SQL w praktyce

Koncepcja bazy danych

W tej części przedstawiona zostanie koncepcja bazy danych, jej podstawowe elementy składowe jak tabele i widoki. Omówione zostaną również typy danych, definiowanie tabel oraz rola i zastosowanie języka SQL w relacyjnych bazach danych.

Tradycyjne bazy danych

W czasie kiedy komputery nie były wykorzystywane tak powszechnie jak dzisiaj, informacje były gromadzone na papierze. Pracownik firmy musiał ręcznie organizować dane. Zapisywał je, wyszukiwał, aktualizował itd. Dzisiaj, oprócz operacji, które musi wykonać ręcznie, pozostałe wykonuje za pomocą funkcji, które dostarcza relacyjny system bazy danych.

Relacyjny system bazy danych

Relacyjny system bazy danych przechowuje wszystkie dane w tabelach. Każda tabela zawiera dane na konkretny temat, np dane o klientach, dane o pracownikach, towarach itp. System bazy danych zarządza tymi danymi, pozwala m.in. na szybsze ich wyszukanie i zorganizowanie Za każdym razem gdy potrzebujemy informacji z bazy danych, musimy "zapytać" system bazy danych w języku, który on rozumie. Tym językiem jest SQL - Structured Query Language

Wydobywanie informacji z tradycyjnej bazy danych oraz z systemu relacyjnej bazy danych

Do dzisiaj istnieją tzw. tradycyjne bazy danych. Są to bazy informacji, których nośnikiem jest po prostu papier. Wiele instytucji w tym niestety policja oraz szpitale korzysta do dziś z takich baz. Scenariusz wydobywania danych w takich instytucjach jest następujący:

- l osoba potrzebująca danych np. przełożony prosi drugą osobę o pewne dane;
- pracownik po otrzymaniu polecenia szuka informacji wśród dokumentów, które zostały złożone albo w archiwum lub w po prostu w szafie;
- pracownik po zebraniu pewnej ilości segregatorów z danego okresu przegląda je, a następnie przygotowuje zbiorczy dokument zawierający żądane informacje;
- l po wydobyciu danych i zorganizowaniu ich do odpowiedniej formy wysyła je do przełożonego.

Czas wykonania tych operacji jest różny i zależy od wielkości organizacji, jej struktury, ilości potrzebnych danych oraz od pracowitości osób je zbierających. W systemie relacyjnej bazy danych wszystkie powyższe operacje sprowadzają się do sformułowania tego samego pytania o dane ale w formie zrozumiałej dla komputera, a ściślej mówiąc, w formie zrozumiałej dla systemu bazy danych. Cała operacja wydobywania danych trwa w tym przypadku znacznie krócej. Jakość tych danych jest przy tym lepsza. Mamy więcej pewności, że dane są prawdziwe, że ktoś się nie pomylił lub pominął pewną część danych przy zestawianiu żądanych informacji.

SQL - Strukturalny język zapytań

Język SQL jest wykorzystywany w większości relacyjnych systemów baz danych. SQL jest zaimplementowany m.in. w takich systemach baz danych, jak: DB2, Oracie, Inter-Base, MySQL, dBase, Paradox. Składnia języka SQL dzieli się na trzy typy: - język definiowania struktur danych - DDL (Data Definition Language);

- R język do wybierania i manipulowania danymi - DML (Data Manipulation Language);
- l język do zapewniania bezpieczeństwa dostępu do danych - DCL (Data Control Language).

Składnia języka SQL wchodząca w skład języka DDL jest używana przez administratorów systemu relacyjnej bazy danych w celu utrzymania struktury bazy danych, obiektów bazy danych takich jak m.in. tabele. Język DCL jest używany przez administratorów do zapewnienia bezpieczeństwa dostępu do danych, m.in. do nadawania uprawnień do danych. Język DML jest używany przez wszystkich użytkowników, którzy mają dostęp do bazy danych. Za pomocą tego typu składni języka SQL użytkownicy mogą otrzymywać, zmieniać dane, dodawać nowe itp.

Tabela

Tabela składa się z wierszy i kolumn. Wiersze w tabeli są przechowywane w dowolnym porządku. Dla każdego wiersza każda z kolumn posiada jedno pole z wartością. Wszystkie wartości w kolumnie są tego samego typu. W różnych systemach relacyjnej bazy danych jak np. DB2, Oracie, InterBase czy dBase lub Paradox, każda tabela jest przechowywana w osobnym zbiorze na dysku twardym lub kilka tabel w jednym zbiorze. Sposób przechowywania danych z tabeli na dysku twardym jest tematem drugorzędnym. Ćwiczenia te maja przede wszystkim na celu nauczenie języka SQL. Teraz wystarczy tylko wiedzieć, że sposób przechowywania tabel zależy od implementacji systemu relacyjnej bazy danych.

Konstrukcja nazwy tabeli

Nazwa tabeli składa się z dwóch części. Pierwsza część to kwalifikator, a druga z kolei to nazwa tabeli. Kwalifikator i nazwa tabeli oddzielone są kropka. Każda tabela musi mieć unikatową nazwę w granicach kwalifikatora. Taka konstrukcja nazwy tabeli nie stosuje się we wszystkich relacyjnych bazach danych. Między innymi w opisywanym tutaj systemie InterBase. W InterBase odwołanie do tabeli następuje wprost, np. SELECT * PROM KLIENCI . . .

w systemie DB2
SELECT * FROM DB2ADMIN.KLIENCI

Typy danych

W różnych systemach relacyjnej bazy danych inaczej nazywają się typy danych. Jednak ich zakres i typ jest często identyczny. Każdy system relacyjnej bazy danych posiada w swojej dokumentacji sekcję, która opisuje typy danych używanych w tym systemie. Poniżej znajdują się przykładowe typy danych wraz z ich opisem

Typ danych	Opis
SMALLINT	liczby całkowite z przedziału -32768 do +32767 (czasami ten zakres jest mniejszy)
INTEGER	liczby całkowite z przedziału -2147483648 do +2147483647 (lub mniejszy)
DECIMAL (m,n)	liczby rzeczywiste, gdzie m oznacza całkowitą liczbę cyfr, a n oznacza liczbę cyfr po przecinku
CHAR (n)	typ znakowy o stałej długości (max. 255 znaków)
VARCHAR (n)	typ znakowy o zmiennej długości
DATE	typ daty (występują różne standardy zapisywania daty)
TIME	typ czasu (występują różne standardy zapisywania czasu)

Tworzenie tabeli - CREATE TABLE

Tworzenie tabeli polega na definiowaniu jej kolumn. Dla każdej kolumny należy określić nazwę kolumny, typ danych i długość (w zależności od typu) oraz to, czy jest dozwolone pozostawienie wartości pustej w kolumnie.

```
CREATE TABLE UZYTKOWNIK.PRACOWNICY (
ID_PRACOW CHAR(6)NOT NULL,
IMIE VARCHAR(18)NOT NULL,
NAZWISKO VARCHAR(24) NOT NULL,
STANOWISKO VARCHAR(12)NOT NULL,
DZIAL VARCHAR(12) NOT NULL,
DATA_URODZ DATE,
TELEFON_DOM CHAR(12));
```

Wartość pusta NULL

Wartość NULL jest to wartość nieokreślona, która może zostać użyta w każdym polu tabeli niezależnie od typu kolumny. Wartość NULL jest różna od zera lub spacji

Przy konstruowaniu tabeli poleceniem CREATE TABLE w poprzedniej sekcji określiliśmy dla pewnych kolumn parametr NOT NULL. Oznacza to, że przy wstawianiu nowych wierszy musimy określić wartości dla tych kolumn, nie mogą one być wartością NULL. Definicja kolumny w poleceniu CREATE TABLE pozostawiona bez klauzuli NOT NULL określa, że dozwolone jest wstawienie do tej kolumny wartości NULL. Istnieje jeszcze opcja o następującej składni:

NOT NULL WITH DEFAULT ((wartość)]

gdzie parametr wartość określa domyślną wartość dla kolumny. Wartość domyślna zostanie nadana dla kolumny automatycznie, gdy nie określimy jej wprost przy wstawianiu nowego wiersza do tabeli.

Autoryzacja dostępu do tabeli

Możemy udostępnić nasze dane innym użytkownikom, a ściślej mówiąc możemy udostępnić tabele innemu użytkownikowi. W tym celu stosuje się polecenie języka SOŁ GRANT. Poniższy przykład nadaje uprawnienia użytkownikowi o nazwie UŻYTKOWNIK do tabeli PRACOWNICY. Od tej pory UŻYTKOWNIK może wybierać (wykonywać zapytania SELECT) dane z naszej tabeli.

```
GRANT SELECT ON PRACOWNICY TO UŻYTKOWNIK;
```

Prawa do tabeli można odebrać poleceniem REVOKE. Oto przykład:

```
REVOKE SELECT ON PRACOWNICY FROM UŻYTKOWNIK;
```

Powyższy przykład użycia polecenia GRANT umożliwia tylko wybieranie danych z tabeli. Poniższy przykład umożliwia wybieranie, wstawianie i aktualizowanie danych w tabeli.

```
GRANT SELECT, INSEKT, UPDATE ON PRACOWNICY TO UZYTKOWNIK;
```

Widoki

Za pomocą widoków możemy ograniczyć zakres danych dostępnych dla użytkownika. Widok może ograniczać dane z jednej tabeli lub może to być kompozycja danych z kilku tabel. Dane w widoku mogą być ograniczone do kilku kolumn lub do pewnego zakresu wierszy. Widoki stosuje się w różnych celach:

- w celu zabezpieczenia danych przed niepowołanym dostępem;
- uproszczenia korzystania z danych dla końcowego użytkownika.

Przykładem zwiększenia bezpieczeństwa może być widok, który nie obejmuje kolumny z danymi o zarobkach. Wiadomo, że nie wszyscy użytkownicy powinni mieć dostęp do takich danych.

Zapytania SQL

Polecenie SELECT jest używane do pobierania danych z bazy danych (z tabel lub widoków). W tym rozdziale zapoznamy się ze składnią polecenia SELECT. Sekcja ta ma na celu nauczenie formułowania zapytań SQL do wyświetlana wszytkach wierszy z tabeli, wybierania określonych kolumn, używania warunków, używania stów kluczowych BETWEEN, IN, LIKE Oraz DISTINCT.

Struktura polecenia SELECT

SELECT : opisuje nazwy kolumn, wyrażenia arytmetyczne, funkcje
FROM : nazwy tabel lub widoków
WHERE : warunek (wybieranie wierszy)
GROUP BY : nazwy kolumn
HAVING : warunek (grupowanie wybieranych wierszy)
ORDER BY : nazwy kolumn lub pozycje kolumn
Każde polecenie SELECT musi posiadać klauzule SELECT oraz FROM, pozostałe klauzule są opcjonalne. Inne klauzule wchodzące w skład polecenia SELECT zostaną szczegółowo omówione później.

Wybieranie wszystkich kolumn

Poniższe polecenie SELECT wyświetla wszystkie kolumny i wiersze z tabeli PRACOWNICY.

```
SELECT *
FROM DB2ADMIN.PRACOWNICY;
```

Wybieranie wszystkich kolumn i wierszy ma sens tylko w przypadku małych tabel, W praktyce buduje się zapytania, które znacznie ograniczają wynik zapytania.

Wybieranie określonych kolumn

Polecenie SELECT, którego użyjemy za chwilę, wyświetla kolumny IMIĘ, NAZWISKO i DZIAŁ z tabeli PRACOWNICY.

```
SELECT IMIE, NAZWISKO, DZIAL FROM DB2ADMIN.PRACOWNICY
```

Wybieranie i jednoczesnym porządkowaniem

Następujące polecenie SELECT wyświetla kolumny IMIĘ, NAZWISKO i DZIAŁ z tabeli PRACOWNICY i jednocześnie porządkuje dane według nazwiska.

```
SELECT IMIE, NAZWISKO, DZIAL FROM DB2ADMIN.PRACOWNICY
ORDER BY NAZWISKO ASC;
```

Wynik wykonania zapytania jest uporządkowany według kolumny wskazanej w klauzuli ORDER BY. Słowo kluczowe ASC mówi o tym, że sortowanie zostanie dokonane w porządku rosnącym. Sortowanie rosnące jest domyślne więc słowo kluczowe ASC nie musi być wyspecyfikowane. Porządek malejący uzyskuje się przez zastosowanie słowa DESC. W zależności od implementacji bazy danych kolumna występująca w klauzuli ORDER BY musi być częścią wyniku wykonania zapytania. Możliwe jest wskazanie większej liczby kolumn w klauzuli ORDER BY. Przykładowo może istnieć potrzeba wybrania danych w tabeli z jednoczesnym sortowaniem według stanowiska, na którym dana osoba pracuje, a następnie według nazwiska.

```
SELECT IMIĘ, NAZWISKO, STANOWISKO, DZIAŁ
FROM DB2ADMIN.PRACOWNICY ...
ORDER BY STANOWISKO ASC, NAZWISKO ASC;
```

Istnieje inny sposób na wskazanie kolumn w klauzuli ORDER BY. Zamiast nazywać kolumny, możemy je wskazać poprzez ich pozycje na liście SELECT.

```
SELECT IMIE, NAZWISKO, STANOWISKO, DZIAL FROM DB2ADMIN.PRACOWNICY
ORDER BY 3 ASC, 2 ASC;
```

Inne przykłady:

```
ORDER BY 3 ASC, NAZWISKO ASC
ORDER BY 3 ASC, 2 ASC, DZIAL ASC;
```

Dozwolona jest tylko jedna klauzula ORDER BY w zapytaniu SELECT. Klauzulę ORDER BY określa się jako ostatnią w całym zapytaniu SELECT.

Wybieranie niepowtarzających się wierszy

Słowo kluczowe DISTINCT zapewnia, że wynik zwrócony z zapytania zawierać będzie tylko niepowtarzające się wiersze. Wszystkie powtarzające się wartości nie zostaną wyświetlone. SELECT DISTINCT STANOWISKO FROM DB2ADMIN.PRACOWNICY;

Słowo kluczowe DISTINCT musi występować zaraz po słowie kluczowym SELECT.

```
SELECT DISTINCT STANOWISKO, DZIAŁ FROM DB2ADMIN. PRACOWNICY;
```

Takie zapytanie wyświetli wszystkie stanowiska obejmowane w danych działach. Jeżeli w danym dziale pojawią się dwa takie same stanowiska, tylko jedno zostanie wyświetlone. Słowo DISTINCT eliminuje wiersze, które posiadają duplikaty we wszystkich kolumnach wyspecyfikowanych w wyrażeniu SELECT. Tylko jedno słowo DISTINCT może zostać użyte w całym zapytaniu SELECT.

Wybieranie określonych wierszy Do wybrania określonych wierszy z tabeli używa się klauzuli WHERE, która służy do określenia kryterium wyboru wierszy. W klauzuli WHERE specyfikujemy warunek, który musi być spełniony dla szukanych wierszy.

```
SELECT IMIĘ, NAZWISKO, STANOWISKO, DZIAL
```

```
PROM DB2ADMIN.PRACOWNICY
WHERE STANOWISKO = 'SPRZEDAWCA';
```
W przypadku kolumn typu znakowego, daty lub czasu, wartości dla których sprawdzany jest warunek muszą być otoczone apostrofem. Przy porównywaniu kolumn typu znakowego należy pamiętać, że rozróżniane są wielkie i małe litery. Dla kolumn typu numerycznego jak np. INTEGER, SMALLINT, wartości do porównania nie są otaczane apostrofem.
```
SELECT NR_KLIENTA, NR_SAMOCHODU, NR_PRACOW_WYP, CENA_JEDN
FROM DB2ADMIN.WYPOZYCZENIA
WHERE CENA_JEDN >= 100;
```
Operatory logiczne używane w klauzuli WHERE
```
SELECT NR_KLIENTA, NR_SAMOCHODU, NR_PRACOW WYP, CENA_JEDN
FROM DB2ADMIN. WYPOZYCZENIA
WHERE CENA_JEDN = 100 - równa
CENA_JEDN <> 100 - nie równa
CENA_JEDN > 100 - większa niż
CENA_JEDN >= 100 - większa lub równa
CENA_JEDN < 100 - mniejsza niż
CENA_JEDN <= 100 - mniejsza lub równa
```
Operatory AND oraz OR
Kiedy w warunku używamy operatora AND, aby wiersz został zawarty w wyniku, oba warunki połączone operatorem AND muszą zostać spełnione, tzn. muszą zwrócić wartość prawdy (TRUE). Warunek z operatorem OR zwróci wartość TRUE, gdy przynajmniej jedna ze stron zwróci wartość TRUE.
```
SELECT IMIE, NAZWISKO, STANOWISKO, DZIAL
FROM DB2ADMIN.PRACOWNICY
WHERE STANOWISKO = 'SPRZEDAWCA'
AND DZIAL = 'OBSŁUGA KLIENTA';
```
Takie zapytanie SQL zwróci w wyniku wszystkich pracowników pracujących na stanowisku sprzedawca w dziale obsługi klienta.
```
SELECT IMIE, NAZWISKO, STANOWISKO, DZIAL
FROM DB2ADMIN.PRACOWNICY
WHERE STANOWISKO = 'SPRZEDAWCA'
OR DZIAL = 'TECHNICZNY';
```
Następne zapytanie zwróci wszystkich pracowników pracujących na stanowisku sprzedawca oraz wszystkich pracowników pracujących w dziale technicznym niezależnie od tego, czy pracują na stanowisku sprzedawca. Operatorów AND i OR możemy używać razem do budowy bardziej złożonych warunków. Następujące zapytanie zwróci wszystkich pracowników pracujących na stanowisku kierownika w dziale obsługi klienta oraz wszystkich pracowników z działu technicznego. Wiersze zostaną uporządkowane wg działu a następnie wg nazwiska.
```
SELECT IMIE, NAZWISKO, STANOWISKO, DZIAL
FROM DB2ADMIN.PRACOWNICY
WHERE STANOWISKO = 'KIEROWNIK'
AND DZIAL = 'OBSŁUGA KLIENTA'
OR DZIAL = 'TECHNICZNY'
ORDER BY DZIAL, NAZWISKO;
```
W poprzednim przykładzie widoczna jest wyższość operatora AND nad operatorem OR. Następne zapytanie posiada w klauzuli WHERE warunki otoczone nawiasami. Nawiasy pozwalają określić kolejność sprawdzania warunków.
```
SELECT IMIE, NAZWISKO, STANOWISKO, DZIAL
FROM DB2ADMIN.PRACOWNICY
WHERE STANOWISKO = 'KIEROWNIK'
AND (DZIAL = 'OBSŁUGA KLIENTA' OR DZIAŁ = 'TECHNICZNY')
ORDER BY DZIAL, NAZWISKO;
```
Zapytanie wyświetli osoby pracujące tylko na stanowisku kierownika w dziale obsługi klienta lub w dziale technicznym.
Predykat IN
Predykat IN pozwala porównać wartość do wartości ze zbioru. Wartości typu znakowego, daty i czasu muszą być otoczone apostrofem.
```
SELECT IMIE, NAZWISKO, STANOWISKO, DZIAL
FROM DB2ADMIN.PRACOWNICY
WHERE STANOWISKO IN ('SPRZEDAWCA', 'KIEROWNIK');
```
Wartości mogą być typu numerycznego, znakowego, typu daty lub czasu
```
SELECT MARKA, TYP, ROK_PROD, POJ_SILNIKA
FROM DB2ADMIN. SAMOCHODY
WHERE POJ_SILNIKA IN (1400, 1600);
```
Predykat BETWEEN

Predykat BETWEEN pozwala sprawdzić, czy dana wartość zawiera się między dwoma wskazanymi wartościami.

```
SELECT MARKA, TYP, ROK_PROD, KOLOR, POJ_SILNIKA
FROM DB2ADMIN.SAMOCHODY
WHERE POJ_SILNIKA BETWEEN 1100 AND 1800;
```

Zapytanie zwróciło dane o samochodach, których pojemność silnika zawiera się miedzy 1100 a 1800 cm sześciennych.
Klauzula:

```
WHERE POJ_SILNIKA BETWEEN 1100 AND 1800;
```

jest równa następującemu zapisowi:

```
WHERE POJ_SILNIKA >= 1100 AND POJ_SILNIKA <= 1800;
```

Wybieranie wartości NULL

Wybieranie wierszy z tabeli, w których jedno z pól zawiera wartość pustą NULL, polega na użyciu predykatu NULL. W przykładzie użycia predykatu NULL wybieramy wszystkich klientów, którzy nie posiadają karty kredytowej. Zwrócone zostaną wiersze z danymi o klientach, którzy w polu NR_ KARTY_KREDYT nie posiadaj ą żadnego wpisu.

```
SELECT IMIĘ, NAZWISKO, ULICA, MIASTO
PROM DB2ADMIN.KLIENCI
WHERE NR_KARTY_KREDYT IS NULL;
```

Możliwe jest wybranie wszystkich klientów posiadających kartę kredytową. Wtedy w klauzuli WHERE dla sprawdzenia wartości w polu NR_KARTY_KREDYT używamy również predykatu NULL, ale z zaprzeczeniem.

```
SELECT IMIE, NAZWISKO, NR_KARTY_KREDYT, MIASTO
FROM DB2ADMIN.KLIENCI
WHERE NR_KARTY_KREDYT IS NOT NULL;
```

Wyszukiwanie częściowe - predykat LIKE

Często istnieje konieczność wyszukania np. nazwisk klientów, które zaczynają się od konkretnej litery.

```
SELECT IMIE, NAZWISKO, ULICA, MIASTO
FROM DB2ADMIN.KLIENCI
WHERE NAZWISKO LIKE 'K%';
```

Inne przykłady użycia predykatu LIKE:

```
SELECT IMIE, NAZWISKO, ULICA, MIASTO
FROM DB2ADMIN.KLIENCI
WHERE NAZWISKO LIKE '%SKI';
```

Zapytanie zwróci wiersze z danymi o klientach, których nazwiska kończą się na "ski".
W następnym przykładzie wyszukamy klientów, którzy w swoim nazwisku posiadają litery "K" oraz "A" w wymienionym porządku.

```
SELECT IMIE, NAZWISKO, ULICA, MIASTO
FROM DB2ADMIN.KLIENCI
WHERE NAZWISKO LIKE '%K%A%';
```

W zapytaniach z predykatem LIKE można stosować zaprzeczenie NOT oraz operatory AND i OR. Oto przykłady:

```
SELECT IMIE, NAZWISKO, ULICA, MIASTO
FROM DB2ADMIN.KLIENCI ,
WHERE NAZWISKO NOT LIKE 'K%';
```

Następujące zapytanie wyszuka wszystkich klientów, których nazwiska nie zaczynają się na literę "K" oraz "D".

```
SELECT IMIE, NAZWISKO, ULICA, MIASTO
FROM DB2ADMIN.KLIENCI
WHERE NAZWISKO NOT LIKE 'K%'
AND NAZWISKO NOT LIKE 'D%' ;
```

Możliwe jest również wyszukanie np. klientów, których nazwiska zawierają drugą literę "O". Znak "_" zastępuje dowolny pojedynczy znak.

```
SELECT IMIE, NAZWISKO, ULICA, MIASTO
FROM DB2ADMIN.KLIENCI
WHERE NAZWISKO LIKE '_0%';
```

Oto drugi przykład, w którym pomijamy dwie pierwsze litery nazwiska:

```
SELECT IMIE, NAZWISKO, ULICA, MIASTO
FROM DB2ADMIN.KLIENCI
WHERE NAZWISKO LIKE '_C%';
```

Wybieranie danych z wielu tabel

W tej sekcji zajmiemy się wyszukiwaniem danych z wielu tabel. Do tej pory wszystkie zapytania wyszukujące koncentrowały się na jednej tabeli. Poza tym nauczymy się używać skrótów w odwoływaniu się do tabel w

zapytaniach SQL. Poznamy również predykat JOIN. W naszej przykładowej bazie danych WYPAUT, dla każdego numeru miejsca (miejsca pracy pracownika) w tabeli PRACOWNICY istnieje jeden wiersz w tabeli MIEJSCA. DB2 odczytuje numer miejsca pracy pracownika z tabeli PRACOWNICY, a następnie przeszukuje tabelę MIEJSCA w celu znalezienia odpowiadającego temu numerowi wiersza, który opisuje dokładnie miejsce pracy tzn. adres, telefon itd. W języku baz danych, jakim jest SQL, pytanie przedstawione na poprzednim rysunku może wyglądać tak:

```
SELECT DB2ADMIN.PRACOWNICY.NAZWISKO,
DB2ADMIN.PRACOWNICY.STANOWISKO,
DB2ADMIN.PRACOWNICY.DZIAL,
DB2ADMIN.MIEJSCA.MIASTO,
DB2ADMIN.MIEJSCA.ULICA
FROM DB2ADMIN.PRACOWNICY,DB2ADMIN.MIEJSCA
WHERE DB2ADMIN.PRACOWNICY.NR_MIEJSCA=DB2ADMIN.MIEJSCA.NR_MIEJSCA
ORDER BY DB2ADMIN.PRACOWNICY.NAZWISKO;
```

Wybieranie danych z wielu tabel nazywa się powszechnie złączeniem (ang. join). W celu złączenia dwóch lub większej ilości tabel:

- w klauzuli SELECT musimy wyspecyfikować kolumny, które chcemy zawrzeć w zapytaniu;
- w klauzuli FROM określamy nazwy złączanych tabel;
- w klauzuli WHERE określamy warunki złączenia.

Składnie złączenia - predykat JOIN

Istnieją dwa typy składni zapytania złączającego. Pierwszy typ został zaprezentowany już w poprzedniej sekcji. Oto zapytanie, które zostało zbudowane przy użyciu tej składni:

```
SELECT DB2ADMIN.PRACOWNICY.NAZWISKO,
DB2ADMIN.PRACOWNICY.STANOWISKO,
DB2ADMIN.PRACOWNICY.DZIAL,
DB2ADMIN.MIEJSCA.MIASTO,
DB2ADMIN.MIEJSCA.ULICA
FROM DB2ADMIN.PRACOWNICY,
DB2ADMIN.MIEJSCA
WHERE DB2ADMIN.PRACOWNICY.NR_MIEJSCA = DB2ADMIN.MIEJSCA.NR_MIEJSCA
ORDER BY DB2ADMIN.PRACOWNICY.NAZWISKO;
```

Przy złączaniu dwóch tabel, do poprawnego wyświetlenia wyniku klauzula WHERE musi zawierać jeden warunek. Gdy złączamy trzy tabele, klauzula WHERE musi zawierać przy-najmniej dwa warunki. Dwa pierwsze warunki w tym przykładzie dotyczą złączenia tabel, trzeci dotyczy warunku wyboru wierszy. Oto przykład:

```
SELECT DB2ADMIN. WYPOŻYCZENIA.NR_WYPOZYCZENIA,
DB2ADMIN.PRACOWNICY.NAZWISKO,
DB2ADMIN.PRACOWNICY.STANOWISKO,
DB2ADMIN.PRACOWNICY.DZIAL,
DB2ADMIN.MIEJSCA.MIASTO,
DB2ADMIN.MIEJSCA.ULICA
FROM DB2ADMIN.PRACOWNICY,
DB2ADMIN.MIEJSCA,
DB2ADMIN.WYPOZYCZENIA
WHERE DB2ADMIN.PRACOWNICY.NR_MIEJSCA = DB2ADMIN.MIEJSCA.NR_MIEJSCA
AND DB2ADMIN.PRACOWNICY.NR_PRACOWNIKA = DB2ADMIN.WYPOZYCZENIA.NR_PRACOW_WYP
AND DB2ADMIN.MIEJSCA.MIASTO = 'WARSZAWA'
ORDER BY DB2ADMIN.PRACOWNICY.NAZWISKO;
```

Inny typ złączenia polega na zastosowaniu konstrukcji JOIN ... ON.

```
SELECT DB2ADMIN.PRACOWNICY.NAZWISKO,
DB2ADMIN.PRACOWNICY.STANOWISKO, DB2ADMIN.PRACOWNICY.DZIAL,
DB2ADMIN.MIEJSCA.MIASTO, DB2ADMIN.MIEJSCA.ULICA
FROM DB2ADMIN.PRACOWNICY JOIN
DB2ADMIN.MIEJSCA ON
DB2ADMIN.PRACOWNICY.NR_MIEJSCA = DB2ADMIN.MIEJSCA.NR_MIEJSCA
WHERE DB2ADMIN.PRACOWNICY.STANOWISKO = 'SPRZEDAWCA'
ORDER BY DB2ADMIN.PRACOWNICY.NAZWISKO;
```

Kiedy używamy słowa JOIN w klauzuli FROM, warunki złączenia muszą być wyspecyfikowane po klauzuli ON. W klauzuli WHERE można określić dodatkowe warunki. Oto wynik wykonania powyższego zapytania:

Stosowanie aliasów w zapytaniu

Aliasy definiuje się w celu skrócenia nazwy tabeli. Jak wiemy na nazwę tabeli składa się kwalifikator i nazwa tabeli. Kwalifikator mówi o tym, kto jest właścicielem tabeli. W naszym przykładzie użycia aliasów, alias P wskazuje na tabelę DB2ADMIN.PRACOWNICY natomiast alias M opisuje tabelę DB2ADMIN.MIEJSCA.

```
SELECT P.NAZWISKO, P.STANOWISKO, P.DZIAŁ , M.MIASTO, M.ULICA
FROM DB2ADMIN.PRACOWNICY P,
DB2ADMIN.MIEJSCA M
```

WHERE
P.NR_MIEJSCA = M.NR_MIEJSCA AND P.STANOWISKO = 'SPRZEDAWCA'
ORDER BY P.NAZWISKO;
Wynik wykonania tego zapytania jest taki sam jak w ten w poprzedniej sekcji. Począwszy od tej sekcji w przykładach będziemy stosować aliasy dla nazw tabel.

Funkcje skalarne i arytmetyczne

Zajmiemy się używaniem funkcji skalarnych i arytmetycznych. Będziemy używać funkcji arytmetycznych do przeliczania wartości w kolumnach. Poznamy podstawowe funkcje skalarne m.in. funkcje operujące na datach i czasie. Pod koniec tego rozdziału poznamy sposób na wybieranie wartości przy użyciu wyrażenia CASE.

Wybieranie wyliczonych wartości

W zapytaniu SQL możemy użyć następujących operatorów arytmetycznych w celu obliczenia wartości:
+ dodawanie
- odejmowanie
* mnożenie
/ dzielenie
Operatorów tych możemy użyć do budowy bardziej rozbudowanych wyrażeń matematycznych włącznie z użyciem nawiasów w celu zaznaczenia kolejności wykonywania działań.
SELECT P.IMIĘ, P.NAZWISKO, P.PENSJA, P.DODATEK, P.PENSJA + P.DODATEK
FROM DB2ADMIN.PRACOWNICY'P WHERE P.PENSJA > 1100
ORDER BY P,NAZWISKO;
Wynik zapytania zawiera obliczoną kolumnę, która jest sumą kolumn; PENSJA I DODATEK. Kolumna z wynikiem została domyślnie nazwana "5", ponieważ jest ona piąta z kolei. Nazwa taka została nadana w DB2 zainstalowanym pod kontrolą systemu Windows NT. W innych systemach operacyjnych, DB2 może wy liczoną kolumnę nazywać inaczej. Dla dwóch pracowników, którzy zajmują stanowisko kierowników nie zostały obliczone wartości. Nie posiadają oni żadnego dodatku. Ściślej mówiąc, w polu DODATEK wartość dodatku dla tych osób wynosi MOLL. Wartości NULL nie mogą brać udziału w obliczeniach. W dalszej części tego rozdziału dowiemy się, jak obejść taki przypadek.

Nazywanie wyliczone. Kolumny

Kolumnę wynikową możemy nazwać. Poniżej znajduje się identyczne zapytanie jak w poprzedniej sekcji. Po słowie kluczowym AS podana jest nazwa dla nowej wyliczonej kolumny.
SELECT P.IMIE, P.NAZWISKO, P.PENSJA, P.DODATEK,
P.PENSJA + P.DODATEK AS DO_WYPLATY
FROM DB2ADMIN.PRACOWNICY P
WHERE P. PENSJA > 1100
ORDER BX P.NAZWISKO;
W tym przykładzie tak samo jak w przykładzie z poprzedniej sekcji do obliczeń nie mogła być wzięta pod uwagę wartość NULL. Stąd puste pola widoczne na powyższym rysunku. Zostanie to rozwiązane w następnej sekcji.
Nazwa tabeli wyliczonej może być otoczona cudzysłowem co pozwala na użycie nazwy składającej się z kilku słów. Ilustruje to poniższy przykład.
SELECT P.IMIE, P.NAZWISKO, P.PENSJA, P.DODATEK,
P.PENSJA + P.DODATEK AS "DO WYPLATY"
FROM DB2ADMIN.PRACOWNICY P
WHERE P.PENSJA > 1100 ORDER BY P.NAZWISKO;
Nowa nazwa kolumny wyliczonej nie może być użyta w klauzuli WHERE. W systemie DB2 może być natomiast użyta w ORDER BY.
SELECT P.IMIE, P.NAZWISKO, P.PENSJA, P.DODATEK
P.PENSJA + P.DODATEK AS DO_WYPLATY
FROM D32ADMIN. PRACOWNICY P
WHERE P.PENSJA > 1100
ORDER BY DO_WYPLATY;
Jak widać w powyższym przykładzie, nazwa kolumny DO_WYPLATY w klauzuli ORDER BY nie może być poprzedzona aliasem jak pozostałe kolumny. W systemie InterBase, nowa nazwa kolumny wyliczonej nie może być użyta w klauzuli ORDER BY. Zamiast nazwy możemy wskazać numer kolumny, względem której będziemy porządkować dane. Oto przykład tego samego polecenia dla systemu InterBase:
SELECT P.IMIE, P.NAZWISKO, P.PENSJA, P.DODATEK,
P.PENSJA + P.DODATEK AS DO_WYPLATY
FROM PRACOWNICY P WHERE P.PENSJA > 1100

Funkcja COALESCE

Funkcja COALESCE została zaimplementowana tylko w systemie DB2. Funkcja COALESCE jest funkcją operującą na wartości NULL. Zwraca pierwszy argument który nie jest wartością NULL. Funkcja ta jest równoważna funkcji VALUE. Funkcja YALUE jest synonimem funkcji COALESCE. , Poniższy przykład użycia funkcji COALESCE

rozwiązuje nasz problem z poprzedniej sekcji. Działanie funkcji COALESCE najpierw w kolumnie DODATEK zamienia wszystkie wystąpienia wartości NULL na wartość zera, a następnie robi to samo przy obliczaniu wartości do wypłaty.

```
SELECT P.IMIE, P.NAZWISKO, P.PENSJA,
COALESCE ( P.DODATEK, O ) AS DODATEK,
P.PENSJA + COALESCE (P.DODATEK, 0) AS DO_WYPŁATY
FROM DB2ADMIN.PRACOWNICY P
WHERE P,PENSJA > 1100 ORDER BY P.NAZWISKO;
```

W kolejnym przykładzie funkcja COALESCE została użyta w celu zastąpienia wszystkich-wystąpień wartości NULL na ciąg "nie posiada". Wyświetleni zostali wszyscy klienci. Dla tych, którzy nie posiadają karty kredytowej, w polu NRJCARTY został wpisany ciąg "nie posiada".

```
SELECT K.IMIE, K.NAZWISKO,
COALESCE
 FROM DB2ADMIN.KLIENCI K;
```

Dziesiętna reprezentacja wartości

Funkcja DECIMAL została zaimplementowana tylko w systemie DB2. Funkia DBCIMAL zwraca dziesiętną reprezentację wartości numerycznej. Pierwszy parametr zawiera wartość do reprezentacji, drugi parametr określa ilość cyfr przed przecinkiem, trzeci parametr określa liczbę miejsc po przecinku.

```
SELECT P.IMIE, P.NAZWISKO, P.PENSJA,
DECIMAL ( (P.PENSJA * 11.3)/100, 8, 2} AS KWOTA_PODWYZKI
FROM DB2ADMIN.PRACOWNICY P ORDER BY P.NAZWISKO;
```

Przykład oblicza kwotę 11.3% podwyżki.

Zaokrąglanie wyników

Funkcja ROUND została zaimplementowana tylko w systemie DB2. Służy ona do zaokrąglania wyników, Funkcja ta w pierwszym argumencie musi zawierać wartość do zaokrąglenia, w drugim natomiast podaje się liczbę miejsc po przecinku, do jakiej ma zostać zaokrąglona wartość. Poniższy przykład zaokrągla wartości do liczb całkowitych. Wartości dziesiętne poniżej 0,50 zostały zaokrąglone do zera, natomiast powyżej 0,50 do jedności.

```
SELECT P.IMIE, P.NAZWISKO, P.PENSJA,
ROUND ( (P.PENSJA * 11.31/100, 0) AS KWOTA_PODWYZKI
FROM DB2ADMIN.PRACOWNICY P
ORDER BY P.NAZWISKO;
```

Porównania daty

Kolumny typu daty lub czasu mogą być porównywane z innymi wartościami reprezentującymi datę lub czas. Wartości przedstawiające datę lub czas muszą być otoczone pojedynczym cudzysłowem. W poniższym przykładzie zostaną wyświetlone dane pracowników zatrudnionych w lub po dacie 1998-01-01.

```
SELECT P.IMIE, P.NAZWISKO, P.DZIAL,
P.STANOWISKO, P.DATA_ZATR
FROM DB2ADMIN. PRACOWNICY P
WHERE P.DATA_ZATR >= '1998-01-01'
ORDER BY P.NAZWISKO;
```

Kolejne zapytanie wybiera pracowników zatrudnionych co najmniej 2 lata. Porównywana wartość 020000 przedstawia 02 rok, 00 miesięcy i 00 dni. Funkcja CURRENT DATĘ zwraca bieżącą datę. Funkcja ta nie jest dostępna w systemie InterBase.

```
SELECT P.IMIE, P.NAZWISKO, P.DZIAL,
P.STANOWISKO, P.DATA__ZATR
FROM DB2ADMIN.PRACOWNICY P
WHERE CURRENT DATE - P.DATA_ZATR >= 020000
ORDER BY P.NAZWISKO;
```

Oprócz funkcji CORRENT DATĘ, która zwraca bieżącą datę, mamy do wykorzystania funkcję zwracającą bieżący czas CURRENT TIME oraz funkcję CURRENT TIMESTAMP zwracającą dokładny bieżący czas. Obie pozostałe funkcje również nie są dostępne w systemie InterBase.

Funkcje daty

Funkcja YEAR pozwala odczytać rok z pełnego formatu daty. Funkcja YEAR oraz wszystkie pozostałe w tej sekcji nie zostały niestety zaimplementowane w InterBase. Kolejny przykład jest identyczny do tego z poprzedniej sekcji z tym wyjątkiem, że dodatkowa kolumna przedstawia ilość przepracowanych lat przez pracownika, który pracuje dłużej niż dwa lata.

```
SELECT P.IMIE, P.NAZWISKO, P.DZIAL, P.STANOWISKO, P.DATA_ZATR,
YEAR (CURRENT DATĘ - P.DATA_ZATR) AS ILOSC_LAT
FROM DB2ADMIN.PRACOWNICY P
WHERE CURRENT DATE - P.DATA_ZATR >= 020000
ORDER BY P.NAZWISKO;
```

Poza funkcją YEAR mamy do dyspozycji funkcje MDNTH oraz DAY, które odpowiednio wydobywają z daty miesiąc i dzień. Oto przykład:

```
SELECT P. IMIĘ, P. NAZWISKO, P.DATA_ZATR
```

```
YEAR(P.DATA_ZATR) AS ROK,
MONTH(P.DATA_ZATR) AS MIESIAC,
DAY(P.DATA_ZATR) AS DZIEN
FROM DB2ADMIN.PRACOWNICY P;
```

W naszej przykładowej bazie danych znajduje się tabela WYPOŻYCZENIA, która m.in. przechowuje dane o dacie wypożyczenia samochodu i o dacie jego oddania. Następny przykład będzie obliczał ilość dni, przez które samochód był wypożyczony.

```
SELECT K.NAZWISKO, W.NR_WYPOZYCZENIA,
W.DATA_WYP, W.DATA_ODD,
DAYS(W.DATA_ODD)- DAYS(W.DATA_WYP) + l AS ILOSC_DNI
FROM DB2ADMIN.KLIENCI K, DB2ADMIN.WYPOŻYCZENIA W
WHERE K.NR_KLIENTA = W.NR_KLIEHTA AND W.DATA_ODD IS NOT NULL;
```

Ciąg DAYS(W.DATA_ODD) - DAYS (W. DAT_WYP) + l AS ILOSC_DNI występujący w zapytaniu odejmuje od daty oddania datę wypożyczenia samochodu i dodaje jeden. Dodanie jednego dnia ma na celu zaznaczenie sytuacji, gdy klient oddał samochód w dniu wypożyczenia. W takim przypadku różnica tych dat równa jest zero. W pozostałych przypadkach również dodawana musi być liczba jeden, aby zawrzeć w wyniku pierwszy dzień wypożyczenia. Funkcja DAYS odczytuje z daty ilość dni od daty l stycznia 0001 roku plus jeden. Następny przykład użycia funkcji DAYS polega na odjęciu od istniejących dat dwóch dni. Możemy również posłużyć się funkcją YEARS oraz MONTHS, które odpowiednio oznaczają lata i miesiące.

```
SELECT K. NAZWISKO, W. NR_WYPOZYCZENIA,
W.DATA_WYP, W.DATA_ODD
W.DATA_WYP - 2 DAYS, W. DATA_ODD - 2 DAYS
FROM DB2ADMIN.KLIENCI K,
DB2ADMIN.WYPOŻYCZENIA W
WHERE K.NR_KLIENTA = W. NR_KLIENTA
AND W.DATA_ODD IS NOT NULL
AND K.MIASTO = 'WARSZAWA'
;
```

Wybieranie podłańcucha W razie potrzeby wybrania tyko pewnej części łańcucha musimy zastosować funkcję SUBSTR. Na poniższym rysunku funkcja SUBSTR wybiera ciąg o długości sześciu znaków począwszy od trzeciego znaku.

```
SELECT SUBSTR(K.NAZWISKO, 3, 4), K.NAZWISKO
FROM DB2ADMIN.KLIENCI K;
```

W InterBase funkcję SUBSTR należy "uaktywnić". Polega to na zadeklarowaniu funkcji, która zostanie pobrana z zewnętrznej biblioteki dołączanej dynamicznie DLL. Aby funkcja ,.'." SUBSTR była aktywna w InterBase, wykonaj poniższe polecenie w Interactive SQL.

```
DECLARE EXTERNAL FUNCTION SUBSTR
CSTRING(80), SMALLINT, SMALLINT
RETURNS CSTRING(SO) FREE_IT
ENTRY_POINT 'IB_UDF_Substr' MODULE_NAME 'ib_udf.dll';
```

Po wykonaniu powyższego polecenia, możemy przejść do opcji IBConsole, aby zobaczyć tę funkcję, klikając w panelu po lewej stronie w ikonę External Function. Inaczej niż w DB2, w InterBase funkcja SUBSTR wybiera ciąg począwszy od pozycji podanej w drugim argumencie a skończywszy na trzecim argumencie. Zatem polecenie:

```
SELECT SUBSTR(K.NAZWISKO, 3, 4), K.NAZWISKO
FROM KLIENCI K;
```

Łączenie łańcuchów

Funkcja CONCAT pozwala łączyć ciągi znaków w jeden łańcuch wynikowy. Funkcja ta jest dostępna tylko w DB2. Poniższy przykład zapytania wyświetli listę klientów wraz z adresem zamieszkania. Taka lista może posłużyć jako źródło do korespondencji seryjnej.

```
SELECT K.IMIE CONCAT ' ' CONCAT K.NAZWISKO AS KLIENT,
'ul. ' CONCAT K.DLICA CONCAT ' ' CONCAT K. NUMER AS ULICA,
K. KOD CONCAT ' ' CONCAT K. MIASTO AS MIASTO
FROM DB2ADMIN.KLIENCI K
ORDER BY K.NAZWISKO;
```

Zamiast funkcji CONCAT można użyć znaków | |:

```
SELECT K.IMIĘ | | ' ' | | K.NAZWISKO AS KLIENT, ...
```

Wyrażenie CASE

Wyrażenie CASE pozwala na wybranie pewnej wartości w zależności od wartości w innej kolumnie. Wyrażenie CASE dostępne jest tylko w systemie DB2. W przykładzie poniżej sprawdzamy, czy klient pochodzi z Warszawy; jeżeli tak, to w kolumnie wpisywana jest wartość "Klient oddziału macierzystego", w przeciwnym razie jest to "Klient z przedstawicielstwa".

```
SELECT K.IMIE, K.NAZWISKO, K.MIASTO,
CASE K.MIASTO
WHEN 'WARSZAWA' THEN 'Klient oddziału macierzystego'
```

```
ELSE 'Klient z przedstawicielstwa'
END
FROM DB2ADMIN.KLIENCI K ORDER BY K.NAZWISKO;
```

Funkcje kolumnowe i grupujące

Poznamy funkcje operujące na kolumnach, które mogą być użyte w celu wydobycia wyników z jednego lub większej ilości wierszy Poznamy również zasady grupowania wierszy

Funkcje kolumnowe

Do funkcji kolumnowych zalicza się funkcje SUM, AVG, MIN, MAX oraz COUNT Funkcje te są używane w klauzulach SELECT lub HAVING

SUM - funkcja służąca do obliczenia sumy wartości w określonych kolumnach,
AVG - oblicza średnią wartości \v kolumnie,
MIN - znajduje minimalną wartość,
MAX - znajduje maksymalną wartość,
COUNT - śluzy do zliczania wystąpień pewnej wartości w wierszach

Poniższy przykład wyświetli całkowitą sumę wszystkich pensji pracowników, średnią pensję, minimalną i maksymalna pensję oraz ilość pracowników

```
SELECT SUM(P.PENSJA) AS PENSJA,
AVG(P.PENSJA) AS SREDNIA,
MIN (P.PENSJA) AS PENSJA_MIN,
MAX(P.PENSJA) AS PENSJA_MAX,
COUNT(*) AS ILOSC FROM DB2ADMIN PRACOWNICY P,
```

W poprzednim przykładzie funkcja COUNT została użyta do zliczenia wszystkich wierszy w tabeli (COUNT(*)), może być ona użyta również do zliczenia wierszy zawierających powtarzającą się wartość w kolumnie. W tym przykładzie zliczamy liczbę działów i stanowisk w firmie.

```
SELECT COUNT(DISTINCT P.DZIAL) AS ILOSC_DZIALOW,
COUNT(DISTINCT P.STANOWISKO) AS ILOSC_STANOWISK
FROM DB2ADMIN.PRACOWNICY P;
```

Stosowanie funkcji kolumnowych można przeprowadzić również na pewnym podzbiorze wierszy,

```
SELECT SUM(P.PENSJA) AS PENSJA,
AVG(P.PENSJA) AS SREDNIA,
MIN(P.PENSJA) AS PENSJA_MIN,
MAX(P.PENSJA) AS PENSJA_MAX,
COUNT(*) AS ILOSC FROM DB2ADMIN.PRACOWNICY P
WHERE P.DZIAL = 'OBSLUGA KLIENTA';
```

Klauzula GROUP BY

Klauzula GROUP BY grupuje wiersze o tej samej wartości wyszczególnionych kolumn. Funkcje agregujące SQL (AYG, MAX, MIN, SUM oraz COUNT) w klauzuli SELECT operują na każdej grupie osobno. Następujący przykład zapytania pogrupuje wiersze według stanowiska.

```
SELECT P.STANOWISKO, SUM (P.PENSJA) AS PENSJA,
AYG(P.PENSJA) AS SREDNIA,
MIN(P.PENSJA) AS PENSJA_MIN,
MAX(P.PENSJA) AS PENSJA_MAX,
COUNT(*) AS ILOSC
FROM DB2ADMIN. PRACOWNICY P
GROUP BY P.STANOWISKO
ORDER BY P.STANOWISKO;
```

Klauzula HAUING

Klauzula HAYING używana jest w połączeniu z klauzulą GROUP BY w celu ograniczenia wyświetlanych grup. Warunek szukania musi zawierać funkcję agregującą. Po zgrupowaniu wierszy przez klauzulę GROUP BY, klauzula HAYING wyświetla tylko te wiersze spośród zgrupowanych, które spełniają warunki wyszczególnione w klauzuli HAYING. Klauzula HAYING może być użyta tylko wówczas, gdy w zapytaniu znajduje się klauzula GROUP BY. Następny przykład zapytania wyświetla wszystkich pracowników, którzy wypożyczyli samochody na łączną jednostkową wartość powyżej 400 zł.

```
SELECT P.NAZWISKO, SUM (W.CENA_JEDN)
FROM DB2ADMIN.PRACOWNICY P,
DB2ADMIN.WYPOZYCZENIA W
WHERE P.NR_PRACOWNIKA = W.NR_PRACOW_WYP
```

```
GROUP BY P.NAZWISKO;
HAVING SUM(W.CENA_JEDN) > 400
ORDER BY P.NAZWISKO;
```

Klauzula UNION

Zapoznamy się z klauzulą UNION, która pozwala na łączenie dwóch lub więcej wyników wykonania zapytania SELECT. Poznamy składnię wyrażenia UNION, zasady dla listy w klauzuli SELECT oraz różnice między klauzulą UNION i UNION ALL.

Łączenie wielu wyników zapytania

Klauzula UNION łączy dwa lub więcej polecenia SELECT w jedną tabelę wynikową. Klauzula SELECT musi zwracać tę samą liczbę kolumn. Kolumny pokrywające się muszą mieć tę samą szerokość i typ danych. Nazwy tych kolumn mogą być różne. Klauzula UNION łączy dwa zestawy wyników w jeden i jednocześnie usuwa duplikaty. Poniższy rysunek ilustruje zastosowanie klauzuli UNION. Jak widać, powtarzające się wiersze na szarym tle zostały umieszczone tylko raz w końcowym wyniku zapytania z klauzulą UNION. W kolejnym przykładzie są zwracane dane o imieniu i nazwisku wszystkich klientów i pracowników, których nazwiska kończą się na "ski". Tylko jedna osoba o imieniu i nazwisku Jan Kowalski występuje jednocześnie w tabeli klientów i pracowników.

```
SELECT IMIE, NAZWISKO
FROM DB2ADMIN.KLIENCI
WHERE NAZWISKO LIKE '%SKI'
UNION
SELECT IMIE, NAZWISKO
FROM DB2ADMIN. PRACOWNICY
WHERE NAZWISKO LIKE '%SKI';
```

Za każdym razem zapytania łączące wyniki z klauzulą UNION wyświetlają wyniki posortowane rosnąco. Jeżeli chcemy zawrzeć klauzulę ORDER BY, która posortuje nam wynik malejąco, musi ona być umieszczona na końcu zapytania.

```
SELECT IMIE, NAZWISKO
FROM DB2ADMIN.KLIENCI
WHERE NA2WISKO LIKE '%SKI'
UNION
SELECT IMIE, NAZWISKO
FROM DB2ADMIN.PRACOWNICY
WHERE NAZWISKO LIKE '%SKI'
ORDER BY NAZWISKO DESC;
```

W systemie InterBase powyższe zapytanie należy zmodyfikować poprzez zastąpienie ostatniej klauzuli ORDER BY następującą:

```
ORDER BY 2 DESC;
```

InterBase nie pozwala w zapytaniach łączących wyniki na specyfikowanie nazwy kolumny w klauzuli ORDER BY.

Klauzula UNION ALL

Różnica pomiędzy klauzulą UNION a UNION ALL polega na tym, że wynik łączenia zapytań klauzulą UNION ALL zawiera powtarzające się wiersze. Klauzula UNION ALL działa szybciej niż UNION. Tak więc, gdy łączymy kilka wyników zapytania, i gdy jesteśmy pewni, że łączone wyniki nie zawierają duplikatów, możemy używać klauzuli UNION ALL.

Podzapytania

Znajdują się tutaj informacje, jak konstruować podzapytania, jak używać podzapytań w klauzuli WHERE oraz w klauzuli HAYING oraz jak budować podzapytania ze słowami kluczowymi IN, ALL, ANY lub SOME. **Używanie podzapytań**

Przypuśćmy, że musimy znaleźć pracowników, którzy otrzymują wynagrodzenie na kwotę większą niż wynosi średnia. Musimy najpierw sprawdzić, jaka jest średnia dla każdego pracownika.

```
SELECT AVG(P.PENSJA)
FROM DB2ADMIN.PRACOWNICY P;
```

Wynik wynosi: 1530,00

Teraz szukamy pracowników, którzy zarabiają poniżej tej średniej:

```
SELECT P.IMIE, P.NAZWISKO, P.DZIAL, P.STANOWISKO
FROM DB2ADMIN.PRACOWNICY P WHERE P.PENSJA > 1530;
```

Wykonaliśmy zadanie. Znaleźliśmy pracowników, którzy zarabiają powyżej średniej. Ale dokonaliśmy tego w dwóch krokach za pomocą dwóch zapytań. Teraz otrzymamy ten sam wynik, ale przy użyciu podzapytania.

```
SELECT P,IMIE, P.NAZWISKO, P.DZIAL, P.STANOWISKO
```

```
FROM DB2ADMIN.PRACOWNICY P
WHERE P.PENSJA > (SELECT AVG(P.PENSJA)
FROM DB2ADMIN.PRACOWNICY P);
```

Podzapytania z użyciem słowa kluczowego IN

Słowo kluczowe IN pozwala na zidentyfikowanie wszystkich elementów w zbiorze A które nie występują w zbiorze B. Zapytanie wyświetla listę samochodów, których do tej pory nie wypożyczył żaden klient. Zapytanie wybiera te samochody, które nie znajdują się w tabeli WYPOŻYCZENIA, czyli te, które nie były do tej pory przedmiotem wypożyczenia.

```
SELECT S.NR_SAMOCHODO, S.MARKA, S,TYP
FROM D32ADMIN.SAMOCHODY S
WHERE S.NR_SAMOCHODU
NOT IN
(SELECT W.NR_SAMOCHODU
FROM DB2ADMIN.WYPOZYCZENIA W);
```

Podzapytania z użyciem słowa kluczowego ALL

Przykładowe podzapytanie ze słowem ANY będzie wykonane w dwóch krokach. Jako pierwsze jest wykonywane podzapytanie, które znajduje średnią pensję w każdym dziale. W drugim kroku, każda pensja pracownika porównywana jest z listą średnich pensji. Wyświetleni zostaną pracownicy, których pensja jest wyższa od wszystkich średnich pensji obliczonych w podzapytaniu.

Podzapytania w klauzuli HAVING Musimy znaleźć działy, w których średnia pensja pracowników jest wyższa od średniej pensji w firmie. Do średnich pensji nie będą brani pod uwagę kierownicy działów. Gdybyśmy musieli wykonać to zadanie "ręcznie", to musielibyśmy przejść przez trzy kroki. W pierwszym kroku musielibyśmy znaleźć średnią pensję w firmie, nie biorąc pod uwagę kierowników.

```
SELECT AVG(P.PENSJA)
FROM DB2ADMIN.PRACOWNICY P
WHERE P.STANOWISKO <> 'KIEROWNIK';
```

W drugim kroku obliczylibyśmy średnie pensje pracowników w poszczególnych działach, nie biorąc przy tym pod uwagę kierowników.

```
SELECT P.DZIAL, AVG(P.PENSJA) AS SREDNIA_PENSJA
FROM DB2ADMIN.PRACOWNICY P
WHERE P.STANOWISKO <> 'KIEROWNIK'
GROUP BY P.DZIAL
ORDER BY SREDNIA_PENSJA;
```

Jeżeli używasz InterBase, zamień ostatni wiersz powyższego polecenia na:

```
ORDER BY 2;
```

W trzecim kroku musielibyśmy porównać wartości średnich pensji poszczególnych działów ze średnią pensją w firmie. Ostatecznie wykonujemy to zadanie za pomocą pojedynczego zapytania z podzapytaniem w klauzuli HAVING.

```
SELECT P.DZIAL, AVG(P.PENSJA) AS SREDNIA_PENSJA
FROM DB2ADMIN.PRACOWNICY P
WHERE P.STANOWISKO <> 'KIEROWNIK'
GROUP BY P.DZIAL
HAVING AVG(P.PENSJA) > (SELECT AVG(P.PENSJA)
FROM DB2ADMIN.PRACOWNICY P
WHERE P.STANOWISKO <> 'KIEROWNIK') ORDER BY SREDNIA_PENSJA;
```

Utrzymywanie danych

Nauczymy się tworzyć tabele i widoki. Poznamy składnię języka SQL niezbędną do ich tworzenia. Nauczymy się również wstawiać wiersze do tabeli, zmieniać dane w tabeli, usuwać wiersze oraz usuwać tabele. **Tworzenie tabel**

Następujące wyrażenie CREATE TABLE tworzy tabelę KLIENCI_TEST.

```
CREATE TABLE DB2ADMIN.KLIENCI_TEST (
NR_KLIENTA CHAR(8) NOT NULL,
IMIE VARCHAR(20) NOT NULL,
NAZWISKO VARCHAR(20) NOT NULL,
NR_KARTY_KREDYT CHAR(20) ,
ULICA VARCHAR(24) NOT NULL,
NUMER CHAR(8) NOT NULL,
MIASTO VARCHAR(24) NOT NUIi,
KOD CHAR(6) NOT NULL,
NRJTELEFONU CHAR (16),
PRIMARY KEY (NR_KLIENTA) ) ;
```

Definiując tabelę musimy określić jej nazwę np. KLIENCI_TEST. Następnie określić kolumny dla tej tabeli. Każda kolumna musi posiadać: unikatową nazwę w obrębie tabeli oraz typ danych, jakie będą przechowywane w kolumnie. Dodatkowo przy definiowaniu kolumn określić można, czy dozwolone jest pozostawienie jej pustej; jeżeli nie, dodajemy klauzulę NOT NULL do definicji kolumny. Np. kolumna NR_KARTY_ KREDYT nie jest wymagana - podczas wstawiania nowego wiersza - pole w tej kolumnie możemy pozostawić puste. Może dziś (prawie) każdy posiada kartę płatniczą, ale nie każdy posiada kartę kredytową. Dodatkowo nie każdy klient ma życzenie płacić kartą kredytową. Słowo kluczowe PRIMARY KEY określa klucz główny dla tabeli. Klucz główny oraz klucz obcy zostanie opisany w następnym rozdziale. Tabelę możemy przebudować, dodając nową kolumnę lub ją usuwając, możemy zmienić typ danych kolumny, jak również zmienić inne cechy tabeli oraz kolumn w niej zawartych. Do zmiany struktury tabeli służy wyrażenie SQL ALTER TABLE. Kolejne polecenie ALTER TABLE doda dwie kolumny: FIRMA oraz NIP do tabeli KLIENCI_TEST.

```
ALTER TABLE DB2ADMIN.KLIENCI_TEST
ADD FIRMA VARCHAR(40)
ADD NIP CHAR(12) ;
```

W InterBase kolejne wiersze ze słowem ADD w powyższym poleceniu należy oddzielić przecinkiem. Aby zapobiec błędom, musimy wykonać polecenie ALTER TABLE. Następne przykłady będą operować również na tych kolumnach.

Tworzenie widoków

Dane zawarte w widoku nie są jej fizycznymi danymi a danymi należącymi do tabeli lub kilku tabel z których widok czerpie dane. Widoki przede wszystkim są tworzone w celu ograniczenia dostępu do danych w tabelach bazy danych. Do tworzenia widoków służy polecenie CREATE VIEW. Poniższy przykład tworzy widok zawierający dane klientów, którzy posiadają firmę.

```
CREATE VIEW DB2ADMIN.KLIENCI_FIRMY AS
SELECT K.IMIE, K.NAZWISKO, K.FIRMA, K.NIP, K.MIASTO
FROM DB2ADMIN.KLIENCI K
WHERE K.FIRMA IS NOT NULL;
```

Teraz możemy wybierać dane z widoku tak, jak do tej pory wybieraliśmy dane z tabeli.

```
SELECT *
FROM DB2ADMIN.KLIENCI_FIRMY;
```

Następny przykład tworzy widok, który ogranicza dane pracowników do wszystkich danych oprócz informacji na temat dodatku i pensji.

```
CREATE VIEW DB2ADMIN.V__PRACOWNICY AS
SELECT P.NR_PRACOWNIKA, P.IMIE, P.NAZWISKO,
P.DATA_ZATR, P.DZIAL, P.STANOWISKO,
P.NR_MIEJSCA, P.NRJTELEFONU
FROM DB2ADMIN.PRACOWNICY P;
```

Dodawanie i usuwanie rekordów

Aby dodać jeden lub więcej rekordów do istniejącej tabeli, należy posłużyć się wyrażeniem SQL INSERT. Aby dodać rekord do tabeli KLIENCI_TEST zdefiniowanej w sekcji "Tworzenie tabel", napisz i wykonaj poniższe wyrażenie SQL. Upewnij się, że tabela KLIENCIJTEST posiada kolumny FIRMA oraz NIP, które dodaliśmy do struktury tabeli poleceniem ALTER TABLE.

```
INSERT INTO DB2ADMIN.KLIENCI_TEST
VALUES ( '00000031', 'MARIUSZ', 'DOLATA', NULL, 'KOCHANOWSKIEGO', '3', 'WROCŁAW', '37-300', '167-763-234', 'KWIATY', '2224-444-224');
```

Dodaj jeszcze kilka rekordów:

```
INSERT INTO DB2ADMIN.KLIENCI_TEST YALUES ('00000032', 'TOMASZ', 'DOMAGAŁA' , 'HX 145345678', 'RÓŻANA', '4/9', 'WARSZAWA', '01-900', '46-744-431', NULL, NULL);
INSERT INTO DB2ADMIN.KLIENCI_TEST

VALUES ('00000033', 'PAWEŁ', 'MALCZYKOWSKI' , 'HF 14565661', 'SŁONECZNA', '9', 'WARSZAWA1, '01-900', '16-742-114', NULL, NULL); INSERT INTO DB2ADMIN.KLIENCI_TEST VALUES ('00000034', 'PIOTR', 'MUSZYŃSKI' , 'DD 72325221', 'SZYBOWCOWA', '22A', 'WARSZAWA', '01-200', '44-342-116', 'WULKANIZACJA', '4356-098-876');
INSERT INTO DB2ADMIN.KLIENCI_TEST
VALUES ('00000035', 'ANNA', 'MIKOLAJCZYK' ,NULL , 'JAŁOWCOWA', '24', 'WROCŁAW', '37-200', '144-188-415', 'FRYZJERSTWO', '2343-112-345');
```

Powyższe wyrażenia dodały nowe wiersze do tabeli KLIENCI_TEST. Każde z tych wyrażeń wypełnia wartościami wszystkie kolumny tabeli. Aby wstawić dane tylko do wybranych kolumn, należy je określić, a następnie podać wartości:

```
INSERT INTO DB2ADMIN. KLIENCI_TEST {NR_KLIENTA, IMIE, NAZWISKO, ULICA, NUMER, MIASTO, KOD)
VALUES ('00000036', 'MAGDALENA', 'BRZOZA' , 'ALEJE LIPOWE', '4/3', _ 'ŚWIDNICA, '58-100');
```

Powyższe polecenie INSEKT dodało nowy wiersz do tabeli KLIENCI_TEST. Wypełnione zostały wszystkie kolumny oprócz kolumny NR_KARTY_KREDYT i kolumny TELEFON. Wartości dla tych kolumn nie są wymagane więc wstawienie nowego wiersza przebiegło bez błędu. Istnieje możliwość dodania wielu wierszy za jednym razem. Wstawienie kilku rekordów w jednym poleceniu polega na użyciu klauzuli SELECT. Oto przykład:

INSERT INTO DB2ADMIN.KLIENCI_TEST (NR_KLIENTA, IMIE, NAZWISKO, ULICA, NUMER, MIASTO, KOD)
SELECT NR_KLIENTA, IMIE, NAZWISKO, ULICA, NUMER, MIASTO, KOD
FROM DB2ADMIN.KLIENCI
WHERE FIRMA IS NULL;
Aby usunąć rekordy z tabeli, użyj polecenia DELETE FROM np.
DELETE FROM DB2ADMIN.KLIENCI_TEST WHERE FIRMA IS NOT NULL;
Polecenie DELETE FROM bez klauzuli WHERE usuwa wszystkie rekordy z tabeli, np.
DELETE FROM DB2ADMIN.KLIENCI_TEST;

Zmienianie danych w tabeli

Polecenie UPDATE zmienia wartości we wskazanych kolumnach tabeli dla jednego lub większej ilości wierszy. Poniższe polecenie UPDATE zwiększa kwotę dodatku pracownika zatrudnionego na stanowisku sprzedawcy o 50 zł.
UPDATE DB2ADMIN.PRACOWNICY
SET DODATEK = DODATEK + 50
WHERE STANOWISKO = 'SPRZEDAWCA';
Teraz możemy sprawdzić, czy wartości dodatku dla sprzedawców zostały zmienione:
SELECT *
FROM DB2ADMIN.PRACOWNICY
WHERE STANOWISKO = 'SPRZEDAWCA';
Jeżeli zmieniamy wartości więcej niż jednej kolumny, muszą one być oddzielone przecinkiem. Poniższe polecenie zwiększa dodatek dla kierowników o 30 zł oraz zwiększa pensje o 10%.
UPDATE DB2ADMIN.PRACOWNICY
SET DODATEK = DODATEK + 30,
PENSJA = PENSJA + (PENSJA *10) /100
WHERE STANOWISKO = 'KIEROWNIK';

Usuwanie tabel

Tabela KLIENCI_TEST nie będzie nam już więcej potrzebna. Aby usunąć tabelę, musimy użyć polecenia DROP TABLE:
DROP TABLE KLIENCI TEST;
Polecenie usuwające tabelę usuwa jednocześnie wszystkie dane zawarte w tabeli oraz usuwa wszystkie widoki które czerpią dane z usuwanej tabeli.

Ograniczenia i integralność referencyjna

Dowiemy się istotnych informacji o ograniczeniach, integralności danych tabeli oraz o integralności referencyjnej. Wszystkie te zagadnienia składają się na bezpieczeństwo i jakość danych gromadzonych w bazie danych. **Ograniczenia**

Możesz zdefiniować ograniczenie sprawdzające poprawność wpisywanych danych do tabeli poprzez określenie warunku sprawdzającego CHECK. Poniższy przykład ilustruje wyrażenie zmieniające strukturę tabeli PRACOWNICY poprzez dodanie ograniczenia zapobiegającego wpisaniu kwoty dodatku większej od kwoty pensji.
ALTER TABLE DB2ADMIN.PRACOWNICY ADD CHECK (PENSJA > DODATEK);
Jeżeli wpiszesz teraz wyrażenie dodające wiersz do tabeli pracownicy, który będzie zawierał w kolumnie DODATEK wartość większą niż w kolumnie PENSJA np.
INSERT INTO DB2ADMIN.PRACOWNICY
VALUES ('0011', 'JOLANTA', 'NOWAKOWSKA1, '1999-05-01', 'OBSLUGA KLIENTA', 'SPRZEDAWCA', 1100, 1200, '000001' , '433-451-154') ;
Baza DB2 wygeneruje komunikat o błędzie, który mówi o naruszeniu ograniczenia sprawdzającego CHECK:
DB21034E The command was processed as an SQL statement
because it was not a valid Comnand Linę Processor command.
During SQL processing it returned:
SQL0545N The reąuested operation is not allowed because a row does not satisfy the check constraint "DB2ADMIN.PRACOWNICY.SQLQ10121215529810". SQLSTATE=23513

Integralność danych - klucz główny

Każda tabela bazy danych powinna zawierać klucz główny. Klucz główny tabeli to kolumna lub grupa kolumn, która w sposób jednoznaczny identyfikuje wiersz w tabeli. Na przykład, dla tabeli zawierającej dane o pracownikach kluczem głównym może być, kolumna o nazwie NR_PRACOWNIKA, która jednoznacznie określa danego pracownika. Kluczem głównym może być numer telefonu w tabeli przechowującej dane abonentów operatora telefonicznego. Jak już wspomniałem, klucz główny może składać się z wielu kolumn. Przykładem takiego klucza głównego może być kolumna NUMER oraz ROK w tabeli przechowującej dane o wystawionych fakturach, gdzie kolumna NUMER określa numer faktury a kolumna ROK określa rok wystawienia. Wartości z tych kolumn wzięte razem są różne w każdym wierszu. Jak już wspomniałem, dla tabeli PRACOWNICY kluczem głównym może być kolumna NR_PRACOWN1KA. Ustalenie klucza głównego (PRIMARY KEY) podczas tworzenia tabeli:

```
CREATE TABLE DB2ADMIN. PRACOWNICY (
NR_PRACOWNIKA CHAR(4)NOT NULL,
IMIE VARCHAR(20) NOT NULL,
NAZWISKO VARCHAR(20) NOT NULL,
DATA_ZATR DATE NOT NULL,
DZIAL VARCHAR(20) NOT NULL,
STANOWISKO VARCHAR(20) NOT NULL,
PENSJA DECIMAL(8,2),
DODATEK DECIMAL(8,2)
NR_MIEJSCA CHAR(6) NOT NULL,
NRJTELEFONU CHAR(16)
PRIMARY KEY (NR_PRACOWNIKA));
```

zapobiegnie wstawieniu dwóch identycznych wierszy. W przypadku gdy dodamy drugi wiersz z danymi pracownika o numerze już istniejącym w tabeli, DB2 wyświetli błąd z informacją o naruszeniu integralności danych. DB21034E The command was processed as an SQL statement because it was not a valid Command Linę Processor command. During SQL processing it returned:SQL0803N One or morę values in the INSERT statement, UPDATE statement, or foreign key update caused by a DELETE statement arę not valici because they would produce duplicate rows for a table with a primary key, uniąue constraint,or uniąue index. SQLSTATE=23505

Integralność refereicyjna - klucz obcy

Klucz obcy to jedna lub więcej kolumn tabeli odwołujących się do kolumny lub kolumn klucza głównego w innej tabeli. Klucze obce są wykorzystywane do utrzymywania integralności referencyjnej w bazie danych. Tworząc klucz obcy, definiujemy związek między tabelą klucza głównego i tabelą klucza obcego. Związek taki powstaje podczas złączania kolumn takich samych typów danych z każdej tabeli. Złączanie tabel przez odpowiednie kolumny chroni dane z tabeli klucza obcego przed "osieroceniem", jakie mogłoby nastąpić w wyniku usunięcia odpowiadających im danych z tabeli klucza głównego. Definiowanie kluczy obcych jest po prostu sposobem łączenia danych przechowywanych w różnych tabelach bazy danych. Związek klucza obcego chroni wiersze z tabeli PRACOWNICY przed osieroceniem na wypadek usunięcia jakiegokolwiek wiersza z tabeli MIEJSCA. Aby zapewnić taką ochronę, musimy zdefiniować klucze obce we wszystkich tabelach, które odwołują się do innych tabel. Taki związek występuje m.in. w naszych przykładowych tabelach PRACOWNICY oraz MIEJSCA. ALTER TABLE DB2ADMIN.PRACOWNICY
ADD POREIGN KEY (NR_MIEJSCA)
REFERENCES MIEJSCA (NR MIEJSCA) ON DELETE RESTRICT;

Polecenie to ustanawia klucz obcy w tabeli PRACOWNICY w kolumnie NR_ MIEJSCA. Czytając dalej to polecenie dowiadujemy się że kolumna ta odwołuje się do kolumny NR_MIEJSCA w tabeli MIEJSCA. Słowa kluczowe ON DELETE RESTRICT mówią, że niemożliwe jest usunięcie wiersza z tabeli MIEJSCA, gdy istnieje wiersz do niego się odwołujący w tabeli PRACOWNICY. Dla systemu InterBase zamiast słowa RESTRICT jest honorowane słowo NO ACTION

Tu znajdują się opisy wszystkich możliwych akcji, jakie zostaną zainicjowane w chwili usuwania wiersza w tabeli zależnej:

RESTRIC : Ograniczone usuwanie, które mówi, że dopóki istnieją w tabeli PRACOWNICY wiersze odwołujące do usuwanego adresu lub dla InterBase

NO ACTION : nie można go usunąć. Aby usunąć dane o adresie z tabeli MIEJSCA, najpierw należy usunąć wszystkich pracowników pracujących w miejscu o którym informacje chcemy usunąć

CASCADE : kaskadowe usuwanie, mówi, że gdy usuwamy wiersze z tabeli MIEJSCA, to są jednocześnie usuwane wszystkie wiersze z danymi o pracownikach, którzy pracują w usuwanym miejscu

SET NULL : Wstaw wartość NULL, mówi, że jeśli usuwamy dane o miejscach, to w tabeli PRACOWNICY w kolumnie NR_MIEJSCA zostanie wstawiona wartość NULL

Narzędzia DB2

Poznamy takie narzędzia jak: Control Center, Command Center, Command Linę Procesor oraz Information Center. Control Center jest aplikacją pozwalającą na zarządzanie obiektami bazy danych. Narzędzie Command Center będziemy wykorzystywać do przygotowywania zapytań SQL i ich wykonywania. To narzędzie pozwala również na wykonywanie skryptów SQL jak również na przeglądanie wyników wykonania zapytania. Narzędzie Command Linę Procesor (CLP) służy do wykonywania poleceń systemowych DB2. Information Center jest systemem pomocy z bardzo wygodnym interfejsem.

Control Center

Control Center jest aplikacją, która pozwala na przeglądanie, dodawanie, usuwanie i zmienianie obiektów baz danych zdefiniowanych w DB2. Tymi obiektami sąm.in. tabele i widoki. Dzięki temu narzędziu możemy zbudować całą bazę danych, nie używając języka SQL. W liście tabel, oprócz tabel bazy danych WYPAUT, znajdują się również tabele systemowe, z których można wydobyć informacje na temat struktury bazy danych. Tabela SY-

SIBM.SYSTABLES zawiera wszystkie tabele zdefiniowane w bazie danych. Tabela SY-SIBM.SYSCOLUMNS zawiera wszystkie informacje o kolumnach zdefiniowanych we wszystkich tabelach bazy danych. Tabela SYSIBM.SYSYIEWS zawiera informacje o widokach zdefiniowanych w bazie danych. Panel po lewej stronie okna zawiera informacje o systemie. Ikona Systems wskazuje na nazwę komputera (WROR-JAKUBOAR). Ikona Instances zawiera instancje DB2 zainstalowane w systemie. Można definiować wiele instancji np. w celu oddzielenia bazy testowej od produkcyjnej. Następna ikona -Databses zawiera bazy danych (w tym przypadku jedna - WYPAUT). Baza WYPAUT z kolei skupia wszystkie obiekty typu tabele, widoki, indeksy, itd.

Command Center

Aplikacja Command Center będzie najczęściej wykorzystywanym narzędziem przy studiowaniu niniejszych ćwiczeń. Na poniższym zdjęciu aplikacji Command Center widzimy zakładki lnteractive, Script, Ouery Results oraz Access Plan, których przeznaczenie zostanie opisane w następnych sekcjach.

Przygotowywanie zapytań SQL i ich wykonywanie

Podczas budowy zapytań przy bieżącej pracy z bazą danych będziemy korzystać z zakładki lnteractive. Aby wykonać zapytanie, musimy się upewnić, że jesteśmy podłączeni do bazy danych, na której chcemy pracować. Są dwa sposoby na podłączenie się do bazy danych. Jeden z nich to wpisanie polecenia SQL:

CONNECT TO WYPAUT USER db2admin USING db2admin;

w oknie Command na zakładce lnteractive. Po naciśnięciu kombinacji klawiszy Ctrl+Enter powyższe polecenie zostanie wykonane i zostaniemy podłączeni do bazy danych. Zostanie to zakomunikowane w oknie poniżej okna Command następującym komunikatem:

```
---------- Command Entered ----------
CONNECT TO WYPAUT USER db2admin USING ******** o
-------------------------------------------
Database Connection Information
Database server = DB2/NT 7.1.0
SQL authorization ID = DB2ADMIN
Local database alias = WYPAUT
```

Oprócz tego komunikatu, w polu Database connection zobaczymy wpis informujący o aktualnym połączeniu. Można to zobaczyć na zdjęciu (JAKUB - DB2 - WYPAUT). Drugi sposób podłączenia się do bazy danych polega na wybraniu z okna Select Database konkretnej bazy danych. Okno Select Database wywołuje się przez naciśnięcie myszką klawisza z trzema kropkami znajdującego się po prawej stronie pola Database connection.

Teraz gdy jesteśmy podłączeni do bazy WYPAUT, możemy wydawać inne polecenia lub zapytania SQL w oknie Command.

Wykonywanie skryptów SQL

Przed tym, jak zaczniemy pracować z ćwiczeniami musimy, po stworzeniu bazy danych, utworzyć tabele i wypełnić je danymi. Możemy to wykonać poprzez okno Com-mand na zakładce lnteractive lub poprzez wykonanie skryptów uprzednio stworzonych. Wpisywanie wszystkich poleceń tworzących tabele oraz poleceń wstawiających dane jest zbyt czasochłonne. Polecam wykonanie skryptów, które zostały zamieszczone na serwerze ftp wydawnictwa. Aby wykonać skrypt, musimy go otworzyć i uruchomić. Wybieramy w tym celu menu Script Import. Pojawi się okno. W tym oknie musimy najpierw wybrać komputer, na którym znajdują się skrypty poprzez rozwinięcie listy System name i wybranie konkretnego systemu. Następnie przechodzimy do katalogu ze skryptami i pojedynczo je otwieramy .Zawartość skryptu zostanie wyświetlona w oknie Script na zakładce Script. Aby wykonać skrypt naciskamy kombinację klawiszy Ctrl+Enter lub wybieramy myszką przycisk Execute znajdujący się pod menu głównym z lewej strony okna. Zobaczymy serię komunikatów u dołu okna mówiącym o tym, że wykonanie poszczególnych poleceń SQL w skrypcie zostało zakończone pomyślnie, np.

DB20000I The SQL coinmand completed successfully-

Wyświetlanie wyników wykonania zapytania

Wyniki wykonania zapytań SQL, które zostały wprowadzone na zakładce lnteractive są wyświetlane na zakładce Ouery Results aplikacji Command Center. Wyniki zapytań uruchomionych z poziomu zakładki Script są z kolei wyświetlane u dołu w tym samym oknie.

Command line Processor

Command Linę Processor pozwala na wykonywanie poleceń systemowych DB2. Do poleceń systemowych zalicza się również polecenie CREATE DATABASE tworzące bazę danych. Inne polecenia służą do wyświetlania parametrów systemu DB2 i ustawianiu tychże parametrów. Na rysunku 12.6 znajduje się zdjęcie okna Command Linę Processor z wydanym poleceniem LI ST ACTIVE DATABASES. Poniżej znajdują się opisy niektórych poleceń systemowych DB2.Możesz również uzyskać krótką podpowiedz na temat konkretnego polecenia, poprzedzając ją znakiem zapytania w Command Linę Processor, np.

```
db2 ==> ? CONNECT TO
CONNECT TO database-alias - .- ..
[IN {SHARE MODĘ l EXCLUSIVE MODĘ [ON SINGLE NODE]}]
[USER username [{USING password
[NEW new-password CONFIRM confirm-password] |
CHANGE PASSWORD}]]
```

CONNECT TO -łączy aplikację do bazy danych

Przykład:

CONNECT TO WYPAUT USER db2admin USING db2admin
przyłącza aplikację do bazy danych WYPAUT. Parametry USER oraz USING pozwalają określić użytkownika bazy danych oraz hasło.
CREATE DATABASE - tworzy bazę danych.
Przykład:
CREATE DATABASE WYPAUT
stworzy bazę danych z domyślnymi wartościami parametrów bazy.
DB2START/DB2STOP - startuje (lub zatrzymuje) menedżera bazy danych.
DROP DATABASE - usuwa bazę danych z systemu.
Przykład:
DROP DATABASE WYPAUT
GET CONNECTION STATE - wyświetla informacje o stanie połączenia z bazą danych.
Jeżeli jesteśmy podłączeni do bazy WYPAUT poleceniem CONNECT TO WYPAUT, to wydanie polecenia GET CONNECTION STATE spowoduje wyświetlenie podobnego komunikatu:
Database Connection State
Connection state = Connectable and Connected
Connection modę = SHARE
Local database = WYPAUT
alias
Database name = WYPAUT
GET INSTANCE - wyświetla informacje o instancji bazy danych zainstalowanej w systemie.
LIST ACTIYE DATABASES - wyświetla informacje o aktywnych bazach danych, przyłączonych do nich aplikacjach i o ścieżce dostępu do zbiorów w których przechowywane są dane z bazy danych.
LIST APPLICATIONS - wyświetla informacje o aktywnych aplikacjach podłączonych do bazy danych.
LIST DATABASE DIRECTORY - wyświetla informacje o systemowym katalogu DB2, w którym przechowywane są wszystkie informacje o bazach danych. Na wydruku widać, że jedyną bazą, jaka została utworzona do tej pory jest baza WYPAUT. Katalog D:\DB2 to miejsce, gdzie przechowywany jest systemowy katalog.
System Database Directory
Number of entries in the directory = l
Database l entry:
Database alias = WYPAUT
Database name = WYPAUT
Database drive = D:\DB2
Database release level =9.00
Comment =
Directory entry type = indirect
Catalog node number = O
QUIT - powoduje zamknięcie sesji z Command Line Processor.

Tworzenie bazy

Polecenie systemowe CREATE DATABASE tworzy nową bazę danych. Dla potrzeb naszych ćwiczeń musimy taką bazę stworzyć. Ponieważ polecenie CREATE DATABASE należy do poleceń systemowych DB2, musimy je wprowadzić i wykonać w aplikacji Command Linę Processor.

Ustawienia narzędzi DB2

Do poprawnej pracy w aplikacji Command Center, a w szczególności do wykonywania skryptów musimy zmienić pewne domyślne parametry. Z poziomu aplikacji Control Center lub z np. aplikacji Command Center wybieramy menu Tools | Tools Settings. Pojawi się okno Na zakładce General musimy zaznaczyć opcję Use statement termination character, która stanowi o tym, że znak średnika będzie znakiem oddzielającym poszczególne wyrażenia SQL wprowadzane m.in. w aplikacji Command Center.

Information Center

Information Center jest aplikacją wspomagającą użytkownika w wyszukiwaniu pomocnych informacji. Jest to pewnego rodzaju system pomocy. Jego budowa i organizacja pozwala na szybkie wyszukanie potrzebnych informacji. Znajdziemy tutaj pełny opis składni języka SQL, jak również opis poleceń systemowych DB2. Znajdują się tam również odpowiednie łącza do stron WWW firmy IBM, gdzie można znaleźć dodatkowe informacje. Polecam używanie tej aplikacji za każdym razem, gdy istnieje potrzeba sprawdzenia składni danego polecenia lub np. odczytania informacji o błędzie.

InterBase

Do pracy z ćwiczeniami potrzebny nam jest jeden plik który jest wersją Server i Client InterBase dla Windows

Narzędzie IBConsole

IBConsole jest odpowiednikiem narzędzia DB2 Control Center. Tutaj również mamy możliwość podglądania obiektów bazy danych, jakimi są m.in. tabele i widoki. Przede wszystkim IBConsole jest narzędziem, w którym możemy stworzyć bazę danych. Z poziomu IBConsole możemy również wywołać narzędzie lnteractive SQL, które pozwala na wykonywanie poleceń SQL.

Tworzenie bazy danych w InterBase

Jeżeli nie jesteśmy w aplikacji IBConsole, musimy j ą uruchomić z Menu Start | Programy | Interbase | IBConsole. Z menu Server wybieramy pozycję Login. W oknie, które się pojawi wpisujemy użytkownika SYSDBA i hasło masterkey. Po załogowaniu się do menedżera bazy InterBase przejdź do menu Database do pozycji Create Database Aby stworzyć bazę danych, wypełnij to okno. W końcu naciśnij przycisk OK. Baza została utworzona. Teraz możemy przejść do wykonywania skryptów, które utworzą tabele w bazie danych i wypełnią je danymi. Opis wykonywania skryptów znajduje się w następnej sekcji.

Narzędzie InterBase Manager

InterBase Manager można wywołać z menu Start | Programy l InterBase InterBase Server Manager. Pozwala on m.in. na ustalenie czy serwer InterBase ma być wywoływany automatycznie przy starcie systemu operacyjnego.

Narzędzie Interactiue SQ1

Narzędzie lnteractive SQL pozwala na wprowadzanie poleceń SQL i ich wykonywania na bazie danych. Wywołuje sieje z poziomu aplikacji IBConsole z menu Tools | lnteractive SQL Wykonywanie wprowadzonych poleceń SQL dokonuje się przez naciśnięcie kombinacji klawiszy Ctrl+E (Execute). Czasami gdy będziemy wychodzić z Interactive SQL, będziemy pytani, czy zatwierdzić transakcję. Transakcjajest każda operacja na danych w bazie danych. Transakcja musi się wykonać w całości lub zostać wycofana. Oto okno dialogowe z pytaniem, czy zatwierdzić transakcję. Wszystkie polecenia SQL zawarte w tych ćwiczeniach wymagają, aby ich działanie było zatwierdzane Aplikacja Interactive SQL umożliwia również wykonywanie skryptów SQL. Wykonywanie skryptów SQL zostało opisane w kolejnej sekcji.

Wykonywanie skryptów

Aby wykonać skrypty tworzące tabele i wypełniające je danymi, musimy przejść do menu Query w lnteractive SQL i wybrać pozycję Load Script. Okno, które się pojawi pozwala na wybranie pliku skryptu. Po naciśnięciu kombinacji klawiszy Ctrl+E skrypt zostanie wykonany i utworzona zostanie tabela KLIENCI. Pozostałe skrypty również muszą zostać wykonane do utworzenia całej struktury bazy danych. Skrypt zostanie wykonany wtedy, gdy jesteśmy podłączeni do bazy WYPAUT. Na zdjęciu powyżej w pasku stanu na samym dole jest wyświetlona informacja, że baza, do której jesteśmy aktualnie podłączeni to WYPAUT.

Struktura przykładowej bazy danych

Przykładowa baza wypożyczalni samochodów WYPAUT składa się z pięciu tabel. Przechowuje ona dane o klientach, pracownikach, samochodach, miejscach, z których samochody można wypożyczyć oraz dane o wypożyczeniach. Każde wypożyczenie jest odnotowywane w tabeli WYPOŻYCZENIA. Każdy: klient, samochód, miejsce wypożyczenia i oddania, pracownik wypożyczający i przyjmujący posiada numer, po którym jest identyfikowany w tabeli WYPOŻYCZENIA. Pojedynczy rekord z tabeli WYPOŻYCZENIA opisuje jedno wypożyczenie samochodu. Tak więc, gdy odczytujemy ten rekord, możemy odnaleźć dane o kliencie, który wypożyczył dany samochód, dane o pracowniku obsługującym klienta oraz o miejscu wypożyczenia i oddania samochodu.

Opis tabel

Szczegółowy opis tabel wchodzących w skład przykładowej bazy danych wypożyczalni samochodów.

Tabela KLIENCI

Tabela KLIENCI przechowuje dane na temat klientów wypożyczających samochody. Między innymi na podstawie tych danych może zostać wystawiona faktura. **Tabela SAMOCHODY**

Tabela SAMOCHODY zawiera informacje o dostępnych samochodach, które klient może wypożyczyć.

Tabela PRACOWNICY

Tabela PRACOWNICY zawiera dane wszystkich pracowników firmy wypożyczającej samochody.

Tabela MIEJSCA

W tabeli MIEJSCA znajdują się informacje o miejscach, z których klient wypożyczył samochód, oraz informacje o miejscach oddania

Relacje pomiędzy tabelami

Jeden KLIENT może dokonać wielu WYPOŻYCZEŃ niekoniecznie w tym samym czasie. Jeden SAMOCHÓD może być WYPOŻYCZANY wielokrotnie. Jeden PRACOWNIK może obsłużyć wiele WYPOŻYCZEŃ. Samochód może zostać WYPOŻYCZONY/ODDANY wielokrotnie w różnych MIEJSCACH.

Skrypty tworzące strukturę bazy WYPAUT

W następnych sekcjach znajdują się listingi skryptów tworzących tabele bazy WYPAUT. Skrypty te jednocześnie wypełniają tabele przykładowymi danymi. Poniższe skrypty zostały przygotowane do wykonania w systemie DB2. Aby wykonać je w InterBase musimy:

- usunąć wiersz, który łączy się z bazą danych CONNECT TO...;
- usunąć wiersz, który usuwa tabelę DROP TABLE, ponieważ InterBase przerywa przetwarzanie skryptu, gdy

wystąpi błąd. Taki Wad wystąpi, gdy po raz pierwszy uruchomimy skrypt. Polega on na usuwaniu tabeli, która jeszcze nie istnieje;
- usunąć kwalifikatory DB2ADMIN przed nazwą tabeli w poleceniach CREATE TABLE oraz w poleceniach INSERT.
Fragment polecenia SQL tworzącego tabelę oraz polecenia wstawiającego wiersz w InterBase powinien wyglądać tak:
CREATE TABLE KLIENCI (...
...INSERT INTO KLIENCI VALUES (. . .

Struktura przykładowej bazy danych

Skrypt tworzący tabelę KLIENCI i wypełniający ją danymi

```
CONNECT TO WYPAUT USER DB2ADMIN USING DB2ADMIN;
DROP TABLE DB2ADMIN.KLIENCI;
CREATE TABLE DB2ADMIN.KLIENCI (
NR_KLIENTA CHAR(8) NOT NULL,
IMIE VARCHAR(20) NOT NULL,
NAZWISKO VARCHAR(20) NOT NULL,
NR_KARTY_KREDYT CHAR(20) ,
FIRMA VARCHAR(40) ,
ULICA VARCHAR(24) NOT NULL,
NUMER CHAR(8) NOT NULL,
MIASTO VARCHAR(24) NOT NULL,
KOD CHAR(6) NOT NULL,
NIP CHAR(12) ,
NR_TELEFONU CHAR(16),
PRIMARY KEY (NR_KLIENTA));
INSERT INTO DB2ADMIN.KLIENCI
VALUES ('00000001', 'JAN', 'KOWALSKI', NULL, NULL, 'KOCHANOWSKIEGO', '3', 'WROCLAW', '37-300', NULL,
'167-763-234');
INSERT INTO DB2ADMIN.KLIENCI
VALUES ('00000002', 'TOMASZ', 'ADAMCZAK' , 'HH 12345678', 'KOWALSKI S.C.', 'KWIATOWA', '4/9',
'WARSZAWA', '01-900', '543-123-456', '46-744-431');
INSERT INTO DB2ADMIN.KLIENCI
VALUES ('00000003', 'PIOTR', 'MALCZYK' , 'HF 12445661', 'ADA S.C.', 'ROZANA', '9', 'WARSZAWA', '01-900', '443-
133-251', '16-742-114');
INSERT INTO DB2ADMIN.KLIENCI
VALUES ('00000004', 'PAWEL', 'FIODOROWICZ' , 'DD 76545321', 'KRAWIECTWO', 'ARMII KRAJOWEJ', '22A',
'WARSZAWA', '01-200', '555-233-256', '44-342-116');
INSERT INTO DB2ADMIN.KLIENCI
VALUES ('00000005', 'ANIELA', 'DALGIEWICZ' ,NULL , 'MODNA PANI', 'BOHATEROW GETTA', '24', 'WROCLAW',
'37-200', '456-134-153', '144-188-415');
INSERT INTO DB2ADMIN.KLIENCI
VALUES ('00000006', 'JOANNA', 'KWIATKOWSKA', NULL, NULL, 'TUWIMA', '2/5', 'SWIDNICA', '58-100', NULL,
'963-733-231');
INSERT INTO DB2ADMIN.KLIENCI
VALUES ('00000007', 'BOZENA', 'MALINOWSKA', NULL, NULL, 'LELEWELA', '34/1', 'SWIDNICA', '58-100', NULL,
'965-553-778');
INSERT INTO DB2ADMIN.KLIENCI
VALUES ('00000008', 'TOMASZ', 'NOWAK', NULL, NULL, 'ZEROMSKIEGO', '5A/8', 'SWIDNICA', '58-100', NULL,
'911-135-536');
INSERT INTO DB2ADMIN.KLIENCI
VALUES ('00000009', 'KRZYSZTOF', 'DOMAGALA', NULL, NULL, 'LESNA', '5', 'SWIDNICA', '58-100', NULL, '922-
233-232');
INSERT INTO DB2ADMIN.KLIENCI
VALUES ('00000010', 'ARKADIUSZ', 'DOCZEKALSKI', NULL, NULL, 'LESNA', '2', 'SWIDNICA', '58-100', NULL, '922-
233-267');
INSERT INTO DB2ADMIN.KLIENCI
VALUES ('00000011', 'ANNA', 'KOWALSKA' ,'KJ 98765412' , 'MODNIARSTWO', 'POWSTANCOW SLASKICH', '4',
'WROCLAW', '37-200', '422-132-354', '444-283-901');
INSERT INTO DB2ADMIN.KLIENCI
VALUES ('00000012', 'KRZYSZTOF', 'DOBROWOLSKI' , NULL, 'KAMIENIARSTWO', 'STRZEGOMSKA', '124',
'WROCLAW', '37-400', '433-133-332', '443-285-202');
INSERT INTO DB2ADMIN.KLIENCI
```

```
VALUES ('00000013', 'MARCIN', 'KRZYKALA' , NULL, NULL, 'KONOPNICKIEJ', '1/4', 'WROCLAW', '37-400', NULL,
'442-211-109');
INSERT INTO DB2ADMIN.KLIENCI
VALUES ('00000014', 'ANETA', 'PAPROCKA' , NULL, NULL, 'TUWIMA', '2', 'WROCLAW', '37-400', NULL, '442-671-
899');
INSERT INTO DB2ADMIN.KLIENCI
VALUES ('00000015', 'SEBASTIAN', 'KOWNACKI' , NULL, NULL, 'GLOWACKIEGO', '2/9', 'WROCLAW', '37-400',
NULL, '423-681-129');
INSERT INTO DB2ADMIN.KLIENCI
VALUES ('00000016', 'MICHAL', 'MICHALSKI' , NULL, NULL, 'KWIATOWA', '9/3', 'WROCLAW', '37-500', NULL, '499-
621-921');
INSERT INTO DB2ADMIN.KLIENCI
VALUES ('00000017', 'MICHAL', 'SZYKOWNY' , 'WW 12398765', NULL, 'LESNA', '3', 'WARSZAWA', '00-100', NULL,
'191-221-622');
INSERT INTO DB2ADMIN.KLIENCI
VALUES ('00000018', 'MARCIN', 'MARCINKOWSKI' , 'WQ 14368781', NULL, 'OKREZNA', '33', 'WARSZAWA', '00-
200', NULL, '122-127-647');
INSERT INTO DB2ADMIN.KLIENCI
VALUES ('00000019', 'RAFAL', 'RAFALSKI' , 'WS 12358672', 'NAPRAWA SAMOCHODOW', 'PRZEMYSLOWA', '1',
'WARSZAWA', '00-200', '999-765-120', '822-324-742');
INSERT INTO DB2ADMIN.KLIENCI
VALUES ('00000020', 'ROBERT', 'NOWAK' , 'AS 61333699', 'TAPICERSTWO', 'MOSTOWA', '9B', 'WARSZAWA', '00-
100', '987-765-333', '811-311-147');
```

Skrypt tworzący tabelę SAMOCHODY i wypełniający ją danymi

```
CONNECT TO WYPAUT USER db2admin USING db2admin;
DROP TABLE DB2ADMIN.SAMOCHODY;
CREATE TABLE DB2ADMIN.SAMOCHODY (
NR_SAMOCHODU CHAR(6) NOT NULL,
MARKA VARCHAR(20) NOT NULL,
TYP VARCHAR(16) NOT NULL,
ROK_PROD DATE NOT NULL,
KOLOR VARCHAR(16) NOT NULL,
POJ_SILNIKA SMALLINT NOT NULL,
PRZEBIEG INTEGER NOT NULL,
PRIMARY KEY (NR_SAMOCHODU));
INSERT INTO DB2ADMIN.SAMOCHODY
VALUES ('000001', 'MERCEDES', '190D', '1999-01-01', 'BIALY', 1800, 23000);
INSERT INTO DB2ADMIN.SAMOCHODY
VALUES ('000002', 'MERCEDES', '230D', '1999-01-01', 'NIEBIESKI', 2000, 35000);
INSERT INTO DB2ADMIN.SAMOCHODY
VALUES ('000003', 'FIAT', 'SEICENTO', '2000-01-01', 'CZERWONY', 1100, 13000);
INSERT INTO DB2ADMIN.SAMOCHODY
VALUES ('000004', 'FIAT', 'SEICENTO', '1999-01-01', 'BIALY', 900, 10000);
INSERT INTO DB2ADMIN.SAMOCHODY
VALUES ('000005', 'FIAT', 'TIPO', '1998-01-01', 'BORDOWY', 1400, 43000);
INSERT INTO DB2ADMIN.SAMOCHODY
VALUES ('000006', 'POLONEZ', 'CARO', '1997-01-01', 'ZIELONY', 1600, 55000);
INSERT INTO DB2ADMIN.SAMOCHODY
VALUES ('000007', 'OPEL', 'CORSA', '2000-01-01', 'ZIELONY', 1100, 11000);
INSERT INTO DB2ADMIN.SAMOCHODY
VALUES ('000008', 'OPEL', 'VECTRA', '1999-01-01', 'SZARY', 1800, 36000);
INSERT INTO DB2ADMIN.SAMOCHODY
VALUES ('000009', 'MERCEDES', '190D', '1996-01-01', 'BRAZOWY', 1800, 69000);
INSERT INTO DB2ADMIN.SAMOCHODY
VALUES ('000010', 'FORD', 'ESCORT', '2000-01-01', 'NIEBIESKI', 1600, 8000);
INSERT INTO DB2ADMIN.SAMOCHODY
VALUES ('000011', 'FORD', 'ESCORT', '1999-01-01', 'BIALY', 1600, 23000);
INSERT INTO DB2ADMIN.SAMOCHODY
VALUES ('000012', 'FORD', 'KA', '1998-01-01', 'BORDOWY', 1100, 54000);
INSERT INTO DB2ADMIN.SAMOCHODY
VALUES ('000013', 'FIAT', 'SEICENTO', '1999-01-01', 'ZLOTY', 1100, 25000);
INSERT INTO DB2ADMIN.SAMOCHODY
VALUES ('000014', 'FIAT', 'SEICENTO', '2000-01-01', 'BIALY', 1100, 18000);
INSERT INTO DB2ADMIN.SAMOCHODY
```

```
VALUES ('000015', 'SEAT', 'IBIZA', '1998-01-01', 'ZOLTY', 1800, 63000);
INSERT INTO DB2ADMIN.SAMOCHODY
VALUES ('000016', 'FORD', 'SIERRA', '1995-01-01', 'CZERWONY', 1600, 87000);
INSERT INTO DB2ADMIN.SAMOCHODY
VALUES ('000017', 'OPEL', 'CORSA', '2000-01-01', 'ZIELONY', 1400, 9000);
INSERT INTO DB2ADMIN.SAMOCHODY
VALUES ('000018', 'FORD', 'KA', '1999-01-01', 'ZOLTY', 1400, 20000
```

Skrypt tworzący tabelę PRACOWNICY i wypełniający ją danymi

```
CONNECT TO WYPAUT USER db2admin USING db2admin;
DROP TABLE DB2ADMIN.PRACOWNICY;
CREATE TABLE DB2ADMIN.PRACOWNICY (
NR_PRACOWNIKA CHAR(4) NOT NULL,
IMIE VARCHAR(20) NOT NULL,
NAZWISKO VARCHAR(20) NOT NULL,
DATA_ZATR DATE NOT NULL,
DZIAL VARCHAR(20) NOT NULL,
STANOWISKO VARCHAR(20) NOT NULL,
PENSJA DECIMAL(8,2) ,
DODATEK DECIMAL(8,2) ,
NR_MIEJSCA CHAR(6) NOT NULL,
NR_TELEFONU CHAR(16),
PRIMARY KEY (NR_PRACOWNIKA));
INSERT INTO DB2ADMIN.PRACOWNICY
VALUES ('0001', 'JAN', 'KOWALSKI', '1997-02-01', 'OBSLUGA KLIENTA', 'SPRZEDAWCA', 1100, 123, '000001',
'987-231-123');
INSERT INTO DB2ADMIN.PRACOWNICY
VALUES ('0002', 'ANNA', 'KAMINSKA', '1997-01-01', 'OBSLUGA KLIENTA', 'SPRZEDAWCA', 1200, 115, '000002',
'987-231-124');
INSERT INTO DB2ADMIN.PRACOWNICY
VALUES ('0003', 'KRZYSZTOF', 'ADAMSKI', '1997-05-01', 'OBSLUGA KLIENTA', 'KIEROWNIK', 2000, NULL,
'000001', '987-231-125');
INSERT INTO DB2ADMIN.PRACOWNICY
VALUES ('0004', 'PIOTR', 'MICHALSKI', '1998-06-01', 'TECHNICZNY', 'MECHANIK', 1700, 76, '000001', '987-231-
131');
INSERT INTO DB2ADMIN.PRACOWNICY
VALUES ('0005', 'BOZENA', 'DOMANSKA', '1997-02-01', 'OBSLUGA KLIENTA', 'SPRZEDAWCA', 1300, 134, '000003',
'987-231-126');
INSERT INTO DB2ADMIN.PRACOWNICY
VALUES ('0006', 'WOJCIECH', 'BURZALSKI', '1998-12-01', 'TECHNICZNY', 'MECHANIK', 1800, 80, '000003', '987-
231-132');
INSERT INTO DB2ADMIN.PRACOWNICY
VALUES ('0007', 'MARZENA', 'KOWNACKA', '1997-05-01', 'KSIEGOWOSC', 'KASJER', 1400, 105, '000001', '987-
231-141');
INSERT INTO DB2ADMIN.PRACOWNICY
VALUES ('0008', 'DAMIAN', 'MACHALICA', '1997-05-01', 'TECHNICZNY', 'KIEROWNIK', 2200, NULL, '000001', '987-
231-133');
INSERT INTO DB2ADMIN.PRACOWNICY
VALUES ('0009', 'ALICJA', 'MAKOWIECKA', '1999-07-01', 'OBSLUGA KLIENTA', 'SPRZEDAWCA', 1400, 120,
'000004', '933-241-525');
INSERT INTO DB2ADMIN.PRACOWNICY
VALUES ('0010', 'WOJCIECH', 'BAGIELSKI', '1998-04-01', 'OBSLUGA KLIENTA', 'SPRZEDAWCA', 1200, 100,
'000001', '457-531-143');
```

Skrypt tworzący tabelę MIEJSCA i wypełniający ją danymi

```
CONNECT TO WYPAUT USER db2admin USING db2admin;
DROP TABLE DB2ADMIN.MIEJSCA;
CREATE TABLE DB2ADMIN.MIEJSCA (
NR_MIEJSCA CHAR(6) NOT NULL,
ULICA VARCHAR(24) NOT NULL,
NUMER CHAR(8) NOT NULL,
MIASTO VARCHAR(24) NOT NULL,
KOD CHAR(6) NOT NULL,
TELEFON CHAR(16) ,
UWAGI VARCHAR(40),
PRIMARY KEY (NR_MIEJSCA));
```

```
INSERT INTO DB2ADMIN.MIEJSCA
VALUES ('000001', 'LEWARTOWSKIEGO', '12', 'WARSZAWA', '10-100', '228-277-097', NULL);
INSERT INTO DB2ADMIN.MIEJSCA
VALUES ('000002', 'ALEJE LIPOWE', '3', 'WROCLAW', '32-134', '388-299-086', NULL);
INSERT INTO DB2ADMIN.MIEJSCA
VALUES ('000003', 'KOCHANOWSKIEGO', '8', 'KRAKOW', '91-200', '222-312-498', NULL);
INSERT INTO DB2ADMIN.MIEJSCA
VALUES ('000004', 'LOTNICZA', '9', 'POZNAN', '22-200', '778-512-044', NULL);
```

Skrypt tworzący tabelę WYPOŻYCZENIA i wypełnia jacy ją danymi

```
CONNECT TO WYPAUT USER db2admin USING db2admin;
DROP TABLE DB2ADMIN.WYPOZYCZENIA;
CREATE TABLE DB2ADMIN.WYPOZYCZENIA (
NR_WYPOZYCZENIA CHAR(8) NOT NULL,
NR_KLIENTA CHAR(8) NOT NULL,
NR_SAMOCHODU CHAR(6) NOT NULL,
NR_PRACOW_WYP CHAR(4) NOT NULL,
NR_PRACOW_ODD CHAR(4) ,
NR_MIEJSCA_WYP CHAR(6) NOT NULL,
NR_MIEJSCA_ODD CHAR(6) ,
DATA_WYP DATE NOT NULL ,
DATA_ODD DATE ,
KAUCJA DECIMAL(8,2) ,
CENA_JEDN DECIMAL(8,2) NOT NULL,
PRIMARY KEY (NR_WYPOZYCZENIA));
INSERT INTO DB2ADMIN.WYPOZYCZENIA
VALUES ('00000001', '00000001', '000003', '0002', '0002', '000001', '000001', '1998-09-18', '1998-09-23', 200,
100);
INSERT INTO DB2ADMIN.WYPOZYCZENIA
VALUES ('00000002', '00000003', '000004', '0001', '0001', '000001', '000001', '1998-09-26', '1998-09-27', NULL,
100);
INSERT INTO DB2ADMIN.WYPOZYCZENIA
VALUES ('00000003', '00000002', '000004', '0009', '0009', '000002', '000002', '1998-10-04', '1998-10-04', NULL,
100);
INSERT INTO DB2ADMIN.WYPOZYCZENIA
VALUES ('00000004', '00000004', '000003', '0010', '0010', '000003', '000003', '1998-10-19', '1998-10-25', NULL,
100);
INSERT INTO DB2ADMIN.WYPOZYCZENIA
VALUES ('00000005', '00000006', '000007', '0010', '0010', '000003', '000003', '1998-10-29', '1998-11-02', 200,
100);
INSERT INTO DB2ADMIN.WYPOZYCZENIA
VALUES ('00000006', '00000005', '000008', '0010', '0002', '000001', '000003', '1998-11-07', '1998-11-09', 200,
100);
INSERT INTO DB2ADMIN.WYPOZYCZENIA
VALUES ('00000007', '00000008', '000011', '0009', '0002', '000001', '000001', '1998-11-20', '1998-11-25', 200,
100);
INSERT INTO DB2ADMIN.WYPOZYCZENIA
VALUES ('00000008', '00000006', '000011', '0001', '0005', '000004', '000004', '1998-11-28', '1998-12-02', 200,
100);
INSERT INTO DB2ADMIN.WYPOZYCZENIA
VALUES ('00000009', '00000007', '000017', '0002', '0002', '000001', '000002', '1998-12-01', '1998-12-03', 200,
100);
INSERT INTO DB2ADMIN.WYPOZYCZENIA
VALUES ('00000010', '00000009', '000017', '0002', '0010', '000001', '000002', '1998-12-15', '1998-12-17', 200,
100);
INSERT INTO DB2ADMIN.WYPOZYCZENIA
VALUES ('00000011', '00000010', '000001', '0005', '0005', '000003', '000003', '1998-12-20', '1998-12-23', 200,
100);
INSERT INTO DB2ADMIN.WYPOZYCZENIA
VALUES ('00000012', '00000012', '000002', '0005', '0005', '000004', '000004', '1999-01-04', '1999-01-14', 200,
100);
INSERT INTO DB2ADMIN.WYPOZYCZENIA
VALUES ('00000013', '00000011', '000005', '0001', '0005', '000003', '000001', '1999-01-24', '1999-01-29', NULL,
100);
INSERT INTO DB2ADMIN.WYPOZYCZENIA
```

```
VALUES ('00000014', '00000013', '000005', '0001', '0001', '000004', '000001', '1999-02-01', '1999-02-05', 200,
100);
INSERT INTO DB2ADMIN.WYPOZYCZENIA
VALUES ('00000015', '00000014', '000004', '0001', '0001', '000002', '000002', '1999-02-04', '1999-02-04', 200,
100);
INSERT INTO DB2ADMIN.WYPOZYCZENIA
VALUES ('00000016', '00000015', '000018', '0009', '0009', '000002', '000002', '1999-03-20', '1999-03-23', 200,
100);
INSERT INTO DB2ADMIN.WYPOZYCZENIA
VALUES ('00000017', '00000016', '000013', '0010', '0010', '000004', '000001', '1999-03-20', '1999-03-22', 200,
100);
INSERT INTO DB2ADMIN.WYPOZYCZENIA
VALUES ('00000018', '00000020', '000014', '0001', '0001', '000001', '000001', '1999-04-01', '1999-04-05', NULL,
100);
INSERT INTO DB2ADMIN.WYPOZYCZENIA
VALUES ('00000019', '00000019', '000015', '0005', '0005', '000004', '000004', '1999-05-04', '1999-05-09', NULL,
100);
INSERT INTO DB2ADMIN.WYPOZYCZENIA
VALUES ('00000020', '00000017', '000017', '0002', '0002', '000003', '000001', '1999-08-14', '1999-08-17', NULL,
100);
INSERT INTO DB2ADMIN.WYPOZYCZENIA
VALUES ('00000021', '00000018', '000009', '0002', NULL, '000001', NULL, '1999-12-04', NULL, NULL, 100);
INSERT INTO DB2ADMIN.WYPOZYCZENIA
VALUES ('00000022', '00000017', '000001', '0001', NULL, '000002', NULL, '1999-12-22', NULL, NULL, 100);
INSERT INTO DB2ADMIN.WYPOZYCZENIA
VALUES ('00000023', '00000009', '000003', '0010', NULL, '000002', NULL, '2000-01-08', NULL, 200, 100);
INSERT INTO DB2ADMIN.WYPOZYCZENIA
VALUES ('00000024', '00000014', '000004', '0005', NULL, '000001', NULL, '2000-01-24', NULL, 200, 100);
INSERT INTO DB2ADMIN.WYPOZYCZENIA
VALUES ('00000025', '00000010', '000004', '0009', NULL, '000002', NULL, '2000-02-09', NULL, 200, 100)'
```