	12	Nowak	1	Nowak	12-15
2	18	Kowalski	2	Kowalski	13-25
3	18	Kowalski	3	Kowalski	24-13
4	[NULL]	[NULL]	4	Green	36-28

Samozłączenie

- Samozłączenie to złączenie tabeli z samą sobą.
- Przykład. Z tabeli OSOBY wybrać te osoby o danym nazwisku, które nie są najstarsze.

```
SELECT oso_1.id, oso_1.nazwisko
FROM osoby AS oso_1 INNER JOIN osoby AS oso_2
        ON oso_1.nazwisko=oso_2.nazwisko AND oso_1.wiek<oso_2.wiek;
```

Query 1

	id	nazwisko
1	6	kowalski
2	9	nowak
3	10	nowak
4	11	okos
5	9	nowak

Podzapytania

- Zapytania to instrukcje SELECT służące do wyszukiwania danych w tabelach.
- Podzapytania to zapytania zwracające dane, które nie są końcowym wynikiem, lecz będą wykorzystywane w dalszej części instrukcji.
- Podzapytanie musi być ujęte w nawiasy zwykłe.
- Najpierw uruchamiane jest podzapytanie (zapytanie wewnętrzne), a później zapytanie zewnętrzne.
- Jeżeli wynik podzapytania będzie porównywany operatorem =, <, >, <=, >= lub <>, to musi ono zwrócić jedną wartość.

Podzapytania

- Przykład. Instrukcja wybiera osoby mające więcej lat, niż wynosi średnia wieku wszystkich osób z tabeli OSOBY.
- Przykład. Instrukcja wybiera osoby mające więcej lat, niż wynosi średnia wieku wszystkich kobiet z tabeli OSOBY.

```
SELECT id, imie, nazwisko, wiek
FROM osoby
WHERE wiek >
            (SELECT AVG(wiek)
             FROM osoby);
```

Query 1

	id	imie	nazwisko	wiek
1	2	piotr	nowak	92
2	4	ala	okos	48
3	8	iwona	wolna	77
4	10	ola	nowak	45

```
SELECT *
FROM osoby
WHERE wiek >
            (SELECT AVG(wiek)
             FROM osoby
             WHERE plec=0);
```

Query 1

	id	imie	nazwisko	nr	wiek	plec
1	2	piotr	nowak	4567	92	1
2	4	ala	okos	9618	48	0
3	8	iwona	wolna	2435	77	0
4	10	ola	nowak	7659	45	0

Tworzenie tabel

- Do budowania tabel służy polecenie CREATE TABLE.
- Definiując tabelę musimy podać listę kolumn opisywaną przez nazwę kolumny, jej typ i czasami długość przechowywanej wartości.

```
CREATE TABLE nazwa_tablicy
(nazwa_kolumny typ_danych(rozmiar),
  nazwa_kolumny typ_danych(rozmiar),
…);
```

Tworzenie tabel – typy danych

Podstawowe typy danych, to:

- Typy tekstowe
  - CHAR(n) – łańcuch znaków o ustalonej długości (n)
  - VARCHAR(n) – łańcuch znaków o zmiennej długości (maksymalnie n)
- Typy całkowite
  - TINYINT, INT, SMALLINT, BIGINT
- Typy rzeczywiste
  - FLOAT, REAL, NUMERIC
- Data
  - DATE, TIME

Poszczególne implementacje mogą zawierać dodatkowe typy lub typy o innych nazwach.

Klauzula DEFAULT

- Klauzula DEFAULT służy do wskazania, jaka wartość ma być wstawiona do kolumny, jeśli nie została określona konkretna wartość.

  ...

  kolumna typ (rozmiar) DEFAULT wyrażenie

  ...

- Wyrażenie musi być proste, nie wolno stosować podzapytań.
- Przykład.

  miasto VARCHAR(20) DEFAULT ‘Sopot’

Warunki integralności

- Podczas definiowania tabeli mamy możliwość określić, jakie warunki powinny spełniać dane w wierszach wprowadzanych do tablicy. Warunki takie nazywa się warunkami integralności (constraints).
- Możemy zażądać, aby wypełnienie wartości w danej kolumnie było obowiązkowe, aby wartości pochodziły z określonego zakresu, aby były unikalne itd.

Ograniczenia NULL i NOT NULL

- Podczas definiowania kolumn tabeli możemy zażądać, aby wiersze tej tabeli w polach tej kolumny nie dopuszczały (NOT NULL) wartości nieokreślonych.

```
CREATE TABLE nazwa_tablicy
    (nazwa_kolumny typ(rozmiar) [NULL | NOT NULL],
    nazwa_kolumny typ(rozmiar) [NULL | NOT NULL],
…);
```

- Opcja NULL (domyślna) oznacza, że pola tej kolumny mogą przyjmować wartość NULL.
- Opcja NOT NULL oznacza, że pola tej kolumny muszą mieć określoną wartość, nie mogą przyjmować wartości NULL.

Ograniczenia CONSTRAINT

- Do definiowania innych niż NOT NULL warunków integralności służy klauzula CONSTRAINT. Warunki mogą być wpisanie bezpośrednio przy definicji kolumny lub na końcu po zdefiniowaniu wszystkich kolumn.
- Warunek może być umieszczony przy definicji kolumny:

```
CREATE TABLE nazwa_tablicy
(...
 nazwa_kolumny typ (rozmiar)
CONSTRAINT nazwa_warunku typ_warunku [warunek],
…);
```

- Warunek umieszczony po definicjach wszystkich kolumn:

```
 CREATE TABLE nazwa_tablicy
(...
 nazwa_kolumny typ (rozmiar),
 …
CONSTRAINT nazwa_warunku typ_warunku warunek,
CONSTRAINT nazwa_warunku typ_warunku warunek,
 …);
```

Ograniczenia CONSTRAINT

- warunek — dodatkowe informacje w zależności od typu warunku, w przypadku umieszczenia klauzuli CONSTRAINT po definicjach kolumn warunek musi być zawsze określony.
- typ_warunku — jeden z następujących: CHECK, PRIMARY KEY, FOREIGN KEY.
- nazwa_warunku — jest identyfikatorem warunku integralności, nie jest wymagane jego podanie, ale wtedy system nada warunkowi własny, zazwyczaj nieczytelny identyfikator. Identyfikator jest potrzebny przy komendach włączających i wyłączających warunki integralności.

Ograniczenie CHECK

- Określa warunek, jaki musi spełniać wartość w kolumnie każdego wstawianego wiersza, warunek nie może się odwoływać się do innych tabel.

  CONSTRAINT CHECK (warunek logiczny);

- Warunek logiczny musi być prosty, nie wolno stosować podzapytań.
- Przykład.

```
CREATE TABLE tab
(
...
wiek INT,
...
CHECK(wiek>0)
);
```

Ograniczenie PRIMARY KEY

- Definiuje klucz główny tabeli. Jeśli kluczem głównym jest jedna kolumna, wygodniej warunek zapisać po definicji tej kolumny. W przypadku klucza głównego opartego na kilku kolumnach wygodniej zdefiniować go po definicji wszystkich kolumn.
- Definicja warunku przy definicji kolumny:

  kolumna typ(rozmiar) CONSTRAINT nazwa_warunku PRIMARY KEY;
- Definicja warunku po definicji wszystkich kolumn:

  CONSTRAINT nazwa_warunku PRIMARY KEY (kolumna_1, kolumna_2, …);
- W tabeli może być tylko jeden klucz główny. Wszystkie kolumny wchodzące w skład klucza głównego są obowiązkowe — nie musimy dodatkowo nakładać warunku NOT NULL.

Ograniczenie PRIMARY KEY

- Definiuje klucz obcy, reprezentujący związek z inną tabelą. Sprawia że, wartości kolumn z tym kluczem mogą przyjmować tylko wartości z klucza głównego lub unikalnego innej, wskazanej tabeli.
- Jeśli kluczem obcym jest jedna kolumna, wygodniej warunek zapisać po definicji tej kolumny. W przypadku klucza opartego na kilku kolumnach wygodniej zdefiniować go po definicji wszystkich kolumn.
- Definicja warunku przy definicji kolumny:

  kolumna typ(rozmiar) CONSTRAINT nazwa_warunku
  REFERENCES nazwa_tabeli lista_kolumn;
- Definicja warunku po definicji wszystkich kolumn:

  CONSTRAINT nazwa_warunku
  FOREIGN KEY (lista kolumn tabeli definiowanej)
  REFERENCES nazwa_tabeli lista_kolumn;
- Aby klucz obcy mógł być zdefiniowany musi być zdefiniowana wcześniej tabela, do której klucz ten się odwołuje, a na zestawie kolumn wskazanym przez klucz obcy musi być zdefiniowany klucz główny lub unikalny. Jeśli odwołujemy się do klucza głównego obcej tabeli, to nie musimy specyfikować listy kolumn tego klucza.

Wstawianie wierszy

- Do wstawiania nowych wierszy do tabeli służy polecenie INSERT :

  INSERT INTO nazwa_tabeli [(lista_kolumn)]

  VALUES (lista_wartości);

- Przy wstawianiu do wszystkich kolumn tabeli nie musimy podawać listy kolumn, ale wtedy musimy wartości wprowadzać w takiej kolejności, jaka była wyspecyfikowana podczas polecenia CREATE TABLE. Z tego względu, aby uniezależnić się od późniejszych modyfikacji tabeli bezpieczniej jest podawać listę kolumn w klauzuli INSERT INTO.
- Przykład

  INSERT INTO t1 (nr, nazwisko)

  VALUES (50, ‘Stawicki’);

- W każdym poleceniu INSERT można wstawić tylko jeden wiersz.

Modyfikacja wierszy

- Do zmiany zawartości wierszy służy polecenie UPDATE.

```
UPDATE nazwa_tabeli [alias]
SET kolumna = { wyrażenie | podzapytanie }
        [ , kolumna= { wyrażenie | podzapytanie } ...]
[ WHERE warunek ];
```

- Przykład

```
UPDATE  t1
SET     nazwisko='Wolski'
WHERE   nr=2;
```

- Jeśli nie wpiszemy klauzuli WHERE, to zostaną zmodyfikowane wszystkie wiersze tabeli.

Usuwanie wierszy

- Do usuwania wierszy z tabeli służy polecenie DELETE.

```
DELETE  [FROM]  tabela
[ WHERE warunek ];
```

- Warunek w klauzuli WHERE określa, które wiersze będą usunięte.
- Przykład

```
DELETE FROM t1
WHERE nazwisko='Nowak';
```

Zostaną usunięte osoby o nazwisku *Nowak*.