Bazy danych i język SQL

SOFLAB TECHNOLOGY SP. Z O.O.
Centrum Testów Oprogramowania
Al. Jerozolimskie 123a, 02-017 Warszawa
soflab.pl

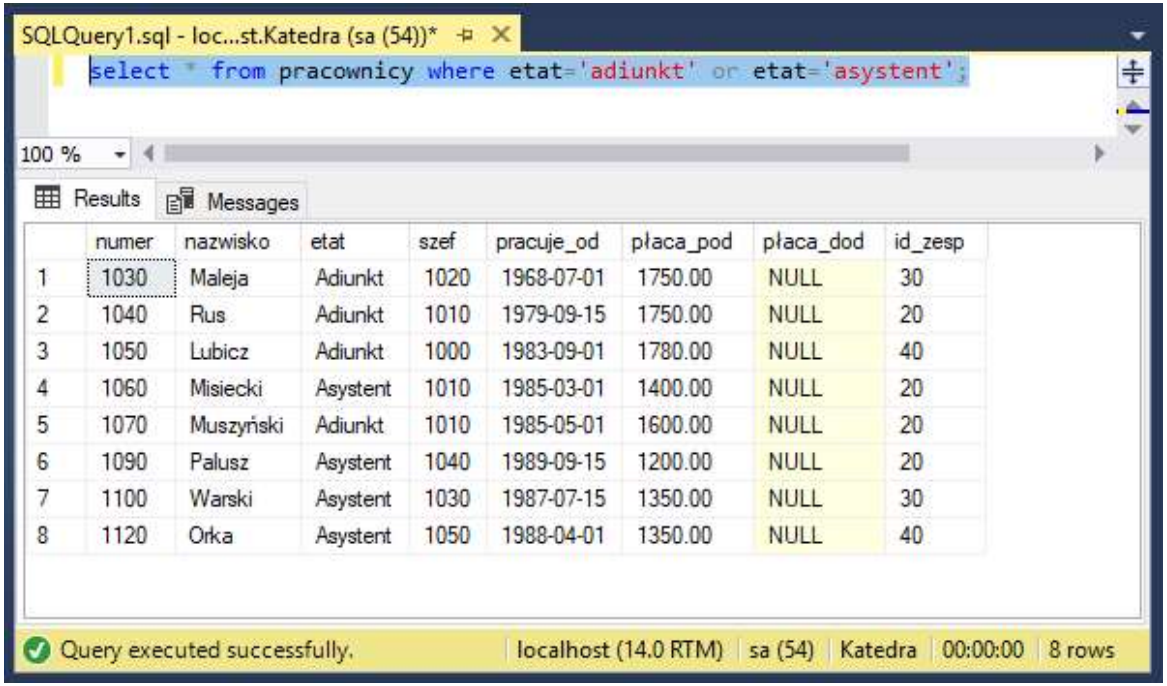
NAJWYŻSZA JAKOŚĆ ŚWIADCZONYCH USŁUG

Krzysztof Świder
Specjalista BI

krzysztof.swider@soflab.pl
mobile: +48 608 460 306

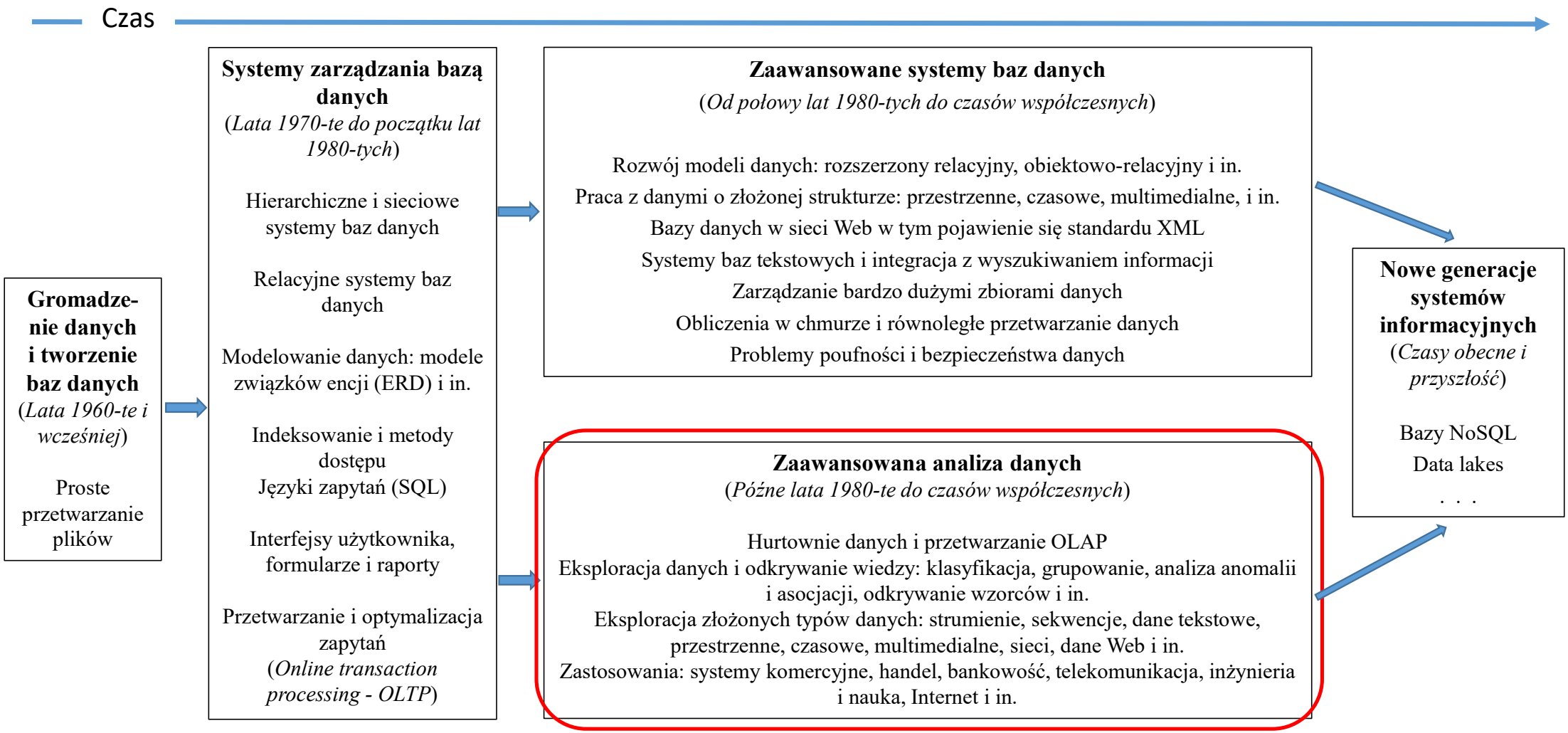
Bazy SQL w kilku zdaniach

- Baza danych to odpowiednio zorganizowana kolekcja (zbiór) danych.
- Specjalne oprogramowanie określane nazwą system zarządzania bazą danych (DBMS) służy do utrzymywania spójności i bezpieczeństwa danych, a także do ich odtworzenia w razie awarii systemu.
- Strukturalny język zapytań (SQL) jest językiem baz danych przeznaczonym do zarządzania i manipulowania danymi.
- Dane są zorganizowane według pewnego schematu oraz rozmieszczone w logicznie powiązanych ze sobą tabelach
  - Dostęp do danych jest możliwy przy pomocy zapytań SQL jak w pokazanym przykładzie.



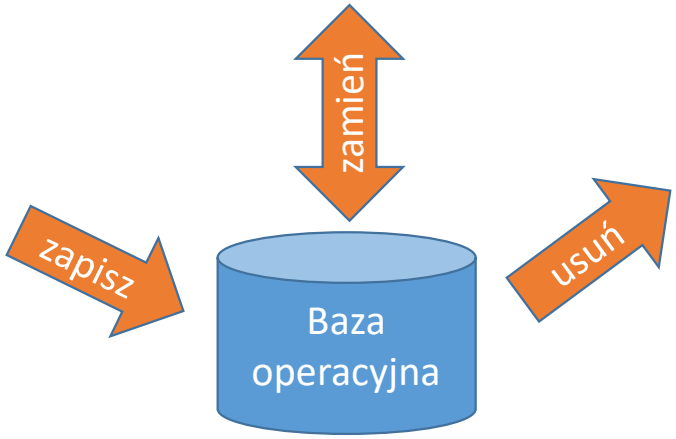
	numer	nazwisko	etat	szef	pracuje_od	płaca_pod	płaca_dod	id_zesp
1	1030	Maleja	Adiunkt	1020	1968-07-01	1750.00	NULL	30
2	1040	Rus	Adiunkt	1010	1979-09-15	1750.00	NULL	20
3	1050	Lubicz	Adiunkt	1000	1983-09-01	1780.00	NULL	40
4	1060	Misiecki	Asystent	1010	1985-03-01	1400.00	NULL	20
5	1070	Muszyński	Adiunkt	1010	1985-05-01	1600.00	NULL	20
6	1090	Palusz	Asystent	1040	1989-09-15	1200.00	NULL	20
7	1100	Warski	Asystent	1030	1987-07-15	1350.00	NULL	30
8	1120	Orka	Asystent	1050	1988-04-01	1350.00	NULL	40

Rozwój technologii baz danych



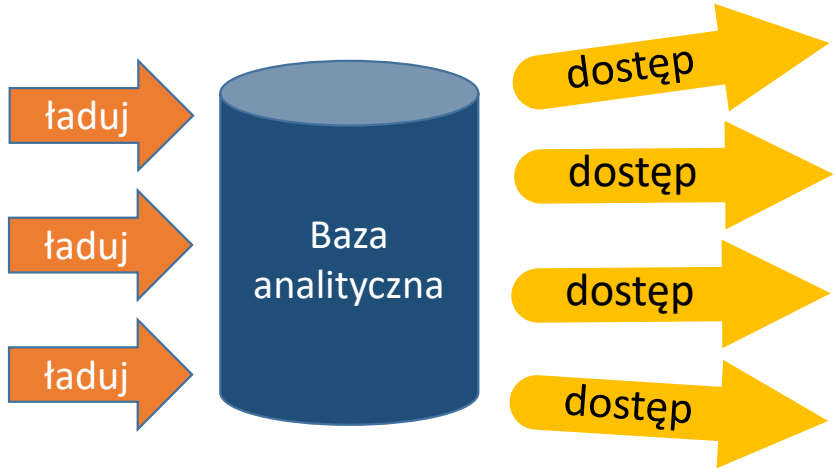
Bazy operacyjne a bazy analityczne

- Bazy operacyjne (np. relacyjne) wspierają codzienną działalność przedsiębiorstwa.



Dane są stale aktualizowane

- Bazy analityczne (np. hurtownie danych) dostarczają informacji, która może posłużyć do analizy (objaśniania zjawisk, predykcji i in.)

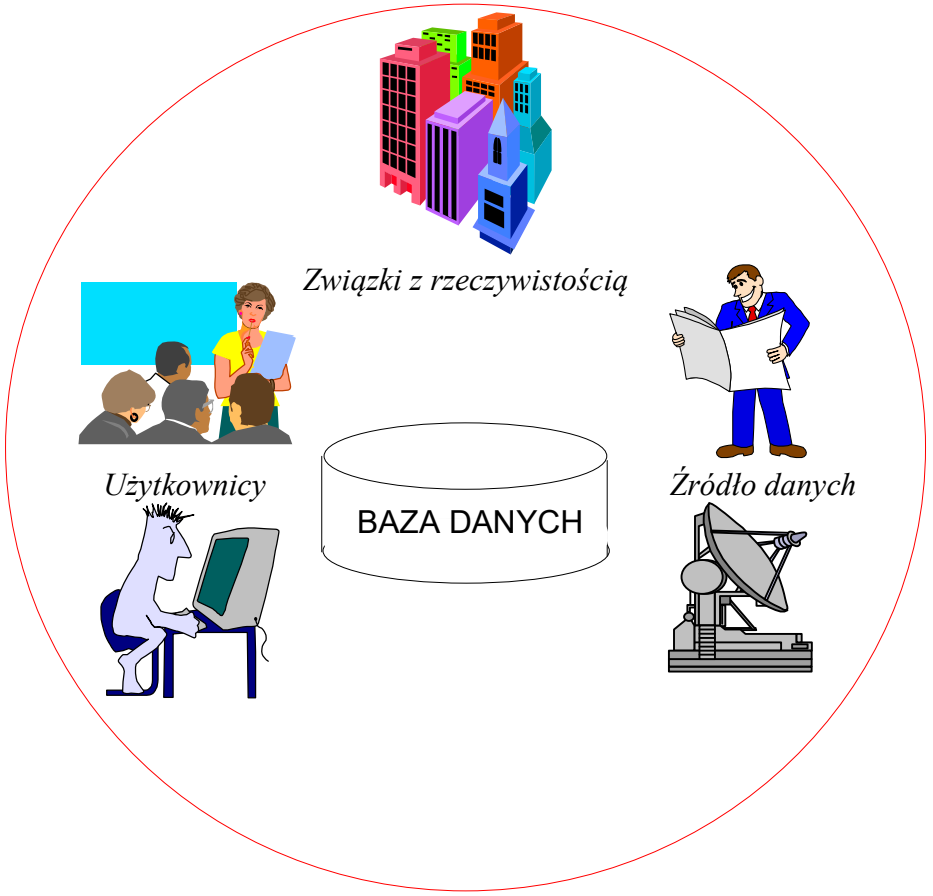


Dane są ładowane do hurtowni i jest zapewniony do nich dostęp ale **bez możliwości aktualizowania**

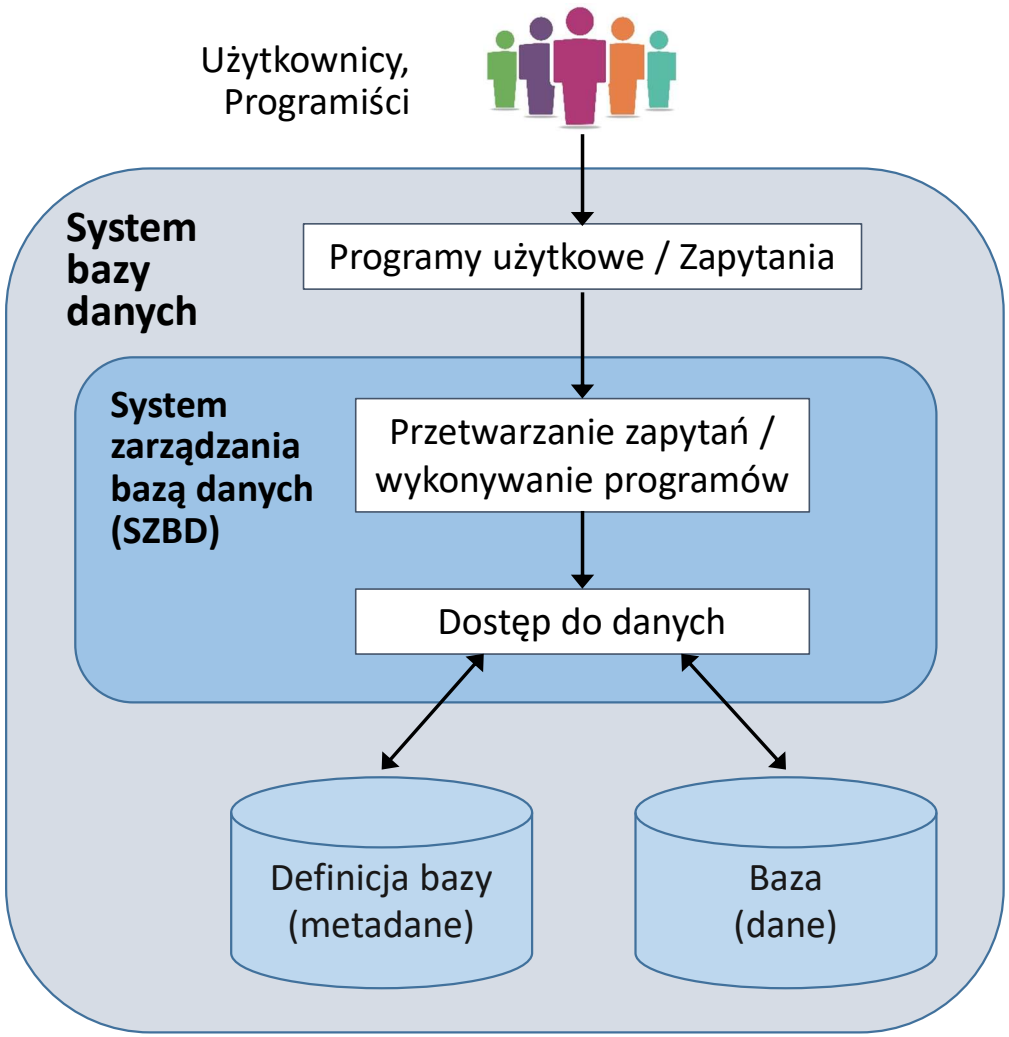
Pojęcie bazy danych

- Baza danych jest zbiorem powiązanych ze sobą danych.
  - Pojęciem dane określa się znane fakty, które mogą zostać zapisane oraz jednoznacznie zinterpretowane.
- Rozważmy, na przykład, nazwiska, numery telefonów oraz adresy naszych znajomych.
  - Możemy je umieścić w książce adresowej z zakładkami lub zapisać na twardym dysku, posługując się komputerem osobistym oraz oprogramowaniem takim jak Microsoft Access lub Excel.
  - Taki zbiór powiązanych danych o nie podlegającym dyskusji znaczeniu, jest bazą danych.
- Z bazą danych związane są: (i) źródła, z których pochodzą dane, (ii) pewien stopień interakcji z wydarzeniami w świecie rzeczywistym oraz (iii) społeczność trwale i aktywnie zainteresowana zawartością bazy.

Otoczenie bazy danych



System bazy danych



- System zarządzania bazą danych (SZBD) jest zbiorem programów, które umożliwiają użytkownikom tworzenie a następnie utrzymywanie bazy.
- Definiowanie bazy oznacza specyfikowanie typów, struktur oraz ograniczeń nakładanych na dane, które mają znaleźć się w bazie.
  - Definicja bazy jest zapisywana przez SZBD w formie katalogu bazy danych określanego jako metadane.
- Konstruowanie bazy danych polega na zapisywaniu danych na nośniku nadzorowanym przez SZBD.
- Program użytkowy korzysta z bazy danych wysyłając do SZBD zapytania oraz inne polecenia związane z operacjami na danych.
- Bazę danych, SZBD oraz programy użytkowe będziemy łącznie określać jako system bazy danych.

Przykładowa zawartość bazy danych

Tabela Pracownicy

numer	nazwisko	etat	szef	pracuje_od	płaca_pod	płaca_dod	id_zesp
1000	Lech	Dyrektor	NULL	1971-01-01	3160.00	570.00	10
1080	Koliberek	Sekretarka	1000	1983-02-20	1150.00	NULL	10
1040	Rus	Adiunkt	1010	1979-09-15	1750.00	NULL	20
1070	Muszyński	Adiunkt	1010	1985-05-01	1600.00	NULL	20
1060	Misiecki	Asystent	1010	1985-03-01	1400.00	NULL	20
1090	Palusz	Asystent	1040	1989-09-15	1200.00	NULL	20
1010	Podgajny	Profesor	1000	1975-05-01	2180.00	420.00	20
1030	Maleja	Adiunkt	1020	1968-07-01	1750.00	NULL	30
1100	Warski	Asystent	1030	1987-07-15	1350.00	NULL	30
1020	Delcki	Profesor	1000	1977-09-01	2050.00	270.00	30
1110	Rajski	Stażysta	1030	1990-07-01	900.00	NULL	30
1050	Lubicz	Adiunkt	1000	1983-09-01	1780.00	NULL	40
1120	Orka	Asystent	1050	1988-04-01	1350.00	NULL	40
1130	Kolski	Stażysta	1050	1991-09-01	900.00	NULL	40

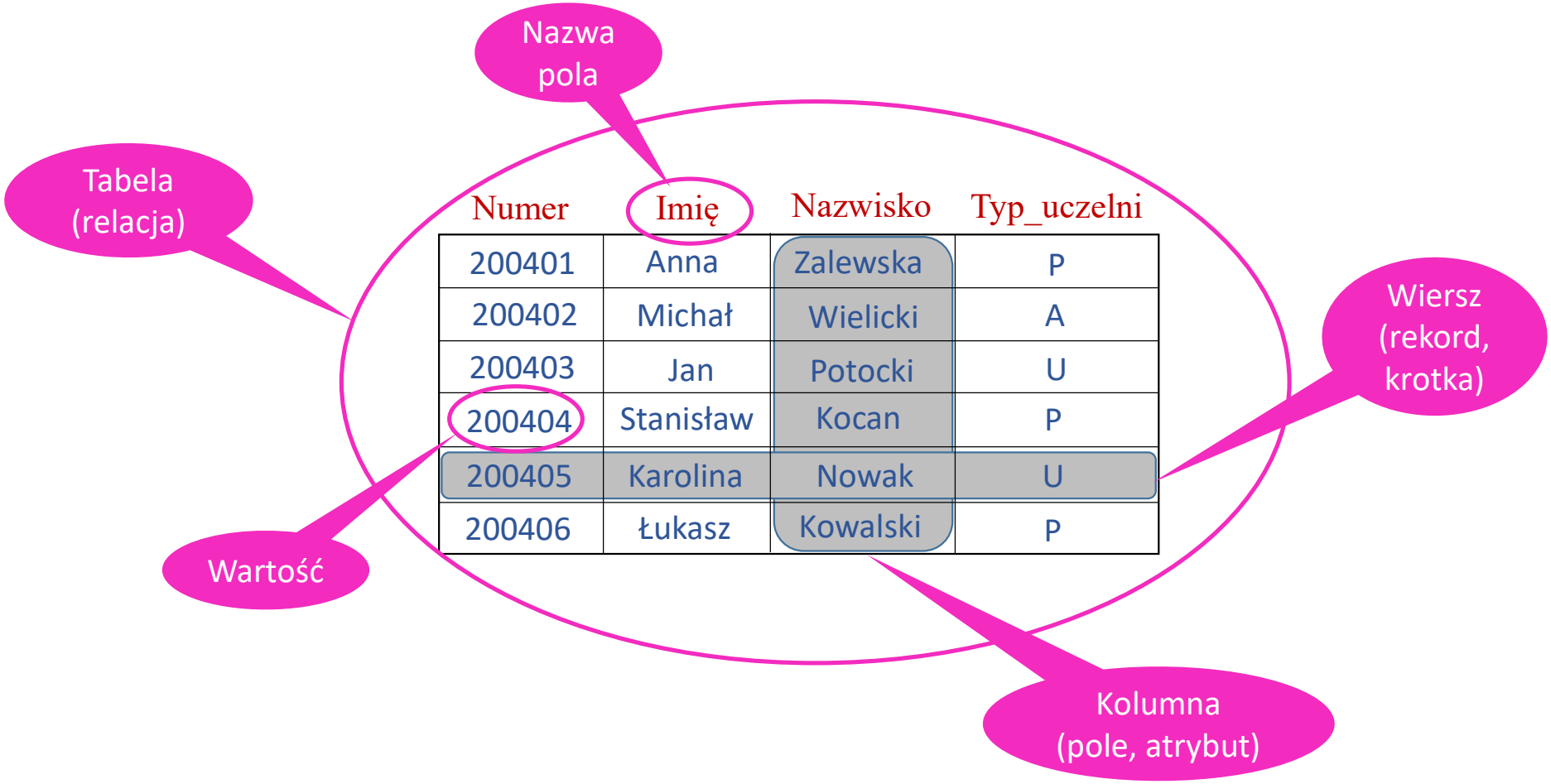
Tabela Etaty

nazwa	płaca_min	płaca_max
Stażysta	800,00	1000,00
Sekretarka	900,00	1200,00
Asystent	1000,00	1600,00
Adiunkt	1600,00	2000,00
Profesor	2000,00	2500,00
Dyrektor	2500,00	3200,00

Tabela Zespoły

id_zesp	nazwa	adres
10	Administracja	Piotrowo 3a
20	Bazy danych	Wieżowa 75
30	Sieci komputerowe	Garbary 3
40	Systemy operacyjne	Piotrowo 3a
50	Translatory	Mansfelda 4

Tabele



Numer	Imię	Nazwisko	Typ_uczelni
200401	Anna	Zalewska	P
200402	Michał	Wielicki	A
200403	Jan	Potocki	U
200404	Stanisław	Kocan	P
200405	Karolina	Nowak	U
200406	Łukasz	Kowalski	P

Widoki

TABELA → WIDOK (PERSPEKTYWA)

Numer	Imię	Nazwisko	Typ_uczelni
200401	Anna	Zalewska	P
200402	Michał	Wielicki	A
200403	Jan	Potocki	U
200404	Stanisław	Kocan	P
200405	Karolina	Nowak	U
200406	Łukasz	Kowalski	P

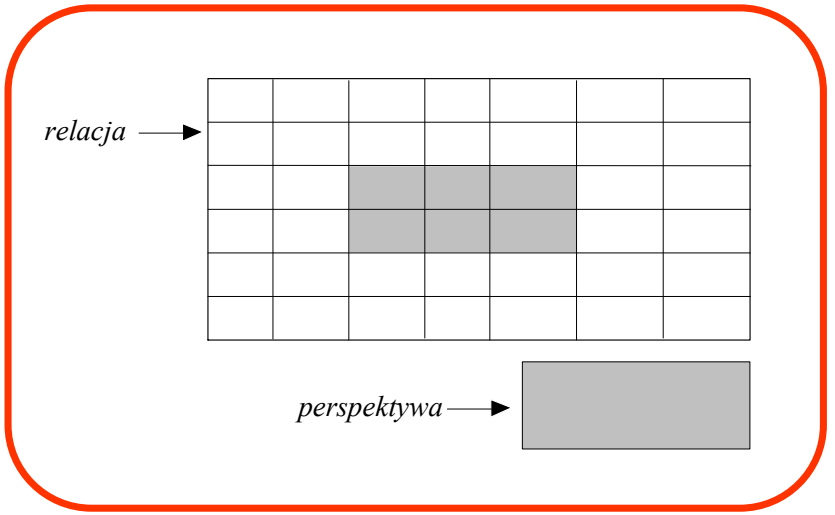
Wiersz z „U”

Wiersz z „U”

Pole „Numer”

Pole „Nazwisko”

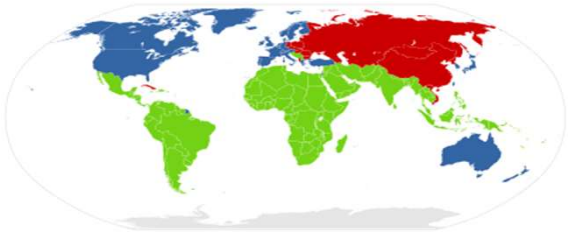
Numer	Nazwisko
200403	Potocki
200405	Nowak



Modele bazy danych

- Model konceptualny
  - powstaje w początkowej fazie opracowania systemu.
  - Często jest reprezentowany przez diagram związków encji (ERD).
- Model implementacyjny
  - powstaje w wyniku transformacji wcześniej przygotowanego modelu konceptualnego na poziom projektu bazy.
  - Istnieje 3 strategie (modele hierarchiczne, sieciowe i relacyjne), z których największe znaczenie praktyczne mają obecnie modele relacyjne.
- Model fizyczny
  - odnosi się do sposobu organizacji danych w zewnętrznej pamięci komputera.
  - Przy najwyższym stopniu szczegółowości rozważa się poszczególne bity przechowywane w pamięci, a na poziomie bardziej ogólnym operuje się rekordami (zapisami) i plikami.

Proces opracowania bazy danych



miniświat

Analiza miniświata – modelowanie konceptualne

diagramy konceptualne

Transformacja do modelu logicznego

relacje

Normalizacja

schemat znormalizowany

Transformacja do modelu fizycznego

fizyczne struktury danych

Strojenie

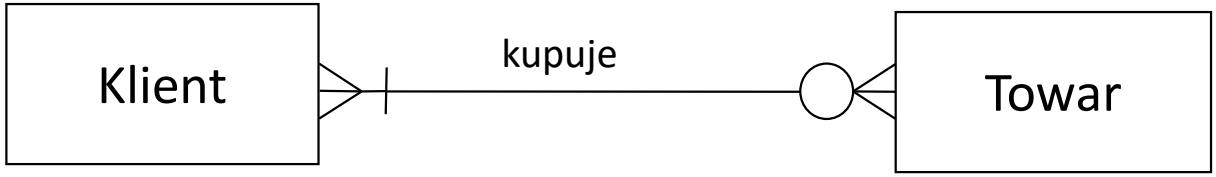
wydajny system

Modele konceptualne

- Podstawową techniką sporządzania i prezentacji konceptualnych modeli danych są diagramy związków encji (ERD).

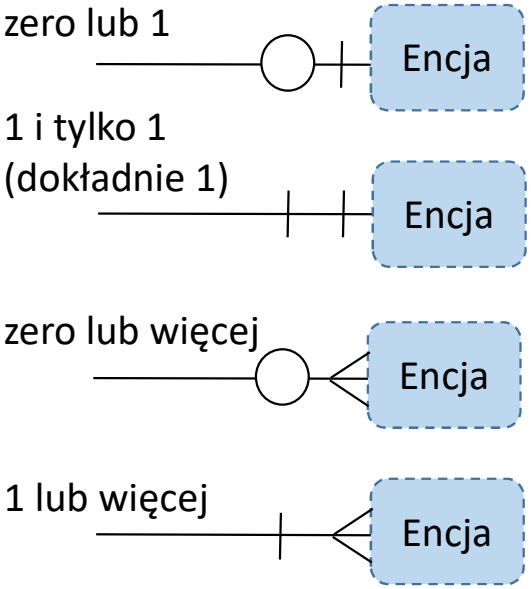
- Encja to cokolwiek, o czym chcemy przechowywać informację. Przykładami encji są: Klient, Towar, Zamówienie oraz Faktura.
  - Atrybuty opisują encje.
  - Na przykład do atrybutów encji Klient należą: nazwisko, adres, limit kredytowy, itp.

- Związki opisują zależności pomiędzy encjami.
  - Związek jest przedstawiony jako linia łącząca encje.



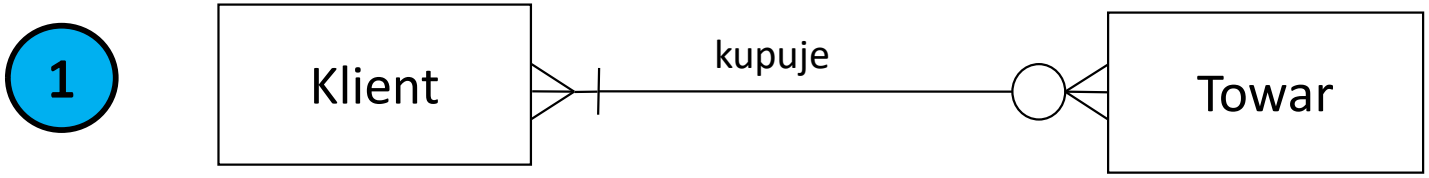
Krotność i opcjonalność związków

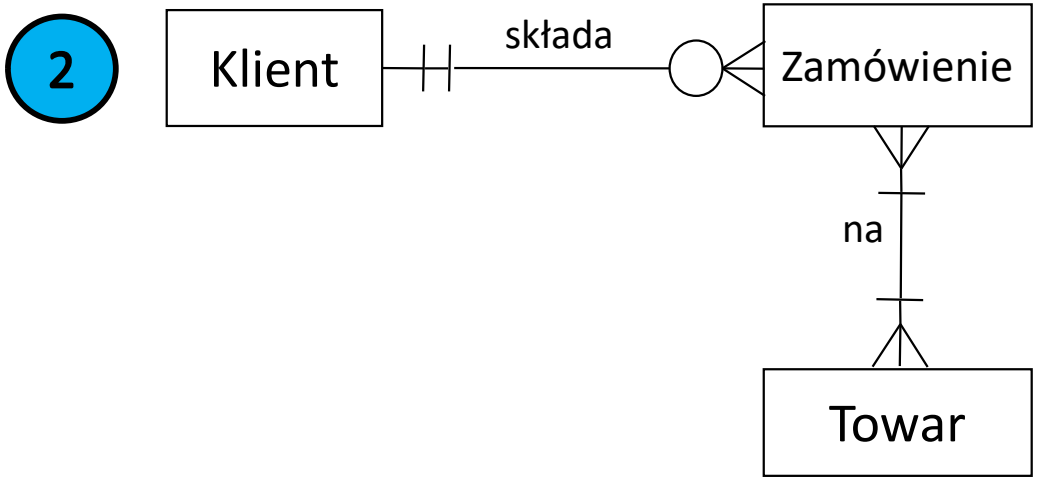
- Krotność i opcjonalność bliżej określają sens merytoryczny związku.
  - Krotność odnosi się do maksymalnej liczby powiązań jaka może pojawić się pomiędzy wystąpieniem jednej encji a wystąpieniami drugiej encji uczestniczącej w związku.
  - Opcjonalność dotyczy minimalnej liczby powiązań jaka może pojawić się pomiędzy wystąpieniem jednej encji a wystąpieniami drugiej encji uczestniczącej w związku.

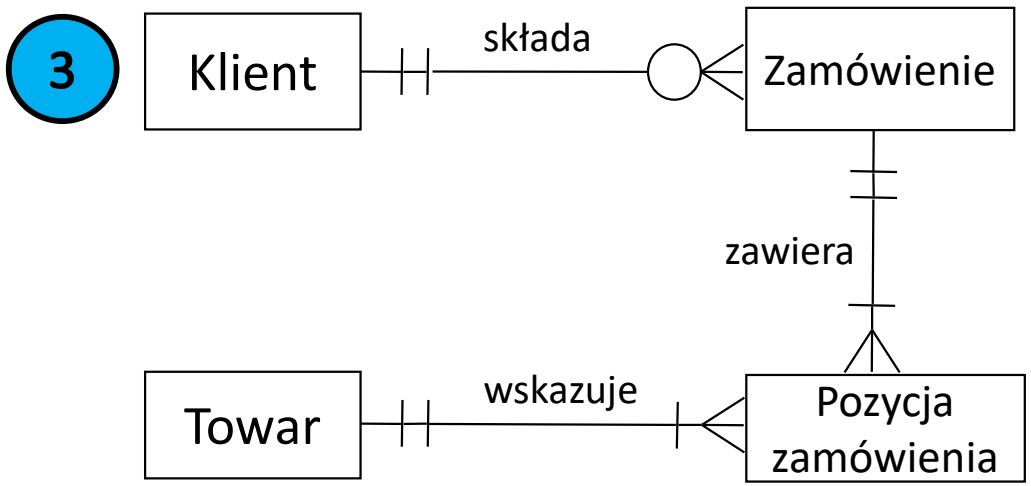


- Krotność może być 1 lub Wiele a określający to symbol jest umieszczony po zewnętrznej stronie linii oznaczającej związek (bliżej encji do której odnosimy krotność).
  - Dla krotności 1 jest to pojedyncza kreska pionowa.
  - Dla krotności Wiele jest to „łapka z trzema palcami”.
- Opcjonalność może być 0 lub 1, a określający to symbol jest umieszczany tuż obok symbolu krotności w kierunku środka linii.
  - Dla opcjonalności 0 jest to kółeczko.
  - Dla opcjonalności 1 jest pojedyncza kreska pionowa.

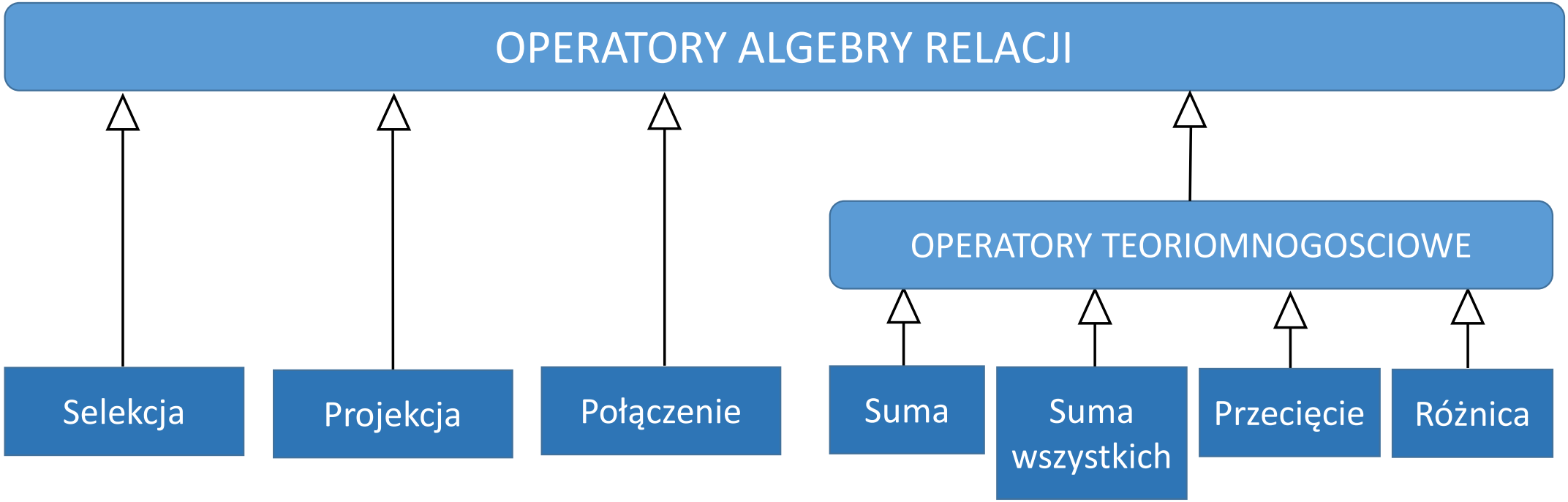
Normalizacja modeli konceptualnych



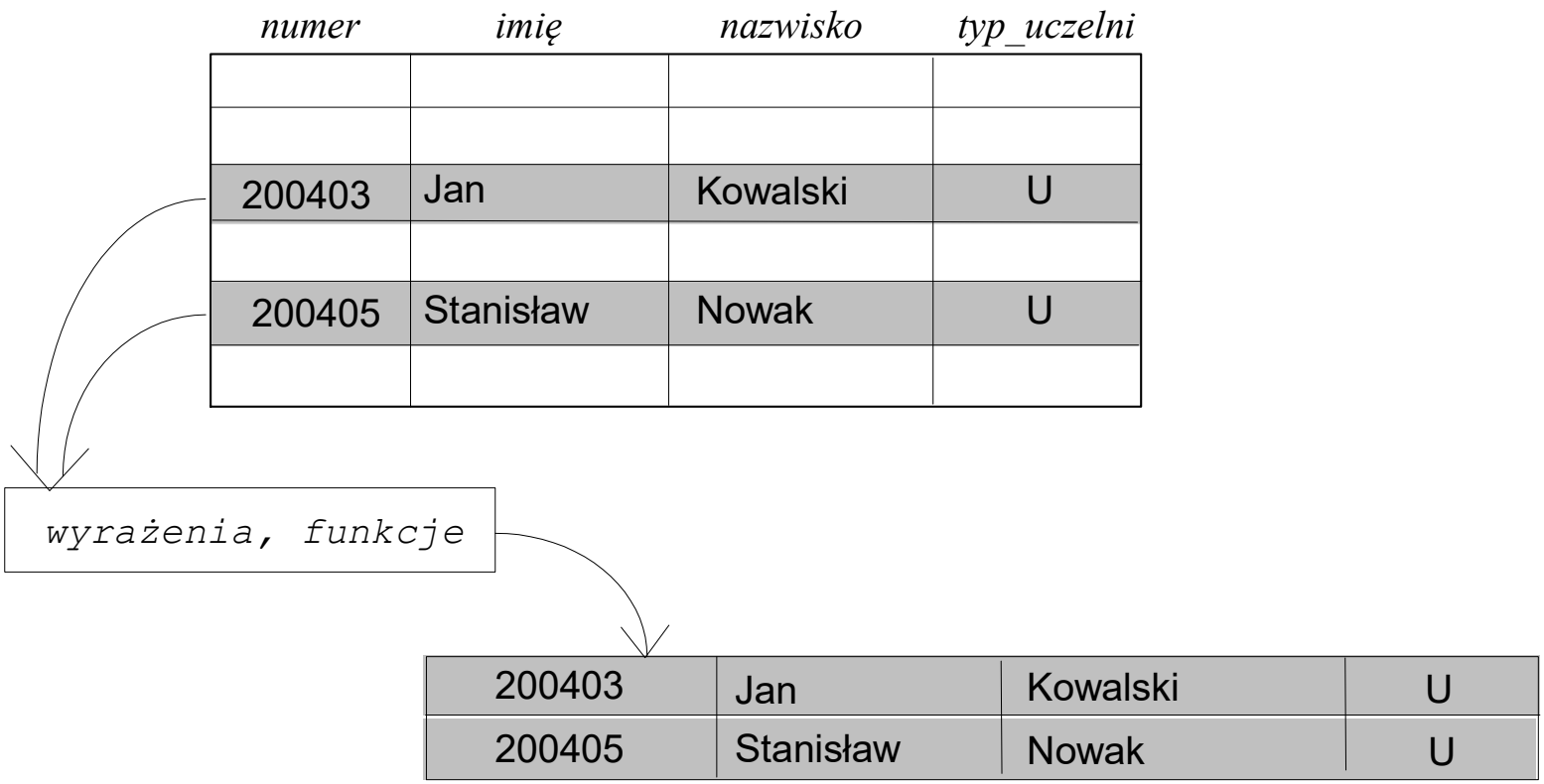




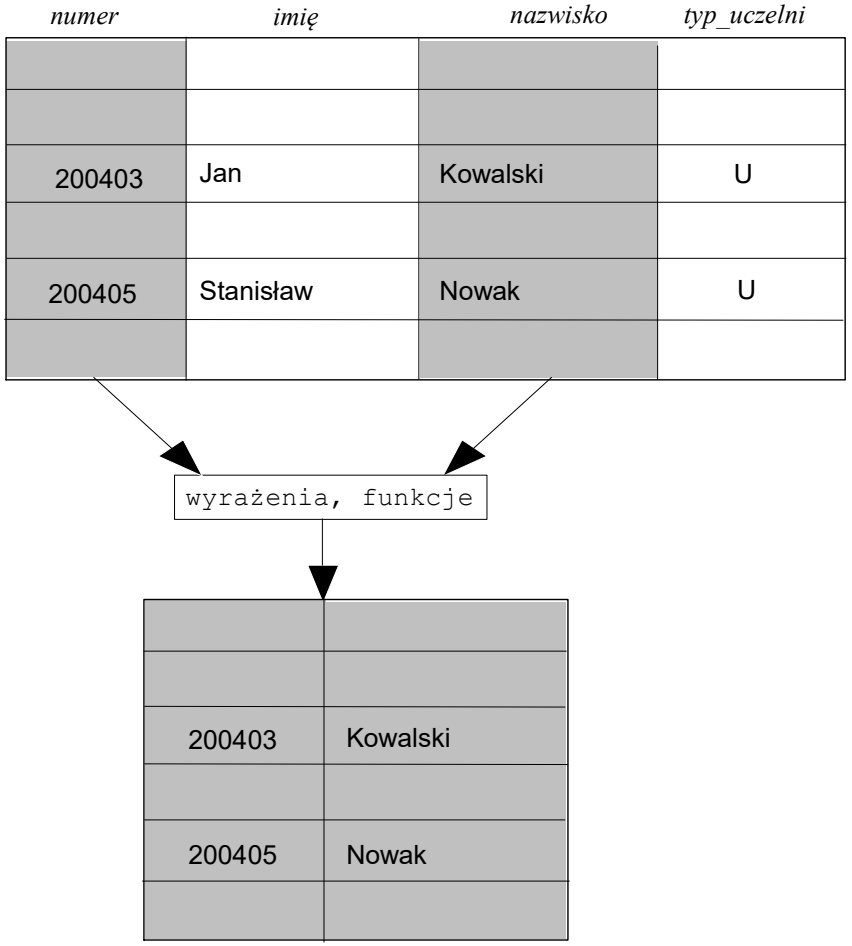
Operatory algebry relacji



Selekcja


numer
imię
nazwisko
typ_uczelni
200403
Jan
Kowalski
U
200405
Stanisław
Nowak
U
wyrażenia, funkcje
200403
Jan
Kowalski
U
200405
Stanisław
Nowak
U

Projekcja


numer
imię
nazwisko
typ_uczelni
200403
Jan
Kowalski
U
200405
Stanisław
Nowak
U
wyrażenia, funkcje
200403
Kowalski
200405
Nowak

Połączenie

numer	*imię*	*nazwisko*	*typ_uczelni*
			P
			P
200403	Jan	Kowalski	U
			P
200405	Stanisław	Nowak	U
			A

typ_uczelni	*nazwa*
P	Politechnika
U	Uniwersytet
A	Akademia

Jan	Kowalski	Uniwersytet
Stanisław	Nowak	Uniwersytet

Operatory teoriomnogościowe algebry relacji

☞ Suma

- Oznacza sumowanie zbiorów krotek dwóch lub więcej relacji.

☞ Suma wszystkich

- Sumuje wszystkie krotki dwóch lub więcej relacji (także, jeśli krotki się powtarzają).

☞ Przecięcie

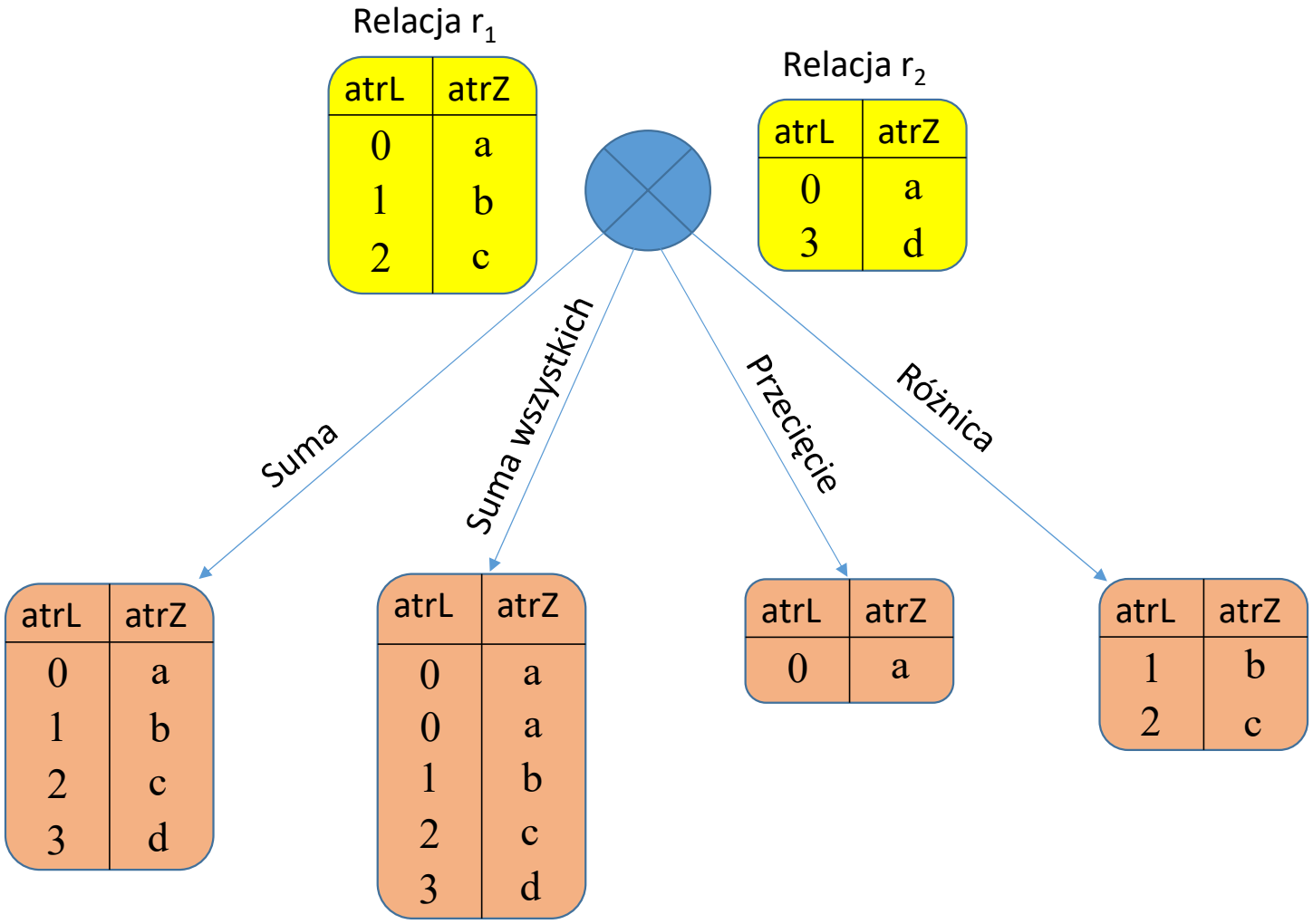
- Oznacza iloczyn zbiorów krotek dwóch lub więcej relacji, tzn. zbiór tych krotek, które występują jednocześnie w tych relacjach.

☞ Różnica

- Daje w wyniku krotki występujące wyłącznie w pierwszej spośród relacji wyjściowych.

Warunkiem wykonania operatorów teoriomnogościowych jest zgodność liczby i typów atrybutów w relacjach pełniących rolę operandów.

Operatory teoriomnogościowe - ilustracja



Języki bazy danych

- ☞ Język definiowania danych DDL (*Data Definition Language*)
  - do definiowania struktury danych przechowywanych w bazie, a więc tworzenia schematu implementacyjnego bazy.
- ☞ Język manipulowania danymi DML (*Data Manipulation Language*)
  - do zapisu, usuwania i aktualizowania danych.
- ☞ Język zapytań QL (*Query Language*)
  - do pobierania z bazy informacji zgodnych z wyspecyfikowanymi warunkami.
- ☞ Język sterowania danymi DCL (*Data Control Language*)
  - do sterowania transakcjami, np. ich wycofywania lub zatwierdzania.

Właściwości relacyjnych baz danych

- Baza danych jest „widziana” przez użytkownika (programistę aplikacji) jako zbiór tabel (relacji).
- Dostępny jest zbiór operatorów umożliwiających łączenie lub wydzielanie części relacji.
- Istotną cechą jest niezależność danych i aplikacji.
- Wzajemne powiązania danych są realizowane za pomocą samych danych (wspólnych wartości atrybutów).
- Do przetwarzania struktur relacyjnych jest stosowany język deklaratywny (nieproceduralny)
  - Oznacza to m.in., że użytkownik nie specyfikuje ścieżek dostępu do danych i nie musi znać fizycznej reprezentacji danych.

Wymienione własności obowiązują w najbardziej pełnym zakresie dla poprawnie zaprojektowanej bazy danych.

Język SQL

- Język SQL powstał w firmie IBM w latach 70-tych, jako część projektu badawczego dotyczącego relacyjnych baz danych.
- Obecnie stał się światowym standardem dla języków baz danych i występuje w produktach większości firm sprzedających oprogramowanie dla baz danych.
- Nazwa SQL jest skrótem od *Structured Query Language*, co oznacza „strukturalny język zapytań”.

Przykładowa baza danych

☞ W przykładach demonstrujących użycie SQL będzie wykorzystywana następująca baza danych:

Pracownicy

numer	nazwisko	etat	szef	pracuje_od	płaca_pod	płaca_dod	id_zesp
1000	Lech	Dyrektor	NULL	1971-01-01	3160.00	570.00	10
1080	Koliberek	Sekretarka	1000	1983-02-20	1150.00	NULL	10
1040	Rus	Adiunkt	1010	1979-09-15	1750.00	NULL	20
1070	Muszyński	Adiunkt	1010	1985-05-01	1600.00	NULL	20
1060	Misiecki	Asystent	1010	1985-03-01	1400.00	NULL	20
1090	Palusz	Asystent	1040	1989-09-15	1200.00	NULL	20
1010	Podgajny	Profesor	1000	1975-05-01	2180.00	420.00	20
1030	Maleja	Adiunkt	1020	1968-07-01	1750.00	NULL	30
1100	Warski	Asystent	1030	1987-07-15	1350.00	NULL	30
1020	Delcki	Profesor	1000	1977-09-01	2050.00	270.00	30
1110	Rajski	Stażysta	1030	1990-07-01	900.00	NULL	30
1050	Lubicz	Adiunkt	1000	1983-09-01	1780.00	NULL	40
1120	Orka	Asystent	1050	1988-04-01	1350.00	NULL	40
1130	Kolski	Stażysta	1050	1991-09-01	900.00	NULL	40

Etaty

nazwa	płaca_min	płaca_max
Stażysta	800,00	1000,00
Sekretarka	900,00	1200,00
Asystent	1000,00	1600,00
Adiunkt	1600,00	2000,00
Profesor	2000,00	2500,00
Dyrektor	2500,00	3200,00

Zespoły

id_zesp	nazwa	adres
10	Administracja	Piotrowo 3a
20	Bazy danych	Wieżowa 75
30	Sieci komputerowe	Garbary 3
40	Systemy operacyjne	Piotrowo 3a
50	Translatory	Mansfelda 4

Zapis poleceń SQL

☞ Polecenie SQL może być pisane zarówno dużymi, jak i małymi literami, w tzw. *formacie swobodnym*, co oznacza, że trzy następujące polecenia:

SELECT nazwa FROM zespoły;	SELECT nazwa FROM zespoły;	select nazwa from zespoły;

są równoważne.

☞ W każdym poleceniu można wyróżnić tzw. klauzule (*clauses*) rozpoczynające się słowem kluczowym (np. SELECT lub FROM)

- W celu zwiększenia czytelności polecenia zaleca się pisanie klauzul w osobnych wierszach (tak jak w pierwszej kolumnie tabeli).
- Słowa kluczowe mogą być zapisane zarówno dużymi jak też małymi literami.

Proste przykłady użycia polecenia SELECT

- Ważnym poleceniem języka SQL jest polecenie select służące do formułowania zapytań.
- W najprostszej formie, umożliwiającej projekcję danych pojedynczej relacji, polecenie to musi zawierać:
  - klauzulę select wskazującą atrybuty projekcji,
  - klauzulę from, wskazującą relację, której dotyczy polecenie.

Przykłady

```
select id_zesp, nazwa
from zespoły;
```

- W wyniku zostanie zwrócona zawartość kolumn id_zesp i nazwa z tabeli zespoły

```
select *
from pracownicy;
```

- W wyniku zostanie zwrócona cała zawartość tabeli pracownicy.

Operacje arytmetyczne w języku SQL (1/2)

- Proste obliczenia w języku SQL realizuje się przez umieszczanie wyrażeń arytmetycznych w klauzuli select.
- Wyrażenie składa się z liczb oraz nazw kolumn o wartościach liczbowych uzupełnionych znakami operacji takich jak: +, -, *, / .

Przykład Przypuśćmy, że adiunktom zabiera się 20% pensji na podatek dochodowy.

Obliczenie podatku można przeprowadzić wykorzystując dane z tabeli pracownicy następująco:

```
select płaca_pod, płaca_pod*0.20
from pracownicy
where etat = ‘adiunkt';
```

- Wynik

płaca_pod	(No column name)
1750.00	350.0000
1750.00	350.0000
1780.00	356.0000
1600.00	320.0000

Operacje arytmetyczne w języku SQL (2/2)

Przykład

- W tabeli pracownicy są podane miesięczne zarobki wszystkich pracowników, a dla niektórych z nich - także jednorazowa nagroda roczna zapisywana jako wartość atrybutu płaca_dod .
- W ten sposób roczny dochód pracownika jest obliczany jako płaca_pod pomnożona przez 12, przy czym wynik może być ewentualnie powiększony o wartość płaca_dod.
- Na przykład całkowite dochody roczne wszystkich profesorów można policzyć stosując następujące zapytanie:

```
select nazwisko as Nazwisko, płaca_pod*12+płaca_dod as Płaca_roczna
from pracownicy
where etat = 'profesor';
```

☞ Wynik:

Nazwisko	Płaca_roczna
Podgajny	26580.00
Delcki	24870.00

Definiowanie tabel

☞ Polecenia służące do definiowania oraz modyfikowania schematu bazy danych są podzbiorem SQL nazywanym językiem definiowania danych (DDL).

- Na kolejnych kilku slajdach zostaną omówione wybrane polecenia języka DDL.

☞ W języku SQL tabele są tworzone za pomocą polecenia `CREATE TABLE` postaci:

```
CREATE TABLE nazwa_tabeli (
nazwa_atrybutu typ(rozmiar) [DEFAULT wartość_domyślna]
     [[CONSTRAINT nazwa_ogr_atr] ograniczenie_atrybutu],
nazwa_atrybutu typ(rozmiar) [DEFAULT wartość_domyślna]
     [[CONSTRAINT nazwa_ogr_atr] ograniczenie_atrybutu],
          . . .
[[CONSTRAINT nazwa_ogr_tab] ograniczenie_tabeli, ...]
);
```

Przykład

Utworzyć tabelę etaty.

```
CREATE TABLE etaty (
  nazwa nvarchar(20) NOT NULL PRIMARY KEY,
  płaca_min money NOT NULL CHECK(płaca_min>500),
  płaca_max money NOT NULL CHECK(płaca_max<=10000)
);
```

Przykładowe typy danych (T-SQL)

Typ danych	Opis
bigint	Liczba całkowita od -2^63 (-9 223 372 036 854 775 808) do 2^63-1 (9 223 372 036 854 775 807).
int	Liczba całkowita od -2^31 (-2 147 483 648) do 2^31-1 (2 147 483 647).
smallint	Liczba całkowita od -2^15 (-32 768) do 2^15-1 (32 767).
tinyint	Liczba całkowita od 0 do 255.
bit	Liczba całkowita 0 lub 1.
decimal(precision, scale)	Liczbowy typ danych ze stałą dokładnością i podziałką.
numeric	Taki sam jak typ danych 'decimal'.
money	Pieniężny typ danych od -2^63 (-922 337 203 685 477.5808) do 2^63-1 (922 337 203 685 477.5807) z dokładnością jednej dziesięciotysięcznej jednostki.
smallmoney	Pieniężny typ danych od -2^31 (-214 748.3648) do 2^31-1 (214 748.3647) z dokładnością jednej dziesięciotysięcznej.
float(n)	Liczbowy typ danych ze zmienną dokładnością, gdzie n wynosi ilość bitów mantysy
real	Liczbowy typ danych ze zmienną dokładnością.
datetime	Typ danych określający datę i czas od 1.1.1753 do 31.12.9999 z dokładnością około 3ms. Wartości są zaokrąglone na .000, .003 a .007.
smalldatetime	Typ danych określający datę i czas od 1.1.1900 do 6.6.2079 z dokładnością 1m. Wartości do 29.998 są zaokrąglane w dół a wartości od 29.999 są zaokrąglane w górę na najbliższą minutę.
char	Łańcuch znaków o stałej długości, długość 8000 znaków.
varchar	Łańcuch znaków o zmiennej długości, długość maksymalna 8000 znaków.
nchar	Łańcuch znaków Unicode o stałej długości, długość maksymalna 4000 znaków.
nvarchar	Łańcuch znaków Unicode o zmiennej długości, długość maksymalna 4000 znaków.
image	Dane binarne o zmiennej długości, długość maksymalna 2^31-1 (2 147 483 647) bajtów.

Ograniczenia integralnościowe dla atrybutów

Ograniczenie	Znaczenie
NULL	Wskazuje, że atrybut może przyjmować wartości puste.
NOT NULL	Uniemożliwia nadawanie atrybutowi wartości pustych.
UNIQUE	Atrybut pełni rolę klucza unikalnego relacji (tzn. wartość atrybutu jest unikalna dla wszystkich krotek relacji).
PRIMARY KEY	Atrybut pełni rolę klucza podstawowego relacji.
REFERENCES	Określa tzw. ograniczenie referencyjne, tj. referencję do klucza podstawowego lub unikalnego innej relacji. Ograniczenie to jest wykorzystywane do definiowania tzw. klucza obcego relacji.
ON DELETE CASCADE	Ograniczenie to wprowadza się dla klucza obcego, np.: id_zesp number(4) references zespół(id_zesp) on delete cascade Jeżeli zostanie usunięta krotka z relacji z kluczem podstawowym, to automatycznie są usuwane te krotki z relacji z kluczem obcym, dla których wartość klucza obcego jest równa wartości klucza podstawowego usuwanej krotki.
CHECK	Określa warunek, który musi być spełniony dla każdej krotki w tabeli.

Ograniczenia integralnościowe dla tabel

- Jako ostatni element definicji tabeli można, określić ograniczenia integralnościowe dla tabeli.
- Ograniczenia dla tabeli różnią się od ograniczeń dla atrybutów tym, że mogą dotyczyć więcej niż jednego atrybutu danej tabeli.
  - Do ograniczeń tego typu należą:

```
UNIQUE,
PRIMARY KEY,
REFERENCES,
ON DELETE CASCADE,
CHECK.
```

- Dodatkowo dla tabeli istnieje ograniczenie `FOREIGN KEY` umożliwiające zdefiniowanie klucza obcego złożonego z wielu atrybutów.
- Każdemu ograniczeniu można opcjonalnie przypisać nazwę.

Modyfikowanie schematu tabeli (1/4)

☞ Dodanie atrybutu:

- W celu dodania atrybutu do tabeli jest stosowane polecenie `ALTER TABLE` postaci:

```
ALTER TABLE nazwa_tabeli
ADD nazwa_atrybutu typ(rozmiar) [DEFAULT wartość_domyślna]
       [[CONSTRAINT nazwa_ogr_atr] ograniczenie_atrybutu];
```

Przykład

Do tabeli pracownicy dodać nowy atrybut o nazwie tytuł_naukowy.

```
alter table pracownicy
add tytuł_nauk nvarchar(10);
```

Modyfikowanie schematu tabeli (2/4)

☞ Dodanie ograniczenia integralnościowego dla tabeli:

- Dodanie ograniczenia integralnościowego dla tabeli jest możliwe przy pomocy polecenia `ALTER TABLE` o następującym formacie:

```
ALTER TABLE nazwa_tabeli
ADD [CONSTRAINT nazwa_ogr_tab] ograniczenie_tabeli;
```

Przykład

Atrybut id_zesp tabeli pracownicy zdefiniować jako klucz obcy tej tabeli, wskazujący na atrybut kluczowy id_zesp tabeli zespoły.

```
alter table pracownicy
add constraint prac_fk foreign key (id_zesp) references zespoły (id_zesp);
```

Modyfikowanie schematu tabeli (3/4)

☞ Modyfikowanie typu atrybutu:

- Innym możliwym zastosowaniem polecenia `ALTER TABLE` jest modyfikowanie typu atrybutu (kolumny). Stosuje się wtedy klauzulę `ALTER COLUMN`:

```
ALTER TABLE nazwa_tabeli
  ALTER COLUMN nazwa_atrybutu typ_atrybutu;
```

Przykład

Zmodyfikować definicję atrybutu tytuł_nauk zwiększając dopuszczalną długość do 15 znaków.

```
alter table pracownicy
alter column tytuł_nauk nvarchar(15);
```

Modyfikowanie schematu tabeli (4/4)

☞ Usuwanie kolumny w tabeli:

- Polecenie `ALTER TABLE` może posłużyć także do usuwania kolumny z tabeli. Stosuje się wtedy klauzulę `DROP COLUMN`:

```
ALTER TABLE nazwa_tabeli
  DROP COLUMN nazwa_kolumny;
```

Przykład

Zmodyfikować definicję tabeli pracownicy usuwając kolumnę tytuł_nauk.

```
alter table pracownicy
drop column tytuł_nauk;
```

Zmiana nazwy i usuwanie tabeli

☞ Nazwa tabeli może zostać zmieniona za pomocą procedury sp_rename następująco:

```
sp_rename stara_nazwa , nowa_nazwa;
```

Przykład

Zmienić nazwę tabeli pracownicy na personel a następnie powrócić do poprzedniej nazwy.

```
sp_rename pracownicy , personel;
sp_rename personel , pracownicy;
```

☞ Tabela może zostać usunięta za pomocą polecenia `DROP TABLE` użytego zgodnie z następującym schematem:

```
DROP TABLE nazwa_tabeli;
```

Przykład

Usunąć tabelę pracownicy.

```
drop table pracownicy;
```

Operacje DML – zapisywanie danych do tabeli

☞ Kolejna grupa poleceń, jaka zostanie zaprezentowana, stanowi podzbiór języka SQL określany niekiedy jako język manipulowania danymi (DML).

☞ Zapisywanie danych do tabeli odbywa się przy pomocy polecenia `INSERT INTO` , które ma następujący format:

```
INSERT INTO tabela [(kolumna1, kolumna2, …)]
VALUES
(wartość1, wartość2,…),
(wartość1, wartość2, …),
  … ;
```

Przykład

Zapisać wszystkie dane do tabeli zespoły.

```
INSERT INTO zespoły
VALUES
 (10,'Administracja','Piotrowo 3a'),
 (20,'Bazy danych','Wieżowa 75'),
 (30,'Sieci komputerowe','Garbary 3'),
 (40,'Systemy operacyjne','Piotrowo 3a'),
 (50,'Translatory','Mansfelda 4');
```

UPDATE – Modyfikowanie zawartości tabel (1/2)

☞ Do modyfikowania zawartości tabeli służy polecenie `UPDATE`.

- W przypadku, gdy zmieniamy wartości pewnych atrybutów w poleceniu `UPDATE` można zamieścić wyrażenia wyznaczające nowe wartości posługując się następującą składnią:

```
UPDATE tabela
SET kolumna1 = wyrażenie1,
    kolumna2 = wyrażenie2,
    ...
[WHERE warunki];
```

Przykład

Wszystkim adiunktom należy podwyższyć płace podstawowe o 100 zł.

```
update pracownicy
set płaca_pod = płaca_pod+100
where etat = 'adiunkt';
```

UPDATE – Modyfikowanie zawartości tabel (2/2)

- Atrybuty można także modyfikować nadając im wartości pobrane z bazy. W takich przypadkach używamy polecenia `UPDATE` zgodnie z następującą składnią:

```
UPDATE tabela_1
SET kolumna1 = (SELECT wyrażenie1
                FROM tabela_2
                WHERE warunki)
[WHERE warunki];
```

Przykład

Wszystkim asystentom przydzielić maksymalną płacę przewidzianą na tym etacie.

```
UPDATE pracownicy
SET płaca_pod = (SELECT płaca_max
                 FROM etaty
                 WHERE nazwa = 'asystent')
WHERE etat = 'asystent';
```

Operację modyfikowania płac asystentów realizuje pokazane polecenie UPDATE a efekt można zaobserwować wyświetlając nowe wartości poleceniem SELECT.

```
SELECT nazwisko, etat, płaca_pod
FROM pracownicy
WHERE etat = 'asystent'
```

	nazwisko	etat	płaca_pod
1	Misiecki	Asystent	1600.00
2	Palusz	Asystent	1600.00
3	Warski	Asystent	1600.00
4	Orka	Asystent	1600.00

Tworzenie zapytań

- Tworzenie zapytań w języku SQL odbywa się przy pomocy polecenia SELECT .
- Polecenie to można stosować w bardzo prostej postaci, w której wystarczy użyć tylko dwóch obowiązkowych klauzul SELECT i FROM.
- Możliwe są też formy znacznie bardziej rozbudowane, pozwalające formułować nawet złożone zapytania.
- Oprócz obowiązkowych klauzul SELECT i FROM w zapytaniach SELECT stosuje się szereg innych klauzul, do których zalicza się m.in.:
  - WHERE
  - ORDER BY
  - GROUP BY
  - HAVING

Klauzula WHERE (1/3)

☞ Klauzula WHERE jest klauzulą opcjonalną polecenia SELECT, realizującą operację selekcji algebry relacji. Ogólna postać polecenia SELECT jest w tym przypadku następująca:

```
SELECT atrybuty_projekcji
FROM tabela
WHERE warunki_do_spełnienia;
```

- W specyfikacji „warunki_do_spełnienia” można użyć jednego z dziesięciu operatorów opisanych w tabeli.

Operator	Opis
=	Równość
!=	Różność
>	Większość
>=	Większość lub równość
<	Mniejszość
<=	Mniejszość lub równość
is null	Sprawdzenie, czy wartość atrybutu jest wartością pustą
between... and...	Sprawdzenie, czy lewy operand jest zawarty w przedziale określonym po słowach **between** i **and**
in(zbiór)	Sprawdzenie, czy lewy operand należy do zbioru literałów podanego jako prawy operand
like	Porównanie wartości atrybutu podanego jako lewy operand z tzw. wzorcem dopasowania. Wzorzec jest konstruowany za pomocą dwóch znaków specjalnych: „%” i „_”. Pierwszy z nich reprezentuje dowolny łańcuch znaków (także pusty), a drugi reprezentuje pojedynczy znak

Klauzula WHERE (2/3)

Przykład

Wybrać wszystkich pracowników, których pensja podstawowa jest mniejsza od sześciokrotnej pensji dodatkowej.

```
select nazwisko, płaca_pod, płaca_dod
from pracownicy
where płaca_pod<6*płaca_dod;
```

Wynik

	nazwisko	płaca_pod	płaca_dod
1	Lech	3160.00	570.00
2	Podgajny	2180.00	420.00

Przykład

Wyszukać wszystkich pracowników, których nazwiska zaczynają się na literę „L”.

```
select nazwisko
from pracownicy
where nazwisko like 'L%';
```

Wynik

	nazwisko
1	Lech
2	Lubicz

Klauzula WHERE (3/3)

- Warunek selekcji zadany w klauzuli **WHERE** może zawierać także operatory logiczne AND, OR i NOT.
- W efekcie można wyróżnić cztery poziomy priorytetów operatorów stosowanych w warunku selekcji, które w kolejności od najwyższego do najniższego są następujące:
  - poziom 1: =, !=, <, >, <=, >=, BETWEEN … AND …, IN, LIKE, IS NULL
  - poziom 2: NOT
  - poziom 3: AND
  - Poziom 4: OR.

Przykład

Wyszukać wszystkich adiunktów i asystentów, których płaca podstawowa jest wyższa od 1750.

```
select nazwisko, etat, płaca_pod
from pracownicy
where płaca_pod>1750 and (etat='adiunkt' or etat='asystent');
```

Wynik

	nazwisko	etat	płaca_pod
1	Lubicz	Adiunkt	1780.00

Klauzula ORDER BY

☞ Klauzula ORDER BY wskazuje atrybut, według którego zostaną posortowane wiersze otrzymane w wyniku zapytania.

- Klauzula ta, jeśli występuje, powinna być zawsze umieszczana jako ostatnia w poleceniu SELECT.

Przykład

Sporządzić spis wszystkich asystentów w kolejności określonej datą ich zatrudnienia.

```
select nazwisko, pracuje_od
from pracownicy
where etat='asystent'
order by pracuje_od;
```

Wynik

	nazwisko	pracuje_od
1	Misiecki	1985-03-01
2	Warski	1987-07-15
3	Orka	1988-04-01
4	Palusz	1989-09-15

Klauzula GROUP BY (1/2)

☞ Klauzula GROUP BY umożliwia podział krotek na grupy, według wartości podanego atrybutu.

- Wydzielenie grup umożliwia zastosowanie funkcji grupowych (np. zliczających liczbę krotek w grupie).

Przykład

Podać liczbę osób zatrudnionych w poszczególnych zespołach.

```
select id_zesp, count(*) Liczba_zatr
from pracownicy
group by id_zesp
order by id_zesp;
```

Wynik

	id_zesp	Liczba_zatr
1	10	2
2	20	5
3	30	4
4	40	3

Klauzula GROUP BY (2/2)

☞ Grupować można „rekurencyjnie”, według kolejnych atrybutów wskazanych w klauzuli.

- Grupowaniu może towarzyszyć nie tylko projekcja realizowana przez klauzulę SELECT ale także selekcja realizowana przy użyciu WHERE.

Przykład

Dla każdego zespołu należy wyznaczyć liczbę osób zatrudnionych na poszczególnych etatach występujących w tym zespole. Z zestawienia wyłączyć dyrektora.

```
select id_zesp, etat, count(*) Liczba_zatr
from pracownicy
where etat!='dyrektor'
group by id_zesp, etat
order by id_zesp;
```

Wynik

	id_zesp	etat	Liczba_zatr
1	10	Sekretarka	1
2	20	Adiunkt	2
3	20	Asystent	2
4	20	Profesor	1
5	30	Adiunkt	1
6	30	Asystent	1
7	30	Profesor	1
8	30	Stażysta	1
9	40	Adiunkt	1
10	40	Asystent	1
11	40	Stażysta	1

Klauzula HAVING

☞ W odróżnieniu od klauzuli WHERE, która operuje na pojedynczych krotkach, za pomocą klauzuli HAVING można dokonywać selekcji na wcześniej wydzielonych grupach.

- Należy przy tym zaznaczyć, że tworzenie grup i obliczanie funkcji grupowych jest realizowane przed selekcją.

Przykład

Podać zestawienie pokazujące wysokość średniej płacy podstawowej w zespołach liczących powyżej 3-ch pracowników.

```
select id_zesp, avg(płaca_pod) Średnia_płaca
from pracownicy
group by id_zesp
having count(*) > 3;
```

Wynik

	id_zesp	Średnia_płaca
1	20	1626.000000
2	30	1512.500000

Poziome łączenie relacji (1/2)

- W przypadkach, gdy wymagana informacja musi być pozyskana z więcej niż jednej tabeli, stosuje się operacje łączenia relacji (tabel).
- Poziome łączenie polega na utworzeniu relacji wynikowej, której krotki są wynikiem konkatenacji (złączenia) krotek relacji źródłowych.
  - Jedną z możliwości jest wyznaczenie iloczynu kartezjańskiego tj. wszystkich możliwych kombinacji połączeń krotek obu relacji. Łączone w ten sposób relacje wymienia się kolejno w klauzuli FROM polecenia SELECT, np.:

```
select *
From pracownicy, zespoły;
```

- Ze względu na duży rozmiar danych wynikowych, takie łączenie stosuje się rzadko.
  - W praktyce krotki jednej relacji są łączone z krotkami innej relacji tylko wtedy, gdy wartości korespondujących atrybutów tych krotek spełniają warunek określony w klauzuli WHERE.
  - Taka operacja połączenia ma następujący format ogólny:

```
SELECT atrybut(y)
FROM łączone_tabele
WHERE warunek_połączenia;
```

Poziome łączenie relacji (2/2)

☞ Jeśli łączone relacje mają atrybuty o tych samych nazwach, nazwy te są poprzedzone nazwami relacji.

- Takie prefiksowanie atrybutów jest stosowane także w klauzulach SELECT, GROUP BY i ORDER BY.
- Jeśli prefiksowanie ma być stosowane wielokrotnie jest możliwe użycie aliasów (krótkich nazw) zdefiniowanych w klauzuli FROM.

Przykład

Podać nazwy zespołów w jakich pracują poszczególni pracownicy.

```
select nazwisko,nazwa
from pracownicy p, zespoły z
where p.id_zesp = z.id_zesp
order by p.id_zesp;
```

Wynik →

	nazwisko	nazwa
1	Lech	Administracja
2	Koliberek	Administracja
3	Palusz	Bazy danych
4	Podgajny	Bazy danych
5	Rus	Bazy danych
6	Misiecki	Bazy danych
7	Muszyński	Bazy danych
8	Delcki	Sieci komputerowe
9	Maleja	Sieci komputerowe
10	Warski	Sieci komputerowe
11	Rajski	Sieci komputerowe
12	Orka	Systemy operacyjne
13	Kolski	Systemy operacyjne
14	Lubicz	Systemy operacyjne

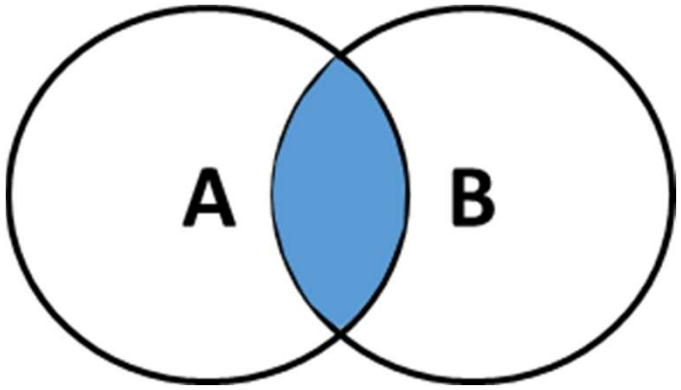
Użycie konstrukcji INNER JOIN

Inny zapis zapytania z poprzedniego slajdu

```
SELECT nazwisko, nazwa
FROM pracownicy p, zespoły z
WHERE p.id_zesp = z.id_zesp
ORDER BY p.id_zesp;
```

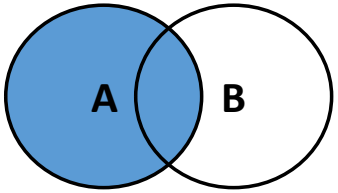
```
SELECT nazwisko, nazwa
FROM pracownicy p
INNER JOIN zespoły z
ON p.id_zesp = z.id_zesp
ORDER BY p.id_zesp;
```

Ilustracja konstrukcji INNER JOIN

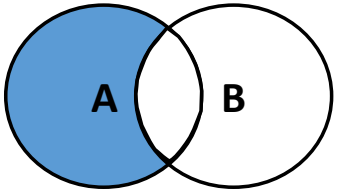



```
SELECT <atrybuty_projekcji>
FROM tabelaA a
INNER JOIN tabelaB b
ON a.klucz = b.klucz
```

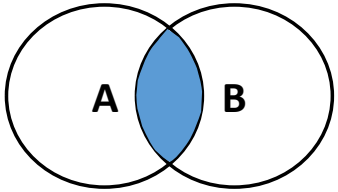
Połączenia JOIN



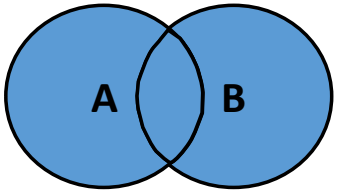

```
SELECT <atrybuty_projekcji>
FROM tabelaA a
LEFT JOIN tabelaB b
ON a.klucz = b.klucz
```



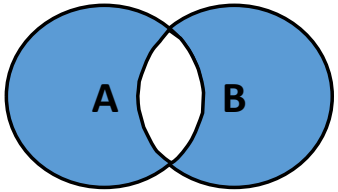

```
SELECT <atrybuty_projekcji>
FROM tabelaA a
INNER JOIN tabelaB b
ON a.klucz = b.klucz
WHERE b.klucz IS NULL
```



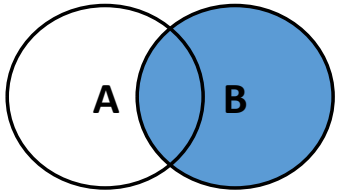

```
SELECT <atrybuty_projekcji>
FROM tabelaA a
INNER JOIN tabelaB b
ON a.klucz = b.klucz
```



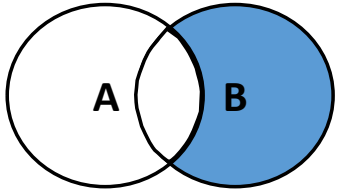

```
SELECT <atrybuty_projekcji>
FROM tabelaA a
FULL OUTER JOIN tabelaB b
ON a.klucz = b.klucz
```




```
SELECT <atrybuty_projekcji>
FROM tabelaA a
FULL OUTER JOIN tabelaB b
ON a.klucz = b.klucz
WHERE a.klucz IS NULL OR b.klucz IS NULL
```




```
SELECT <atrybuty_projekcji>
FROM tabelaA a
RIGHT JOIN tabelaB b
ON a.klucz = b.klucz
```




```
SELECT <atrybuty_projekcji>
FROM tabelaA a
INNER JOIN tabelaB b
ON a.klucz = b.klucz
WHERE a.klucz IS NULL
```

Pionowe łączenie relacji (1/2)

- Pionowe łączenie relacji polega na utworzeniu tabeli wynikowej, której krotki są sumą, częścią wspólną lub różnicą krotek relacji źródłowych.
  - Oznacza to, że łączenie odbywa się przez zastosowanie jednego z operatorów mnogościowych algebry relacji, tj. unii, przekroju lub różnicy.
- Zapytanie zawiera wtedy dwie lub więcej klauzule SELECT, mające tę samą liczbę atrybutów zgodnych typów.
  - W wyniku pojawiają się nazwy atrybutów wyłącznie z pierwszej klauzuli SELECT.
  - Jeśli zostaje użyta klauzula ORDER BY, to musi ona wystąpić jako ostatnia i zamiast nazw atrybutów zawierać ich numery porządkowe.
- Ogólny format zapytania realizującego połączenie pionowe pokazano obok.
  - Występujący tutaj „`operator`” przyjmuje jedną z wartości: `union`, `union all`, `intersect` lub `except`.

```
SELECT atrybut_A1,..., atrybut_An
FROM tabela_A
WHERE warunki
      operator
SELECT atrybut_B1,..., atrybut_Bn
FROM tabela_B
WHERE warunki
ORDER BY 1,...,n;
```

Pionowe łączenie relacji (2/2)

Przykład

Określić te etaty w zespołach 30 i 40, dla których pewnym pracownikom należącym do różnych zespołów przyznano jednakowe płace podstawowe.

```
select etat, płaca_pod
from pracownicy
where id_zesp=30
    intersect
select etat, płaca_pod
from pracownicy
where id_zesp=40
order by 2;
```

Wynik

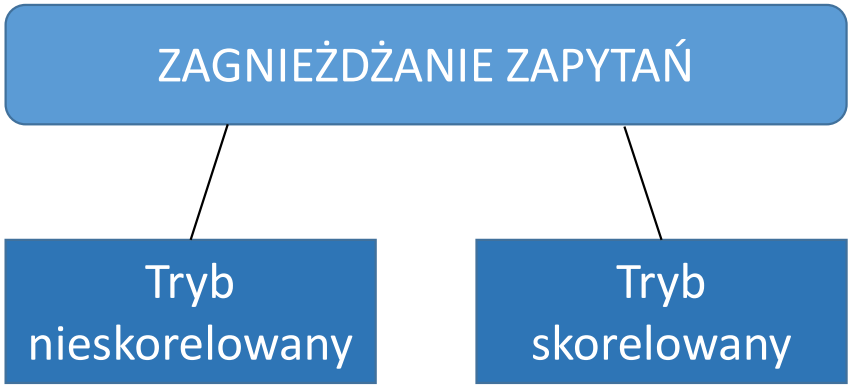
	etat	płaca_pod
1	Stażysta	900.00
2	Asystent	1350.00

Stażyści Rajski i Kolski, z których każdy reprezentuje inny spośród wymienionych zespołów, mają jednakowe płace podstawowe wynoszące po 900. Także asystenci Warski i Orka będący w różnych zespołach mają jednakowe płace podstawowe wynoszące po 1350. Wyświetlona informacja wynikowa została uporządkowana (rosnąco) według wartości atrybutu płaca_pod.

Zagnieżdżanie zapytań

- Polecenia select mogą być w sobie zagnieżdżane, co pozwala realizować bardziej złożone operacje na relacjach.
- Zapytanie zagnieżdżone jest także nazywane podzapytaniem (subquery) lub zapytaniem wewnętrznym w odróżnieniu od zawierającego go zapytania zewnętrznego.

- Można wyodrębnić dwa następujące sposoby zagnieżdżania zapytań:
  - zagnieżdżanie w trybie nieskorelowanym oraz
  - zagnieżdżanie w trybie skorelowanym.



Zagnieżdżanie zapytań – tryb nieskorelowany

- W podstawowym trybie zagnieżdżania zwanym nieskorelowanym podzapytanie jest wykonywane jako pierwsze, jednokrotnie, a wyniki podzapytania są przekazywane do zapytania zewnętrznego.
  - Ogólny format zagnieżdżania zapytań w tym trybie pokazano obok

```
SELECT atrybut_A1,..., atrybut_An
FROM tabela_A
WHERE atrybut
        operator
(SELECT atrybut_B1,..., atrybut_Bn
FROM tabela_B
WHERE warunek);
```

- W przypadku, gdy podzapytanie wyznacza dokładnie jedną krotkę, w warunku selekcji zapytania zewnętrznego stosujemy najczęściej jeden z operatorów porównania, np. = lub >=.
- Jeśli natomiast podzapytanie wyznacza więcej niż jedną krotkę stosujemy operatory: `in`, `any` lub `all`.
  - Operator `in` sprawdza przynależność do zbioru wartości wyznaczonego przez podzapytanie.
  - Operator `any` powoduje porównanie pojedynczej, zadanej wartości z każdą wartością wyznaczoną przez podzapytanie. Warunek selekcji jest spełniony, jeśli przynajmniej jedno z tych porównań da pozytywny rezultat.
  - Operator `all` powoduje porównanie pojedynczej zadanej wartości z każdą wartością wyznaczoną przez podzapytanie. Warunek selekcji jest spełniony, jeśli każde z tych porównań da pozytywny rezultat.

Tryb nieskorelowany - przykłady

Przykład

Wyświetlić wszystkich pracowników zatrudnionych na tym samym etacie co pracownik Orka (asystenci), wraz z ich płacą podstawową.

```
select nazwisko,płaca_pod from pracownicy
where etat =
(select etat from pracownicy
 where nazwisko='Orka');
```

Wynik

	nazwisko	płaca_pod
1	Misiecki	1400.00
2	Palusz	1200.00
3	Warski	1350.00
4	Orka	1350.00

Przykład

Wyznaczyć wszystkich pracowników, którzy zarabiają mniej niż każdy pracownik z zespołu 20.

```
select nazwisko,płaca_pod from pracownicy
where płaca_pod < all
(select płaca_pod from pracownicy
 where id_zesp=20);
```

Wynik

	nazwisko	płaca_pod
1	Koliberek	1150.00
2	Rajski	900.00
3	Kolski	900.00

W odróżnieniu od przykładu poprzedzającego, gdzie podzapytanie wyznacza pojedynczą wartość, tutaj zapytanie wewnętrzne wyznacza grupę wartości, dla której użyto operatora `all`.

Zagnieżdżanie zapytań – tryb skorelowany

- W trybie skorelowanym najpierw jest wykonywane zapytanie zewnętrzne, a dopiero potem skorelowane z nim podzapytanie.
  - Podzapytanie skorelowane jest wykonywane tyle razy, ile razy jest wykonywane zapytanie zewnętrzne.
- Składniowo zapytania skorelowane od nieskorelowanych różni konieczność zastosowania aliasów relacji, na których operuje zapytanie zewnętrzne i odwoływania się do nich w podzapytaniu.

Przykład

Wyznaczyć nazwiska, płace podstawowe i etaty pracowników, których płaca podstawowa jest wyższa niż przeciętna dla etatu, na którym są zatrudnieni. Uzyskany wynik uporządkować według wartości atrybutu nazwisko.

```
select nazwisko,płaca_pod,etat from pracownicy p
where płaca_pod >
(select avg(płaca_pod) from pracownicy
 where etat = p.etat)
order by nazwisko;
```

Wynik

	nazwisko	płaca_pod	etat
1	Lubicz	1780.00	Adiunkt
2	Maleja	1750.00	Adiunkt
3	Misiecki	1400.00	Asystent
4	Orka	1350.00	Asystent
5	Podgajny	2180.00	Profesor
6	Rus	1750.00	Adiunkt
7	Warski	1350.00	Asystent

Zapytanie zewnętrzne przegląda krotki pracowników, przekazując je do podzapytania skorelowanego. W podzapytaniu jest wyznaczana przeciętna płaca na etacie pracownika analizowanego przez zapytanie zewnętrzne.

Tryb skorelowany – jeszcze jeden przykład

☞ W przypadku zapytań skorelowanych możemy wykorzystać nowy operator `exists` lub jego negację `not exists` celem sprawdzenia, czy podzapytanie wyznacza jakąkolwiek wartość (bądź nie wyznacza żadnej).

Przykład

Podać numery, nazwiska oraz nazwy etatów pracowników zatrudnionych na etatach, na których nie pracuje nikt inny. Uzyskany wynik uporządkować według wartości atrybutu numer.

```
select numer,nazwisko,etat
from pracownicy p
where not exists
(select numer
 from pracownicy
 where etat=p.etat and numer!=p.numer)
order by numer;
```

Wynik

	numer	nazwisko	etat
1	1000	Lech	Dyrektor
2	1080	Koliberek	Sekretarka

Użyty w przykładzie operator `exists` umożliwia sprawdzenie, czy dla analizowanego pracownika w tabeli pracownicy występuje inna osoba pracująca na tym samym etacie i mająca inny numer pracownika.

Funkcje w języku SQL

- Ważnym elementem języka SQL są funkcje wbudowane
  - Są to funkcje gotowe do użycia po uruchomieniu systemu, które mogą być wywołane w poleceniach SQL.
- Zestawy dostępnych funkcji SQL mogą nieco różnić się od siebie w konkretnych realizacjach języka
  - Na przykład funkcje w systemie Oracle mogą nie być identyczne jak w systemie MS SQL Server.
- Dla ilustracji rozważymy wybrane funkcje SQL dostępne w systemie MS SQL Server. Przykłady dotyczyć będą użycia funkcji należących do następujących kategorii:
  - funkcje znakowe
  - funkcje numeryczne
  - funkcje daty i czasu oraz
  - funkcje konwersji

Funkcje znakowe

☞ Funkcja CHAR zwraca znak na podstawie podanego kodu liczbowego ASCII.

- Znaki można łączyć w „słowa” przy pomocy operatora +

Przykład

Dla pracowników zespołu 10 złożyć pierwsze trzy litery nazwy ich zespołu (Adm) z wprowadzonych kodów dla znaków „A”, „d” i „m”.

```
SELECT nazwisko, CHAR(65)+CHAR(100)+CHAR(109) as Skrót_nazwy_zespołu
FROM pracownicy
WHERE id_zesp = 10;
```

Wynik

	nazwisko	Skrót_nazwy_zespołu
1	Lech	Adm
2	Koliberek	Adm

Funkcje numeryczne

☞ Funkcja ROUND zwraca liczbę zaokrągloną do zadanej ilości miejsc po przecinku.

Przykład

Podzielić płace pracowników zespołu 10 na 7 a następnie zaokrąglić wyniki do 2-ch miejsc po przecinku.

```
SELECT nazwisko, płaca_pod, płaca_pod/7 as Podz_na_7, ROUND(płaca_pod/7,2) as Zaokr
FROM pracownicy
WHERE id_zesp = 10;
```

Wynik

	nazwisko	płaca_pod	Podz_na_7	Zaokr
1	Lech	3160.00	451.428571	451.430000
2	Koliberek	1150.00	164.285714	164.290000

Funkcje operujące na datach

☞ Funkcja DATEADD zwraca datę po dodaniu do niej określonej liczby jednostek (np. dni).

Przykład

Do dat zatrudnienia pracowników zespołu 10 dodać 400 dni.

```
SELECT nazwisko, pracuje_od, DATEADD(day, 400, pracuje_od) as Pracuje_od_plus400
FROM pracownicy
WHERE id_zesp = 10;
```

Wynik

	nazwisko	pracuje_od	Pracuje_od_plus400
1	Lech	1971-01-01	1972-02-05
2	Koliberek	1983-02-20	1984-03-26

Funkcje konwersji typów

☞ Funkcja CAST dokonuje konwersji wyrażenia pewnego typu na inny typ.

Przykład

Dla każdego z pracowników zespołu 10 podać trzy dane: (i) datę zatrudnienia, (i) bieżącą datę i czas oraz (iii) tylko bieżącą datę. Do uzyskania bieżącej daty i czasu można użyć funkcji CURRENT_TIMESTAMP, która zwraca daną typu datetime.

```
SELECT nazwisko, pracuje_od, CURRENT_TIMESTAMP as CTS, CAST(CURRENT_TIMESTAMP as date) as CTS_tylko_data
FROM pracownicy
WHERE id_zesp = 10;
```

Wynik

	nazwisko	pracuje_od	CTS	CTS_tylko_data
1	Lech	1971-01-01	2019-04-21 18:37:13.710	2019-04-21
2	Koliberek	1983-02-20	2019-04-21 18:37:13.710	2019-04-21

Funkcje operujące grupach krotek (1/2)

☞ W odróżnieniu od poprzednio omawianych, tego typu funkcje operują nie na pojedynczych wartościach lecz na grupach wartości.

- Wynik jest wyznaczany na podstawie wartości pochodzących z pewnej liczby krotek.
- Wydzielenie grup krotek jest możliwe za pomocą klauzuli GROUP BY.

Przykład

Wyznaczyć maksymalne płace w poszczególnych zespołach.

```
select id_zesp, max(płaca_pod) as Płaca_maks
from pracownicy
group by id_zesp;
```

Wynik

	id_zesp	Płaca_maks
1	10	3160.00
2	20	2180.00
3	30	2050.00
4	40	1780.00

Funkcje operujące grupach krotek (2/2)

☞ Przydatną funkcją operującą na grupie krotek jest funkcja COUNT.

- W szczególności można jej użyć do wyznaczenia całkowitej liczby krotek w wydzielonych grupach, co zilustrowano na przykładzie.

Przykład

Wyznaczyć liczbę osób pracujących na poszczególnych etatach w zespole 20.

```
select id_zesp, etat, count(*) Liczba_zatr
from pracownicy
where id_zesp=20
group by id_zesp, etat;
```

Wynik

	id_zesp	etat	Liczba_zatr
1	20	Adiunkt	2
2	20	Asystent	2
3	20	Profesor	1