****

****

LISTA KOMEND DO MYSQL

v2.2/2017

****

****

Interaktywny spis treści:

Kliknij w treść aby się tam przenieść

Na szybko do sprawdzenia
(kliknij w jedna z opcji):

⇨ **Wróć do spisu treści (kliknij)**

INNER JOIN => łączy dane z kilku tabel za pomocą porównania kolumn z różnych tabel, można go użyć do np. złączenia tabel , które mają rozdzielone dane, a połączyć je możemy za pomocą

indeksów np. IdSprzedawcy = IdKupca, np.:

dla 2 tabel:

tabela sprzedajacy:

IdSprzedawcy	Identyfikator	Pensja	DataZatrudnienia	SprzedalprzedmiotNumertelefonusprzedawcy

tabela kupujacy:

IdKupca	*Imie*	*Nazwisko*	*Nr.tel.kupca*	*Adres*	*Kupilprzedmiot*

SELECT Imie,Nazwisko,Kupilprzedmiot,Numertelefonusprzedawcy **FROM** sprzedajacy **INNER JOIN** kupujacy **ON** kupujacy.IdKupca=sprzedajacy.IdSprzedawcy;

dla 3 tabel:

tabela sprzedajacy:

IdSprzedawcy	Identyfikator	Pensja	DataZatrudnienia	SprzedalprzedmiotNumerTelefonusprzedawcy

tabela kupujacy:

IdKupca	*Imie*	*Nazwisko*	*Nrtel_kupca*	*Adres*	*IdSprzedawcy*	*IdTowaru*

tabela towary:

IdTowaru	Nazwatowaru	Cenatowaru

SELECT Imie,Nazwisko,Kupilprzedmiot,Nrtel_kupca,Cenatowaru **FROM** sprzedajacy **INNER JOIN** kupujacy **ON** kupujacy.IdKupca=sprzedajacy.IdSprzedawcy **INNER JOIN** towary **ON** towary.IdTowaru = kupujacy.IdTowaru;

Uwaga

Jeśli chcemy połączyć dwie tabele, które mają takie same kolumny i te kolumny mają nadane klucze główne to, możemy skorzystać z opcji:

Tabela ogrod 1	Tabela ogrod2
#IdKwiata Nazwa Rodzaj Kwitnie	#IdKwiata Nazwa Rodzaj Kwitnie

- Kratką czyli # oznaczamy klucz główny a (#) klucz obcy

SELECT Nazwa,Rodzaj,Kwitnie **FROM** ogrod1 **INNER JOIN** ogrod2 **USING** IdKwiata;

łączenie tabel przy pomocy where

Tabele możemy dołączać również za pomocą konstrukcji **where**. Użyteczna jest tutaj umiejętność wykorzystania **aliasów**, żebyśmy nie pogubili się w zapisie. np. dla powyższych trzech tabel sprzedajacy,kupujący,towary będzie to:

SELECT NumerTelefonusprzedawcy,Imie,Nazwisko,Kupilprzedmiot,Nazwatowaru,Cenatowaru **FROM** sprzedajacy s,kupujący k,towary t **WHERE** s.IdSprzedawcy=k.IdSprzedawcy **AND** k.IdTowaru=t.IdTowaru;

s,k,t to tzw. aliasy – tymczasowe nazwy dla tabel

==========

****Dodawanie danych do tabeli****

INSERT INTO nazwatabeli **VALUES** (...) => polecenie, które służy do dopisywania danych do tabeli. Jeśli mamy np. tabele:

towary:

IdTowaru	Nazwatowaru	Cenatowaru

(pamiętamy o typach danych jakie nadaliśmy kolumnom, aby NIE podać niewłaściwej!!!!):

INSERT INTO towary **VALUES** (5,'Puszka CocaCola', 2.50);

wcześniej oczywiście musieliśmy utworzyć tabele, do której będziemy dopisywać dane.

#Uwaga: Apostrofów w dodawaniu danych używamy, gdy mamy do wklepania tekst. Np. imie o typie danych VARCHAR

****tworzenie tabeli****

służy do tego polecenie:

CREATE TABLE nazwatabeli (... TYP DANYCH, ... TYP DANYCH, itd..);

czyli dla naszego przykładu wyglądałoby to tak:

CREATE TABLE towary (IdTowaru INT(11), Nazwatowaru VARCHAR(50), Cenatowaru DEC(5,2));

- towary to nazwa tabeli
- IdTowaru, Nazwatowaru, Cenatowaru to kolumny

A co jeśli chcemy dodać wartości tylko do pewnych kolumn???

Załóżmy, że mamy tabele klienci o takich kolumnach:

Nazwisko	Imie	PESEL	Nr Telefonu	Wiek

I chcemy tylko wypełnić kolumny Nazwisko,Imie,Wiek, stosujemy następującą operację:

INSERT INTO klienci(Nazwisko,Imie,Wiek) **VALUES** (‘Tomaszewski’, ‘Andrzej’,19);

Czyli pod Nazwisko idzie Tomaszewski, pod imię Andrzej a pod wiek 19(bo w takiej kolejności podaliśmy przy **klienci**, może być inna)

==========

⇨ Wróć do spisu treści (kliknij)

Typy danych w MYSQL:

==========

CHAR() - pojedynczy znak

VARCHAR() - zmiennoznakowe do np. tekstu, peselu w nawiasie podajemy maksymalna liczbę znaków jak mozemy pomieścić

INT(n) - całkowita, gdzie n to maksymalna liczba cyfr

DATE - typu data (podajemy w taki sposób ROK-MIESIAC-DZIEN czyli np. '1996-02-12' ale możemy powiedzieć SQL-owi, żeby w inny sposób zapisywał datę)

DOUBLE(m,n) - podwójnej precyzji liczba gdzie m oznacza maksymalna szerokość liczby a n oznacza liczbę miejsc po przecinku

DEC(m,n) - całkowita liczba gdzie m oznacza maksymalna szerokość liczby a n oznacza liczbę miejsc po przecinku

// uwaga: m nie może być mniejsze niż n

==========

desc nazwatabeli => służy do wyświetlenia budowy tabeli to znaczy, jaka kolumna ma jaki format danych, gdzie jest klucz główny itd.

np. **desc** pilkarze;

desc == **description** (z ang. opis)

show tables => do wyświetlenia listy tabel które mamy

show databases => do wyświetlenia listy baz danych jakimi dysponujemy

CREATE DATABASE nazwabazy => do stworzenia bazy danych

np. create database pracownicy;

==========

ALTER TABLE *nazwatabeli* to jedno z obszerniejszych poleceń do modyfikowania naszej tabeli,

mozemy za jej pomocą:

1) zmienić typ danych danej tabeli:

szablon:

ALTER TABLE *nazwa_tabeli* **MODIFY** *nazwa_kolumny typ_danych_kolumny*;

np. dla tabeli towary o kolumnach:

IdTowaru	Nazwatowaru	Cenatowaru

będzie to tak:

ALTER TABLE *towary* **MODIFY** *Nazwatowaru VARCHAR(60);*

czyli zmieniłem typ danych dla Nazwatowaru na VARCHAR(60)

2) dodać kolumnę do tabeli:

szablon:

ALTER TABLE *nazwa_tabeli* **ADD** *nazwa_kolumny typ_danych_kolumny*;

np. dla tabeli towary o kolumnach:

IdTowaru	Nazwatowaru	Cenatowaru

będzie to tak:

ALTER TABLE *towary* **ADD** *Ilosc INT(50);*

czyli dodajemy kolumnę Ilosc która jest typu całkowitego i może przechowywać liczby dużego rozmiaru

3) usunąć kolumnę z tabeli:

szablon:

ALTER TABLE *nazwa_tabeli* **DROP COLUMN** *nazwa_kolumny*;

np. dla tabeli towary o kolumnach:

IdTowaru	Nazwatowaru	Ilosc

będzie to tak:

ALTER TABLE *towary* **DROP COLUMN** *Ilosc*;

czyli usuwamy kolumnę Ilosc

Uwaga!

Jeśli będziemy chcieli usunąć dane z tabeli w taki sposób aby schemat tabeli nie został naruszony to możemy użyć polecenia **DELETE**

DELETE * **FROM** *towary*; *//* **spowoduje usunięcie wszystkich rekordów z towary**

4) zmienić nazwę kolumny z tabeli:

szablon:

ALTER TABLE *nazwa_tabeli* **RENAME COLUMN** *stara_nazwa* **TO** *nowa_nazwa*;

np. dla tabeli towary o kolumnach:

IdTowaru	Nazwatowaru	Cenatowaru	Ilosc

będzie to tak:

ALTER TABLE *towary* **RENAME COLUMN** *Nazwatowaru* **TO** *TowarOpis*;

//komentarz: dla mySQL coś nie pyka…??

5) zmienić nazwę tabeli:

szablon:

ALTER TABLE *nazwa_tabeli* **RENAME TO** *nowa_nazwa_tabeli*;

np. dla tabeli towary o kolumnach:

IdTowaru	Nazwatowaru	Cenatowaru	Ilosc

będzie to tak:

ALTER TABLE *towary* **RENAME TO** *Towary2017*;

==========

DATEDIFF (staradata, nowadata) => służy do obliczenia różnicy w dniach między datami, np.

DATEDIFF ('2017-05-07', '2017-05-08') zwróci wartość 1 (no bo jeden dzień między tymi datami jest), mozemy stosować do kolumn np:

DATEDIFF(DataZak, DataRozp);

Operacje na dacie:

DAY() - pobiera dzień z daty

MONTH() - pobiera miesiąc z daty

YEAR() - pobiera rok z daty

CURDATE() - zwraca obecna datę

NOW() – zwraca obecną datę i czas

CURTIME() – zwraca obecny czas

mozemy między nimi wykonywać różne operacje, dla **DAY, MONTH, YEAR** w nawiasach podajemy z jakiej kolumny chcemy „wyłuskać" daną rzeczy np. miesiąc, to **MONTH**(DataUrodzenia)

==========

CONCAT() - polecenie które służy do łączenia kolumn (inaczej scalania kolumn),

np. dla tabeli pilkarze, która ma takie kolumny:

Nazwisko	Imie	RokUrodzenia	PESEL

żeby połączyć Nazwisko i Imie to korzystamy z funkcji **CONCAT()** czyli:

SELECT CONCAT(*Nazwisko*, ' ', *Imie*), *RokUrodzenia*, *PESEL* **FROM** *pilkarze*;

odstęp w ' ' służy do tego żeby Nazwisko i imię nie były połączone czyli żeby nie było np. takiej sytuacji => *KowalskiJan*

w apostrofach podajemy tekst który zawsze się będzie wyświetlał czyli w tym przypadku np. odstęp miedzy Nazwiskiem a imieniem

==========

WHERE => dodatkowa instrukcja w zapytaniu która pozwala na wyszczególnienie danych które chcemy otrzymać np. z tabeli:

towary:

IdTowaru	Nazwatowaru	Ilosc	Cenatowaru

których cena jest większa niż 2500 to zapytamy

w ten sposób:

SELECT * **FROM** *towary* **WHERE** *Cenatowaru* > *2500*;

Symbol * oznacza, że bierzemy wszystkie kolumny do wyświetlenia

==========

BETWEEN .... AND ... => dodatkowa instrukcja w zapytaniu która pozwala na wyszczególnienie danych z tzw. „ pomiedzy „ czyli np.

z tabeli towary:

IdTowaru	Nazwatowaru	Ilosc	Cenatowaru

towary których cena jest większa niż 2500 ale mniejsza niż 2700 to zapytamy

SELECT * **FROM** *towary* **WHERE** *Cenatowaru* **BETWEEN** *2500* **AND** *2700*;

czyli: wybierz z towary te rekordy gdzie cenatowaru jest pomiedzy 2500 a 2700.

==========

ORDER BY => instrukcja w zapytaniu która służy do porządkowania danych, np.

z tabeli towary:

IdTowaru	Nazwatowaru	Ilosc	Cenatowaru

uporządkujemy wpierw według nazwy towaru(rosnąco) a potem według Cenatowaru rosnąco

atrybuty **ORDER BY**: (możemy manipulować czy chcemy sortować rosnąco czy malejąco, domyślnie jest rosnąco)

DESC (z ang. DESCENDING) - malejąco

ASC (z ang. ASCENDING) - rosnąco

czyli będzie to wyglądać tak:

SELECT * **FROM** *towary* **ORDER BY** *Nazwatowaru* **ASC**, *Cenatowaru **ASC***;

czyli: posortuj wpierw według nazwy towaru rosnąco a potem według ceny towaru rosnąco

==========

GROUP BY => instrukcja w zapytaniu której zwykle używamy z funkcjami **agregującymi** czyli *(COUNT, MAX, MIN, SUM, AVG)* aby pogrupować według czegoś wyniki:

Np. dla tabeli klienci o kolumnach:

Id_Klienta	Kraj_pochodzenia

COUNT znaczy licz, jako, że pole Id_Klienta jest unikalne to jesteśmy pewni, że nie zliczy błędnie ile tych klientów się nazbierało.

SELECT COUNT(Id_Klienta), *Kraj_pochodzenia*

FROM *klienci*

GROUP BY *Kraj_pochodzenia*;

policzy nam id klientów ze względu na kraj np:

możemy otrzymać taki rezultat:

Id_Klienta	**Kraj_pochodzenia**
9	Polska
11	Niemcy
50	Argentyna

Czyli np. 9 to liczba klientów którzy pochodzą z Polski

DISTINCT => dodatkowa instrukcja do zapytania która wyklucza powtórzenia np. dla tabeli:

towary:

IdTowaru	Nazwatowaru	Ilosc	Cenatowaru

chcemy aby Nazwy towarów się nie powtarzały to bierzemy:

SELECT DISTINCT *NazwaTowaru,IdTowaru, Cenatowaru* **FROM** *towary*;

mozemy użyć tej instrukcji z **COUNT** gdybyśmy np. chcieli policzyć ile jest państw z tabeli klienci o kolumnach: Id_Klienta, Kraj_pochodzenia

COUNT to instrukcja która liczy liczbę rekordów, w polaczeniu z **DISTINCT** nie bierze pod uwagę powtórzeń np.

SELECT COUNT(**DISTINCT** *Kraj_pochodzenia*) **FROM** *Id_Klienta*;

tutaj nie weźmie pod uwagę np. trzy razy Polska tylko jeden raz!!

==========

Funkcje agregujące:

*w nawiasie podajemy oczywiście nazwę kolumny:

MAX() - zwraca maksymalna wartość

MIN() - zwraca minimalna wartość

SUM() - zwraca sumę wartości rekordów z kolumny np. CenaTowaru

AVG() - zwraca średnią wartość z kolumny (od average)

COUNT() – zwraca ile czegoś występuje (count = liczyć) ze względu na kolumnę którą podamy w nawiasach

UWAGA: COUNT TO NIE JEST SUMA!!!

Przykładowo liczymy sobie ile towarów jest i wyświetli nam załóżmy 225 (przykład:
SELECT COUNT(IdTowaru) **FROM** towary;

Jeśli obawiamy się , że policzy powtórzenia to możemy zawsze zastosować opcję z DISTINCT, wtedy COUNT nie uwzględni nam powtórzonych rekordów (te same podlicza jako jeden) np.

SELECT COUNT(**DISTINCT** *IdTowaru*) **FROM** *towary*;

Legenda:
Towary – tabela z której biorę dane,
IdTowaru – kolumna którą poddaje COUNTOWI (czyli policzeniu)
==========

IS NULL => mozemy użyć do zapytania, czyli gdzie wartość jest **NULL**, np.

Dla Tabeli towary:

IdTowaru	Nazwatowaru	Ilosc	Cenatowaru

poszukamy tych towarów gdzie Cenatowaru jest NULL

czyli:

SELECT * **FROM** *towary* **WHERE** *Cenatowaru* **IS NULL**;

==========

IS NOT NULL => mozemy użyć do zapytania , czyli gdzie wartość NIE jest NULL

Dla Tabeli towary:

IdTowaru	Nazwatowaru	Ilosc	Cenatowaru

poszukamy tych towarów gdzie Cenatowaru nie jest NULL

czyli:

SELECT * **FROM** *towary* **WHERE** *Cenatowaru* **IS NOT NULL**;

==========

AND,OR,WHERE => do zapytań złożonych

AND czyli "i"

OR czyli "lub"

WHERE czyli "gdzie"

np.

Dla Tabeli towary:

IdTowaru	Nazwatowaru	Ilosc	Cenatowaru

chcemy aby znalazł towary których Nazwa = GD a Cenatowaru = 520 , to

SELECT * **FROM** *towary* **WHERE** *Cenatowar*u = 520 **AND** *Nazwa* = GD;

Dla Tabeli towary:

IdTowaru	Nazwatowaru	Ilosc	Cenatowaru

chcemy aby znalazł towary których Nazwa jest GD lub AB, to

SELECT * **FROM** *towary* **WHERE** *Nazwatowaru* = GD **OR** *Nazwatowaru* = AB;

==========

UPDATE (kwerenda)=> polecenie do aktualizowania danych w tabelach, jeśli chcemy zmienić jakaś wartość

Szablon:

UPDATE *nazwa_tabeli* **SET** *kolumna1 = wartosc1, kolumna2 = wartosc2, ...* **WHERE** *warunek*;

Jest tabela towary:

IdTowaru	Nazwatowaru	Ilosc	Cenatowaru

I wyobraźmy sobie, że mamy takie rekordy:

3	Bibuła	5 000	0.50
4	Pizza	125	5.60
5	CocaCola	100	2.50

a my chcemy podnieść cenę CocaColi do 3 zł, to stosujemy **UPDATE** czyli:

UPDATE *towary* **SET** *Cenatowaru* = 3.00 **WHERE** *IdTowaru* = 5;

czyli znaczy to dosłownie: ustaw Cenatowaru na 3 gdzie IdTowaru = 5

==========

LIKE => instrukcja do zapytania która pozwala nam na znalezienie rekordów "**podobnych**", mozemy ustawić

żeby filtrował z dokładną nazwa(czyli znakiem =) ale dodatkowo LIKE pozwala nam na znalezienie tych rekordów które np.

w nazwie maja pierwsze trzy litery kra... albo trzy ostatnie albo datę podobna... No, ma wiele zastosowań

np tabela towary:

IdTowaru	Nazwatowaru	Ilosc	Cenatowaru

chcemy znaleźć te Nazwy towarów których nazwa zaczyna się na Opl ..

SELECT * **FROM** *towary* **WHERE** *NazwaTowaru* **LIKE** 'Opl%';

PS: * oznacza wyświetlenie wszystkiego!!

PS2: dla operatora LIKE podajemy co chcemy szukać podobnego w pojedynczych apostrofach czyli ‘ ‘

szablon budowy LIKE:

SELECT *kolumna1, kolumna2, ...* **FROM** *nazwa_tabeli* **WHERE** *nazwa_kolumny* **LIKE** ‘coś’; *//* **podajemy w apostrofach!!**

Przykładowe opisy dla **LIKE** z zastosowanie znaczków "%" i "_", czyli procent i podłoga

_ znaczy dokładnie jeden znak (tzw. **podłoga**!!!!!)

% znaczy dowolna liczba znaków po procencie!!

a więc...
dla przykładu:

Treść zapytania:	**CO ROBI?**
... **WHERE** *nazwaklienta* **LIKE** 'b%'	Szuka wszystkich wartości które zaczynają się na literę b
... **WHERE** *nazwaklienta* **LIKE** '%b'	Szuka wszystkich wartości które kończą się na literę b
... **WHERE** *nazwaklienta* **LIKE** '%or%'	Szuka tych wartości które maja or w dowolnym miejscu
... **WHERE** *nazwaklienta* **LIKE** '_s%'	Szuka tych wartości które maja s na drugim miejscu
... **WHERE** *nazwaklienta* **LIKE** 'a_%_%'	Szuka tych wartości które zaczynają się na a i maja przynajmniej 3 znaki (bo litera a i dwie podłogi)
... **WHERE** *nazwaklienta* **LIKE** 'a%o'	Szuka tych wartości które zaczynają się na literę a a kończą na literę o

==========

IN => operator do zapytania który pozwala nam na wyszczególnienie wielokrotnej liczby wartości w klauzuli (zapytaniu) **WHERE**

szablon:

SELECT *nazwy_kolumn* **FROM** *nazwa_tabeli* **WHERE** *nazwa_kolumny* **IN** *(wartosc1, wartosc2, ...);*

np dla tabeli towary:

IdTowaru	Nazwatowaru	Ilosc	Cenatowaru

chcemy wyświetlić te towary które maja największa wartość, zatem mozemy

zastosować OPERATOR IN + MAX

czyli:

SELECT * **FROM** *towary* **WHERE** *Cenatowaru* **IN** (**MAX**(*Cenatowaru*));

a gdy np. chcemy konkretne kilka wartości to:

SELECT * **FROM** *towary* **WHERE** *Cenatowaru* **IN** ('2000', '500', '20');

czyli wyświetl te rekordy z tabeli towary których Cena towaru wynosi 2000 lub 500 lub 20

przeciwieństwem do IN, JEST **NOT IN** czyli jeśli użyjemy NOT IN dla tego:

SELECT * FROM towary WHERE Cenatowaru IN ('2000', '500', '20');

to otrzymamy te rekordy z tabeli towary ktore nie maja ceny towaru rownej 2000 lub 500 lub 20

****rekord** znaczy wiersz

****krotka znaczy **rekord**

==========

Aliasy w MYSQL są po to, aby dać nazwie tabeli, kolumny **TYMCZASOWA** nazwę, użyteczne gdy np. musimy się odnieść

do tej samej kolumny z tabeli **kilka razy** a mysql krzyczy ze ta kolumna nie jest potem przez to unikalna. Właśnie aliasy są dobrym środkiem aby taki problem ominąć.

szablon dla aliasu dla kolumny:

SELECT nazwy_kolumn **AS** nazwa_aliasu **FROM** nazwa_tabeli;

szablon dla aliasu dla tabeli:

SELECT nazwy_kolumn **FROM** nazwa_tabeli **AS** nazwa_aliasu;

Użycie zwrotu „AS” w aliasach jest opcjonalne.

Czyli np.

SELECT K.Babla **FROM** krakra K;

To K będzie aliasem,

K.Babla mówi ze bierzemy kolumnę Babla z tabeli krakra (dobre gdy np. chcemy sprecyzować skąd wziąć kolumnę gdy się powtarza w kilku tabelach..)

Mamy tabele klienci o kolumnach:

klientID	Nazwaklienta

Kilka uwag:

SELECT klientID as ID, **Nazwaklienta** AS Klient **FROM** klienci;

spowoduje ze nadajemy nazwy kolumnom:

=> dane z klientID wyświetli w kolumnie która będzie nazwana ID (to tylko nazwa! Do tego się nie odwołujemy!)

=> dane z Nazwaklienta wyświetli w kolumnie która będzie nazwana Klient (tylko nazwa)

PS: W momencie gdy chcemy nadać kolumnie nazwę wieloczłonową, to wypisujemy nazwę w apostrofach czyli np.

SELECT klientID as ‘ID Klienta’, **Nazwaklienta** AS Klient **FROM** klienci;

albo np.

gdy napotykamy problem z unikalnością:

dla tabel:

Kluby:

Idklubu	Nazwa	Miasto

Mecze:

IdGosc	idGosp	Wynik	Datameczu

SELECT K1.Nazwa AS 'nazwaGospodarza', K1.Miasto AS 'Miasto_gospodarza', K2.Nazwa AS 'nazwaGoscia', K2.Miasto AS 'Miasto_goscia' **FROM** Kluby

K1 **INNER JOIN** Mecze **ON** K1.idklubu = Mecze.idGosp **INNER JOIN** Kluby K2 **ON** K2.idklubu=Mecze.idGosc;

==========

JOIN => zapytania, co zwracają?

wyobraźmy sobie ze:

mamy takie zbiory:

TABELA 1	CZĘŚĆ WSPÓLNA: TABELA 1 i TABELA 2	TABELA 2

INNER JOIN => zwraca cześć wspólną TABELI 1 i TABELI 2 *(dlatego dajemy warunek połączenia w INNER JOINIE.. bo baza musi o coś oprzeć część wspólną)*

LEFT JOIN => zwraca rekordy z lewej tabeli i te które pasują z prawej tabeli *(czyli mówiąc prosto: TABELA 1 i cześć wspólna TABEL 1 i 2)*

RIGHT JOIN => zwraca rekordy z prawej tabeli i te które pasują z lewej tabeli *(czyli mówiąc prosto: cześć wspólna TABEL 1 i 2 i TABELA 2)*

FULL JOIN => zwraca te rekordy które pasują i z lewej i z prawej tabeli *(czyli mówiąc prosto: TABELA 1, cześć wspólna TABEL 1 i 2, TABELA 2)*

rekordy = wiersze

==========

UNION => instrukcja która pozwala na polaczenie wyników kilku tabel w jedna,

jeśli mamy np. dwie tabele:

towary2017:

IdTowaru	Nazwatowaru	Cenatowaru

towary2016:

IdTowaru	Nazwatowaru	Cenatowaru

i chcemy **połączyć** ich wyniki to zastosujemy operatora **UNION**

czyli:

SELECT * **FROM** *towary2017* **UNION**

SELECT * **FROM** *towary2016*;

==========

HAVING => zapytanie które mozemy łączyć z funkcjami agregującymi czyli np. MIN, MAX, zostało dodane do MYSQL PONIEWAZ

zapytania (czyli inaczej klauzuli) WHERE NIE MOZEMY STOSOWAC Z FUNKCJAMI AGREGUJACYMI!!!

klauzula WHERE operuje tylko na danych rekordów a nie na danych zagregowanych.

Mamy tabele klienci:

IDKlienta	Pochodzenie

np.

SELECT COUNT(*IDKlienta*), *Pochodzenie* **FROM** *klienci* **GROUP BY** *Pochodzenie* **HAVING COUNT**(*IDKlienta*) > 5;

czyli policz liczba klientów i pogrupuj według Pochodzenia gdzie policzona liczba klientów wyniosła więcej niż 50

==========

EXISTS (do podzapytań)

Operator **EXISTS** (istnieje) => zwraca rekordy z zapytania jeśli istnieją rekordy spełniające podzapytanie z wykorzystaniem operatora EXISTS

czyli np.

dla tabel:

towary2017:

IdTowaru	Nazwatowaru	Cenatowaru

towary2016:

IdTowaru	Nazwatowaru	Cenatowaru

chcemy żeby wyświetliło te nazwy towarów z tabeli: towary2016 gdzie istnieje Nazwatowaru z tabeli: towary2017 gdzie indeksy z obu tabel się zgadzają i Cena towaru jest mniejsza niż 500

otrzymamy to za pomocą operatora **EXISTS** który łączy się z **WHERE**!

SELECT *Nazwatowaru* **FROM** *towary2016* **WHERE EXISTS** (**SELECT** *Nazwatowaru* **FROM** *towary2017* **WHERE** *towary2016.IdTowaru = towary2017.IdTowaru* **AND** *Cenatowaru < 500*);

==========

Operatory porównywania z jakich mozemy korzystać w SQL to:

=, <>, !=, >, >=, <, or <=

==========

Operatory **ANY**, **ALL**: => używamy ich z klauzulami (zapytaniami) **WHERE** lub **HAVING**

ANY => jakikolwiek

ALL => wszystkie

***Operator **ANY** zwraca prawdę jeśli **jakikolwiek** rekord z podzapytania spełnia warunek*

***Operator **ALL** zwraca prawdę jeśli **wszystkie** rekordy z podzapytania spełnia warunek*

##########

Uwaga:

za słowo **operator** w poniższych szablonach

wstawiamy, jeden z tych znaków:

=, <>, !=, >, >=, <, or <=

##########

=> szablon OPERATORA ANY:

SELECT nazwy_kolumn

FROM nazwa_tabeli

WHERE nazwa_kolumny ***operator*** **ANY**

(**SELECT** nazwa_kolumny **FROM** nazwa_tabeli **WHERE** warunek);

=> szablon OPERATORA ALL:

SELECT nazwy_kolumn

FROM nazwa_tabeli

WHERE nazwa_kolumny ***operator*** **ALL**

(**SELECT** nazwa_kolumny **FROM** nazwa_tabeli **WHERE** warunek);

np... dla następujących tabel:

Produkty:

IDProduktu	Nazwaproduktu

Szczegolyzamowien:

IDProduktu	Nazwaproduktu

załóżmy, że w tabeli są takie zamówienia których ilość jest równa 10, **ale NIE WSZYSTKIE!**

stosując operator ANY:

SELECT Nazwaproduktu

FROM Produkty

WHERE IDProduktu **= ANY** (**SELECT** IDProduktu **FROM** Szczegolyzamowien **WHERE** Ilosc = 10);

czyli tłumacząc na normalny język:

wyświetl te nazwy produktów których ilość w tabeli Szczegolyzamowien jest równa 10, jako ze chociaż kilka się znajdzie to zostaną one wyświetlone

stosując operator ALL:

SELECT Nazwaproduktu

FROM Produkty

WHERE IDProduktu = **ALL** (**SELECT** IDProduktu **FROM** Szczegolyzamowien **WHERE** Ilosc = 10);

czyli tłumacząc na normalny język:

wyświetl nazwy produktów których ilość w tabeli Szczegolnyzamowien jest równa 10,

nie wszystkich zamówień ilość wynosi 10, zatem program nie wyświetli nic.

==========

INSERT INTO => polecenie służy do przekopiowania danych z jednej tabeli do drugiej

***INSERT INTO** wymaga, że dane które przekopiujemy będą pasować do tabeli do której chcemy je wrzucić!!

jeśli chcemy **przekopiować** WSZYSTKIE dane do nowej tabeli to stosujemy gwiazdkę **(*)** a jeśli chcemy wyszczególowić dane to możemy

dodać warunek za pomocą operatora **WHERE:**

czyli:

1) mamy tabela1, chcemy przekopiować dane do tabeli2

2) tworzymy tabela2 poleceniem:

CREATE TABLE tabela2 **LIKE** tabela1;

(stworzy nam dokładnie tabelę z takimi kolumnami i typami danych jakie ma tabela1)

3) korzystamy z polecenia:

INSERT INTO *tabela2*

SELECT * FROM *tabela1*

WHERE *warunek*;

(**WHERE** warunek opcjonalnie - bez tego przekopiuje nam wszystkie dane po prostu)

PS: To jest pojedyncza komenda. W taki sposób również możemy składać zapytania w MYSQL(czyli fragment polecenia i ENTER) tylko trzeba pamiętać o prawidłowej budowie zapytania + średniku na końcu!

jeśli chcemy przekopiować kilka kolumn danych (nie wszystkie) to piszemy tak:

INSERT INTO *tabela2 (kolumna1, kolumna2, kolumna3, ...)*

SELECT *kolumna1, kolumna2, kolumna3, ...*

FROM *tabela1*

WHERE *warunek;*

(**WHERE** warunek opcjonalnie - bez tego przekopiuje nam wszystkie dane po prostu)

==========

Jeśli chcemy utworzyć **podobną** tabelę to stosujemy następującą instrukcję:

mamy tabele: zawklub

chcemy zawklub_kopia,

CREATE TABLE zawklub_kopia **LIKE** zawklub;

==========

Jeśli podczas tworzenia tabeli chcemy dać klucz główny to, podajemy go po wpisaniu wszystkich kolumn a w nawiasie

co będzie miało klucz główny:

np:

CREATE TABLE *Osoby* (

Numerek int NOT NULL,

Imie varchar(255) NOT NULL,

Nazwisko varchar(255),

Wiek int,

PRIMARY KEY (Numerek) **<= klucz główny**

);

//Uwaga: Jeśli byśmy chcieli dodać klucz główny po stworzeniu tabeli to używamy składni:

ALTER TABLE Osoby **ADD PRIMARY KEY** (**Numerek**);

- Osoby to nazwa tabeli
- **Numerek** to nazwa kolumny

//PS: Jeśli chcemy większej liczbie kolumn dać klucze główne to podajemy po przecinku czyli: np. .**PRIMARY KEY(ID, Babla)**

NOT NULL znaczy, że pola są wymagane, czyli baza **(W TEORII)** nie pozwoli nam na dodanie rekordu bez tych danych czyli (ID,LastName)

==========

Jeśli **podczas tworzenia tabeli** *chcemy dać klucz obcy z odwołaniem to podajemy go po wpisaniu wszystkich kolumn a*

w nawiasie do czego się odwołuję.

np.

CREATE TABLE Zamowienia (

IDZamowienia int NOT NULL,

OrderNumber int NOT NULL,

Numerek int,

PRIMARY KEY (IDZamowienia),

FOREIGN KEY (**Numerek**) **REFERENCES** Osoby(**Numerek**) **<= klucz obcy**

);

Numerek to kolumna z tabeli, którą teraz tworzymy czyli Zamowienia!

Numerek to kolumna z tabeli Osoby, którą stworzyliśmy wcześniej **(patrz wyżej to z kluczem głównym)**

//Uwaga: Jeśli byśmy chcieli dodać klucz obcy **PO** stworzeniu tabeli to używamy składni:

ALTER TABLE Zamowienia **ADD FOREIGN KEY** (**Numerek**) **REFERENCES** Osoby(**Numerek**);

- Zamowienia to nazwa tabeli
- **Numerek** to kolumna z tabeli Zamowienia
- **Numerek** to kolumna z tabeli Osoby

==========

Autoinkrementacja (**AUTO_INCREMENT**) - służy do tego, żeby komputer sam numerował kolejne rekordy a nie my:

np:

CREATE TABLE **Persons** (

ID int NOT NULL **AUTO_INCREMENT**,

LastName varchar(255) NOT NULL,

FirstName varchar(255),

Age int,

PRIMARY KEY (ID)

);

Wtedy trzeba pamiętać, że , przy wrzucaniu danych do tabeli Persons musimy specyfikować gdzie dane wrzucić, czyli:

INSERT INTO Persons(LastName,FirstName,Age) **VALUES**(‘Kaczor’,’Dawid’,’20’);

I ten rekord(krotka) dostanie swój ***unikalny (kolejny)*** numer!!

//Notka: A więc aby wrzucać dane **tylko** do konkretnych kolumn a nie do wszystkich to stosujemy składnię:

INSERT INTO nazwatabeli(nazwa kolumny, nazwa kolumny …) **VALUES** (tresc, tresc);

Przykład dla tabeli:
towary2017:
(chce tylko dodać IdTowaru i nazwę towaru, bez ceny!)

IdTowaru	Nazwatowaru	Cenatowaru

Odp**: INSERT INTO** towary2017(IdTowaru, Nazwatowaru) **VALUES**(1, 'Banan');

// Warto pamiętać, że w VALUES w apostrofach umieszczamy te dane, które są w **VARCHAR**

==========

SUBSTRING() – komenda która, wyciaga z żądanej kolumny/łańcucha interesujące nas fragmenty np. z peselu, z daty urodzenia

budowa **SUBSTRING():**

SUBSTRING*(nazwa_kolumny, zktóregoznakuSTART, ileznakówpobrać)*

a bardziej profesjonalnie wygląda to tak:

SUBSTRING(łańcuch - czyli **string**, pozycja, długość)

ponieważ równie dobrze możemy, dać **SUBSTRING** od tekstu (łańcuch = ciąg znaków) np.

SUBSTRING('w3resource', 1, 2); co da w rezultacie następujący wynik w mysql: w3

SUBSTRING('w3resource', 4, 4); co da w rezultacie następujący wynik w mysql: sour

itd...

Uwaga

Ważne, żeby polecenie **SUBSTRING** było koniecznie przy nawiasie, jeśli między nimi

będzie odstęp **mysql** będzie krzyczał, że niewłaściwie użyte zapytanie:

TEGO CHCEMY **UNIKNĄĆ** czyli np. SUBSTRING (...) <= to jest źle!

Między SUBSTRING a (…) jest odstęp.

Mamy np. tabele pilkarze o kolumnie PESEL i takie rekordy:

PESEL
96031100121
13541255451

czyli jeśli np. chcemy z peselu wyciągnąć miesiąc, np dla tego:

96031100121

to zastosujemy: **SUBSTRING**(pesel, 3, 2)

i w ten sposób MYSQL zwróci nam suchą liczbę 03 czyli *nasz miesiąc*,

możemy ją wykorzystać do porównywania np. z **MONTH()** z datą urodzenia.

możemy też wyciągać np. konkretny fragment daty,

Mamy np. tabele pilkarze o kolumnie DataUrodzenia i takie rekordy:

DataUrodzenia
1986-02-15
1996-05-05

To dla np. 1986-02-15, chcemy z niej wyciągnąć **nie** cały rok a ostatnie dwie cyfry roku to co robimy?

SUBSTRING(DataUrodzenia, 3, 2)

czyli bierzemy z daty urodzenia od 3 znaku, dwa znaki czyli 8 i 6, czyli 86

a teraz wyciągnijmy z daty 1986-02-15 dzień,

SUBSTRING(DataUrodzenia, 9, 2)

co w rezultacie zwróci nam: 15

PS:

w przypadku np. dnia możemy też go wyciągnąć z daty stosując
DAY(DataUrodzenia) czyli ogólnie:

DAY(nazwa_kolumny)

Uwaga w Dacie w takim formacie te myślniki też się liczą!

w **SUBSTRING** liczenie znaków zaczynamy od 1.

a dla np. daty identyfikacja każdego znaku jest następująca:

pozycjonowanie dla przykładowej daty dla MYSQL'a:

Nr. Pozycji od lewej	1	2	3	4	5	6	7	8	9	10
Nasza data w MYSQL	**1**	**9**	**8**	**6**	**-**	**0**	**2**	**-**	**1**	**5**
Nr. Pozycji od prawej	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

jeśli chcemy skorzystać z pozycji od prawej to stosujemy znak **MINUS** w **POZYCJI** (czyli miejscu w substring)

(wtedy **SUBSTRING** liczy nam od prawej długość, którą podamy)

//UWAGA! to nie sprawia, że z MINUSEM SUBSTRING pisze rezultat wycięcia w odwrocie!!!

Np.

SUBSTRING(DataUrodzenia, -5, 3)

co da nam w rezultacie wynik: 6-0

tłumacząc: -5 to **pozycja** od której zaczyna, a 3 to ile **znaków wycina**

ale też

SUBSTRING(DataUrodzenia, 4, 3)

da nam w rezultacie wynik: 6-0

tłumacząc: 4 to **pozycja** od której zaczyna, a 3 to **ile znaków wycina**

Dlaczego -5, 3?

brzmi to tak:

weź z kolumny **DataUrodzenia** wytnij fragment który zaczyna się od **-5**

(jak jest **MINUS** to wtedy **SUBSTRING** idzie w lewą stronę)

o długości 3. Czyli dla naszych numerów pozycji jest to:

6 - 0

-7 -6 -5

zaczynamy w -5 i idziemy w lewo o 3.

(pamiętamy, że pozycje w SUBSTRING liczmy od 1 !!!)

GROUP_CONCAT() –

Funkcja, którą możemy wypisać dane po przecinku.

> Mamy zrobić widok: „Podającą nazwy towarów oddzielone przecinkiem dla każdego numeru faktury”....

Gdy mamy

tabelę o nazwie faktury2015, która wygląda tak:

NrFaktury	DataFaktury	NIP	SposobPlatnosci

tabelę towary2015, która wygląda tak:

IdTowaru	KodTowaru	nazwaTowaru	VAT

tabelę szczegoly2015, która wygląda tak:

NrFaktury	IdTowaru	Cośtam

Przepis:

SELECT *f.NrFaktury*, **GROUP_CONCAT**(*nazwaTowaru*) **FROM** *towary2015* **INNER JOIN** *szczegoly2015* **ON** *szczegoly2015.IdTowaru=towary2015.IdTowaru* **INNER JOIN** faktury2015 f **ON** *f.NrFaktury=szczegoly2015.NrFaktury* **GROUP BY** *NrFaktury*;

Gdzie f to *alias*, nadaliśmy go, ponieważ MYSQL by krzyczał, że nie wie, skąd pobrać kolumnę NrFaktury bo znajduje się ona zarówno w faktury2015 jak i szczegoly2015.

Dodatki:

LIMIT – polecenie, które służy do wydobycia określonej liczby rekordów(wierszy) z interesującej nas tabeli ..

Przykład dla tabeli o nazwie faktury o 4 rekordach, która wygląda tak:

NrFaktury	DataFaktury	Rozliczajacy	Wystawiono
F2/2008	2008-02-12	A.Paser	2016-02-12
F3/2009	2009-05-17	D.Podsiak	2010-09-05
F4/2015	2015-02-01	K.Kopernik	2015-05-05

Cel: *chcemy, aby wyświetliło nam tylko pierwsze dwie faktury(trzeciej nie chcemy bo tak się uparliśmy!). Składnia zapytania wyglądałaby zatem następująco:*

SELECT * **FROM** *faktury* **LIMIT 2**;

Treść naszego zapytania jest następująca:

> *Wyświetl mi wszystkie kolumny z faktur, ale tylko pierwsze dwa rekordy (czyli F2/2008 oraz F3/2009)!*

Kolejny przykład:

SELECT *NrFaktury* **FROM** *faktury* **WHERE** *Rozliczajacy* **LIKE** *'%er%'* **LIMIT 1**;

Co znaczy dosłownie:

> *Wyświetl mi te numery faktury z tabeli faktury gdzie w polu Rozliczajacy jest cos co ma fragment re (obojętnie w jakim miejscu, w środku, na początku czy na końcu, nie ma znaczenia), I wyświetl TYLKO jeden rekord! (Zasługa naszego LIMIT 1)*