Wprowadzenie

01 – SELECT – podstawy, filtrowanie wierszy

Na początku krótko omówmy ogólnie obiekty, na których będziemy pracować. Każda instancja naszych danych (np. pracownik) to **wiersz/krotka**, która jest opisana przy pomocy pewnych **kolumn/atrybutów**. Wszystkie instancje tego samego typu przechowujemy w **tabeli**. Tabele dotyczące modelowanego przez nas świata przechowujemy w **bazie danych**. Różne bazy danych mogą znajdować się na jednym **serwerze**.

Inaczej mówiąc, na serwerze może być wiele różnych baz danych. Baza danych jest swego rodzaju kontenerem powiązanych logicznie ze sobą tabel. Tabele są opisane przez atrybuty/kolumny; tabele przechowują wiersze/krotki, gdzie jeden wiersz stanowi jedną instancję obiektu opisywanego przez tę tabelę.

Na pierwszych zajęciach pracujemy na serwerze lokalnym. Założyliśmy na nim bazę danych Projekty, której celem jest przechowywanie informacji dotyczących projektów realizowanych w pewnej jednostce naukowej. W tej bazie danych mamy kilka tabel, które opisują tę rzeczywistość, tj. Pracownicy, Projekty, Realizacje i Stanowiska. Każda tabela jest opisana przez kolumny/atrybuty, np. Pracownicy mają id, nazwisko, placa, itd. W każdej tabeli przechowujemy wiersze, np. pojedynczy wiersz w tabeli Pracownicy opisuje jakiegoś pracownika.

SELECT – podstawy

Na zajęciach omawiamy dialekt SQL – [Transact-SQL](https://docs.microsoft.com/en-us/sql/t-sql/language-reference" \t "_blank) (T-SQL).

Przed rozpoczęciem pracy upewnij się, że masz utworzoną bazę [projekty-create-insert.sql](https://www.elearning.amu.edu.pl/wmi/mod/resource/view.php?id=3383). Obejrzyj tabele, atrybuty i zawartość tych tabel. Zauważ, że skrypt jest tak przygotowany, że w każdej chwili możesz go ponownie uruchomić i utworzyć tabele na nowo (będzie to szczególnie przydatne, gdy zaczniemy ćwiczyć modyfikowanie danych).

Polecenie [SELECT](https://docs.microsoft.com/en-us/sql/t-sql/queries/select-transact-sql) pozwala wyświetlić dane zawarte w jednej lub wielu tabelach. Podstawowa składnia:

SELECT lista atrybutow

FROM lista tabel

WHERE warunki filtrowania

GROUP BY lista atrybutow

HAVING warunki dotyczace grup

ORDER BY lista atrybutow;

Wynikiem polecenia SELECT jest zawsze **relacja** (inaczej – tabela, czyli wiersze i kolumny). Z tego powodu na wyniku polecenia SELECT można m.in. wykonać kolejne polecenie SELECT.

Projekcja kolumn

01 – SELECT – podstawy, filtrowanie wierszy

Najbardziej podstawowym użyciem polecenia SELECT jest wyświetlanie określonych kolumn tabeli. Dodatkowo można dokonać tzw. przemianowania, czyli zmienić nazwy kolumn (ustawić aliasy).

Przykład 1

Wyświetlenie nazwiska i stanowiska każdego pracownika:

SELECT nazwisko,

stanowisko

FROM Pracownicy;

Średnik na końcu zapytania nie jest wymagany, ale zaleca się jego stosowanie w celu zachowania czytelności zapytania.

Przykład 2

Gdy tabela ma bardzo dużo kolumn lub po prostu chcemy wyświetlić jej wszystkie atrybuty, możemy ułatwić sobie pracę i użyć aliasu \*. Poniższe zapytanie wyświetla wszystkie atrybuty z tabeli *Projekty*:

SELECT \*

FROM Projekty;

Przykład 3

W T-SQL podczas operacji SELECT można dokonać zmiany nazw kolumn przy pomocy tzw. aliasów. Możemy to zrobić na co najmniej sześć sposobów:

SELECT [Nazwa projektu] = nazwa,

dataRozp AS [Data rozpoczecia projektu],

dataZakonczPlan [Planowana data zakonczenia],

dataZakonczFakt data\_zakonczenia\_faktycznego,

kierownik 'Kierownik projektu',

stawka "Stawka w PLN"

FROM Projekty;

Funkcje wierszowe, wyrażenia, konwersja typów

01 – SELECT – podstawy, filtrowanie wierszy

Na stronie Microsoft Docs znajduje się wyczerpująca lista [funkcji wbudowanych w SQL Server](https://docs.microsoft.com/en-us/sql/t-sql/functions/functions). Poniżej znajdują się często wykorzystywane funkcje.

Funkcje wierszowe na łańcuchach znaków

- [SUBSTRING()](https://docs.microsoft.com/en-us/sql/t-sql/functions/substring-transact-sql) – zwraca podciąg napisu,

- [UPPER()](https://docs.microsoft.com/en-us/sql/t-sql/functions/upper-transact-sql) – zwraca napis pisany dużymi literami,

- [LEN()](https://docs.microsoft.com/en-us/sql/t-sql/functions/len-transact-sql) – zwraca liczbę znaków napisu.

Przykład 4

Wyświetlenie nazw projektów pisanych dużymi literami:

SELECT UPPER(nazwa) AS [nazwa projektu]

FROM Projekty;

Operacje na kolumnach

Istnieje szereg operatorów, którymi można przekształcać wartości w kolumnach. Na przykład, operując na kolumnach przechowujących liczby możemy wykonywać na nich operacje arytmetyczne, np. dodawać ze sobą kolumny lub mnożyć przez wartość stałą. Z kolei na kolumnach typu znakowego możemy dokonywać konkatenacji (złączenia).

- [operatory arytmetyczne](https://docs.microsoft.com/en-us/sql/t-sql/language-elements/arithmetic-operators-transact-sql)

- [operatory znakowe](https://docs.microsoft.com/en-us/sql/t-sql/language-elements/string-operators-transact-sql)

Przykład 5

Wyświetlenie nazw projektów oraz stawki za dzień pracy w każdym projekcie:

SELECT nazwa,

stawka \* 8 AS [dniówka]

FROM Projekty;

Przykład 6

W T-SQL klauzula FROM nie jest obowiązkowa. Poniższe zapytanie wykorzystuje operator "+" w trzech kontekstach: wyświetla jutrzejszą datę, wynik przykładowej operacji arytmetycznej oraz ciąg będący wynikiem złączenia (konkatenacji) znaków:

SELECT GETDATE() + 1 AS [co będzie jutro],

2 + 3 AS [suma],

'piękny ' + 'tekst' AS [napis];

Funkcje daty i czasu

- [GETDATE()](https://docs.microsoft.com/en-us/sql/t-sql/functions/getdate-transact-sql) – zwraca aktualną datę,

- [YEAR()](https://docs.microsoft.com/en-us/sql/t-sql/functions/year-transact-sql) – zwraca rok podanej daty,

- [DATEDIFF()](https://docs.microsoft.com/en-us/sql/t-sql/functions/datediff-transact-sql) – oblicza różnicę pomiędzy datami (np. w dniach, minutach, itp.),

- [ISDATE()](https://docs.microsoft.com/en-us/sql/t-sql/functions/isdate-transact-sql) – sprawdza czy podany argument jest datą.

Przykład 7

Zapytanie wyświetla nazwę projektu oraz rok jego rozpoczęcia:

SELECT nazwa,

YEAR(dataRozp) AS [rok rozpoczęcia]

FROM Projekty;

Obsługa wartości pustych – funkcja ISNULL()

W SQL-u kolumny mogą przyjmować specjalną wartość NULL. Może ona oznaczać np. wartość pustą, wartość nieznaną, wartość nie mającą zastosowania w danym kontekście, wartość zastrzeżoną, itp. Na etapie selekcji kolumn możemy podmienić NULL-e na pożądane przez nas wartości.

- funkcja [ISNULL()](https://docs.microsoft.com/en-us/sql/t-sql/functions/isnull-transact-sql) – zamienia wartości puste danego atrybutu (pierwszy parametr) na zadaną wartość (drugi parametr).

Należy pamiętać, że **dowolne operacje arytmetyczne na wartości NULL (np. + lub \*) dają w rezultacie również NULL**:

SELECT 1 + 2 + 3 + NULL

Przykład 8

Wyświetlamy informacje o pracowniku i jego szefie – jeśli *szef* ma wartość NULL to przyjmujemy, że pracownik "sam jest swoim szefem" (wyświetlane jest *id* pracownika):

SELECT id,

nazwisko,

ISNULL(szef, id) AS [kto jest szefem]

FROM Pracownicy;

Porównaj wynik bez funkcji ISNULL:

SELECT id,

nazwisko,

szef AS [kto jest szefem]

FROM Pracownicy;

Typy danych i ich konwersja

Język T-SQL obsługuje wiele [typów danych](https://docs.microsoft.com/en-us/sql/t-sql/data-types/data-types-transact-sql), z których najważniejsze to:

- napisy o stałej i zmiennej długości [CHAR i VARCHAR](https://docs.microsoft.com/en-us/sql/t-sql/data-types/char-and-varchar-transact-sql),

- liczby całkowite i logiczne, np. [INT](https://docs.microsoft.com/en-us/sql/t-sql/data-types/int-bigint-smallint-and-tinyint-transact-sql) i [BIT](https://docs.microsoft.com/en-us/sql/t-sql/data-types/bit-transact-sql),

- liczby rzeczywiste stałoprzecinkowe [DECIMAL/NUMERIC](https://docs.microsoft.com/en-us/sql/t-sql/data-types/decimal-and-numeric-transact-sql) i zmiennoprzecinkowe [FLOAT/REAL](https://docs.microsoft.com/en-us/sql/t-sql/data-types/float-and-real-transact-sql),

- pieniądze i waluty, np. [MONEY](https://docs.microsoft.com/en-us/sql/t-sql/data-types/money-and-smallmoney-transact-sql),

- data i czas, np. [DATETIME](https://docs.microsoft.com/en-us/sql/t-sql/data-types/datetime-transact-sql).

Wykorzystanie części powyższych typów widzieliśmy w skrypcie tworzącym bazę *Projekty*, np. w tabeli *Pracownicy* (dla zachowania czytelności w poniższym kodzie pozostawiono tylko deklaracje typów danych poszczególnych kolumn):

CREATE TABLE Pracownicy

(

id INT,

nazwisko VARCHAR(20),

szef INT,

placa MONEY,

dod\_funkc MONEY,

stanowisko VARCHAR(10),

zatrudniony DATETIME

);

W operacji SELECT dla danej kolumny domyślnie używany jest jej zadeklarowany typ. Jednak jak widzieliśmy wcześniej, podczas tworzenia zapytania możemy manipulować kolumnami, np. łączyć jej, dodawać ze sobą, itp. W związku z tym podczas pracy często pojawia się konieczność jasnego wskazania jaki typ danych powinien zostać zastosowany do danej kolumny. Istnieje możliwość przekonwertowana (rzutowania) jednego typu danych na drugi, który obecnie jest przez nas pożądany. Do konwersji typów danych wykorzystuje się funkcje [CAST() i CONVERT()](https://docs.microsoft.com/en-us/sql/t-sql/functions/cast-and-convert-transact-sql).

Obie funkcje różnią się w niewielkim stopniu. Poza różnicami w składni:

- CAST() jest bardziej związany ze standardem ANSI-SQL,

- CONVERT() pozwala na bardziej wyrafinowane konwersje, w szczególności gdy np. musimy przekonwertować datę na zadany format.

Przykład 9

W poniższym zapytaniu następuje:

- CAST() – konwersja atrybutu *placa* z typu MONEY na typ znakowy VARCHAR,

- CONVERT() – konwersja atrybutu *placa* z typu MONEY na typ znakowy VARCHAR (podobnie jak w przypadku CAST),

- CONVERT() – konwersja typu daty na typ znakowy w predefiniowanym formacie [103 = dd/mm/yyyy](https://docs.microsoft.com/en-us/sql/t-sql/functions/cast-and-convert-transact-sql#date-and-time-styles).

SELECT nazwisko,

CAST(placa AS VARCHAR) + ' zł' AS placa\_1,

CONVERT(VARCHAR, placa) + ' zł' AS placa\_2,

CONVERT(VARCHAR, zatrudniony, 103) AS data\_zatrudnienia

FROM Pracownicy;

Przykład 10

Podobnie jak w Przykładzie 8, wyświetlamy informacje o pracowniku i jego szefie; tym razem, jeśli szef ma wartość NULL chcemy wyświetlić napis "szef szefów"; wymaga to konwersji typu kolumny z INT na CHAR:

SELECT id,

nazwisko,

ISNULL(CAST(szef AS CHAR(11)), 'szef szefów') AS [kto jest szefem]

FROM Pracownicy;

DISTINCT, ORDER BY, TOP

01 – SELECT – podstawy, filtrowanie wierszy

Usuwanie powtórzeń z wyniku – DISTINCT

Przy pomocy polecenia [DISTINCT](https://docs.microsoft.com/en-us/sql/t-sql/queries/select-clause-transact-sql#arguments) można usunąć powtórzenia z wyników zapytania.

Przykład 11

Wyświetlenie wszystkich stanowisk obsadzonych przez pracowników, bez powtórzeń:

SELECT DISTINCT stanowisko

FROM Pracownicy;

Porównaj z zapytaniem:

SELECT stanowisko

FROM Pracownicy;

Przykład 12

Wyświetlenie nazwiska i płacy wszystkich pracowników; informacje są posortowane rosnąco według płacy:

SELECT nazwisko,

placa

FROM Pracownicy

ORDER BY placa;

Przykład 13

Zapytanie zwraca 3 najlepiej zarabiających pracowników:

SELECT TOP 3 nazwisko,

placa

FROM Pracownicy

ORDER BY placa DESC;

**Uwaga**: polecenie TOP zawsze powinno iść w parze z klauzulą ORDER BY, ponieważ wiersze w tabeli nie mają kolejności.

WHERE – filtrowanie wierszy

01 – SELECT – podstawy, filtrowanie wierszy

Po klauzuli [WHERE](https://docs.microsoft.com/en-us/sql/t-sql/queries/where-transact-sql) podajemy zbiór warunków jaki ma zostać spełniony przez wynikowe krotki.

Dostępne operatory:

- [porównania](https://docs.microsoft.com/en-us/sql/t-sql/language-elements/comparison-operators-transact-sql)

|  |
| --- |
|  |
| **Operator** | **Znaczenie** |
|  |  |
| = | równość atrybutów |
| <> lub != | nierówność atrybutów |
| > | atrybut po lewej stronie znaku ma większą wartość od atrybutu po prawej stronie znaku |
| >= | atrybut po lewej stronie znaku ma większą bądź równą wartość od atrybutu po prawej stronie znaku |
| < | atrybut po lewej stronie znaku ma mniejszą wartość od atrybutu po prawej stronie znaku |
| <= | atrybut po lewej stronie znaku ma mniejszą bądź równą wartość od atrybutu po prawej stronie znaku |
| [IS NULL, IS NOT NULL](https://docs.microsoft.com/en-us/sql/t-sql/queries/is-null-transact-sql) | przyrównanie do [wartości pustej](https://docs.microsoft.com/en-us/sql/t-sql/language-elements/null-and-unknown-transact-sql) |

- [logiczne i języka SQL](https://docs.microsoft.com/en-us/sql/t-sql/language-elements/logical-operators-transact-sql)

|  |
| --- |
|  |
| **Operator** | **Znaczenie** |
|  |  |
| OR | operator logiczny *lub* |
| AND | operator logiczny *i* |
| NOT | negacja |
| BETWEEN ... AND ... | sprawdza czy wartość jest pomiędzy podanymi wielkościami |
| [LIKE](https://docs.microsoft.com/en-us/sql/t-sql/language-elements/like-transact-sql) | sprawdza, czy podana wartość jest zgodna z wzorcem, gdzie: <br> % – dowolna liczba znaków (0 lub więcej), \_ – pojedynczy znak, [] – zbiór dozwolonych znaków, [^] – zbiór niedozwolonych znaków |
| IN | sprawdza czy wartość jest w zbiorze |

Przykład 14

Wyświetlenie pracowników pracujących na stanowisku adiunkta:

SELECT \*

FROM Pracownicy

WHERE stanowisko = 'adiunkt';

Przykład 15

Wyświetlenie pracowników, których szefem jest osoba o *id* 1 lub 5 oraz których dodatek funkcyjny jest większy niż 100:

SELECT \*

FROM Pracownicy

WHERE (szef = 1 OR szef = 5)

AND dod\_funkc > 100;

**Uwaga**: wartości liczbowe otoczone apostrofami, np. szef = '1', są poprawnie rozpoznawane przez MSSQL, ale co do zasady **nie powinno** się ich tak zapisywać (inne systemy bazodanowe mogą inaczej obsługiwać takie wartości, a my nie chcemy uczyć się złych nawyków).

Przykład 16

Wyświetlenie pracowników, których nazwisko zaczyna się na literę *W* oraz których płaca znajduje się między 2000 a 3000:

SELECT \*

FROM Pracownicy

WHERE nazwisko LIKE 'W%'

AND placa BETWEEN 2000 AND 3000;

Przykład 17

Wyświetlenie projektów, które rozpoczęły się po 1 stycznia 2015 r. oraz których kierownik ma *id* równe 4 lub 5:

SELECT \*

FROM Projekty

WHERE dataRozp > '2015-01-01'

AND kierownik IN (4, 5);

Obsługa wartości pustych – operator IS NULL

01 – SELECT – podstawy, filtrowanie wierszy

Jak już wiemy, w SQL-u kolumny mogą przyjmować specjalną wartość NULL. Może ona oznaczać np. wartość pustą, wartość nieznaną, wartość nie mającą zastosowania w danym kontekście, wartość zastrzeżoną, itp. Na etapie filtrowania wierszy, wartości NULL mogą być źródłem wielu niezamierzonych błędów, co jest konsekwencją wprowadzonej w SQL-u logiki trójwartościowej. Poniżej przeanalizujemy kilka przypadków.

Przykład 17

**TLDR**: w klauzuli WHERE filtrujemy wiersze poprzez WHERE kolumna IS NULL albo WHERE kolumna IS NOT NULL (nie używamy WHERE kolumna = NULL, bo to zwraca niezamierzone wyniki). Np. lista pracowników bez dodatku funkcyjnego:

SELECT nazwisko,

dod\_funkc

FROM Pracownicy

WHERE dod\_funkc IS NULL;

Spójrzmy na listę pracowników i przysługujące im dodatki funkcyjne:

SELECT nazwisko,

dod\_funkc

FROM Pracownicy;

nazwisko dod\_funkc

------------- ----------

Wachowiak 900,00

Jankowski NULL

Fiołkowska NULL

Mielcarz 400,00

Różycka 200,00

Mikołajski NULL

Wójcicki NULL

Listkiewicz NULL

Wróbel 300,00

Andrzejewicz NULL

Załóżmy, że chcemy wyświetlić tylko tych pracowników, którzy nie mają zdefiniowanego dodatku funkcyjnego. Jeżeli w klauzuli WHERE użyjemy warunku dod\_funkc = NULL…

SELECT nazwisko,

dod\_funkc

FROM Pracownicy

WHERE dod\_funkc = NULL;

…to, dość zaskakująco, w rezultacie nie dostaniemy żadnego pracownika:

nazwisko dod\_funkc

------------- ----------

Moglibyśmy podejrzewać, że skoro żaden pracownik nie spełnia warunku dod\_funkc = NULL, to jeżeli dokonamy negacji tego warunku…

SELECT nazwisko,

dod\_funkc

FROM Pracownicy

WHERE dod\_funkc != NULL;

-- lub

SELECT nazwisko,

dod\_funkc

FROM Pracownicy

WHERE NOT(dod\_funkc = NULL);

…to dostaniemy wszystkich pracowników. Jednak znów w rezultacie nie dostaniemy żadnego pracownika:

nazwisko dod\_funkc

------------- ----------

Dlaczego tak się dzieje i jak rozwiązać ten problem?

**W wyniku zapytania pozostają tylko te wiersze, dla których warunki w WHERE zwrócą łącznie wartość TRUE**; pominięte zostaną wiersze, dla których WHERE zwróci FALSE lub UNKNOWN. **Wartość UNKNOWN** jest to trzecia wartość logiczna, którą **uzyskamy np. porównując NULL z dowolną inną wartością przy pomocy operatora rodzaju =, <, itp.** Poniżej przedstawiono matryce logiczne dla logiki trójwartościowej:

|  |  |
| --- | --- |
|  | |
| **x** | | **NOT x** | |
|  | |  | |
| TRUE | | FALSE | |
| FALSE | | TRUE | |
| UNKNOWN | | UNKNOWN | |
|  |
| **x** | **y** | **x AND y** | **x OR y** |
|  |  |  |  |
| TRUE | TRUE | TRUE | TRUE |
| TRUE | UNKNOWN | UNKNOWN | TRUE |
| TRUE | FALSE | FALSE | TRUE |
| UNKNOWN | TRUE | UNKNOWN | TRUE |
| UNKNOWN | UNKNOWN | UNKNOWN | UNKNOWN |
| UNKNOWN | FALSE | FALSE | UNKNOWN |
| FALSE | TRUE | FALSE | TRUE |
| FALSE | UNKNOWN | FALSE | UNKNOWN |
| FALSE | FALSE | FALSE | FALSE |

*Wskazówka*: powyższe tablice można łatwo zapamiętać przyjmując, że TRUE = 1, FALSE = 0, UNKNOWN = 1/2, NOT x = 1-x, x AND y = min(x, y) i x OR y = max(x, y).

Wracając do przykładu, warunek dod\_funkc = NULL generuje nam dla każdego pracownika wartość logiczną UNKNOWN (z powodu porównania z NULL-em). Nawet jeżeli zaprzeczymy warunek (NOT(dod\_funkc = NULL)), to również uzyskamy wartości logiczne UNKNOWN. Wiemy, że rekordy, dla których WHERE zwróci FALSE lub UNKNOWN, nie są wyświetlane, dlatego w wyniku naszego zapytania nie uzyskujemy żadnych pracowników.

Oczywiście jeżeli jeden z warunków w WHERE zwróci UNKNOWN, to nie oznacza to, że dany rekord definitywnie wypada z wyniku zapytania. W powyższych matrycach logicznych widzimy, że jest przypadek gdy jedną wartością logiczną jest UNKNOWN, a wynik jest TRUE: UNKNOWN OR TRUE = TRUE:

SELECT nazwisko,

placa,

dod\_funkc

FROM Pracownicy

WHERE dod\_funkc = NULL

OR placa > 3000;

nazwisko placa dod\_funkc

------------- -------- ----------

Wachowiak 5500,00 900,00

Jankowski 3500,00 NULL

Fiołkowska 3550,00 NULL

Mielcarz 5000,00 500,00

Różycka 3900,00 300,00

Listkiewicz 3200,00 NULL

Andrzejewicz 3900,00 NULL

Jankowski 3200,00 NULL

Nie dajmy się jednak zwieść, to zapytanie nie działa tak jak prawdopodobnie zamierzał programista: dod\_funkc = NULL – w powyższych wynikach wciąż brakuje Mikołajskiego i Wójcickiego (dla których dod\_funkc wynisi NULL).

Wszystkie powyższe przypadki sankcjonują potrzebę wprowadzenia operatora [IS NULL, IS NOT NULL](https://docs.microsoft.com/en-us/sql/t-sql/queries/is-null-transact-sql), którego używamy w operacjach logicznych jeżeli chcemy przyrównać kolumnę do wartości pustej NULL. Dzięki temu możemy poprawnie sprawdzić, którzy pracownicy mają zdefiniowany dodatek funkcyjny lub go nie mają:

SELECT nazwisko,

dod\_funkc

FROM Pracownicy

WHERE dod\_funkc IS NULL;

nazwisko dod\_funkc

------------- ----------

Jankowski NULL

Fiołkowska NULL

Mikołajski NULL

Wójcicki NULL

Listkiewicz NULL

Andrzejewicz NULL

Jankowski NULL

SELECT nazwisko,

dod\_funkc

FROM Pracownicy

WHERE dod\_funkc IS NOT NULL;

nazwisko dod\_funkc

---------- ----------

Wachowiak 900,00

Mielcarz 500,00

Różycka 300,00

Wróbel 400,00

Kolumna warunkowa – CASE

01 – SELECT – podstawy, filtrowanie wierszy

Czasem w zależności od wartości danej kolumny chcemy wyświetlić inną pożądaną wartość. W takim wypadku pomocne jest użycie wyrażenia [CASE](https://docs.microsoft.com/en-us/sql/t-sql/language-elements/case-transact-sql). Możemy wyróżnić dwie formy użycia tego wyrażenia.

Przykład 18

W poniższym zapytaniu wyświetlamy wszystkich pracowników z informacją czy należy im się dodatek socjalny (jeżeli zarabiają do 3000):

SELECT nazwisko,

placa,

CASE WHEN placa > 3000

THEN 'przyzwoite zarobki'

ELSE 'dać dodatek socjalny'

END AS [jaka płaca?]

FROM Pracownicy;

W tym przykładzie w CASE użyliśmy tylko jedną parę WHEN ... THEN ..., ale jeśli jest taka potrzeba, to można ich użyć więcej razy.

Przykład 19

W poniższym zapytaniu decydujemy jaki ma być dalszy status projektu:

SELECT id,

nazwa,

CASE nazwa

WHEN 'e-learning' THEN 'do kasacji'

WHEN 'neural network' THEN 'kontynuować'

ELSE 'do sprawdzenia'

END AS [co dalej z projektem?]

FROM Projekty;

id nazwa co dalej z projektem?

--- --------------- ---------------------

50 analiza danych do sprawdzenia

10 e-learning do kasacji

40 neural network kontynuować

30 semantic web do sprawdzenia

20 web service do sprawdzenia

W CASE powyższy zapis należy rozumieć jako sprawdzanie warunków nazwa = 'e-learning' oraz nazwa = 'neural network'.

## Złączenie krzyżowe – CROSS JOIN

02 – SELECT – złączenia tabel

CROSS JOIN – iloczyn kartezjański; najprostsze złączenie, które nie posiada żadnego warunku złączenia; wynikiem są wszystkie krotki połączonych tabel, złączone każda z każdą. Uproszczona składnia jest następująca:

SELECT Tabela\_P.\*,

Tabela\_Q.\*

FROM Tabela\_P

CROSS JOIN Tabela\_Q;

Złączenia krzyżowego używamy wtedy, gdy nie porównujemy ze sobą kolumn z łączonych tabel.

Przykład 1

Zacznijmy od klasycznego przykładu z talią kart do gry. Załóżmy, że mamy dwie tabele:

*Kolory*

|  |
| --- |
|  |
| **kolor** |
|  |
| ♥ |
| ♠ |
| ♦ |
| ♣ |

*Cechy*

|  |
| --- |
|  |
| **cecha** |
|  |
| as |
| król |
| dama |
| walet |
| 10 |
| 9 |
| … |
| 3 |
| 2 |

Aby uzyskać wszystkie karty występujące w standardowej talii 52 kart musimy połączyć każdy wiersz z tabeli *Kolory* z każdym wierszem z tabeli *Cechy*. Uzyskujemy to poprzez operację iloczynu kartezjańskiego:

|  |
| --- |
|  |
| **kolor** | **cecha** |
|  |  |
| ♥ | as |
| ♥ | król |
| ♥ | dama |
| ♥ | walet |
| ♥ | 10 |
| ♥ | 9 |
| ♥ | 8 |
| ♥ | 7 |
| ♥ | 6 |
| ♥ | 5 |
| ♥ | 4 |
| ♥ | 3 |
| ♥ | 2 |
| ♠ | as |
| ♠ | król |
| ♠ | dama |
| ♠ | walet |
| … | … |
| ♣ | 4 |
| ♣ | 3 |
| ♣ | 2 |

W jaki sposób moglibyśmy dokonać takiej operacji w SQL-u? Najpierw stwórzmy tabele i zapełnijmy je danymi:

CREATE TABLE Kolory

(

kolor VARCHAR(5)

);

CREATE TABLE Cechy

(

cecha VARCHAR(5)

);

INSERT INTO Kolory VALUES ('kier'), ('pik'), ('karo'), ('trefl');

INSERT INTO Cechy VALUES ('as'), ('król'), ('dama'), ('walet'), ('10'), ('9'),

('8'), ('7'), ('6'), ('5'), ('4'), ('3'), ('2');

Zobaczmy co znajduje się w tych tabelach:

SELECT \* FROM Kolory;

SELECT \* FROM Cechy;

Teraz możemy użyć złączenia CROSS JOIN aby uzyskać iloczyn kartezjański:

SELECT \*

FROM Kolory

CROSS JOIN Cechy;

Zobaczmy, że w przypadku CROSS JOIN gdy zamienimy miejscami tabele, to uzyskamy taki sam wynik, jednak kolejność kolumn będzie inna:

SELECT \*

FROM Cechy

CROSS JOIN Kolory;

Tabele mogą również mieć aliasy:

SELECT \*

FROM Kolory K

CROSS JOIN Cechy C;

Dzięki aliasom możemy odwoływać się do poszczególnych kolmn w łączonych tabelach:

SELECT C.\*,

K.kolor

FROM Kolory K

CROSS JOIN Cechy C;

Możemy również dodać filtrowanie wierszy:

SELECT \*

FROM Kolory K

CROSS JOIN Cechy C

WHERE C.cecha = 'walet';

Istnieje też skrócona, **niezalecana** forma zapisu złączenia krzyżowego. Zamiast stosować frazę CROSS JOIN, można wymienić tabele po przecinku:

SELECT \*

FROM Kolory, Cechy;

**Obecnie odradza się stosowania powyższej notacji przecinkowej**, ponieważ prowadzi to do złych nawyków programistycznych. Więcej na ten temat znajduje się w materiałach z sekcji Dodatkowa lektura.

Złączenie wewnętrzne – INNER JOIN

02 – SELECT – złączenia tabel

INNER JOIN – wynikiem zapytania są tylko te krotki z łączonych tabel, które spełniają warunek powiązania podany po słowie kluczowym ON. Najczęstszym rodzajem złączenia jest tzw. *equi-join*, w którym porównywane są wartości klucza obcego jednej tabeli z wartościami klucza podstawowego drugiej tabeli za pomocą operatora równości =.

SELECT Tabela\_P.\*,

Tabela\_Q.\*

FROM Tabela\_P

INNER JOIN Tabela\_Q

ON warunki\_złączenia;

W T-SQL podczas tworzenia złączenia wewnętrznego można pominąć słowo INNER.

Wizualizacja:

Przykład 2

Chcemy wyświetlić nazwiska pracowników oraz nazwy projektów, którymi kierują.

Najpierw przypomnijmy sobie jak wyglądają tabele *Pracownicy* oraz *Projekty*.

SELECT \* FROM Pracownicy;

SELECT \* FROM Projekty;

Aby zrozumieć logikę złączenia wewnętrznego, wróćmy na chwilę do złaczenia krzyżowego. Przy pomocy CROSS JOIN możemy wygenerować wszystkie pary pracownik-projekt:

SELECT P.nazwisko,

P.id,

R.kierownik,

R.nazwa

FROM Pracownicy P

CROSS JOIN Projekty R;

Zobaczmy, że w wynikach znajdują się interesujące nas wiersze, tzn. te, w który Pracownicy.id = Projekty.kierownik:

nazwisko id kierownik nazwa

------------- --- ---------- ---------------

Wachowiak 1 5 e-learning

Jankowski 2 5 e-learning

Fiołkowska 3 5 e-learning

Mielcarz 4 5 e-learning

Różycka 5 5 e-learning <---

Mikołajski 6 5 e-learning

Wójcicki 7 5 e-learning

Listkiewicz 8 5 e-learning

Wróbel 9 5 e-learning

Andrzejewicz 10 5 e-learning

Wachowiak 1 4 web service

Jankowski 2 4 web service

Fiołkowska 3 4 web service

Mielcarz 4 4 web service <---

Różycka 5 4 web service

Mikołajski 6 4 web service

Wójcicki 7 4 web service

Listkiewicz 8 4 web service

Wróbel 9 4 web service

Andrzejewicz 10 4 web service

Wachowiak 1 4 semantic web

Jankowski 2 4 semantic web

Fiołkowska 3 4 semantic web

Mielcarz 4 4 semantic web <---

Różycka 5 4 semantic web

Mikołajski 6 4 semantic web

Wójcicki 7 4 semantic web

Listkiewicz 8 4 semantic web

Wróbel 9 4 semantic web

Andrzejewicz 10 4 semantic web

Wachowiak 1 1 neural network <---

Jankowski 2 1 neural network

Fiołkowska 3 1 neural network

Mielcarz 4 1 neural network

Różycka 5 1 neural network

Mikołajski 6 1 neural network

Wójcicki 7 1 neural network

Listkiewicz 8 1 neural network

Wróbel 9 1 neural network

Andrzejewicz 10 1 neural network

Wachowiak 1 10 analiza danych

Jankowski 2 10 analiza danych

Fiołkowska 3 10 analiza danych

Mielcarz 4 10 analiza danych

Różycka 5 10 analiza danych

Mikołajski 6 10 analiza danych

Wójcicki 7 10 analiza danych

Listkiewicz 8 10 analiza danych

Wróbel 9 10 analiza danych

Andrzejewicz 10 10 analiza danych <---

Jankowski 11 10 analiza danych

W pierwszej chwili może pojawić się myśl, aby do naszego zapytania z CROSS JOIN dołączyć warunek WHERE P.id = R.kierownik:

SELECT P.nazwisko,

P.id,

R.kierownik,

R.nazwa

FROM Pracownicy P

CROSS JOIN Projekty R

WHERE P.id = R.kierownik;

Takie zapytanie by zadziałało i nawet dało poprawny wynik, ale wiemy, że **co do zasady, złączenia krzyżowego używamy tylko wtedy, gdy nie porównujemy ze sobą kolumn z łączonych tabel**. Natomiast **gdy porównujemy ze sobą kolumny z łączonych tabel, to w takim przypadku używamy złączenia wewnętrznego**.

Używając INNER JOIN możemy połączyć odpowiednie wiersze z tabeli *Projekty* z odpowiednimi wierszami z tabeli *Pracownicy*. Projekt połączymy z tym pracownikiem, który jest kierownikiem tego projektu – warunek Pracownicy.id = Projekty.kierownik.

SELECT P.nazwisko,

P.id,

R.kierownik,

R.nazwa

FROM Pracownicy P

INNER JOIN Projekty R

ON P.id = R.kierownik;

W wyniku uzyskaliśmy tylko tych pracowników, którzy kierują jakimś projektem i tylko te projekty, które mają kierownika.

Za powyższym przykładem idzie też pewna **intuicja** mówiąca czym jest złączenie wewnętrzne – w pewnym sensie jest to po prostu złączenie krzyżowe z filtrowaniem wierszy.

Połączenie tabeli z samą sobą – SELF JOIN

02 – SELECT – złączenia tabel

Nie jest to osobny rodzaj złączenia, ale zyskał osobną nazwę – SELF JOIN – aby podkreślić, że wykonywane jest złączenie tabeli samej ze sobą. Złączenia takie są wykorzystywane, gdy konieczne jest porównanie dwóch wierszy z tej samej tabeli.

Przykład 3

Chcemy znaleźć pary projektów, które są kierowane przez tę samą osobę.

Nie ma możliwości, aby w pojedynczym poleceniu SELECT odwołać się jednocześnie do dwóch wierszy tej samej tabeli. Stworzymy zatem "kopię" tabeli *Projekty* i zaczniemy od połączenia wszystkich wierszy z tabeli *Projekty* (P1) z wszystkimi wierszami jej kopii (P2):

SELECT P1.id,

P1.kierownik,

P1.nazwa,

P2.id,

P2.kierownik,

P2.nazwa

FROM Projekty P1

CROSS JOIN Projekty P2;

Mamy znaleźć pary projektów, więc logicznie rzecz biorąc, w takiej parze powinny znaleźć się dwa *różne* projekty. Nasze złączenie wewnętrzne możemy rozpocząć od znalezienia par różnych projektów:

SELECT P1.id,

P1.kierownik,

P1.nazwa,

P2.id,

P2.kierownik,

P2.nazwa

FROM Projekty P1

JOIN Projekty P2

ON P1.id != P2.id;

W wyniku mamy spermutowane pary, więc przyjmijmy, że zostawimy te pary, które mają większe id pierwszego projektu:

SELECT P1.id,

P1.kierownik,

P1.nazwa,

P2.id,

P2.kierownik,

P2.nazwa

FROM Projekty P1

JOIN Projekty P2

ON P1.id > P2.id;

Mamy wszystkie pary projektów, więc pozostawiamy tylko te, które mają tego samego kierownika:

SELECT P1.id,

P1.kierownik,

P1.nazwa,

P2.id,

P2.kierownik,

P2.nazwa

FROM Projekty P1

JOIN Projekty P2

ON P1.id > P2.id

AND P1.kierownik = P2.kierownik;

**Uwaga**: w treści przykładów zaczynamy często wychodząc od CROSS JOIN. Oczywiście podczas nauki SQL-a nie ma w tym nic złego, jednak w przypadku większych tabel albo pracy w realnym środowisku produkcyjnym musimy być świadomi, że tego typu operacje mogą być bardzo zasobożerne.

Złączenie zewnętrzne – OUTER JOIN

02 – SELECT – złączenia tabel

Złączenie wewnętrzne zwracało w wyniku jedynie te krotki, które spełniały warunek złączenia – np. tylko tych pracowników, którzy kierują jakimś projektem, jak w przykładzie 2.

Wynik złączenia zewnętrznego jest szerszy – zawiera wynik INNER JOIN oraz dodatkowo te wiersze, które nie zostały połączone – z lewej tabeli, z prawej tabeli, lub z obu tabel.

|  |
| --- |
|  |
| **Złączenie zewnętrzne** | **Zapis w języku SQL** | **Znaczenie** |
|  |  |  |
| lewostronne | LEFT OUTER JOIN | **każdą** krotkę z lewej tabeli uzupełniamy tymi krotkami z tabeli prawej, które odpowiadają warunkowi złączenia; jeżeli takiej krotki nie ma, to krotka z lewej tabeli jest uzupełniana wartościami NULL |
| prawostronne | RIGHT OUTER JOIN | **każdą** krotkę z prawej tabeli uzupełniamy tymi krotkami z tabeli lewej, które odpowiadają warunkowi złączenia; jeżeli takiej krotki nie ma, to krotka z prawej tabeli jest uzupełniana wartościami NULL |
| pełne | FULL OUTER JOIN | suma obu powyższych ([prawy do lewego, lewy do prawego](https://www.youtube.com/watch?v=cKF2GjRAUtE&t=1m06s)) |

Złączenie A LEFT OUTER JOIN B jest równoważne złączeniu B RIGHT OUTER JOIN A.

Wizualizacja:

Złączenie zewnętrzne lewostronne - LEFT OUTER JOIN

Złączenie zewnętrzne prawostronne - RIGHT OUTER JOIN

Złączenie zewnętrzne pełne - FULL OUTER JOIN

Przykład 4

Poniższe zapytanie zwraca nazwiska wszystkich pracowników oraz nazwę projektu, którym pracownik kieruje (jeśli taki projekt istnieje) – jeśli pracownik nie kieruje żadnym projektem, to podawana jest wartość NULL:

SELECT P.nazwisko,

P.id,

R.kierownik,

R.nazwa

FROM Pracownicy P

LEFT OUTER JOIN Projekty R

ON P.id = R.kierownik;

Porównaj wynik zapytania z przykładem dla INNER JOIN:

SELECT P.nazwisko,

P.id,

R.kierownik,

R.nazwa

FROM Pracownicy P

INNER JOIN Projekty R

ON P.id = R.kierownik;

Przykład 5

Chcemy znaleźć stanowiska, na których nie zatrudniono pracowników. Zauważ, że oczywiście zwykła negacja warunku nie da nam tego, czego szukamy (zastanów się dlaczego!):

SELECT DISTINCT S.nazwa

FROM Pracownicy P INNER JOIN Stanowiska S

ON P.stanowisko != S.nazwa

Jeden ze sposobów na rozwiązanie tego typu zadań polega na wykorzystaniu złączenia zewnętrznego.

Zacznijmy od połączenia stanowisk (zewnętrznie lewostronnie) z pracownikami. Zobaczymy wtedy kto gdzie pracuje, a dodatkowo będziemy mieli listę *wszystkich* stanowisk (również tych, które nie połączyły się z żadnym pracownikiem):

SELECT S.nazwa,

P.id,

P.nazwisko

FROM Stanowiska S

LEFT OUTER JOIN Pracownicy P

ON S.nazwa = P.stanowisko;

Zatem, aby znaleźć stanowiska bez pracowników wystarczy pozostawić tylko te wiersze, które posiadają wartość NULL w atrybutach tabeli *Pracownicy*. Ponadto możemy zostawić same nazwy stanowisk. Ostatecznie:

SELECT S.nazwa

FROM Stanowiska S

LEFT OUTER JOIN Pracownicy P

ON S.nazwa = P.stanowisko

WHERE P.id IS NULL;

*Uwaga*: w przypadku LEFT OUTER JOIN, w klauzuli WHERE ... IS NULL wstawiamy tę kolumnę z tabeli po prawej stronie złączenia zewnętrznego, która wygeneruje wartości NULL dla wierszy nie spełniających warunku złączenia, np. klucz główny z tej tabeli. W przykładzie tabelą po prawej stronie są Pracownicy P, a kluczem głównym jest tutaj P.id, więc wstawiamy WHERE P.id IS NULL.

Przykład 6

Aby dobrze zrozumieć omawiany problem, rozbudujmy zapytanie z [Przykładu 5](https://its.wmi.amu.edu.pl:8443/repository/courses/1/673/sco_5.html?var=0.08412962612497554&timestamp=1618075579265#przyk%C5%82ad-5). Zapytanie zwracało nazwiska wszystkich pracowników oraz nazwę projektu, którym pracownik kieruje (jeśli taki projekt istnieje; jeżeli pracownik nie kieruje żadnym projektem, to podawana jest wartość NULL):

SELECT P.nazwisko,

P.id,

R.kierownik,

R.nazwa

FROM Pracownicy P

LEFT OUTER JOIN Projekty R

ON P.id = R.kierownik;

W powyższym zapytaniu połączyliśmy całą tabelę *Pracownicy* z całą tabelą *Projekty*.

Teraz poprzez klauzulę WHERE możemy sprawdzić kto kieruje projektem *neural network*:

SELECT P.nazwisko,

P.id,

R.kierownik,

R.nazwa

FROM Pracownicy P

LEFT OUTER JOIN Projekty R

ON P.id = R.kierownik

WHERE R.nazwa = 'neural network';

Powyższe zapytanie zwróciło jeden rekord.

A gdybyśmy chcieli sprawdzić kto kieruje projektem *neural network*, a przy pozostałych pracownikach podać wartość NULL? Musielibyśmy połaczyć całą tabelę *Pracownicy* z tabelą *Projekty* zredukowaną do jednego rekordu (czyli zrobić coś na kształt filtrowania tabeli tuż **przed** operacją złączenia zewnętrznego):

SELECT P.nazwisko,

P.id,

R.kierownik,

R.nazwa

FROM Pracownicy p

LEFT OUTER JOIN Projekty R

ON P.id = R.kierownik

AND R.nazwa = 'neural network';

Jak widać, ten sam warunek R.nazwa = &apos;neural network&apos; w ON i WHERE daje różne wyniki. W przypadku złączenia zewnętrznego jest to spowodowane tym, że JOIN jest wykonywany przed WHERE.

Powyższa konstrukcja pozwala nam na stworzenie zapytania zwracającego nazwiska pracowników, którzy nie kierują projektem *neural network*.

SELECT P.nazwisko

FROM Pracownicy P

LEFT OUTER JOIN Projekty R

ON P.id = R.kierownik

AND R.nazwa = 'neural network'

WHERE R.id IS NULL;

Przykład 7

Chcemy znaleźć projekt o najniższej stawce godzinowej (nie używamy TOP ani MIN).

Możemy zrobić sprytne złączenie zewnętrzne tabeli *Projekty* samej ze sobą, tj. łaczyć w pary te projekty, z których pierwszy ma większą stawkę niż drugi:

SELECT \*

FROM Projekty P1

LEFT OUTER JOIN Projekty P2

ON P1.stawka > P2.stawka;

W jednym przypadku nie uda się znaleźć pary dla projektu – nie będzie on miał projektu o niższej stawce niż on sam, a więc będzie to projekt o najniższej stawce. Ponieważ jest to złączenie zewnętrzne, projekt pojawi się w wyniku z dołączonymi wartościami NULL. Teraz możemy pozostawić wyniki z pierwszej tabeli *Projekty* i odfiltrować interesujący nas wiersz:

SELECT P1.\*

FROM Projekty P1

LEFT OUTER JOIN Projekty P2

ON P1.stawka > P2.stawka

WHERE P2.id IS NULL;

Podzapytania nieskorelowane (niepowiązane, niezależne)

03 – SELECT – podzapytania w klauzuli WHERE

Przykład 1

Poniższe zapytanie zwraca informacje o projektach, których stawka jest wyższa niż stawka dla projektu *e-learning*. Podzapytanie zostało tutaj użyte do tego, aby pobrać informację o stawce dla projektu *e-learning*; zwraca pojedynczą wartość.

SELECT \*

FROM Projekty

WHERE stawka > (SELECT stawka

FROM Projekty

WHERE nazwa = 'e-learning');

Przykład 2

Zapytanie zwraca informacje o pracownikach pracujących na stanowiskach, na których minimalna płaca jest większa od 2500.

SELECT \*

FROM Pracownicy

WHERE stanowisko IN (SELECT nazwa

FROM Stanowiska

WHERE placa\_min > 2500);

Podzapytanie może zwrócić zbiór wartości (może być wiele takich stanowisk), więc należy użyć operatora IN (a nie =).

Przykład 3

W Przykładzie 1 szukaliśmy projektu, który ma większą stawkę niż projekt *e-learning*. Podzapytanie zwróciło nam jedną wartość. Co w przypadku gdy podzapytanie zwróci więcej wartości? Szukając projektu, który ma większą stawkę niż stawki w *e-learning* oraz *semantic web*, poniższe zapytanie zwróci błąd:

SELECT \*

FROM Projekty

WHERE stawka > (SELECT stawka

FROM Projekty

WHERE nazwa IN ('e-learning', 'semantic web'));

Msg 512, Level 16, State 1, Line 1

Subquery returned more than 1 value. This is not permitted when the subquery

follows =, !=, <, <= , >, >= or when the subquery is used as an expression.

W takim przypadku, tuż przed podzapytaniem musimy dodać słowo ALL:

SELECT \*

FROM Projekty

WHERE stawka > ALL (SELECT stawka

FROM Projekty

WHERE nazwa IN ('e-learning', 'semantic web'));

Przykład 4

Wyświetl nazwy stanowisk nie obsadzonych przez żadnego pracownika:

SELECT nazwa

FROM Stanowiska

WHERE nazwa != ALL (SELECT stanowisko

FROM Pracownicy);

Przykład 5

Podaj nazwiska pracowników nie będących adiunktami, którzy zarabiają więcej niż adiunkt (jakikolwiek):

SELECT nazwisko,

stanowisko

FROM Pracownicy

WHERE stanowisko != 'adiunkt'

AND placa > SOME (SELECT placa

FROM Pracownicy

WHERE stanowisko = 'adiunkt');

Podzapytania skorelowane (powiązane)

03 – SELECT – podzapytania w klauzuli WHERE

Wynik zwracany przez podzapytanie skorelowane jest zależny od aktualnie analizowanego wiersza w zapytaniu nadrzędnym.

Przykład 6

Zapytanie zwraca informacje o tych pracownikach, których płaca jest większa niż to przewidują widełki dla ich stanowiska.

SELECT \*

FROM Pracownicy P

WHERE P.placa > (SELECT S.placa\_max

FROM Stanowiska S

WHERE S.nazwa = P.stanowisko);

Podzapytanie zostało użyte do pobrania informacji o górnej granicy płacy (placa\_max) dla tego stanowiska, na którym pracuje pracownik analizowany w nadzapytaniu – warunek S.nazwa = P.stanowisko (podzapytanie będzie zwracało różne wartości, w zależności od stanowiska pracownika; w pewnym sensie można to rozumieć jako złączenie wewnętrzne). W zapytaniu nadrzędnym nie możemy odnieść się do tabeli Stanowiska S, ponieważ nie znajduje się ona w klauzuli FROM zapytania nadrzędnego (to nie jest złączenie).

To, że w zapytaniu nadrzędnym mamy jedną tabelę Pracownicy P, a w podzapytaniu drugą tabelę Stanowiska S oraz warunek odnoszący się do tych dwóch tabel S.nazwa = P.stanowisko oznacza, że mamy do czynienia z **podzapytaniem skorelowanym**.

Aby lepiej zobrazować w jaki sposób procesowane są kolejne wiersze z zapytania nadrzędnego, poniżej zaprezentowane jest zapytanie pomocniczne w postaci złączenia tabel (to nie jest podzapytanie skorelowane):

SELECT P.nazwisko,

P.stanowisko,

P.placa,

S.nazwa,

S.placa\_max,

CASE WHEN P.placa > S.placa\_max

THEN 1

ELSE 0

END AS 'placa > placa\_max ?'

FROM Pracownicy P

JOIN Stanowiska S

ON P.stanowisko = S.nazwa;

W wynikach widać, że faktycznie tylko jeden pracownik spełnia warunek zarabiania więcej niż przewidziana płaca maksymalna dla jego stanowiska.

Ok, ale jak dokładnie działa podzapytanie skorelowane? Przypomnijmy jeszcze raz:

*Wynik zwracany przez podzapytanie skorelowane jest zależny od aktualnie analizowanego wiersza w zapytaniu nadrzędnym.*

Spójrzmy na wybrane 3 iteracje w nadzapytaniu, tj. co się dzieje podczas analizy wierszy nr 1, 2 i 7 z tabeli *Pracownicy* (reszta wierszy analogicznie); dla zachowania czytelności liczba kolumn została zredukowana tylko do tych istotnych.

Wiersz nr 1:

Podzapytanie skorelowane: Stanowiska S

nazwa placa\_min placa\_max

Nadzapytanie: Pracownicy P ----------- ---------- ----------

adiunkt 2000,00 3000,00

id nazwisko placa stanowisko doktorant 900,00 1300,00

---- ------------- -------- ---------- dziekan 2700,00 4800,00

┌──────────────────────────────────────┐ P.stanowisko = S.nazwa ┌────────────────────────────────┐

│ 1 Wachowiak 4500,00 profesor │-------------------------------------│ profesor 3000,00 5000,00 │

└──────────────────────────────────────┘ P.placa > S.placa\_max ? NIE └────────────────────────────────┘

2 Jankowski 2500,00 adiunkt sekretarka 1500,00 2500,00

3 Fiołkowska 2550,00 adiunkt techniczny 1500,00 2500,00

4 Mielcarz 4000,00 profesor

5 Różycka 2800,00 profesor

6 Mikołajski 1000,00 doktorant

7 Wójcicki 1350,00 doktorant

8 Listkiewicz 2200,00 sekretarka

9 Wróbel 1900,00 techniczny

10 Andrzejewicz 2900,00 adiunkt

Wiersz nr 2:

Nadzapytanie: Pracownicy P

Podzapytanie skorelowane: Stanowiska S

id nazwisko placa stanowisko

---- ------------- -------- ---------- nazwa placa\_min placa\_max

1 Wachowiak 4500,00 profesor ----------- ---------- ----------

┌──────────────────────────────────────┐ P.stanowisko = S.nazwa ┌────────────────────────────────┐

│ 2 Jankowski 2500,00 adiunkt │-------------------------------------│ adiunkt 2000,00 3000,00 │

└──────────────────────────────────────┘ P.placa > S.placa\_max ? NIE └────────────────────────────────┘

3 Fiołkowska 2550,00 adiunkt doktorant 900,00 1300,00

4 Mielcarz 4000,00 profesor dziekan 2700,00 4800,00

5 Różycka 2800,00 profesor profesor 3000,00 5000,00

6 Mikołajski 1000,00 doktorant sekretarka 1500,00 2500,00

7 Wójcicki 1350,00 doktorant techniczny 1500,00 2500,00

8 Listkiewicz 2200,00 sekretarka

9 Wróbel 1900,00 techniczny

10 Andrzejewicz 2900,00 adiunkt

Wiersz nr 7:

Nadzapytanie: Pracownicy P

id nazwisko placa stanowisko

---- ------------- -------- ----------

1 Wachowiak 4500,00 profesor

2 Jankowski 2500,00 adiunkt Podzapytanie skorelowane: Stanowiska S

3 Fiołkowska 2550,00 adiunkt

4 Mielcarz 4000,00 profesor nazwa placa\_min placa\_max

5 Różycka 2800,00 profesor ----------- ---------- ----------

6 Mikołajski 1000,00 doktorant adiunkt 2000,00 3000,00

┌──────────────────────────────────────┐ P.stanowisko = S.nazwa ┌────────────────────────────────┐

│ 7 Wójcicki 1350,00 doktorant │-------------------------------------│ doktorant 900,00 1300,00 │

└──────────────────────────────────────┘ P.placa > S.placa\_max ? TAK └────────────────────────────────┘

8 Listkiewicz 2200,00 sekretarka dziekan 2700,00 4800,00

9 Wróbel 1900,00 techniczny profesor 3000,00 5000,00

10 Andrzejewicz 2900,00 adiunkt sekretarka 1500,00 2500,00

techniczny 1500,00 2500,00

Przykład 7

W Przykładzie 6, wykorzystując podzapytanie skorelowane, mieliśmy zapytanie, które zwraca informacje o tych pracownikach, których płaca jest większa niż to przewidują widełki dla ich stanowiska. Tę samą odpowiedź możemy uzyskać poprzez złączenie tabel:

SELECT P.\*

FROM Pracownicy P

INNER JOIN Stanowiska S

ON P.stanowisko = S.nazwa

WHERE P.placa > S.placa\_max;

## EXISTS, NOT EXISTS

# 03 – SELECT – podzapytania w klauzuli WHERE

Operator EXISTS jest operatorem **jednoargumentowym** – testuje istnienie wartości (sama wartość nie ma znaczenia). EXISTS zwraca wartość TRUE jeżeli argument zwracany jako wartość podzapytania jest niepusty. Jeśli podzapytanie zwraca wartość pustą (NULL) wówczas EXISTS zwraca FALSE.

EXISTS często idzie w parze z podzapytaniem skorelowanym.

#### Przykład 8

Podaj nazwiska profesorów, którzy nie posiadają pod swoją opieką doktorantów:

SELECT P1.nazwisko

FROM Pracownicy P1

WHERE P1.stanowisko = 'profesor'

AND NOT EXISTS (SELECT \*

FROM Pracownicy P2

WHERE P2.stanowisko = 'doktorant'

AND P2.szef = P1.id);

Zapytania pomocnicze:

SELECT id, nazwisko FROM Pracownicy WHERE stanowisko = 'profesor';

SELECT id, nazwisko, szef FROM Pracownicy WHERE stanowisko = 'doktorant';

#### Przykład 9

Przykład dotyczy bazy Aviation, w której mamy trzy proste tabele, tj. listę pilotów, samoloty w hangarze oraz informacje o tym które samoloty potrafi pilotować dany pilot.

Pilots:

|  |
| --- |
|  |
| **pilot** |
|  |
| Celko |
| Higgins |
| Jones |
| Smith |
| Wilson |

Hangar:

|  |
| --- |
|  |
| **plane** |
|  |
| B-1 Bomber |
| B-52 Bomber |
| F-14 Fighter |

Skills:

|  |
| --- |
|  |
| **pilot** | **plane** |
|  |  |
| Celko | Piper Cub |
| Higgins | B-52 Bomber |
| Higgins | F-14 Fighter |
| Higgins | Piper Cub |
| Jones | B-52 Bomber |
| Jones | F-14 Fighter |
| Smith | B-1 Bomber |
| Smith | B-52 Bomber |
| Smith | F-14 Fighter |
| Wilson | B-1 Bomber |
| Wilson | B-52 Bomber |
| Wilson | F-14 Fighter |
| Wilson | F-17 Fighter |

Kod tworzący bazę

--USE master;

--DROP DATABASE Aviation;

--GO

--CREATE DATABASE Aviation;

--GO

--USE Aviation;

--GO

------------ USUŃ TABELE ------------

DROP TABLE IF EXISTS Skills;

DROP TABLE IF EXISTS Pilots;

DROP TABLE IF EXISTS Hangar;

------------ CREATE - UTWÓRZ TABELE I POWIĄZANIA ------------

CREATE TABLE Pilots

(

pilot VARCHAR(20)

);

CREATE TABLE Hangar

(

plane VARCHAR(20)

);

CREATE TABLE Skills

(

pilot VARCHAR(20),

plane VARCHAR(20)

);

GO

------------ INSERT - WSTAW DANE ------------

INSERT INTO Pilots VALUES

('Celko' ),

('Higgins'),

('Jones' ),

('Smith' ),

('Wilson' );

INSERT INTO Hangar VALUES

('B-1 Bomber' ),

('B-52 Bomber' ),

('F-14 Fighter');

GO

INSERT INTO Skills VALUES

('Celko' , 'Piper Cub' ),

('Higgins' , 'B-52 Bomber' ),

('Higgins' , 'F-14 Fighter'),

('Higgins' , 'Piper Cub' ),

('Jones' , 'B-52 Bomber' ),

('Jones' , 'F-14 Fighter'),

('Smith' , 'B-1 Bomber' ),

('Smith' , 'B-52 Bomber' ),

('Smith' , 'F-14 Fighter'),

('Wilson' , 'B-1 Bomber' ),

('Wilson' , 'B-52 Bomber' ),

('Wilson' , 'F-14 Fighter'),

('Wilson' , 'F-17 Fighter');

GO

------------ SELECT ------------

SELECT \* FROM Pilots;

SELECT \* FROM Hangar;

SELECT \* FROM Skills;

Spróbujemy odpowiedzieć na następujące pytanie:

- znajdź pilota, który umie pilotować wszystkimi samolotami z hangaru.

Możemy to zapytanie zapisać alternatywnie jako:

- znajdź pilota, dla którego nie istnieje samolot w hangarze, którego ten pilot nie potrafi pilotować.

Stosując dwukrotnie NOT EXISTS możemy znaleźć odpowiedź na powyższy problem:

SELECT pilot

FROM Pilots AS P

WHERE NOT EXISTS (SELECT \*

FROM Hangar H

WHERE NOT EXISTS (SELECT \*

FROM Skills AS S

WHERE S.pilot = P.pilot

AND S.plane = H.plane));

pilot

-------

Smith

Wilson

#### Przykład 10

Zmodyfikujmy tabelę Skills z bazy Aviations:

DELETE FROM Skills

WHERE plane = 'B-52 Bomber';

Teraz tabele wyglądają następująco:

Pilots:

|  |
| --- |
|  |
| **pilot** |
|  |
| Celko |
| Higgins |
| Jones |
| Smith |
| Wilson |

Hangar:

|  |
| --- |
|  |
| **plane** |
|  |
| B-1 Bomber |
| B-52 Bomber |
| F-14 Fighter |

Skills:

|  |
| --- |
|  |
| **pilot** | **plane** |
|  |  |
| Celko | Piper Cub |
| Higgins | F-14 Fighter |
| Higgins | Piper Cub |
| Jones | F-14 Fighter |
| Smith | B-1 Bomber |
| Smith | F-14 Fighter |
| Wilson | B-1 Bomber |
| Wilson | F-14 Fighter |
| Wilson | F-17 Fighter |

W efekcie mamy teraz w hangarze samolot B-52 Bomber, którego nikt nie potrafi pilotować. Skonstruujmy dwa zapytania znajdujące taki samolot, którego nikt nie potrafi pilotować. Wersja z NOT EXISTS daje poprawny wynik:

SELECT plane

FROM Hangar H

WHERE NOT EXISTS (SELECT \*

FROM Skills S

WHERE S.plane = H.plane);

plane

------------

B-52 Bomber

Wersja z NOT IN również daje poprawny wynik:

SELECT plane

FROM Hangar

WHERE plane NOT IN (SELECT plane

FROM Skills);

plane

------------

B-52 Bomber

Wprowadźmy teraz drobną, z pozoru nic nie znaczącą zmianę w tabeli Skills:

INSERT INTO Skills VALUES ('Wilson', NULL);

Skills:

|  |
| --- |
|  |
| **pilot** | **plane** |
|  |  |
| Celko | Piper Cub |
| Higgins | F-14 Fighter |
| Higgins | Piper Cub |
| Jones | F-14 Fighter |
| Smith | B-1 Bomber |
| Smith | F-14 Fighter |
| Wilson | B-1 Bomber |
| Wilson | F-14 Fighter |
| Wilson | F-17 Fighter |
| Wilson | NULL |

Dodaliśmy informację, że Wilson potrafi pilotować jakiś niezidentyfikowany samolot. Odpowiadając cały czas na to samo pytanie, wersja zapytania z NOT EXISTS wciąż daje poprawny wynik:

SELECT plane

FROM Hangar H

WHERE NOT EXISTS (SELECT \*

FROM Skills S

WHERE S.plane = H.plane);

plane

------------

B-52 Bomber

Natomiast zapytanie z NOT IN zwróci tym razem pusty wynik:

SELECT plane

FROM Hangar

WHERE plane NOT IN (SELECT plane

FROM Skills);

plane

------

Dlaczego tak się dzieje? Powtórzmy jeszcze raz jak zachowuje się EXISTS:

*EXISTS zwraca wartość TRUE jeżeli argument zwracany jako wartość podzapytania jest niepusty. Jeśli podzapytanie zwraca wartość pustą (NULL) wówczas EXISTS zwraca FALSE.*

W przypadku NOT IN zapytanie:

SELECT plane

FROM Hangar

WHERE plane NOT IN (SELECT plane

FROM Skills);

możemy zapisać równoważnie:

SELECT plane

FROM Hangar

WHERE plane != (SELECT plane FROM Skills WHERE plane = 'Piper Cub')

AND plane != (SELECT plane FROM Skills WHERE plane = 'F-14 Fighter')

AND plane != (SELECT plane FROM Skills WHERE plane = 'Piper Cub')

AND plane != (SELECT plane FROM Skills WHERE plane = 'F-14 Fighter')

AND plane != (SELECT plane FROM Skills WHERE plane = 'B-1 Bomber')

AND plane != (SELECT plane FROM Skills WHERE plane = 'F-14 Fighter')

AND plane != (SELECT plane FROM Skills WHERE plane = 'B-1 Bomber')

AND plane != (SELECT plane FROM Skills WHERE plane = 'F-14 Fighter')

AND plane != (SELECT plane FROM Skills WHERE plane = 'F-17 Fighter')

AND plane != (SELECT plane FROM Skills WHERE plane = NULL);

Z pierwszych zajęć wiemy, że w klauzuli WHERE jeśli przyrównamy kolumnę do wartości NULL, to wartość logiczna takie wyrażenia będzie UNKNOWN. Tutaj widzimy, że w ostatnim wierszu mamy plane = NULL, przez co dla każdego wiersza cała klauzula WHERE (wszystkie warunki połączone AND) dadzą albo FALSE albo UNKNOWN, przez co każdy wiersz zostanie pominięty.

Jakie z tego płyną ogólne wnioski?

1. jeżeli kolumna może przyjmować wartości NULL, to NOT EXISTS i NOT IN mogą dać różne wyniki,
2. jeżeli kolumna nie może przyjmować wartości NULL, to NOT EXISTS i NOT IN dadzą identyczne wyniki.

Stąd też w praktyce bazodanowej ważne jest by jasno w definicji kolumny wskazać czy może ona przyjmować wartości NULL.

## Funkcje agregujące

# 04 – SELECT – funkcje agregujące, operacje na zbiorach, podzapytania w klauzuli FROM i SELECT

Działają na zbiorze krotek i zwracają pojedynczą, zagregowaną wartość. Podstawowe funkcje to:

- [COUNT()](https://docs.microsoft.com/en-us/sql/t-sql/functions/count-transact-sql) – zliczanie elementów w zbiorze,

- [AVG()](https://docs.microsoft.com/en-us/sql/t-sql/functions/avg-transact-sql) – średnia wartości elementów w zbiorze,

- [MIN()](https://docs.microsoft.com/en-us/sql/t-sql/functions/min-transact-sql) – wartość minimalna w zbiorze,

- [MAX()](https://docs.microsoft.com/en-us/sql/t-sql/functions/max-transact-sql) – wartość maksymalna w zbiorze,

- [SUM()](https://docs.microsoft.com/en-us/sql/t-sql/functions/sum-transact-sql) – suma wartości elementów w zbiorze.

Więcej informacji znajduje się w [dokumentacji funkcji agregujących](https://docs.microsoft.com/en-us/sql/t-sql/functions/aggregate-functions-transact-sql).

#### Przykład 1

Zapytanie zwraca liczbę wszystkich pracowników i ich średnie zarobki

SELECT COUNT(\*) 'liczba pracowników',

AVG(placa) 'średnia płaca'

FROM Pracownicy;

**Uwaga**: niedopuszczalne jest odwoływanie się w jednym zapytaniu do wartości zagregowanych i nie zagregowanych. Przykładowo, dla poniższego zapytania:

SELECT COUNT(\*) 'liczba pracowników',

AVG(placa) 'średnia płaca',

nazwisko

FROM Pracownicy;

zwrócony zostanie błąd:

Msg 8120, Level 16, State 1, Line 3

Column 'Pracownicy.nazwisko' is invalid in the select list because

it is not contained in either an aggregate function or the GROUP BY clause.

#### Przykład 2

Chcemy dowiedzieć się ile różnych typów stanowisk zajmują pracownicy. Poniższe zapytanie zliczy nam stanowiska, zliczając również powtarzające się nazwy stanowisk:

SELECT COUNT(stanowisko) AS 'ile stanowisk'

FROM Pracownicy;

ile stanowisk

--------------

11

Chcąc wyeliminować duplikaty w funkcji agregującej, używamy słowa DISTINCT:

SELECT COUNT(DISTINCT stanowisko) AS 'ile różnych stanowisk'

FROM Pracownicy;

ile różnych stanowisk

----------------------

5

#### Przykład 3

Funkcje agregujące pomijają (ignorują) wartości puste w swoich obliczeniach.

Jeśli na przykład chcemy znaleźć informację o łącznej kwocie przeznaczanej na dodatki funkcyjne (SUM(dod\_funkc)), to to zapytanie zwróci spodziewany wynik, ignorując wartości puste. Dodatkowo, wyświetli się ostrzeżenie (jeżeli pracujemy w trybie Grid, tj. Query > Results To > Results To Grid (Ctrl+D) , to ostrzeżenie pojawi się w karcie Messages, tuż obok karty Results; w przypadku trybu Text, tj. Query > Results To > Results To Text (Ctrl+T), ostrzeżenie będzie pod wynikiem).

SELECT SUM(dod\_funkc) AS 'suma dodatków funkcyjnych'

FROM Pracownicy;

suma dodatków funkcyjnych

--------------------------

2100,00

Warning: Null value is eliminated by an aggregate or other SET operation.

Tego typu ostrzeżenia są zgodne ze standardem ANSI i nie zawsze oznaczają poważny problem w zapytaniu. Ostrzeżenia można wyłączyć poleceniem [SET ANSI\_WARNINGS OFF;](https://docs.microsoft.com/en-us/sql/t-sql/statements/set-ansi-warnings-transact-sql) (nie jest to zalecane).

W naszym przypadku, zamiast wyłączać wszystkie ostrzeżenia ANSI, możemy w zapytaniu odfiltrować wiersze, w których dodatek funkcyjny jest niezdefiniowany:

SELECT SUM(dod\_funkc) AS 'suma dodatków funkcyjnych'

FROM Pracownicy

WHERE dod\_funkc IS NOT NULL;

Alternatywnie można użyć funkcji ISNULL() w funkcji agregującej SUM().

#### Przykład 4

Liczba pracowników i ich średnie zarobki z podziałem na stanowiska:

SELECT stanowisko,

COUNT(\*) 'liczba pracowników',

AVG(placa) 'średnia płaca'

FROM Pracownicy

GROUP BY stanowisko;

#### Przykład 5

Chcemy sprawdzić ilu pracowników pracuje na każdym ze stanowisk - włączając w to nieobsadzone stanowiska. Zacznijmy od połączenia tabeli Stanowiska z tabelą Pracownicy. Aby uwzględnić wszystkie stanowiska z tabeli Stanowiska, musimy zrobić złączenie zewnętrzne:

SELECT \*

FROM Stanowiska S

LEFT JOIN Pracownicy P

ON S.nazwa = P.stanowisko;

nazwa placa\_min placa\_max id nazwisko szef placa dod\_funkc stanowisko zatrudniony

---------- ------------- ------------- ------ --------------- ------- ----------- -------------- ---------- -----------------------

adiunkt 3000,00 4000,00 2 Jankowski 1 3500,00 NULL adiunkt 2000-09-01 00:00:00.000

adiunkt 3000,00 4000,00 3 Fiołkowska 1 3550,00 NULL adiunkt 1995-01-01 00:00:00.000

adiunkt 3000,00 4000,00 10 Andrzejewicz 5 3900,00 NULL adiunkt 2012-01-01 00:00:00.000

doktorant 1900,00 2300,00 6 Mikołajski 4 2100,00 NULL doktorant 2017-10-01 00:00:00.000

doktorant 1900,00 2300,00 7 Wójcicki 5 2350,00 NULL doktorant 2015-10-01 00:00:00.000

dziekan 3700,00 5800,00 NULL NULL NULL NULL NULL NULL NULL

profesor 4000,00 6000,00 1 Wachowiak NULL 5500,00 900,00 profesor 1990-09-01 00:00:00.000

profesor 4000,00 6000,00 4 Mielcarz 1 5000,00 500,00 profesor 1990-12-01 00:00:00.000

profesor 4000,00 6000,00 5 Różycka 4 3900,00 300,00 profesor 2011-09-01 00:00:00.000

sekretarka 2500,00 3500,00 8 Listkiewicz 1 3200,00 NULL sekretarka 1990-09-01 00:00:00.000

techniczny 2500,00 3500,00 9 Wróbel 1 2900,00 400,00 techniczny 2009-01-01 00:00:00.000

techniczny 2500,00 3500,00 11 Jankowski 5 3200,00 NULL techniczny 2000-01-01 00:00:00.000

Widzimy, że stanowisko dziekana jest nieobsadzone i powinniśmy przy nim uzyskać wartość 0 pracowników. W jaki sposób poprawnie zagregować wiersze? Spójrzmy na wyniki trzech różnych argumentów umieszczonych w COUNT():

SELECT S.nazwa,

COUNT(\*) AS 'COUNT(\*)',

COUNT(S.nazwa) AS 'COUNT(S.nazwa)',

COUNT(P.id) AS 'COUNT(P.id)'

FROM Stanowiska S

LEFT JOIN Pracownicy P

ON S.nazwa = P.stanowisko

GROUP BY S.nazwa;

nazwa COUNT(\*) COUNT(S.nazwa) COUNT(P.id)

---------- -------- -------------- -----------

adiunkt 3 3 3

doktorant 2 2 2

dziekan 1 1 0 <----

profesor 3 3 3

sekretarka 1 1 1

techniczny 1 1 1

W rezultacie, w danej grupie:

- COUNT(\*) zliczył liczbę wierszy,

- COUNT(S.nazwa) zliczył liczbę wierszy w kolumnie S.nazwa (żadna wartość nie była NULL-em),

- COUNT(P.id) zliczył liczbę wierszy w kolumnie P.id (dla stanowiska dziekana wartość była NULL-em, więc została pominięta przy zliczaniu).

Należy zatem uważać, co zliczamy - gdy po złączeniu zewnętrznym tabel nie chcemy zliczać pół-pustych wierszy, nie używamy COUNT(\*).

#### Przykład 6

Liczba pracowników i ich średnie zarobki z podziałem na stanowiska; zwracana jest informacja dotycząca tylko tych stanowisk, na których minimalna wypłacana płaca jest większa niż 3000:

SELECT stanowisko,

COUNT(\*) 'liczba pracowników',

AVG(placa) 'średnia płaca'

FROM Pracownicy

GROUP BY stanowisko

HAVING MIN(placa) > 3000;

## Operacje na zbiorach

# 04 – SELECT – funkcje agregujące, operacje na zbiorach, podzapytania w klauzuli FROM i SELECT

- [UNION](https://docs.microsoft.com/en-us/sql/t-sql/language-elements/set-operators-union-transact-sql) – suma zbiorów,

- [UNION ALL](https://docs.microsoft.com/en-us/sql/t-sql/language-elements/set-operators-union-transact-sql) – suma zbiorów z pozostawieniem duplikatów,

- [EXCEPT](https://docs.microsoft.com/en-us/sql/t-sql/language-elements/set-operators-except-and-intersect-transact-sql) – różnica zbiorów,

- [INTERSECT](https://docs.microsoft.com/en-us/sql/t-sql/language-elements/set-operators-except-and-intersect-transact-sql) – przekrój (część wspólna) zbiorów.

Operacje ilustruje poniższy schemat:

|  |
| --- |
|  |
| **Wyniki zapytań** | **UNION** | **UNION ALL** | **EXCEPT** | **INTERSECT** |
|  |  |  |  |  |
|  |  |  |  |  |

Operacje teoriomnogościowe UNION, UNION ALL, EXCEPT i INTERSECT mogą być wykonywane tylko na tabelach tego samego typu (o tej samej liczbie kolumn tego samego typu).

#### Przykład 7

Nazwiska pracowników zarabiających powyżej 3500 razem z pracownikami zarabiającymi nie więcej niż 2500 (porównaj UNION z UNION ALL):

SELECT nazwisko,

placa,

'> 3500' [przedzial]

FROM Pracownicy

WHERE placa > 3500

UNION -- ALL

SELECT nazwisko,

placa,

'<= 2500'

FROM Pracownicy

WHERE placa <= 2500;

#### Przykład 8

Pracownicy zarabiający nie więcej niż 2900, po usunięciu tych, co zarabiają 2500:

SELECT nazwisko,

placa

FROM Pracownicy

WHERE placa <= 2900

EXCEPT

SELECT nazwisko,

placa

FROM Pracownicy

WHERE placa > 2500;

#### Przykład 9

Pracownicy zarabiający więcej niż 2500 i nie więcej niż 3200:

SELECT nazwisko,

placa

FROM Pracownicy

WHERE placa > 2500

INTERSECT

SELECT nazwisko,

placa

FROM Pracownicy

WHERE placa <= 3200;

## Podzapytania w klauzuli FROM i SELECT

# 04 – SELECT – funkcje agregujące, operacje na zbiorach, podzapytania w klauzuli FROM i SELECT

Klauzula FROM

Ponieważ wynikiem polecenia SELECT jest relacja (tabela), można je umieścić w klauzuli FROM.

**Uwaga**: składnia T-SQL wymaga nazwania takiego podzapytania (w przykładzie poniżej: AS Tabela) oraz nazwania każdej z kolumn (jeśli nazwa nie istnieje).

#### Przykład 10

Średnia liczba pracowników na stanowiskach:

SELECT AVG(liczba)

FROM (SELECT COUNT(\*) AS liczba

FROM Pracownicy

GROUP BY stanowisko) AS Tabela;

#### Przykład 11

Podzapytanie w klauzuli FROM może być pomocne jeżeli w SELECT musimy wyliczyć nową kolumnę, a następnie chcemy jej użyć w WHERE. Normalnie nie jest to możliwe, ponieważ filtrowanie wierszy odbywa się przed selekcją kolumn.

Chcemy znaleźć listę pracowników, których płaca nie znajduje się w widełkach płacowych zdefiniowanych dla ich stanowisk.

Poniższe zapytanie zwróci błąd:

SELECT nazwisko,

placa,

placa\_min,

placa\_max,

CASE

WHEN placa BETWEEN placa\_min AND placa\_max

THEN 1

ELSE 0

END AS 'płaca w widełkach?'

FROM Pracownicy P

JOIN Stanowiska S

ON P.stanowisko = S.nazwa

WHERE "płaca w widełkach?" = 0;

Msg 207, Level 16, State 1, Line 13

Invalid column name 'płaca w widełkach?'.

Przy użyciu podzapytania w FROM uzyskamy poprawny wynik:

SELECT nazwisko,

placa,

placa\_min,

placa\_max,

"płaca w widełkach?"

FROM (SELECT \*,

CASE

WHEN placa BETWEEN placa\_min AND placa\_max

THEN 1

ELSE 0

END AS 'płaca w widełkach?'

FROM Pracownicy P

JOIN Stanowiska S

ON P.stanowisko = S.nazwa) AS T

WHERE "płaca w widełkach?" = 0;

#### Przykład 12

Stosunek liczby pracowników na poszczególnych stanowiskach do liczby wszystkich pracowników:

SELECT stanowisko,

CAST(COUNT(\*) AS REAL)/(SELECT COUNT(\*) FROM Pracownicy) AS 'udzial'

FROM Pracownicy

GROUP BY stanowisko;

**Uwaga**: funkcje analityczne pozwalają na wykonanie powyższego zapytania w wydajniejszy sposób.