Podstawy języka SQL

Wykorzystywanie baz danych SQL to niemal standard przemysłowy wszędzie tam, gdzie występuje potrzeba zgromadzenia i przetwarzania różnorodnych danych. W chwili obecnej niemal cała informatyka bazuje na wiedzy, jaką czerpie z gromadzenia i przetwarzania danych zarówno od twórców oprogramowania, jak i użytkowników tegoż (kwestią sporną jest świadomość tego procederu u użytkowników). Gromadzone dane muszą być odpowiednio przechowywane, jednocześnie dostęp do nich musi często przebiegać nieprzerwanie i bezproblemowo nawet w przypadku, gdy dostępu żąda kilkunastu/kilkuset użytkowników naraz. W tej sytuacji najlepszym rozwiązaniem są serwery baz danych SQL, czyli oprogramowanie pozwalające na tworzenie i przechowywanie wielu baz danych, które mogą należeć do różnych użytkowników czy programów.

Sama baza danych to nic innego jak pojedynczy plik (czasami z towarzyszącymi plikami dodatkowymi, takimi jak plik ze zdarzeniami czy też z notowanymi transakcjami), odpowiednio sformatowany i poukładany tak, by jak najefektywniej zwracać wyniki zapytania o konkretne dane/wstawiać nowe dane w odpowiednie miejsce.

INFORMACJA: Niektórzy twierdzą, że systemy nie wykorzystujące SQL działają lepiej i są bardziej przejrzyste. Twierdzenie to będzie prawdziwe w przypadku mniejszych projektów, w których danych będzie ni więcej niż kilkadziesiąt tysięcy. W chwili kiedy liczba danych zaczyna się drastycznie zwiększać tylko odpowiednio optymalizowane bazy danych w połączeniu z dobrze zoptymalizowanymi serwerami baz danych są w stanie zapewnić odpowiedni poziom dostępu i wymiany danych.

Zalety baz danych:
- sprawny i wydajny mechanizm przeszukiwania
- indeksowanie zawartości jednoznacznych danych (tzw. klucze indeksowe)
- możliwość łączenia poszczególnych tabel poprzez tzw. klucze obce (podstawowy mechanizm weryfikacji danych)
- proste polecenia dodawana/edytowania/usuwania danych
- kompresowanie danych (zarówno tekstowych jak i binarnych – w tym obrazów)
- efektywny system operacji na łańcuchach znakowych (porównywanie, przeszukiwanie, łączenie oraz rozdzielanie)
- wiele innych...

Poważną wadą SQL jest brak ujednolicenia. Ogólnie rzecz ujmując standard SQL definiuje jednolitą składnię poleceń, jednak pozostawia niektóre kwestie niewyjaśnione. One to stanowią z kolei podstawę do tego, że niemal każdy system bazodanowy SQL posiada inne podejście do takich kwestii, jak domyślne ograniczniki nazw tabel i pól, implementacji (bądź braku) niektórych zapytań (klauzula ON),

W niniejszym materiale przedstawiona zostanie podstawowa składnia języka SQL, działająca w większości baz danych opartych na standardzie SQL. Rozwiązania charakterystyczne dla danej dystrybucji SQL będą omawiane szerzej w innych materiałach.

Podstawowa składnia języka SQL

Zapytania SQL dzielą się na trzy typy: język definiowania danych (DDL – Data Definition Language), język kontrolowania danych (DCL – Data Control Language) oraz język język operacji na danych (DML – Data Manipulation Language). W pierwszej kolejności przedstawione zostaną zapytania DDL z racji tego, że bez definicji baz danych oraz tabel nie za wiele można by zdziałać z przechowywaniem danych.

1. DDL – język definicji danych

Każda z operacji w tym typie językowym rozpoczyna się od słowa kluczowego CREATE (utwórz), DROP (wyrzuć) bądź ALTER (zmiana).

a) tworzenie baz danych:

CREATE DATABASE <nazwa bazy>;

powyższa komenda utworzy dla nas nową, pustą bazę danych na serwerze SQL. W zależności od dostawcy zapytanie posiada dodatkowe atrybuty, które będą omówione wraz z konkretnym rozwiązaniem. Przykład użycia:

CREATE DATABASE baza_testowa;

b) usuwanie bazy danych:

DROP DATABASE <nazwa bazy>;

polecenie odwrotne do poprzedniego. Usuwa bazę danych o zadanej nazwie. Przykład użycia:

DROP DATABASE baza_testowa;

c) tworzenie nowej tabeli w ramach bazy danych:

CREATE TABLE <nazwa tabeli> (<nazwa pola> [typ] [atrybuty pola], <drugie pole> …, … , <n-te pole>;

polecenie tworzy nam nową, pustą bazę danych z podanymi przez nas polami (kolumny danych). Każde pole musi mieć zdefiniowany typ (typy będą omówione później). Dodatkowo każde pole może posiadać atrybuty, od których zależy jego wstępne wypełnienie, zachowanie na określonych typ danych itp. Polecenie w przedstawionej postaci jest najbardziej podstawowym, działającym we wszystkich dostępnych serwerach SQL.
PROSZĘ ZAUWAŻYĆ, że możemy dodawać dowolną ilość nowych pól w ramach pojedynczej tabeli. Przykład utworzenia tabeli:

CREATE TABLE tabela_test (id INT(8), nazwa VARCHAR(10), plec TINYINT, komentarz TEXT);

d) usunięcie wskazanej tabeli z bazy danych:

DROP TABLE <nazwa tabeli>

Polecenie usuwa z aktualnej bazy danych wskazaną tabelę. Wszystkie dane z tabeli zostaną bezpowrotnie utracone. Przykład użycia:

DROP TABLE tabela_test;

e) modyfikacja tabeli poprzez dodanie nowego elementu

ALTER TABLE <nazwa tabeli> ADD [element] <nazwa pola> [typ] [atrybuty];

Polecenie pozwala na dodanie nowego elementu do istniejącej tabeli z określonym typem oraz atrybutami. Nowa kolumna zawsze pojawi się na końcu tabeli! Pozostałych obiektów (jak indeksy, klucze obce i główne) to nie dotyczy – one nie pojawiają się jawnie w standardowym widoku tabeli. Przykład użycia (dodanie nowej koluny):

ALTER TABLE tabela_test ADD COLUMN nowa_kolumna TEXT;

UWAGA! W przypadku dodawania innych elementów (np. kluczy obcych) składnia polecenia będzie nieco inna! Niestety nie ma jednolitego standardu polecenia ALTER (jedynie przytoczony jest taki sam dla wszystkich baz danych) dlatego zapytanie ALTER będzie szczegółowo omawiane dla każdego z serwerów oddzielnie.

f) modyfikacja tabeli poprzez usunięcie pola

ALTER TABLE <nazwa tabeli> DROP [modyfikowany element] <nazwa pola>;

Polecenie jest podobne do poprzedniego. Pozwala ono jednak na usunięcie wskazanego elementu z bazy danych. Jeżeli elementem jest kolumna to zostanie usunięta ona oraz wszystkie wartości przypisanej do niej w kolejnych wierszach! W przypadku pozostałych elementów efekt usuwania zależny jest od ich powiązań (np. klucz główny nie będzie mógł zostać usunięty bez wcześniejszego odłączenia relacji z nim związanej). Przykład użycia:

ALTER TABLE tabela_test DROP COLUMN nowa_kolumna;

2. DCL – język kontroli danych

Do tej kategorii zaliczają się jedynie dwa zapytania – GRANT oraz REVOKE. Pozwalają one na nadawanie bądź odbieranie uprawnień do działań na wskazanej tabeli wymienionym użytkownikom bazy danych.

a) nadanie uprawnień do wykonywania określonych operacji z bazą danych:

GRANT <uprawnienia> ON <nazwa tabeli> TO <nazwy uzytkownikow, oddzielone przecinkami>

Tak jak zostało wspomniane we wstępie, polecenie GRANT pozwala na nadanie odpowiednich praw, takich jak dodawanie danych, aktualizacja czy usuwanie we wskazanej bazie danych/tabeli dla wskazanych użytkowników (bądź pojedynczego użytkownika). Poprzez wprowadzanie ograniczeń dla użytkowników oraz roztropną politykę przywilejów dane w bazie zyskują na bezpieczeństwie. Przykład użycia:

GRANT SELECT,UPDATE ON moja_tabela TO uzytkownik1

b) odmawianie uprawnień do bazy danych/tabeli

REVOKE <uprawnienia> ON <nazwa tabeli> TO <nazwy uzytkownikow,oddzielone przecinkami>

Operacja działa odwrotnie do wcześniej opisywanej. Wskazani przez nas użytkownicy (bądź pojedynczy użytkownik) nie będą posiadali dostępu do określonego zasobu w bazie danych. Przykład użycia:

REVOKE UPDATE ON moja_tabela TO uzytkownik1

3. DML – język przetwarzania danych

Trzecia, najczęściej używana grupa zapytań SQL. O ile nadawanie uprawnień czy też tworzenie nowych baz danych/tabel nie jest wykonywanie na co dzień, o tyle zapytania UPDATE czy INSERT dla bazy danych stanowią chleb powszedni. Jako przykład niech posłużą chociażby bazy danych gier internetowych, for dyskusyjnych czy też bazy kont bankowych. W wyżej wymienionych projektach nikt nie przebudowuje bazy co parę sekund. Jednak zapytań przetwarzających dane wykonuje się nawet parę milionów na dzień!
Poniżej zostanie zaprezentowana podstawowa składnia poleceń; szczegółowe struktury zapytań, w tym charakterystyczne dla wybranych platform, zostaną zaprezentowane w chwili omawiania konkretnych rozwiązań.

a) dodawanie danych do bazy danych

INSERT INTO <nazwa tabeli> (<kolumna1>, <kolumna2>, <kolumna3>) VALUES ([wartosc1], [wartosc2], [wartosc3])'

Zapytanie dodaje do wskazanej tabeli pod wskazane w pierwszym nawiasie kolumny wartości zapisane w drugim nawiasie. Należy pamiętać, że nazwy kolumn można podawać w dowolnej kolejności (nie musi być taka jak rzeczywista kolejność w bazie danych) jednak trzeba pamiętać, że w drugim nawiasie trzeba zachować kolejność z pierwszego nawiasu. Standard dopuszcza dodawanie wartości poprzez takie zapytanie:

INSERT INTO <nazwa tabeli> VALUES ([wartosc1], [wartosc2], [wartosc3], …, [wartoscN])

jednak formę tę stosuje się aktualnie znacznie rzadziej. Po pierwsze wymusza ona podawanie WSZYSTKICH wartości jakie mają znajdować się w tabeli. W przypadku chęci pominięcia niektórych wartości musielibyśmy podawać wartość NULL/'' bądź inne domyślne wartości, które baza przyjęłaby sama (w przypadku nie podania ich). Po drugie wartości muszą być identycznej kolejności, w jakieś kolumny występują w tabeli.

Przykład dodawania nowego wiersza danych:

INSERT INTO moja_tabela (imię, nazwisko, pesel, plec) VALUES ('Jan', 'Dzban', '88888888888', 0);

Proszę zauważyć, że wartości tekstowe MUSZĄ zostać objęte apostrofem. Z kolei wartości liczbowe NIE MOGĄ zostać objęte apostrofem (aczkolwiek niektóre implementacje SQL automatycznie zamieniają wartości na liczbowe).

b) aktualizacja danych w tabeli

UPDATE <nazwa tabeli> SET <kolumna1> = [wartosc1], <kolumna2> = [wartosc2], … , <kolumnaN> = [wartoscN] {WHERE <kolumnaM> <operator> [wartosc]}

Zapytanie to pozwala na podmianę danych we wskazanych przez nas kolumnach. Sama konstrukcja polecenia jest nieco bardziej 'skomplikowana' niż przedstawianych dotychczas.
Domyślnie polecenie działa na wszystkie rekordy (wiersze danych) co znaczy, że jeżeli po SET wpiszemy imię = 'Maria' wszyscy dodani użytkownicy otrzymają imię Maria! Jak łatwo się domyślić, nie jest to pożądane zachowanie. Częściej (raczej zawsze) będziemy potrzebować zmienić dane w jednym wierszu bądź tylko w określonych wierszach. Stąd podany w klamrach człon WHERE. Pozwala on bowiem na dodanie warunków jakie muszą spełnić wiersze, by zmiany

w kolumnie/kolumnach doszły do skutku. Same warunki buduje się w oparciu o operatory logiczne (oraz operatory łączące), które to zostaną opisane w następnym podrozdziale (oraz w materiałach dedykowanych dla konkretnych serwerów SQL). Należy przy tym pamiętać, że ilość argumentów może być większa (będzie to omawiane przy okazji konkretnych serwerów SQL). Przykład polecenia zmieniającego imię na Janusz oraz wartość płeć na 0 w każdym wierszu, w którym nazwisko to Iksiński:

UPDATE moja_tabela SET imię = 'Janusz', plec = 0 WHERE nazwisko = 'Iksiński';

c) usunięcie danych z tabeli

DELETE FROM <nazwa tabeli> {WHERE <kolumnaM> <operator> [wartosc]};

Polecenie pozwala na usunięcie wszystkich danych ze wskazanej tabeli. W celu usunięcia jedynie wskazanych rekordów należy dodać klauzulę WHERE, w której, podobnie do zapytania UPDATE, podajemy warunki, których spełnienie będzie wymagane do usunięcia wiersza danych. Przykład polecenia:

DELETE FROM moja_tabela WHERE plec = 0;

Zostaną usunięte wszystkie wiersze danych, w których pod kolumną płeć widnieje wartość 0.

TRANSAKCJE W SQL

Komputery, cokolwiek by sobie nie myśleć, nie są wielozadaniowe w takim stopniu, w jakim sobie o nich myślimy. W zasadzie wszystkie dostępne mikropocesory (nie zważając na ich rodzinę czy też zastosowanie) w danym momencie mogą przetwarzać tylko jeden rozkaz. Oczywiście aktualnie buduje się procesory wielordzeniowe (1 rdzeń = 1 zadanie, czyli np. czterordzeniowy procesor może pracować w jednej chwili nad 4 niezależnymi zadaniami), procesory mogące 'jednocześnie' zajmować się dwoma zadaniami na jednym rdzeniu (tzw. potokowość i hiperpotokowość przy 4 wątkach na rdzeń); aczkolwiek rozwiązanie to nie jest do końca równoległe gdyż w danej chwili rdzeń może pobierać dane/wysyłać wynik tylko dla jednego zadania, tak więc zadania te uruchamiają się sekwencyjnie, pseudorównolegle. Niektóre procesory (jak np. rodzina Itanium) pozwalają na składanie i wysyłanie rozkazów do procesora w paczkach (np. 3 bądź 6 rozkazów w ramach jednego przesyłu) dzięki czemu zadania wykonują faktycznie równolegle (aczkolwiek udział tych procesorów na rynku jest niewielki, nawet na serwerach).

W tej chwili można by zapytać – co ma przetwarzanie danych przez procesor do baz danych? Otóż wiele. Skoro procesor nie jest w stanie wykonywać wielu zadań równocześnie (w najlepszym wypadku może wykonywać do 8 zadań – w przypadku 8 rdzeń/2 zadań na rdzeń) to jakim sposobem z bazy danych może korzystać jednocześnie wielu użytkowników? O ile sam odczyt nie stanowi problemu (dla nas opóźnienia w przesyle danych przez komputer są niemal niezauważalne, a sam plik może być czytany przez kilka wątków jednocześnie) o tyle zapis stanowi wiele problemów. Pomijając kwestię jednozadaniowości należy tutaj także wziąć pod uwagę problem dostępu do zasobów oraz ograniczeń technologicznych. Zarówno dane tekstowe jak i binarne czytane i zapisywane są przez komputer z danego miejsca (pliku) potokowo. Zapis binarny ma tę przewagę, że pozwala na dotarcie do miejsca zapisu znacznie szybciej – program czytający taki plik może mieć jego 'mapę' dzięki której szybko i skutecznie może przemieścić się do pożądanego bajtu, od którego ma zacząć przeglądać dane. W przypadku zapisu pojawia się problem – to, co aktualnie jest zapisywane może być jednocześnie odczytywane. Ponadto istnieje szansa (im więcej użytkowników tym szansa jest większa), że dane próbuje zapisać/nadpisać w danym czasie więcej niż jeden użytkownik. Wtedy dane mogą ulec zniszczeniu, przeinaczeniu lub, w skrajnych

wypadkach, zostać zniszczone.

Standardowo ochroną przed tymi zdarzeniami jest stosowanie blokady zasobu bądź używanie specjalnych systemów tzw. semaforów. Dzięki temu zasób, zapisywany przez jeden proces systemowy, nie może zostać zapisany przez inny (następny chętny musi poczekać na odblokowanie zasobu). Rozwiązanie proste i skuteczne (stosowane od niepamiętnych czasów). Co jednak z odczytem? Odczyt może być w zasadzie nieblokowany – i tak zadziała. Kwestią tego jest w jakiej postaci pobierze zapisane dane – nowe, stare czy też losowe (np. zniekształcone bo uchwycone w chwili zapisu)? Tutaj mamy kolejny problem trapiący tradycyjne przechowywanie danych.

Teraz dopiero, po uwzględnieniu przedstawionych faktów, w pełni można docenić system transakcji wbudowany w SQL, który jest jedną z jego największych zalet. Nie chroni on bowiem jedynie operacji zapisu w bazie lecz także poprawności odczytu z bazy.

Jego działanie jest bardzo proste. Każda z funkcji ingerująca w dane bazy (UPDATE, INSERT, DELETE, DROP) automatycznie obejmowana jest operacją transakcji, czyli od chwili jej rozpoczęcia do czasu jej zakończenia żadna inna operacja na wskazanych wierszach tabeli nie może zostać dokonana. Z kolei odczyt, w zależności do konfiguracji bazy/serwera może być dostępny, jednak czytane dane będą z wersji zanim rozpoczęła się transakcja. Po jej zakończeniu dostępne będą nowe dane. Ponadto transakcje pozwalają na chronologiczne uporządkowanie dodawania/uporządkowania przesyłanych informacji. Klient wysyła bowiem informacje i zupełnie nie interesuje go kiedy te dane zostaną uzupełnione (chociaż dla nas dane te pojawiają się niemal natychmiast). To baza przechowuje kolejne polecenia w kolejce (tzw. buforze) i wykonuje je w kolejności otrzymania.

Oczywiście nie zawsze mile widziane jest działanie automatycznej akceptacji wszystkich nadchodzących do naszej bazy danych. Dlaczego? Otóż przy sporym ruchu z i do bazy może okazać się, że transakcje nie są odpowiednio wydajne. Rozwiązaniem tego jest możliwość ręcznego sterowania transakcjami. Większość serwerów pozwala wręcz na wyłączenie automatycznych transakcji i zmuszenie projektantów oprogramowania/operatorów bazy do ręcznej akceptacji transakcji.

Ręczne transakcje działają w następujący sposób. Jeżeli wysłane zostanie zapytanie o rozpoczęcie transakcji to baza, jeżeli ma włączoną domyślną akceptację każdej zmiany, wyłączy ją dla wskazanej sesji (naszej). Od tego momentu wszystkie zmiany, jakich będziemy dokonywać w bazie będą widoczne jedynie dla nas; baza będzie je zapisywać w czymś na kształt brudnopisu. W tym momencie, jeżeli stwierdzimy iż popełniliśmy błąd, możemy wysłać zapytanie o cofnięcie naszych zmian; wszystko co dotychczas zrobiliśmy zostanie bezpowrotnie utracone! Podczas dłuższych transakcji (wielu operacji na danych) możemy tworzyć punkty kontrolne (chwilowy zapis); wtedy cofnięcie transakcji cofnie nas do punktu kontrolnego nie zaś pozbawi nas całej pracy. Jeżeli natomiast będziemy już zadowoleni z naszej pracy możemy wydać polecenie zaakceptowania zmian. Baza dokona zablokowania odpowiedniej ilości tabel/pól w tabelach i podmieni ich zawartość.

Trzeba też pamiętać, że wiele serwerów baz danych zapisuje każdą transakcję w specjalnym, dodatkowym pliku zdarzeń. Dzięki temu nawet przypadkowe usunięcie czy też wadliwa podmiana danych jest możliwa do cofnięcia. Wadą takich dzienników jest natomiast zbyt duży rozmiar samej bazy danych. Dlatego na administratorze serwera spoczywa obowiązek odpowiedniego zarządzania serwerem (np. robieniem cyklicznego zapisu danych oraz usuwania pliku zdarzeń co określony czas).

UWAGA! W niniejszym materiale specjalnie nie zostały przytoczone polecenia transakcyjne dla SQL. Niestety rozwiązania bazodanowe różnią się w składni tychże poleceń w związku z czym same zapytania zostaną zaprezentowane dla konkretnych rozwiązań.

OPERATORY BAZ DANYCH

Język SQL, tak jak pozostałe języki, posiada zaimplementowaną obsługę warunków (a także funkcji, procedur czy pętli). Niestety, tak jak w przypadku transakcji, implementacja operatorów jest różna w zależności od wybranego serwera SQL. Dlatego w niniejszym materiale wymienione zostaną jedynie te wspólne we wszystkich językach, natomiast dodatkowe operatory będą wymienione i krótko omówione w materiałach dla poszczególnych rozwiązań.

Ogólna zasada działania wszystkich warunków jest następująca: wybieramy wskazaną wartość (przykładowo daną kolumnę z bazy danych), po czym przyrównujemy ją do wskazanej przez nas wartości (wartość ta może być zarówno podana przez nas jak również może być pobrana z innego miejsca bazy danych). Przyrównanie jednej wartości do drugiej może być realizowane poprzez operatory; operatorem jest np. znak równości (oznacza to, że porównywane wartości muszą być równe), znak mniejszości (wartość a musi być mniejsza od wartości b) czy też (w przypadku baz danych) słowo LIKE (wartość b musi w jakimś stopniu zawierać się w wartości a). Innymi słowy wynikiem warunku musi być ZAWSZE PRAWDA (wyrażenie zostało spełnione nawet jako zaprzeczenie). Należy pamiętać, że warunki można ze sobą łączyć.

Poniżej znajdują się podstawowe operatory wykorzystywane w zapytaniach SQL. Dodatkowe, specyficzne operatory będą wymienione i scharakteryzowane przy okazji opisu konkretnych rozwiązań. Przykłady użycia będą pokazane na końcu tego materiału.

a) AND – jeden z najczęściej stosowanych operatorów w zapytaniach wielokryterialnych. Używany jako spójnik 'oraz' – oba warunki połączone przez niego muszą zakończyć się wynikiem prawdy.

b) OR – drugi (na równi) stosowany operator w zapytaniach. Używany jest jako spójnik 'lub' – jeden z dwóch połączonych przez niego warunków musi zakończyć się wynikiem prawdy.

c) = – operator mówiący, że wartość badana w warunku MUSI BYĆ RÓWNA do wartości z nią zestawionej. W przypadku ciągów znakowych (najczęściej porównywane) wartości w nich zapisane porównywane są bajt po bajcie.

d) < – operator, który ustala iż wartość badana w warunku musi BYĆ MNIEJSZA od wartości z nią zestawionej. Operator wykorzystywany głównie w przypadku wartości liczbowych (dla tekstu, ze względu na bajtowy sposób porównań, jest mało efektywny).

e) > – wartość badana musi BYĆ WIĘKSZA od wartości z nią zestawionej. Zastosowanie podobne do operatora mniejszości

f) <> – operator, którego pozytywny wynik może zapewnić wartość badana RÓŻNA od wartości z nią zestawionej. Operator ten na równi wybierany jest do wartości liczbowych oraz ciągów znakowych.

g) LIKE – specyficzny dla SQL operator (w innych językach programowania zastępują go wyrażenia regularne) jedynie dla wartości znakowych, dzięki któremu można pozycje podobne do zestawionej (stąd jego nazwa). Znaki, które nie mają dla nas znaczenia zastępuje się procentem (%; zastępuje 0 lub więcej znaków). Przykładowo zastawiona wartość '%ów' będzie pasowała zarówno do 'kotów', jak i 'butów' czy 'loków'. Jeżeli byśmy doprecyzowali wzór na 'k%ów' zapewne zawęzilibyśmy poszukiwania do 'kotów' oraz także 'krów' czy 'karków'.

h) BETWEEN – kolejny specyficzny dla SQL operator. Pozwala on na podanie dwóch wartości, np. 3 i 10. W tym momencie taki warunek spełni każda wartości będąca pomiędzy wartościami 3 i 10 (czyli np. 5,9, 4 oraz 5.76, 4.2 itd.). Trzeba pamiętać, że najefektywniej warunek działa dla wartości liczbowych oraz do porównywania dat.

i) IN – ostatni opisywany, również specyficzny dla SQL, operator. Pozwala na podanie w nawiasie zbioru wartości (mogą być dowolnego typu). Jeżeli wartość porównywana będzie taka sama jak ta ze zbioru to warunek będzie rozpatrzony pozytywne.

PRZYKŁADY UŻYCIA WARUNKÓW:

a) przyrównania
WHERE imie = 'Justyna' //wartość kolumny imie musi zawierać frazę 'Justyna'
WHERE imie = 'Marek' OR imie = 'Mateusz' //wartość kolumny imie musi zawierać frazę 'Marek' lub 'Mateusz'
WHERE wiek = 14 //wartość kolumny wiek musi mieć wartość 14
WHERE wiek = 30 AND imie = 'Józef' //pobrane zostaną wartości z wierszy, w których osoby mają na imię Józef oraz są w wieku 30

b) mniejszości
WHERE suma < 140//wybrane zostaną wartości,w których wartość kolumny suma jest niższa od 140 (ale nie równe)

c) większości
WHERE suma > 140//wybrane zostaną wartości, w których wartość kolumny suma są większe od 140 (ale nie równe)
WHERE suma < 140 AND (ilosc > 20 OR czas_realizacji < 3)//wybrane zostaną wiersze, w których wartość sumy jest mniejsza od 140 oraz ilosc jest większa od 20 lub czas_realizacji mniejszy od 3. Proszę zwrócić uwagę na nawias – ogranicza od działanie spójnika OR na dwa warunki oraz powoduje, że to wynik OR będzie łączony spójnikiem oraz z wynikiem warunku suma

d) różne
WHERE nazwisko <> 'Nowak'//wybierze wszystkie wiersze z bazy, w których kolumna nazwisko nie będzie zawierała frazy 'Nowak'
WHERE nazwisko <> 'Nowak' AND nazwisko LIKE '%ski'//jak poprzednio lecz prócz tego zostaną wyświetlone tylko rekordy, w których nazwisko kończy się na -ski

e) LIKE
WHERE imie LIKE '%an%'//wybierze rekordy, które zawierają imiona takie jak Lucjan, Marianna, Anna, Marian
WHERE kraj LIKE '%stan' AND kraj LIKE 'T%'//wybierze takie kraje jak Turkmenistan, Tadżykistan (inaczej wybrałby także pozostałe, z końcówką na stan)

f) BETWEEN
WHERE wiek BETWEEN 20 AND 30//wybierze wszystkich, którzy mają wiek między 20 a 30 rokiem życie
WHERE nazwa BETWEEN 'C' AND 'F'//wybierze nazwy rozpoczynające się od liter C, E, F (bądź E w przypadku niektórych baz danych)

g) IN
WHERE wiek IN (20,25,28)//wybierze osoby, które mają 20, 25 bądź 28 lat (inne wartości zostaną pominięte)