MICRO KURS

Podstawy SQLa

Wprowadzenie do SQL

- **SQL - Structured Query Language -strukturalny język zapytań**
- Światowy standard przeznaczony do definiowania, operowania i sterowania danymi w relacyjnych bazach danych
- Powstał w firmie IBM pod koniec lat 70-tych
- Występuje w produktach większości firm produkujących oprogramowanie do zarządzania bazami danych
- Polecenia **SQL** mają postać podobną do zdań w języku angielskim
- Pomimo prób standaryzacji istnieje szereg różnych dialektów **SQL**
- **SQL** używany jest jako standardowe narzędzie umożliwiające dostęp do danych w różnych środowiskach, z różnym sprzętem komputerowym i różnymi systemami operacyjnymi
- Język **SQL** jest niewrażliwy na rejestr czcionki, czyli wielkie i małe litery nie są rozróżniane

Wprowadzenie do SQL

- **SQL** zapewnia obsługę:
  - **zapytań** - wyszukiwanie danych w bazie
  - **operowania danymi** - wstawianie, modyfikowanie i usuwanie
  - **definiowania danych** - dodawanie do bazy danych nowych tabel
  - **sterowania danymi** - ochrona przed niepowołanym dostępem
- Użytkownik określa operacje jakie mają być wykonane nie wnikając w to, jak mają być wykonane
- Najprostsza postać zapytań w **SQL** służy do wybierania rekordów pewnej tabeli, które spełniają określony w zapytaniu warunek
- Taki typ zapytania stanowi odpowiednik operatora selekcji w algebrze relacyjnej
- Takie najprostsze zapytanie, jak zresztą prawie wszystkie zapytania w tym języku, konstruuje się za pomocą trzech słów kluczowych: **SELECT, FROM** i **WHERE**

Podstawowe klauzule w SQL

```
SELECT nazwy_kolumn
FROM nazwa_tabeli
WHERE warunek;
```

- Pozwalają na wybranie z tabeli określonych kolumn i rekordów spełniających ustalone warunki czyli pozwalają na realizację rzutowania i selekcji
- Warunek formułowany jest jako złożone wyrażenie porównania
- Przykładowa tabela o nazwie **NAZWISKA** zawiera kolumny:
  - NUMER
  - IMIE
  - NAZWISKO
  - STANOWISKO
  - PENSJA
  - MIASTO

Klauzule SELECT i FROM

- **SELECT** - podstawowa klauzula **SQL** - używana do wyszukiwania danych w tabeli
- Występuje wraz z klauzulą **FROM**

```
SELECT *
FROM nazwa-tabeli;
```

- Gwiazdka oznacza, że należy wyszukać wszystkie kolumny tabeli
- Jest to przykład instrukcji wybierającej całą tabelę
- W klauzuli **SELECT** zostają określone nazwy kolumn, których wartości, z rekordów spełniających warunek zapytania (formułowany przy pomocy klauzuli **WHERE**), są dołączane do odpowiedzi
- Klauzula **FROM** służy do określenia tabeli, której dotyczy zapytanie

Klauzula WHERE

- W klauzuli **WHERE** formułuje się warunek, który odpowiada warunkowi wyboru (selekcji) w algebrze relacyjnej i który określa ograniczenia, jakie mają spełniać rekordy, aby zostać wybrane w danym zapytaniu
- Jeżeli rekord spełnia te ograniczenia to zostaje dołączony do tabeli wynikowej
- Postać zapytania

```
SELECT *
FROM nazwa-tabeli
WHERE warunek;
```

- Klauzula **WHERE** pozwala na wybranie z tabeli tych wierszy, które spełniają określone warunki

```
SELECT *
FROM NAZWISKA
WHERE STANOWISKO = 'URZEDNIK';
```

- Dla podanego przykładu z tabeli zostaną wybrane tylko te rekordy, w których w polu **STANOWISKO** jest wpisane **'URZEDNIK'**

Formułowanie warunku

- Po słowie kluczowym **WHERE** występuje wyrażenie warunkowe
- Do zapisu porównywania wartości w języku SQL służy sześć operatorów:
  - równy **=**
  - nierówny **<>**
  - mniejszy **<**
  - większy **>**
  - mniejszy lub równy **<=**
  - większy lub równy **>=**
- W wyrażeniu mogą występować stałe oraz nazwy kolumn tabel wymienionych w klauzuli **FROM**
- Dla wartości numerycznych można budować wyrażenia arytmetyczne korzystając z operatorów **+ - * /** i nawiasów **( )**
- Stałe tekstowe w **SQL** są ujmowane w pojedyncze cudzysłowy

'Przykład tekstu'

Formułowanie warunku

- W wyniku porównania powstaje wartość logiczna **TRUE** (prawda) lub **FALSE** (fałsz)
- Wartości logiczne można łączyć w wyrażenia logiczne za pomocą operatorów logicznych **AND, OR i NOT**
- Priorytet operatorów wykorzystywanych w budowie wyrażeń:
  operatory porównania, NOT, AND, OR
- Porównywanie tekstów - dwa teksty są równe, jeśli występują w nich kolejno te same znaki
- Przy teście „nierównościowym” tekstów, tzn. przy wykonywaniu porównań takich jak < lub >=, o wartości porównania decyduje, czy kolejne znaki z tekstu z lewej strony są alfabetycznie wcześniejsze, czy dalsze w stosunku do znaków z tekstu umieszczonego po prawej stronie wyrażenia
- Przykłady

```
Adamski > Adamowicz
Adam < Adamowicz
```

Formułowanie warunku

- Wartości NULL nie podlegają żadnym operacjom porównania, gdyż jest ona traktowana jako wartość nieznana
- SQL umożliwia testowanie pól w poszukiwaniu wartości NULL
- Użycie w klauzuli **WHERE** zwrotu **IS NULL** jest wykorzystywane do sprawdzania czy pole zawiera tę wartość
- Zamiast standardowego operatora porównania pojawia się słowo **IS**
- Słowo NULL nie jest zawarte w cudzysłowie
- Można dokonać przeszukania danych w celu wybrania obiektów posiadających wartości
- W tym celu używa się wyrażenia **IS NOT NULL**

Wykonywanie obliczen na danych

- Jezyk SQL pozwala na wykonywanie obliczeń na danych i pokazywanie ich wyników w postaci wykonanych zapytan
- Wykonanie obliczen polega na zastapieniu pozycji z listy nazw kolumn (w klauzuli SELECT) przez odpowiednie wyrazenia
- Wyrazenie nie musi koniecznie zawierac nazw kolumn, mozna uzywac tylko liczb, albo wyrazen algebraicznych lub lancuchów znaków
- Postać polecenia:

```
SELECT 'Tekst objasniajacy', Stanowisko, Pensja*2
FROM NAZWISKA
WHERE Pensja >= 900;
```

- Wynik zapytania

Wyr1	Stanowisko	Wyr2
Tekst objasniajacy	urzednik	1 800,00 zl
Tekst objasniajacy	kierownik	6 000,00 zl
Tekst objasniajacy	urzednik	2 200,00 zl
Tekst objasniajacy	ksiegowy	4 000,00 zl

Uzycie slowa kluczowego AS

- W zapytaniu mozna uzyc slowa kluczowego **AS**, aby przypisac nazwy kolumnom i wyrazeniom (zamiast standardowych Wyr1, Wyr2)
- Nazwy te poprawiają czytelnosc danych zwracanych przez zapytanie oraz pozwalają odwolac się do nich przez nazwe
- Skladnia polecenia wyglada nastepujaco:

```
SELECT 'Tekst objasniajacy' AS KOMENTARZ,
Stanowisko, Pensja*2 AS PODWYZKA
FROM NAZWISKA
WHERE Pensja >= 900;
```

- Wynik zapytania

KOMENTARZ	Stanowisko	PODWYŻKA
Tekst objasniajacy	urzednik	1 800,00 zl
Tekst objasniajacy	kierownik	6 000,00 zl
Tekst objasniajacy	urzednik	2 200,00 zl
Tekst objasniajacy	ksiegowy	4 000,00 zl

Wykonywanie obliczen w klauzuli WHERE

- Podobnie jak mozna wykonywac obliczenia na danych wybranych z tabeli, mozna równiez wykonywac obliczenia w klauzuli WHERE, aby pomóc w filtrowaniu rekordów
- Przykład polecenia

```
SELECT 'Tekst objasniajacy' AS KOMENTARZ,
Stanowisko, Pensja*2 AS PODWYZKA
FROM NAZWISKA
WHERE Pensja*2 >= 2*900;
```

- Jest oczywiste, że wyniki polecenia będą takie same jak poprzednio
- Cecha charakterystyczna relacyjnych baz danych jest to, ze kolejnosc kolumn i wierszy nie jest istotna - nie sa one traktowane sekwencyjnie
- Mozna wybierac rekordy z bazy danych w dowolnym porzadku
- Domyslnie pojawiaja sie w kolejnosci, w jakiej byly wprowadzone
- Jednak czesto przegladajac rekordy chcemy te kolejnosc okreslic, np. wzgledem zawartosci jednej z kolumn

Sortowanie wyników zapytań

- Klauzula **ORDER BY** jest wykorzystywana do sortowania wyników
- Wyniki zapytania beda uporzadkowane wzgledem zawartosci kolumny (lub kolumn), które okreslimy w klauzuli **ORDER BY**
- Sortowanie mozna przeprowadzic zarówno alfabetycznie jak i wzgledem wartosci numerycznych oraz kolumn zawierających dane w formacie **Date**
- Kolejnosc kolumn nie zalezy od kolumny uzywanej do sortowania wyników zapytan - kolumny pozostaja zawsze w tym samym porzadku, bez wzgledu na kolumne, której uzywamy w klauzuli **ORDER BY**
- Dodanie do poprzedniego polecenia:

```
ORDER BY Stanowisko;
```

- spowoduje, że wyniki zostaną posortowane według kolumny Stanowisko (w porządku rosnącym)
- Wyniki zapytan moga byc posortowane zarówno rosnaco (opcja domyślna), jak i malejaco
- Dla sortowania malejacego, uzywamy w klauzuli **ORDER BY** slowa kluczowego **DESC** (dla rosnącego słowa **ASC** – normalnie jest pomijane)

Operatory logiczne w klauzuli WHERE

- Operacje wykonywane w klauzuli WHERE podlegaja zasadom logiki boolowskiej - wynik przyjmuje zawsze jedna z wartosci: prawda lub falsz
- W przypadku, gdy wynik wyrazenia to prawda, wiersz jest wybierany, w przeciwnym przypadku – pomijany
- Operator **AND** zwraca wynik prawda, gdy wyrazenia po obu stronach operatora sa prawdziwe - jezeli choc jedno z nich jest nieprawdziwe, wtedy cale wyrazenie zwraca jako wynik wartosc falsz
- Operator **OR** zwraca wynik prawda, gdy jedno z wyrazen po prawej lub po lewej stronie operatora jest prawdziwe - gdy oba wyrazenia sa prawdziwe, wynik tez przyjmuje wartosc prawda
- Operatora **NOT** uzywamy do zaprzeczenia wartosci wyrazenia
- Wielokrotne operatory logiczne moga byc wykorzystywane do utworzenia zlozonych instrukcji **WHERE**, w których wykorzystywanych jest kilka wyrazen jednoczesnie
- Formułując takie wyrażenia należy pamiętać o priorytecie operatorów w celu zapewnienia poprawności obliczenia wartości wyrażenia

Klauzula IN

- Wzrost zlozonosci zapytan powoduje trudności z ustaleniem kolejnosci wykonywanych operacji – konieczne staje się stosowanie nawiasów wykorzystywanych do grupowania wyrazen w klauzuli **WHERE**
- W poprzednim przykładzie nawiasy ustalaja kolejnosc w ten sposób, ze najpierw wykonywane sa instrukcje polaczone operatorem **OR**, a nastepnie wykonana jest operacja z operatorem **AND**
- Jezyk SQL dysponuje kilkoma dodatkowymi elementami, które znacznie upraszczaja zapytania z wieloma operatorami logicznymi
- Klauzula **IN** zastepuje wiele operatorów **OR** w instrukcjach sprawdzajacych, czy wybrana grupa wartosci znajduje sie w kolumnie
- Operator **IN** okresla, czy wartosc testowana jest identyczna z przynajmniej jedna z wartosci z listy
- Przyklad ilustruje jak mozna uproscic poprzednie zapytanie:

```
SELECT Imie, Nazwisko, Pensja, Miasto
FROM NAZWISKA
WHERE Miasto IN ('Gdansk', 'Gdynia') AND Pensja IS NOT NULL
ORDER BY Nazwisko DESC;
```

NOT IN

- Wartosc logiczna wyrazenia zawartego wewnatrz klauzuli **IN** mozna zaprzeczyc operatorem **NOT**
- Klauzula **IN** wybiera wszystkie wiersze, w których wartosc testowana jest równa jednej z wartosci umieszczonych na liscie
- **NOT IN** wybiera te wiersze, w których wartosc testowana jest rózna od kazdej wartosci z listy
- Przyklad zapytania wybierającego wszystkich pracowników nie mieszkających w Gdańsku ani w Gdyni, którzy mają ustalone pensje:

```
SELECT Imie, Nazwisko, Pensja, Miasto
FROM NAZWISKA
WHERE Miasto NOT IN ('Gdansk','Gdynia') AND Pensja IS NOT NULL
ORDER BY Nazwisko DESC;
```

- Klauzula **NOT IN** moze byc zastapiona przez operator **AND**

```
SELECT Imie, Nazwisko, Pensja, Miasto
FROM NAZWISKA
WHERE Miasto <> 'Gdansk' AND Miasto <> 'Gdynia' AND
Pensja IS NOT NULL
ORDER BY Nazwisko DESC;
```

Klauzula BETWEEN

- Klauzule **BETWEEN** i jej zaprzeczenie, **NOT BETWEEN**, wykorzystujemy do sprawdzenia, czy wartosc nalezy lub nie nalezy do okreslonego przedzialu wartosci
- Klauzula **BETWEEN** sluzy do sprawdzenia, czy wartosc nalezy do podanego zakresu z uwzglednieniem wartosci granicznych
- Moze byc zastapiona przez dwa porównania polaczone operatorem **AND**
- Przyklad zapytania wyszukującego wszystkich pracowników których pensje mieszczą się w przedziale 1100-3000 zł, posortowane rosnąco wg pensji:

```
SELECT Imie, Nazwisko, Pensja, Miasto
FROM NAZWISKA
WHERE Pensja BETWEEN 1100 AND 3000
ORDER BY Pensja;
```

- Wynik zapytania:

Imie	Nazwisko	Pensja	Miasto
Marian	Malinowski	1 100,00 zl	Gdynia
Adam	Nowak	2 000,00 zl	Gdansk
Mieczyslaw	Dobija	3 000,00 zl	Warszawa
Waldemar	Pawlak	3 000,00 zl	Sopot

- Inaczej sformułowany warunek:

```
WHERE Pensja >= 1100
  AND Pensja <= 3000
```

NOT BETWEEN

- Sprawdza czy podana wartosc znajduje sie poza okreslonym przedzialem
- Dzialanie tej instrukcji moze byc zastapione dwoma porównaniami polaczonymi instrukcja **OR**
- Sprawdzajac czy liczba znajduje sie pomiedzy innymi liczbami, logiczne wydaje sie, ze musi byc ona wieksza od dolnej wartosci i mniejsza od górnej wartosci
- Przyklad zapytania wyszukującego pracowników mających pensje niższe od 1100 i wyższe od 3000 zł:

```
SELECT Imie, Nazwisko, Pensja, Miasto
FROM NAZWISKA
WHERE Pensja NOT BETWEEN 1100 AND 3000
ORDER BY Pensja;
```

- Wynik zapytania:

Imie	Nazwisko	Pensja	Miasto
Jan	Kowalski	900,00 zl	Gdansk
Peter	Norton	3 500,00 zl	Gdansk
Paul	Davies	8 000,00 zl	Londyn

- Inaczej sformułowany warunek:

```
WHERE Pensja < 1100
   OR Pensja > 3000
```

BETWEEN i inne typy danych

- **BETWEEN** stosuje sie równiez, zeby sprawdzic czy podana data i czas naleza do podanego zakresu
- **BETWEEN** mozna stosowac równiez przy operacjach na lancuchach, podobnie jak zwykle operatory porównania
- Postac zapytania wybierającego pracowników, których nazwiska zaczynają się od liter między ‘D’ a ‘N’:

```
SELECT Imie, Nazwisko, Pensja, Miasto
FROM NAZWISKA
WHERE Nazwisko BETWEEN 'D' AND 'N'
ORDER BY Pensja;
```

- Wynik zapytania

Imie	Nazwisko	Pensja	Miasto
Zenon	Miler		Gdynia
Ewa	Musial		Gdansk
Jan	Kowalski	900,00 zl	Gdansk
Marian	Malinowski	1 100,00 zl	Gdynia
Mieczyslaw	Dobija	3 000,00 zl	Warszawa
Paul	Davies	8 000,00 zl	Londyn

- Jak widac w Accessie 2000 z lewej jest warunek >= a z prawej <

Zlozone klauzule WHERE z operatorem LIKE

- Dziala na kolumnach zawierajacych wartosci lancuchowe.
- Operator **LIKE** sprawdza czy wartosc tekstowa odpowiada podanemu wzorcowi, umozliwia wiec wykonywanie czesciowych porównan, takich jak „zaczynajacy sie od tekstu”, „konczacy sie na tekscie”, lub „zawierajacy tekst”
- Tworzac wzorce stosuje sie znaki wieloznaczne:
  - **%** - zastępuje sekwencję dowolnych znaków o długości *n* (gdzie *n* może być zerem)
  - **_** - odpowiada jednemu znakowi w przeszukiwanym tekscie
- W Accessie
  - ***** - zastępuje sekwencję dowolnych znaków o długości *n* (gdzie *n* może być zerem)
  - **?** – odpowiada jednemu znakowi
- Ogólna postać polecenia z operatorem **LIKE**

```
WHERE tekst LIKE wzorzec
```

Przykład operatora LIKE

- Postać zapytania wyszukującego wszystkie rekordy, w których w polu Nazwisko występuje sekwencja znaków **‘no’**:

```
SELECT Imie, Nazwisko, Pensja, Miasto
FROM NAZWISKA
WHERE Nazwisko LIKE '*no*'
ORDER BY Nazwisko;
```

- Wynik zapytania

Imie	Nazwisko	Pensja	Miasto
Marian	Mali**no**wski	1 100,00 zl	Gdynia
Peter	**No**rton	3 500,00 zl	Gdansk
Adam	**No**wak	2 000,00 zl	Gdansk

- Postać zapytania, które wyszuka wszystkie rekordy, gdzie druga litera nazwiska jest „o”:

```
SELECT Imie, Nazwisko, Pensja, Miasto
FROM NAZWISKA
WHERE Nazwisko LIKE '?o*'
ORDER BY Nazwisko;
```

- Operator **LIKE** zmniejsza wydajnosc realizacji zapytan

Funkcje agregujace

- W SQL dostepnych jest kilka funkcji agregujacych dzialajacych na grupie wartosci zwracanych przez zapytanie a nie na pojedynczej wartosci pola
- Na przyklad mozemy w tabeli policzyc liczbe wierszy spelniajacych okreslone kryteria lub mozna wyliczyc wartosc srednia dla wszystkich wartosci z wybranej kolumny
- Funkcje te dzialaja na wszystkich wierszach w tabeli, na pewnej grupie wierszy wybranej klauzula **WHERE** lub na grupach danych wybranych klauzula **GROUP BY**
- **Funkcja COUNT(nazwa_kolumny)**
- Funkcja ta zlicza ilosc wierszy w zapytaniu
- Jezeli chcemy znac liczbe wierszy zwróconych przez zapytanie, najprosciej uzyc funkcji w postaci **COUNT(*)** (gwiazdka - wszystkie kolumny tabeli)
- Sa tego dwa powody:
  - po pierwsze pozwalamy optymalizatorowi bazy danych wybrac kolumne do wykonania obliczen, co czasem nieznacznie podnosi wydajnosc zapytania
  - po drugie, nie musimy sie martwic o wartosci NULL zawarte w kolumnie oraz o to, czy kolumna o podanej nazwie w ogóle istnieje

Tworzenie nowej tabeli

- Do zdefiniowania nowej tabeli uzywamy instrukcji **CREATE TABLE,** której najprostsza instrukcja wyglada nastepujaco:

```
CREATE TABLE Nazwa_tabeli
(nazwa_kolumny      typ_danych[(rozmiar)],
nazwa_kolumny       typ_danych[(rozmiar)],
...)
```

- Kazda kolumna musi miec okreslony typ danych
- Dla wiekszosci typów danych wymagane jest takze okreslenie rozmiaru
- W instrukcji **CREATE TABLE** istnieje mozliwosc zdefiniowania klucza głównego, okreslenie relacji z innymi tabelami, wprowadzenie ograniczen na wartosci kolumn itp.
- **Typy danych w definiowaniu tabel w SQL**
- Do zdefiniowania tabeli konieczne jest podanie typu danych
- Nie mozna stosowac nazw typów uzywanych w Accessie, takich jak: Autonumerowanie, Tekst, Nota, Liczba, Data/godzina, Walutowy, Tak/Nie, Obiekt OLE, Hiperlacze

Typy danych

- Typ danych determinuje nie tylko sposób przechowywania danych na dysku, ale co wazniejsze, sposób interpretacji tych danych
- Niemniej wazne sa wymagania dotyczace zajmowania pamieci
- Marnotrawstwem byloby zarezerwowanie 255 bajtów dla pola, które wykorzystuje tylko 2 bajty, a z drugiej strony zarezerwowanie 5 bajtów dla numeru telefonu, moze nie byc wystarczajace
- Relacyjne bazy danych dostarczaja bardzo bogaty zestaw typów danych
- Istnieja typy danych tekstowych, liczby, typy okreslajace czas oraz obiekty, dane binarne czy duze teksty
- Kazda baza danych posiada swoje wlasne zestawy typów danych, mogace sie róznic pomiedzy soba nazwami
- Niektóre systemy baz danych udostepniaja równiez podtypy, jak np. dla typu liczbowego, moze to byc liczba calkowita, zmiennoprzecinkowa czy waluta
- Wiekszosc baz danych obsluguje podstawowe typy, choc pomiedzy róznymi produktami nie ma pelnej zgodnosci

Typy danych

- Cztery kategorie typów: dane lancuchowe, numeryczne, okreslajace czas i duze obiekty
- Dane lancuchowe moga przechowywac wlasciwie kazdy typ danych z zastrzezeniem, ze dane te sa traktowane tylko jako lancuch znaków
- Dane numeryczne i okreslenia czasu umozliwiaja wykonywanie dzialan matematycznych oraz innych funkcji do przetwarzania danych
- Duze obiekty, sluzą do gromadzenia duzych ilosci informacji - sa one traktowane odmiennie od innych typów danych, np. nie mozna porównywac takich obiektów
- Wazna róznica miedzy typami danych polega na sposobie traktowania ich przez jezyk SQL - dane lancuchowe, okreslenia czasu i duze obiekty musza byc w instrukcjach SQL zawarte w pojedynczych cudzyslowach, natomiast dane numeryczne nie sa zapisywane w cudzyslowach
- W wiekszosci baz danych mamy do dyspozycji dwa rodzaje typów lancuchowych o ustalonej dlugosci i o zmiennej dlugosci
- Ustalona dlugosc powoduje zawsze rezerwacje takiej samej ilosci pamieci, bez wzgledu na wymagania danych, natomiast zmienna dlugosc zuzywa tylko tyle pamieci, ile jest potrzebne dla konkretnej wartosci

Typy danych – dane znakowe

- **Typy łańcuchowe**
- **CHAR** jest typem danych o ustalonej dlugosci - **CHAR(wymiar)**
- W polu typu **CHAR** miejsce nie zuzyte przez dane jest automatycznie uzupelniane spacjami
- **VARCHAR** jest typem danych o zmiennej dlugosci – **VARCHAR(wymiar)**
- Przy deklaracji tego typu danych okreslamy maksymalna dlugosc
- Róznica miedzy **VARCHAR(50)** a **CHAR(50)** polega na tym, ze pole o zmiennej dlugosci dostosowuje potrzebna pamiec do rzeczywistej dlugosci lancucha danych
- W przypadku, gdy chcemy zapamietac wieksza ilosc danych znakowych mamy do dyspozycji specjalny typ dla duzych obiektów tekstowych
- W Oracle jest to **CLOB** – Character Large Object a w Microsoft SQL Server jest typ **TEXT**.
- W Accessie jest to typ **MEMO**

Typy danych - dane numeryczne

- Czasami dane numeryczne przechowuje się w polu znakowym, np. kod pocztowy, czy numer telefonu lepiej zapamietac w polu tekstowym, mimo, ze skladaja sie z cyfr
- Wiekszosc baz danych dostarcza dwóch typów numerycznych, jeden dla liczb calkowitych, drugi dla zmiennoprzecinkowych
- Czasami mamy jeszcze bardziej szczególowe jak **MONEY**, który automatycznie przydziela dwa miejsca po przecinku i znak waluty
- Liczba cyfr obslugiwana przez pole numeryczne moze sie róznic w zaleznosci od bazy danych, a w wielu przypadkach można o tym zadecydowac przy definicji, podobnie jak w typie **CHAR**

Typ danych	**Definicja**
DECIMAL	Liczba zmiennoprzecinkowa
FLOAT	Liczba zmiennoprzecinkowa
INTEGER(rozmiar)	Liczba calkowita o okreslonej dlugosci
MONEY	Liczba posiadajaca dwie pozycje dziesietne
NUMBER	Standardowa liczba zmiennoprzecinkowa

Kolejny typ danych okresla date i czas - w Accessie jest to typ **DATE**

Okreslanie kluczy

- Tworząc tabele, mozna zdefiniowac zarówno klucz główny jak i klucze kandydujace
- Slowo **UNIQUE** sluzy do okreslenia, która kolumna (lub grupa kolumn) musi byc unikalna i jest przez to kluczem kandydujacym
- Uzycie ograniczenia **UNIQUE** powoduje, ze próba powtórzenia danych w tych kolumnach bedzie przez baze danych powstrzymana
- Definicja klucza głównego znajduje sie po definicjach pól, jezeli klucz główny sklada sie z kilku pól podaje sie liste nazw pól oddzielona przecinkami
- Zdefiniowanie klucza głównego wymaga uzycia klauzuli **PRIMARY KEY**
- Oczywiscie w tabeli moze byc zidentyfikowany jeden klucz główny
- Kolejny przykład przedstawia polecenie tworzące tabelę o nazwie NOWA zawierającą osiem pól różnych typów oraz zdefiniowany klucz główny

Przykład tworzenia nowej tabeli

- Postać polecenia, tworzącego tabelę o nazwie **NOWA**, w której kluczem głównym jest pole **Nr_ident**, a kluczem kandydującym jest pole **Telefon**:

CREATE TABLE NOWA	definicja nazwy tabeli
(Nr_ident **INTEGER,**	pole typu całkowitego
Zawód **CHAR(20),**	pole znakowe o stałej długości
Telefon **VARCHAR(15),**	pole znakowe o zmiennej długości
Data_rozp **DATE,**	pole zapamiętujące datę i czas
Premia **MONEY,**	pole walutowe
Prawo_jazdy **LOGICAL,**	pole typu logicznego
Uwagi **MEMO,**	pole dużego obiektu znakowego
UNIQUE (Telefon),	definicja klucza kandydującego
PRIMARY KEY(Nr_ident))	definicja klucza głównego

- Mozna definiowac klucze równiez w linii definiujacej kolumne
- np.: (Nr_ident **INTEGER PRIMARY KEY,**
- Klucze obce - klauzula **REFERENCES** sluzy do ustalenia relacji miedzy pomiedzy tabelami

Przykład tworzenia nowej tabeli

- **Odrzucanie wartosci NULL** - zapobiega wprowadzaniu wartosci NULL do kolumny. Uzycie **NOT NULL** w definicji kolumny wymusza podanie wartosci dla takiej kolumny przy kazdym wprowadzaniu nowego wiersza
- Zapobiega to zmianie wartosci na NULL przy aktualizacji danych w tabeli
- Taki sam efekt daje zdefiniowanie klucza głównego.
- Postac polecenia tworzącego tabelę z ustaleniem relacji miedzy polem **Nr_ident** z tabeli **NOWA** z polem **Numer** z tabeli **NAZWISKA** oraz zabezpieczeniem przed wartościami NULL dla pól **Zawód** i **Data_rozp:**

```
CREATE TABLE NOWA
(Nr_ident INTEGER PRIMARY KEY REFERENCES Nazwiska(Numer),
Zawód CHAR(20) NOT NULL,
Telefon VARCHAR(15),
Data_rozp DATE NOT NULL,
Premia MONEY,
Prawo_jazdy LOGICAL,
Uwagi MEMO)
```

Tworzenie, zmienianie i usuwanie rekordów

- Dane wprowadza sie przy pomocy instrukcji **INSERT**
- Do wprowadzania zmian sluza instrukcje **UPDATE** i **DELETE** (do kasowania)
- Do usuwania tabeli z bazy danych służy instrukcja **DROP**
- **Instrukcja INSERT - j**est to jedyna instrukcja jezyka SQL sluzaca do dopisywania nowych rekordów do tabel
- Podstawowa struktura instrukcji **INSERT** jest nastepujaca:

```
INSERT INTO nazwa_tabeli
[(lista kolumn)]
VALUES
(lista wartosci)
```

- **Nazwa_tabeli** okresla tabele, do której wprowadza się nowy rekord
- W przypadku, gdy wprowadza się wartosci tylko dla niektórych kolumn, nalezy podac nazwy kolumn, do których mają być wprowadzone wartosci
- Pominiecie listy kolumn w instrukcji **INSERT** wymusza podanie wartosci dla wszystkich kolumn w tabeli

Tworzenie, zmienianie i usuwanie rekordów

- Postac polecenia wprowadzajacego pelny rekord danych

```
INSERT INTO NOWA
VALUES (3, 'prawnik', '345 89 98', '1999-08-05', 1200, 1,
'wyjazd w grudniu');
```

- Postac polecenia wprowadzajacego dane do wybranych kolumn

```
INSERT INTO NOWA
(Nr_ident, Zawód, Data_rozp)
VALUES (4, 'ekonomista', '2002-01-01');
```

- Musza byc wypełnione te pola, które sa NOT NULL i klucz główny
- Pole **Zawód** jest dopełniane spacjami do długości 20 znaków
- **Instrukcja DELETE** - sluzy do usuwania rekordów z tabeli.
- Podstawowa struktura instrukcji **DELETE**:

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

- Opcjonalna czesc z klauzula **WHERE** jest wykorzystywana do ograniczania rekordów, które zostana usuniete
- Pominiecie tej czesci powoduje, ze wszystkie rekordy sa usuwane

Tworzenie, zmienianie i usuwanie rekordów

- Postac polecenia usuwajacego z tabeli NOWA, wszystkie rekordy pracowników nie bedacych ekonomistami:

```
DELETE FROM NOWA
WHERE Zawód <> 'ekonomista';
```

- Postać polecenia usuwającego wszystkie rekordy z tabeli NOWA:

```
DELETE FROM NOWA
```

- **Instrukcja UPDATE** - jest wykorzystywana do wprowadzania zmian w istniejacych rekordach
- Struktura instrukcji jest nastepujaca:

```
UPDATE tabela
SET kolumna = wartosc, ...
[WHERE warunek]
```

- Instrukcja sklada sie z trzech czesci:
  - W pierwszej czesci okresla się, jaka tabela bedzie aktualizowana
  - Druga czesc – klauzula **SET** – sluzy do podania listy kolumn, które beda zmieniane i nowych wartosci, które zostana przypisane tym kolumnom
  - W ostatniej czesci za pomoca klauzuli **WHERE** okresla się wiersze tabeli, w których nastapi zmiana

Tworzenie, zmienianie i usuwanie rekordów

- Postac polecenia zmieniajacego zawartosc pola **Premia** (bylo 1200) na 500 dla pracownika o **Nr_ident** równym 3:

```
UPDATE NOWA
SET Premia = 500
WHERE Nr_ident = 3;
```

- **Instrukcja DROP** - sluzy do usuwania tabel z bazy danych
- Przy ustalaniu nowych wartości określonego pola można zastosować wyrażenia arytmetyczne
- Przykładowe polecenie spowoduje zwiększenie wszystkim pracownikom premii o 100 zł

```
UPDATE NOWA
SET Premia = Premia+100;
```

- Postac polecenia usuwającego tabelę z bazy:

```
DROP TABLE Nazwa_tabeli
```

Laczenie tabel

- W wielu przypadkach w trakcie wyszukiwania informacji z bazy danych okazuje sie, ze potrzebne dane przechowywane sa w kilku tabelach
- W celu polaczenia danych z wielu tabel w jednym zapytaniu wymagane jest zlaczenie
- **Polaczenia i normalizacja**
- Efektem normalizacji jest rozbicie bazy danych na wiele tabel
- Uzywajac zlaczen miedzy tabelami mozna wybierac informacje z wielu tabel za pomoca pojedynczej instrukcji **SELECT**
- Daje to efekt ponownego polaczenia danych, które zostaly rozdzielone do wielu tabel w trakcie normalizacji
- Zlaczenie to zapytanie, które laczy dane z wielu tabel
- Struktura standardowego zapytania jest nastepujaca:

```
SELECT lista_kolumn
FROM tabela1, [tabela2, ...]
WHERE warunek;
```

- W czesci **FROM** pojawiaja sie deklaracje kilku tabel, reszta nie różni się od polecenia działającego na jednej tabeli

KONIEC

micro KURSu SQL-a