Bazy Danych

SQL – Podstawy języka

Piotr Macioł
WIMiIP, KISiM,
pmaciol@agh.edu.pl
B5, pok. 606

Języki zapytań

- Interfejsy typu zapytanie przez przykład (ang. Query by Example - QBE), szablony (formularze, strony WWW)
- Structured Query Language (SQL), języki algebraiczne
- języki predykatowe (o zmiennych atrybutowych i krotkowych)
- DATALOG (język zbliżony do PROLOGu ale nieproceduralny i bez termów)

Czym jest SQL?

- *Structured Query Language* **SQL** - nieproceduralny język typu strukturalnego przeznaczony do uzyskiwania dostępu i operowania danymi (DML) oraz budowy bazy danych (DDL).
- Program składa się z poleceń, które mają określoną strukturę wewnętrzną (dyrektywy wewnętrzne).
- SQL w swoich konstrukcjach opiera się na algebrze relacji.
- Aplikacje sięgają do bazy danych za pomocą SQL w dwóch trybach:
  - » Wykonane są w językach (np. C, Cobol, Pascal) rozszerzonych o możliwość łączenia z SQL (*embedded* SQL)
  - » Korzystają ze specjalnego interfejsu (np. ODBC/JDBC), który pozwala wysyłać do bazy zapytania sformułowane w SQL.

SQL – historia

- 1974: Chamberlain, IBM, San Jose – ***SEQUEL*** Structured English Qery Language
- koniec lat 70-tych: ORACLE (Relational Software Inc.) – pierwsza implementacja komercyjna
- 1982: ANSI* – RDL (Relational Data Language)
- 1983: ISO** – definicja SQL
- 1986: ANSI – pierwszy standard SQL (SQL-86)
- 1987: ISO – pierwszy standard SQL (ISO 9075)
- SQL-86, SQL-89, SQL-92, SQL:1999, SQL:2003, SQL:2008,
- SQL:2011 [ISO/IEC 9075:2011] (siódma wersja)

*American National Standards Committee
**International Standards Organisation

Elementy SQL:

- **DDL** *(Data Definition Language)* – tworzenie, usuwanie i modyfikacja schematów, kluczy, indeksów, widoków, warunków integralności i praw dostępu, a także fizyczną strukturę pamięci dyskowej (`CREATE, DROP, ALTER`)
- **DML** (*Data Manipulation Language*) – język zapytań oparty na algebrze relacji obejmujący ponadto polecenia dodające, usuwające i aktualizujące dane w bazie danych (`SELECT, INSERT, DELETE, UPDATE`)
- **Kontrola transakcji** – SQL obejmuje polecenia rozpoczęcia i zakończenia transakcji, a także blokowania danych dla współbieżnych operacji (`START TRANSACTION, COMMIT, ROLLBACK` etc.)

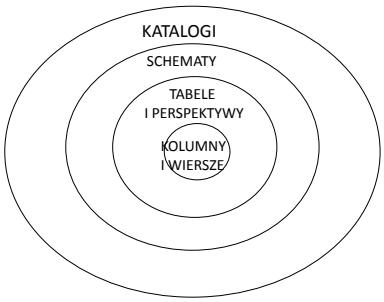
Osadzony SQL

- Standard SQL definiuje również zasady osadzania (*embedded*) SQL w językach programowania, takich jak Pascal, PL/1, Fortran, C i Cobol.
- Język, w którym osadzono SQL nazywany jest *host language* a struktury SQL dozwolone w tym języku tworzą osadzony SQL.
- Etykieta EXEC SQL/END EXEC jest używana do wskazywania struktur osadzonego SQL.
  - » EXEC SQL <struktura-osadzonego-SQL> END EXEC
- <struktura-osadzonego-SQL> wykorzystuje w ogólnym przypadku pełne możliwości SQL uzupełnione o pewne elementy wynikające z osadzenia.

Inne Zalety SQL

- Języki czwartej generacji - specjalne języki pomagające programistom w tworzeniu wzorców dla dialogu z użytkownikiem i w formatowaniu danych dla raportów, dostępne dla większości komercyjnych baz danych (PL/SQL).
- Sesja SQL - dostarcza abstrakcji klienta i serwera (także zdalnego):
  - » klient łączy (*connect)* się z serwerem SQL nawiązując sesję;
  - » wykonuje serię poleceń;
  - » rozłącza się od sesji (*disconnect);*
  - » może zapisać (*commit)* albo wycofać (*rollback)* pracę wykonaną w sesji.
- Środowisko SQL zawiera kilka komponentów między innymi identyfikator użytkownika i bazy danych, które pozwalają zidentyfikować którą z kilku baz danych używa dana sesja.

Hierarchia obiektów w SQL



Schematy

- Transakcje dotyczą wykonywania ciągu instrukcji INSERT, DELETE i UPDATE.
- Odpowiednikiem transakcji dla instrukcji definiujących obiekty i uprawnienia jest pojęcie *schematu*.
- Schemat tworzy grupę powiązanych obiektów. Jest realizowany za pomocą instrukcji:
  `CREATE SCHEMA` *`nazwa_schematu`*
  `ciąg instrukcji` `CREATE TABLE, CREATE VIEW i GRANT (bez rozdzielających średników);`

Schematy

- Tworzenie schematu

```
CREATE SCHEMA nazwa_schematu
AUTHORIZATION ID_wlasciciela

CREATE SCHEMA magazyn AUTHORIZATION dbo
```

- Określanie używanego schematu

```
SET SCHEMA nazwa_schematu

SET SCHEMA magazyn
```


Domeny

- Standard SQL dopuszcza tworzenie własnych zbiorów dopuszczalnych wartości pewnych kolumn w tabelach (dziedzin atrybutów)

```
CREATE DOMAIN nazwa_domeny AS typ_danych
DEFAULT wartosc_domyslna
CHECK warunek_kontrolny

CREATE DOMAIN kontrola AS CHAR(1)
DEFAULT 'T'
CHECK(UPPER(VALUE) = 'T' OR UPPER(VALUE)='N')
```


Typy danych w MySQL (1)

- **`CHAR(`***`n`***`)`** – skończonej długości łańcuch znakowy, z podaną przez użytkownika długością *n*
- **`VARCHAR(`***`n`***`)`** – zmiennej długości łańcuch znakowy, z podaną przez użytkownika maksymalną długością *n*
- **`TEXT`** – typ znakowy różniący się od `CHAR` i `VARCHAR` długością, nie może posiadać wartości `DEFAULT`
- **`INT(`***`n`***`), INTEGER(`***`n`***`)`** – liczba całkowita o podanej długości
- **`BOOL, BOOLEAN`** – równoważne typowi **`TINYINT(1)`**, mogą posiadać wartość **`1`** lub **`0`** rozumiane odpowiednio jako `TRUE` lub `FALSE`
- **`DATE`** - daty zawierające rok (4-cyfrowy), miesiąc i dzień (w formacie YYYY-MM-DD)
- **`TIME`** - czas w godzinach, minutach i sekundach
- **`DATETIME`** – kombinacja `DATE` i `TIME` (np.: `9999-12-31 23:59:59`)

Typy danych w MySQL (2)

- **FLOAT** – mała liczba rzeczywista, zmiennoprzecinkowa
- **DOUBLE** (length, decimal) – duża liczba rzeczywista, zmiennoprzecinkowa
- **DECIMAL** (length, decimal) – liczba typu DOUBLE przechowywana w postaci łańcucha co pozwala na zastosowanie stałej liczby miejsc po przecinku
- **SERIAL** - jest aliasem dla

  'BIGINT UNSIGNED NOT NULL AUTO_INCREMENT UNIQUE'
- **SERIAL DEFAULT VALUE** jako atrybut pola typu integer (od TINYINT do BIGINT) jest aliasem dla **NOT NULL AUTO_INCREMENT**
- Wartości NULL dozwolone są we wszystkich typach. Deklarując atrybut ze specyfikatorem **NOT NULL**, zabrania się wpisywania wartości NULL dla tego atrybutu.

Obsługa wartości NULL

- wartość NULL nie może być umieszczona w kolumnie NOT NULL,
- porównywanie dwóch kolumn zawierających NULL jest nieskuteczne (wartości NULL można identyfikować w klauzuli WHERE przy użyciu wyrażeń IS NULL IS NOT NULL)
- kolumna zawierająca NULL jest ignorowana podczas obliczania wartości agregujących natomiast jest uwzględniana w klauzuli GROUP BY
- jeżeli w warunku złączenia pojawi się kolumna z wartościami NULL to złączenie traktowane jest jako zewnętrzne

Konstrukcja tabeli

- Relacja (tabela) w SQL jest definiowana za pomocą polecenia postaci:

```
CREATE TABLE nazwa_tabeli
      (A1 D1, A2 D2,..., An Dn,
      [warunki-integralności]);
```

  - » Każde A_i jest nazwą atrybutu w schemacie relacji.
  - » Każde D_i określa dziedzinę atrybutu A_i przez podanie typu danych opisujących atrybut być może ze specyfikatorem **NOT NULL**
- [warunki-integralności] mogą przyjmować postaci:
  - » **PRIMARY KEY** $(A_1, \ldots, A_n)$ – definicja klucza zgodnie z zasadami.
  - » **CHECK** (P) gdzie P jest predykatem akceptowalnym w klauzuli **WHERE** zapytania SQL.

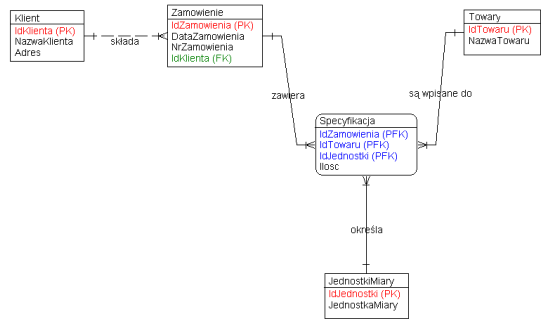
Przykład:

```
CREATE TABLE pracownicy (
    pesel CHAR(11) NOT NULL,
    imie VARCHAR(15) NOT NULL,
    nazwisko VARCHAR(40) NOT NULL,
    tytul VARCHAR(10),
  PRIMARY KEY (pesel),
  CHECK tytul IN (SELECT tytul-nazwa FROM
  tytuly));
```

Ograniczenie można zadać poprzez zdefiniowanie warunku logicznego, w tym także takiego, które sięga do innych tabel lub poprzez standardowego ograniczenia: NOT NULL lub UNIQUE

Integralność danych w SQL

- Wprowadzenie kluczy podstawowych i obcych zapewnia automatyczną kontrolę poprawności struktury danych i operacji przetwarzania danych
- Klucz podstawowy zapewnia unikalność i możliwość identyfikacji każdego zapisu
- Klucze obce zapewniają integrację referencyjną głoszącą, że każda *niepusta wartość klucza obcego* musi odpowiadać jednej z istniejących wartości klucza podstawowego



Klucz podstawowy

- W tablicach rodzicach (parent) jest niezbędny jako łącznik z tablicami dziećmi (child)
- Najlepiej rolę tą spełnia klucz, który jest kolumną tożsamości (identity)

Klucz podstawowy w definicji i zmianie tabeli

```
CREATE TABLE Towary (
      IdTowaru Int NOT NULL AUTO_INCREMENT,
      NazwaTowaru Char(50),
 PRIMARY KEY (IdTowaru);


ALTER TABLE Towary DROP PRIMARY KEY;

ALTER TABLE Towary ADD PRIMARY KEY
(IdTowaru);
```


Klucze obce

- W związkach *nieidentyfikujących* pozwalają na tworzenie złączeń (nie jest do tego konieczne wymuszanie integralności)
- W związkach identyfikujących pozwalają na kontrolę unikalności krotek w relacjach dzieciach
- **FOREIGN KEY** znany jako klucz obcy to pewnego rodzaju odnośnik łączący tabelę w którym występuje klucz obcy z inną tabelą. Klucz obcy zapobiega wszelkim operacjom, które mogłyby zerwać taką więź między tabelami.


```
CREATE TABLE Klient (
      IdKlienta Int NOT NULL AUTO_INCREMENT,
      NazwaKlienta Char(50),
      Adres Char(50),
 PRIMARY KEY (IdKlienta));


CREATE TABLE Zamowienie (
      IdZamowienia Int NOT NULL AUTO_INCREMENT,
      DataZamowienia Date,
      NrZamowienia Char(20),
      IdKlienta Int,
 PRIMARY KEY (IdZamowienia));
```


Złączenie naturalne

```
mysql> select nazwaklienta, datazamowienia from klient join zamowienie using
    -> (idklienta) order by nazwaklienta;
+-----------------------------------+-----------------+
| nazwaklienta                      | datazamowienia  |
+-----------------------------------+-----------------+
| Fabryka Rower~w                   | 2005-03-30      |
| Fabryka Rower~w                   | 2005-03-30      |
| Fabryka Rower~w                   | 2005-03-30      |
| Fabryka Rower~w                   | 2005-03-30      |
| Fabryka Rower~w                   | 2005-03-30      |
| Fabryka w~zk~w dziecięcych        | 2005-04-02      |
| Hurtownia Materia|~w Budowlanych | 2005-03-30      |
| Klient 2                          | 2005-04-06      |
| Klient 3                          | 2005-04-07      |
| Klient 5                          | 2005-04-02      |
| Klient 5                          | 2005-04-02      |
| Klient 5                          | 2005-04-02      |
| Sklep Og~lny                      | 2005-03-27      |
+-----------------------------------+-----------------+
13 rows in set (0.39 sec)
```


Złożony klucz obcy

```
CREATE TABLE JednostkiMiary (
      IdJednostki Int NOT NULL AUTO_INCREMENT,
      JednostkaMiary Char(20),
 PRIMARY KEY (IdJednostki));

CREATE TABLE Specyfikacja (
      IdZamowienia Int NOT NULL,
      IdTowaru Int NOT NULL,
      IdJednostki Int NOT NULL,
      Ilosc Float,
 PRIMARY KEY (IdZamowienia,IdTowaru,IdJednostki));
```

```
mysql> select * from specyfikacja;
+-------------+----------+---------------+-------+
| IdZamowienia | IdTowaru | IdJednostki | Ilosc |
+-------------+----------+---------------+-------+
|           1 |        1 |             1 |   100 |
|           1 |        2 |             1 |   150 |
|           2 |        1 |             2 |   200 |
|           2 |        2 |             1 |   120 |
|          59 |        1 |             2 |   200 |
|          59 |        2 |             1 |   100 |
|          60 |        2 |             1 |    50 |
|          60 |        1 |             2 |   100 |
|          66 |        3 |             1 |   500 |
|          61 |        2 |             1 |   543 |
|          62 |        2 |             1 |   100 |
|          62 |        1 |             1 |   200 |
|          63 |        2 |             1 |   105 |
|          70 |        1 |             1 |   250 |
|          70 |        3 |             1 |   200 |
|          69 |        2 |             2 |   120 |
|          69 |        6 |             3 |   300 |
|          69 |        1 |             1 |   250 |
|          69 |        3 |             1 |   200 |
|          69 |        2 |             1 |   150 |
|          70 |        6 |             3 |   320 |
|          71 |        3 |             1 |   150 |
|          70 |        2 |             1 |   150 |
|          71 |        2 |             1 |   300 |
|          72 |        2 |             1 |   100 |
|          72 |        3 |             1 |   300 |
+-------------+----------+---------------+-------+
26 rows in set (0.41 sec)

mysql> insert into specyfikacja (idzamowienia, idtowaru, idjednostki, ilosc)
    -> values('1', '1', '1', '200');
ERROR 1062 (23000): Powtórzone wystąpienie '1-1-1' dla klucza 1
mysql>
```

próba wprowadzenia identycznego klucza

Wymuszanie integralności

- **REFERENCES** – podaje źródło klucza obcego, tj. tabelę i klucz podstawowy
- **ON DELETE, ON UPDATE** – określenie czynności, które należy podjąć jeśli wartość klucza podstawowego zostanie usunięta lub ulegnie zmianie:
  - » **SET NULL** zastąp wartość klucza obcego przez NULL,
  - » **SET DEFAULT** zastąp wartość klucza obcego przez wartość domyślną,
  - » **CASCADE** skasuj lub zmodyfikuj wszystkie wiersze zawierające zmienianą wartość klucza obcego
  - » **NO ACTION** (tylko modyfikacja) nie zmieniaj wartości klucza
  - » **RESTRICT** nie dopuść do zmiany

Zabronione usuwanie w MySQL

```
ALTER TABLE Zamowienie ADD FOREIGN KEY
(IdKlienta)
REFERENCES Klient (IdKlienta) ON DELETE
RESTRICT ON UPDATE RESTRICT;
```

```
DELETE FROM `klient` WHERE `IdKlienta` =1
LIMIT 1
```

#1217 - Cannot delete a parent row: a foreign key constraint fails

IdKlienta	NazwaKlienta	Adres
1	Klient 1	*NULL*
2	Sklep Ogólny	Zabierzów ul. Spokojna 2
3	Firma Kruk	Modliniczka 127
4	Fabryka Rowerów	Kraków ul. Bracka 1
5	Fabryka Mebli	Zabierzów, ul. Krakowska 12
6	Hurtownia Materiałów Budowlanych	Kraków ul.Wielicka 20
7	Fabryka wózków dziecięcych	Krzeszowice ul, Krakowska 5
10	Klient 5	*NULL*
11	Klient 2	*NULL*
12	Klient 3	*NULL*

IdZamowienia	DataZamowienia	NrZamowienia	IdKlienta
1	2005-03-27	2005/127	1
2	2005-03-27	2005/128	2
59	2005-03-30	2005/201	4
60	2005-03-30	2005/202	4
61	2005-03-30	2005/203	4
62	2005-03-30	2005/204	4
63	2005-03-30	2005/an	4
66	2005-03-30	2005/303	6
67	2005-04-02	2005/876	10
68	2005-04-02	2005/876	10
69	2005-04-02	2005/876	10
70	2005-04-02	2005/876	7
71	2005-04-06	05/12	11
72	2005-04-07	9876/05	12

Kaskadowa aktualizacja w MySQL

```
ALTER TABLE Zamowienie ADD FOREIGN KEY (IdKlienta)
REFERENCES Klient(IdKlienta)
ON DELETE CASCADE ON UPDATE CASCADE;
```

```
DELETE FROM `klient` WHERE `IdKlienta` =1
```


IdKlienta	NazwaKlienta	Adres
2	Sklep Ogólny	Zabierzów ul. Spokojna 2
3	Firma Kruk	Modliniczka 127
4	Fabryka Rowerów	Kraków ul. Bracka 1
5	Fabryka Mebli	Zabierzów, ul. Krakowska 12
6	Hurtownia Materiałów Budowlanych	Kraków ul.Wielicka 20
7	Fabryka wózków dziecięcych	Krzeszowice ul, Krakowska 5
10	Klient 5	*NULL*
11	Klient 2	*NULL*
12	Klient 3	*NULL*

IdZamowienia	DataZamowienia	NrZamowienia	IdKlienta
2	2005-03-27	2005/128	2
59	2005-03-30	2005/201	4
60	2005-03-30	2005/202	4
61	2005-03-30	2005/203	4
62	2005-03-30	2005/204	4
63	2005-03-30	2005/an	4
66	2005-03-30	2005/303	6
67	2005-04-02	2005/876	10
68	2005-04-02	2005/876	10
69	2005-04-02	2005/876	10
70	2005-04-02	2005/876	7
71	2005-04-06	05/12	11
72	2005-04-07	9876/05	12

Porządkowanie kluczy w MySQL

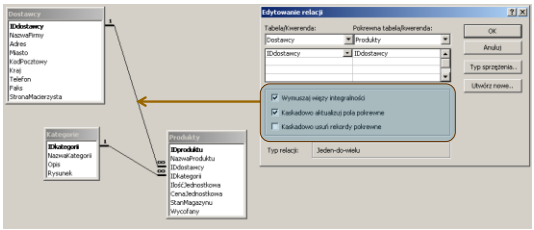
- Zmiana wartości klucza w tabeli rodzicielskiej powoduje odpowiednie zmiany w tabelach dzieciach

```
UPDATE `klient` SET `IdKlienta` = '8' WHERE
  `IdKlienta` =10;
```


IdKlienta	NazwaKlienta	Adres
2	Sklep Ogólny	Zabierzów ul. Spokojna 2
3	Firma Kruk	Modlniczka 127
4	Fabryka Rowerów	Kraków ul. Bracka 1
5	Fabryka Mebli	Zabierzów, ul. Krakowska 12
6	Hurtownia Materiałów Budowlanych	Kraków ul.Wielicka 20
7	Fabryka wózków dziecięcych	Krzeszowice ul, Krakowska 5
8	Klient 5	NULL
11	Klient 2	NULL
12	Klient 3	NULL

IdZamowienia	DataZamowienia	NrZamowienia	IdKlienta
2	2005-03-27	2005/128	2
59	2005-03-30	2005/201	4
60	2005-03-30	2005/202	4
61	2005-03-30	2005/203	4
62	2005-03-30	2005/204	4
63	2005-03-30	2005/an	4
66	2005-03-30	2005/303	6
67	2005-04-02	2005/876	8
68	2005-04-02	2005/876	8
69	2005-04-02	2005/876	8
70	2005-04-02	2005/876	7
71	2005-04-06	05/12	11
72	2005-04-07	9876/05	12

Kaskadowa aktualizacja w MS Access



Indeksy

- Indeks jest strukturą danych umożliwiającą szybki dostęp do krotek pewnej tabeli według jednej lub kilku kolumn
- Indeks zawiera kopie wybranych wartości kolumn ze związanej tabeli uszeregowane, tak by łatwiej było ją przeszukiwać

```
CREATE [UNIQUE] INDEX  nazwa_indeksu
ON nazwa_tabeli (nazwy_kolumn_klucza)

CREATE UNIQUE INDEX  symbol_nazwa_towaru
ON towar (symbol_towaru, nazwa_towaru)
```


Indeksy

- Czas trwania prostego wyszukiwania w tabeli zawierającej 161 016 rekordy z indeksem i bez indeksu

```
SQL Server parse and compile time:
   CPU time = 0 ms, elapsed time = 1 ms.

(1 row(s) affected)

SQL Server Execution Times:
   CPU time = 47 ms,  elapsed time = 39 ms.
```


Modyfikacja schematu bazy (1)

- Polecenie **ALTER** wykonuje następujące operacje:
  - dodaje kolumny i warunki
  - modyfikuje definicje kolumn jak typy i warunki
  - usuwa warunki
- **ALTER TABLE** nazwa_tabeli **ADD** A_i D_i
  jest używane do dodawania atrybutów do istniejącej relacji (tabeli). A_i jest nazwą atrybutu dodawanego do relacji nazwa_tabeli, a D_i jest specyfikacją typu A_i. Wszystkie krotki relacji uzyskują dla nowego atrybutu wartość NULL.
- **ALTER TABLE** nazwa_tabeli **DROP** A_i
  może być użyta do usunięcia atrybutu A_i relacji (tablicy) nazwa_tabeli
- Przykłady:

```
ALTER TABLE pracownicy ADD (placa DECIMAL(7, 2))
ALTER TABLE pracownicy MODIFY (placa DECIMAL(9, 2))
```

Modyfikacja schematu bazy (2)

– **DROP TABLE** tabela1, tabela2

Usuwa relacje (tabele) tabela1, tabela2 z bazy danych

– **RENAME TABLE** tabela1 **TO** tabela1

Polecenie służące do zmiany nazwy jednej lub więcej tabel

– Przykłady:

```
CREATE TABLE new_table (...);
RENAME TABLE old_table TO backup_table,
       new_table TO old_table;

RENAME TABLE current_database.table_name
       TO other_database.table_name;
```


Widoki (perspektywy)

– W większości przypadków pokazywanie wszystkich danych bazy wszystkim użytkownikom jest niepożądane. Również różni użytkownicy wymagają prezentacji danych w różny sposób.

» Np. Klient może być mieć dostęp do informacji o numerze konta innego klienta, ale nie może zobaczyć stanu tego konta.

– Relację, która nie jest składową modelu bazy danych, ale jest prezentowana użytkownikom, nazywa się **widokiem** (*view*).

– Dla utworzenia widoku używa się polecenia postaci:

CREATE VIEW nazwa_widoku **AS** <podzapytanie>

gdzie <podzapytanie> jest dowolną konstrukcją *select-from-where*.

– Po zdefiniowaniu widoku można się do niego odwoływać w zapytaniach jak do normalnej relacji.

Perspektywy (ang. Views)

– Zapytanie posiadające własną nazwę i przechowywane w słowniku danych

– Perspektywy tworzymy po to by:

» zapisać często wykonywane złożone zapytania

» stworzyć logiczne modele dla różnych użytkowników

» zwiększyć bezpieczeństwo danych

Po co widoki?

– W bazie zapamiętane zostaną tylko definicje poszczególnych kolumn widoków, nie zaś wartości. Różni użytkownicy bazy chcą/mogą/powinni mieć różny dostęp do tych samych danych źródłowych.

```
CREATE TABLE Ceny
(ProducentId SMALLINT UNSIGNED NOT NULL,
ProduktId SMALLINT UNSIGNED NOT NULL,
Cena DECIMAL(10,2) NOT NULL,
PRIMARY KEY(ProducentId, ProduktId));
CREATE TABLE Dostawy
(NrDostawy INT UNSIGNED NOT NULL PRIMARY KEY,
ProducentId SMALLINT UNSIGNED NOT NULL,
ProduktId SMALLINT UNSIGNED NOT NULL,
Ilosc INT UNSIGNED NOT NULL);
```

```
CREATE VIEW ksiegowosc
AS SELECT ProducentID, Ilosc*Cena AS Kwota
FROM Dostawy NATURAL JOIN Ceny;
CREATE VIEW produkcja
AS SELECT ProduktId, SUM(Ilosc) AS Zapas
FROM Dostawy
GROUP BY ProduktId;
```


Tworzenie perspektyw

```
CREATE VIEW PodgladZamowienia
SELECT     Klient.NazwaKlienta, Zamowienie.NrZamowienia,
Zamowienie.datazamowienia, Towar.NazwaTowaru,
LiniaZamowienia.Ilosc, LiniaZamowienia.Cena
FROM         Zamowienie INNER JOIN
Klient ON Zamowienie.IdKlienta = Klient.IdKlienta INNER
JOIN
LiniaZamowienia ON Zamowienie.IdZamowienia =
LiniaZamowienia.IdZamowienia INNER JOIN
Towar ON LiniaZamowienia.IdTowaru = Towar.IdTowaru
```


Wykorzystanie perspektyw

```
SELECT     NazwaKlienta, NazwaTowaru, Ilosc, Cena
FROM         PodgladZamowienia
WHERE     (NazwaKlienta = 'FH Klin SA')
```

Nazwaklienta	NazwaTowaru	Ilosc	cena
FH Klin SA	Rura zgrz. fi 6,3 gr 0,2	20	1.50
FH Klin SA	Rura zgrz. fi 12,6 gr 0,2	12	1.75
FH Klin SA	Rura zgrz. fi 6,3 gr 0,3	25	2.10

Modyfikowalność perspektyw

- Zgodnie ze standardem SQL-92 można modyfikować wyłącznie takie perspektywy, które:
  - » nie zawierają złączeń,
  - » są pojedyncze (np. unie nie są dopuszczalne),
  - » nie mogą być oparte na zapytaniu grupującym lub zawierającym słowo DISTINCT,
  - » nie można modyfikować kolumn wyliczonych

Modyfikacja danych (1)

- Modyfikacja bazy danych przez ich usunięcie może być zrealizowana w SQL za pomocą polecenia postaci:

  DELETE FROM <relacja> **WHERE** <warunek>

- Można zatem usuwać jedynie całe krotki, dla których prawdziwy jest <warunek> , oraz jednokrotnie tylko z jednej <relacja> (tabeli).
- Opuszczenie klauzuli **WHERE** skutkuje usunięciem danych z całej tabeli.
- Wykonanie **DELETE** przebiega w dwóch fazach: najpierw realizowany jest wybór wszystkich krotek, a następnie ich usuwanie.

Usuń pracowników z wynagrodzeniem równym zero.

```
DELETE FROM pracownicy
WHERE placa = 0;
```

- <warunek> może mieć formę dowolną akceptowaną przez klauzulę **WHERE** w zapytaniach, w tym może zawierać podzapytania.

Modyfikacja danych (2)

- Modyfikacja bazy danych przez ich dodanie może być zrealizowana w SQL za pomocą poleceń postaci:

  INSERT INTO <relacja> **VALUES** <krotka>

  INSERT INTO <relacja> **VALUES** <relacja-wstawiana>

- Można zatem dodawać jedynie całe krotki oraz jednokrotnie tylko do jednej relacji (tabeli). Zachowany być musi schemat relacji i domeny atrybutów.
- O ile to jest dopuszczone <krotka> może zawierać wartości *NULL*.

Dodaj nową krotkę do relacji *pracownicy* z wynagrodzeniem ustawionym na *NULL*

```
INSERT INTO pracownicy (pesel, nazwisko, imie, placa)
VALUES ('99999900000', 'Regulski', 'Krzysztof','104BT',NULL)

INSERT INTO pracownicy VALUES ('99999900000', 'Regulski',
'Krzysztof','104BT',NULL)
```

- Formę z **values** stosować można również do widoków.

Wstawianie wierszy

```
INSERT INTO Klient
(NazwaKlienta, Telefon, KodPocztowy, Miejscowosc,
Ulica, NrDomuMieszkania, Email)
VALUES('Nowy klient', '48 12 1234567', '30-333',
'Bolechowice', 'Jurajska', '20','ala@tlen.pl')
```

```
+-----------+------------------+---------------+
| IdKlienta | NazwaKlienta     | Telefon       |
+-----------+------------------+---------------+
|         3 | STALHANDEL       | 48 32 7865748 |
|         2 | Firma Krok Sp zoo | 48 12 6374532 |
|         5 | PHPU OSA         | 48 12 6372312 |
|         4 | Rower Polska SA  | 48 12 2853364 |
|         1 | FH Klin SA       | 48 12 1273210 |
|         6 | Nowy klient      | 48 12 1234567 |
+-----------+------------------+---------------+
```


Modyfikacja danych (3)

- Modyfikacja bazy danych przez zmianę wartości atrybutów może być zrealizowana w SQL za pomocą polecenia postaci:

  UPDATE <relacja> **SET** <modyfikacja> **WHERE** <warunek>

- Można modyfikować jeden atrybut krotek, dla których prawdziwy jest <warunek> oraz jednokrotnie tylko z jednej <relacja> (tabeli).
- <modyfikacja> jest **wyrażeniem arytmetycznym** (akceptowalnym w klauzuli **SELECT**), przypisującym (**=**) nową wartość atrybutowi.
- Opuszczenie klauzuli **WHERE** skutkuje modyfikacją całej tabeli.

Podnieś wynagrodzenie Regulskiemu o 30%

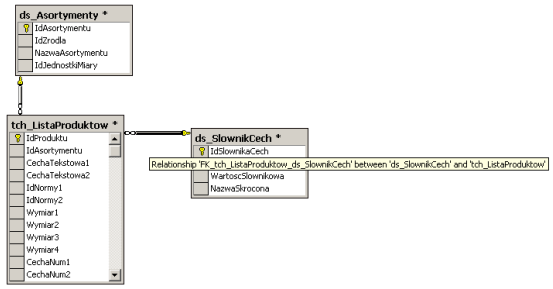
```
UPDATE pracownicy
SET placa=placa*1.3
WHERE nazwisko LIKE 'Regulski';
```


Tabele tymczasowe

- istnieją wyłącznie w trakcie trwania sesji
- obsługuje się je identycznie jak tabele stałe
- są znacznie szybciej obsługiwane niż zapytania czy perspektywy ale nie są automatycznie modyfikowane
- w przeciwieństwie do widoków są w pełni modyfikowalne

Edycja złożenia

- Edycja złożenia i perspektywy zawierającej elementy pochodzące z więcej iż jednej tabeli nie jest możliwa wprost
- Można natomiast stworzyć tabelę tymczasową zawierającą kody i treść atrybutów z połączonych tabel i ją edytować



Tworzenie tabeli tymczasowej w MS SQL

```
CREATE TABLE #tmp (
NazwaAsortymentu char(255),
WartoscSlownikowa char(255),
IdSlownikaCech bigint;
Wymiar1 float,
Wymiar2 float,
wymiar3 float);
```


Przenoszenie złożenia to tabeli

```
INSERT INTO #tmp
SELECT     dbo.ds_Asortymenty.NazwaAsortymentu,
dbo.ds_SlownikCech.WartoscSlownikowa,
dbo.tch_ListaProduktow.Wymiar1,
dbo.tch_ListaProduktow.Wymiar2,
dbo.tch_ListaProduktow.Wymiar3,
dbo.ds_SlownikCech.IdSlownikaCech,
dbo.tch_ListaProduktow.IdProduktu

FROM         dbo.tch_ListaProduktow INNER JOIN
dbo.ds_SlownikCech ON dbo.tch_ListaProduktow.CechaTekstowa1 =
dbo.ds_SlownikCech.IdSlownikaCech INNER JOIN
dbo.ds_Asortymenty ON dbo.tch_ListaProduktow.IdAsortymentu =
dbo.ds_Asortymenty.IdAsortymentu;
```


Edycja tabeli tymczasowej

- W tabeli możemy zmienić dane posługując się nazwami pochodzącymi w systemie z różnych tabel
- Po edycji można odtworzyć tabele źródłowe


```
DECLARE @indeks bigint;

SELECT     @indeks= dbo.ds_SlownikCech.IdSlownikaCech
FROM       dbo.ds_SlownikCech
WHERE
dbo.ds_SlownikCech.WartoscSlownikowa='eliptyczna';

UPDATE #tmp SET IdSlownikaCech=@indeks,
WartoscSlownikowa='eliptyczna' WHERE NazwaAsortymentu='Rura
profilowa' AND WartoscSlownikowa='kroplowa';

UPDATE dbo.tch_ListaProduktow SET
CechaTekstowa1=t.IdSlownikaCech
FROM #tmp t WHERE
dbo.tch_ListaProduktow.IdProduktu=t.IdProduktu;
```

NazwaAsortymentu	WartoscSlownikowa	Wymiar1	Wymiar2	Wymiar3	IdSlo...	IdProdu
Rura profilowa	eliptyczna	34.0	17.5	1.0	4	6978
Rura profilowa	eliptyczna	34.0	17.5	1.2	4	6979
Rura profilowa	eliptyczna	31.0	17.5	1.0	4	6980
Rura profilowa	eliptyczna	37.0	21.0	2.5	4	6981
Rura profilowa	eliptyczna	44.0	32.0	1.2	4	6982
Rura profilowa	eliptyczna	37.0	21.0	2.5	4	2939
Rura profilowa	eliptyczna	34.0	17.5	1.0	4	2940
Rura profilowa	eliptyczna	34.0	17.5	1.2	4	2941
Rura profilowa	kroplowa	81.0	37.0	1.2	5	2944
Rura b/s okreslonego zastosowan...	kroplowa	56.0	30.0	1.0	5	2961
Rura b/s okreslonego zastosowan...	kroplowa	81.0	34.0	2.5	5	2962
Rura b/s okreslonego zastosowan...	kroplowa	105.0	50.0	2.5	5	2963

NazwaAsortymentu	WartoscSlown...	Wymiar1	Wymiar2	Wymiar3	IdSlown...	IdProdu
Rura profilowa	eliptyczna	34.0	17.5	1.0	4	6978
Rura profilowa	eliptyczna	34.0	17.5	1.2	4	6979
Rura profilowa	eliptyczna	31.0	17.5	1.0	4	6980
Rura profilowa	eliptyczna	37.0	21.0	2.5	4	6981
Rura profilowa	eliptyczna	44.0	32.0	1.2	4	6982
Rura profilowa	eliptyczna	37.0	21.0	2.5	4	2939
Rura profilowa	eliptyczna	34.0	17.5	1.0	4	2940
Rura profilowa	eliptyczna	34.0	17.5	1.2	4	2941
Rura profilowa	eliptyczna	81.0	37.0	1.2	4	2944
Rura b/s okreslonego zastosowa...	kroplowa	56.0	30.0	1.0	5	2961
Rura b/s okreslonego zastosowa...	kroplowa	81.0	34.0	2.5	5	2962
Rura b/s okreslonego zastosowa...	kroplowa	105.0	50.0	2.5	5	2963

Przyznawanie praw dostępu

- Serwer baz danych może obsługiwać wielu użytkowników identyfikowanych przez nazwę i hasło
- Nie każdy z użytkowników musi mieć równe prawa
- W momencie utworzenia nowego elementu bazy danych aktualny użytkownik staje się jego właścicielem
- Właściciel może nadawać i odbierać prawa innym użytkownikom

Przyznawanie praw dostępu składnia SQL99

```
GRANT  {ALL [PRIVILEGES] }
| SELECT
| INSERT [ nazwa_kolumny [,...n]) ]
| DELETE
| UPDATE [ nazwa_kolumny [,...n]) ]
| REFERENCES [ nazwa_kolumny [,...n]) ]
| USAGE } [,...n]
ON { [TABLE] nazwa_tabeli
| DOMAIN nazwa_domeny
| COLLATION nazwa_zestawienia
CHARACTER SET nazwa_zestawu_znaków
| TRANSLATION nazwa_translacji }
TO {nazwa_podmiotu | PUBLIC}
[WITH GRANT OPTION]
```


Przykład MySQL

- Aktualny użytkownik, posiadający wszelkie uprawnienia przekazuje część uprawnień użytkownikowi ***andrzej*** na serwerze ***localhost***
- Następnie odbiera mu wszelkie uprawnienia i przekazuje inne

haslo

```
GRANT CREATE , DROP , INDEX , ALTER ,
CREATE TEMPORARY TABLES ON * . * TO
'andrzej'@'localhost' IDENTIFIED BY
'*****' WITH MAX_QUERIES_PER_HOUR 0
MAX_CONNECTIONS_PER_HOUR 0
MAX_UPDATES_PER_HOUR 0 ;


GRANT ALL PRIVILEGES ON `sprzedaz` .
* TO 'andrzej'@'localhost' WITH GRANT
OPTION ;
```


Użytkownik ma prawa do edycji danych, nie może jednak zmieniać struktur bazy danych

```
REVOKE ALL PRIVILEGES ON * . * FROM
'andrzej'@'localhost';

GRANT SELECT ,
INSERT ,
UPDATE ,
DELETE ,
RELOAD ,
SHUTDOWN ,
PROCESS ,
FILE ,
REFERENCES ,
SHOW DATABASES ,
SUPER ,
LOCK TABLES ,
EXECUTE ,
REPLICATION SLAVE ,
REPLICATION CLIENT ON * . * TO 'andrzej'@ 'localhost' WITH GRANT
OPTION MAX_QUERIES_PER_HOUR 0 MAX_CONNECTIONS_PER_HOUR 0
MAX_UPDATES_PER_HOUR 0 ;
```


```
Enter password: *****
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 155 to server version: 4.1.9-max

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql> use planowanie;
Database changed
mysql> create table test (
    -> id char(10));
ERROR 1044 (42000): Access denied for user 'andrzej'@'localhost' to database 'pl
anowanie'
mysql> select * from towary;
+----------+----------------------+-------+
| IdTowaru | NazwaTowaru          | cena  |
+----------+----------------------+-------+
|        1 | Rura stalowa fi 10   |  0.00 |
|        2 | Rura b/s fi 12       |  1.30 |
|        3 | Rura b/s fi 15       |  1.20 |
|       10 | rura b/s fi 32       |  1.10 |
|        6 | Z|czka fi 10         |  2.00 |
+----------+----------------------+-------+
5 rows in set (0.41 sec)

mysql> update towary set cena=1.5 where idtowaru=1;
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> select * from towary;
+----------+----------------------+-------+
| IdTowaru | NazwaTowaru          | cena  |
+----------+----------------------+-------+
|        1 | Rura stalowa fi 10   |  1.50 |
|        2 | Rura b/s fi 12       |  1.30 |
|        3 | Rura b/s fi 15       |  1.20 |
|       10 | rura b/s fi 32       |  1.10 |
|        6 | Z|czka fi 10         |  2.00 |
+----------+----------------------+-------+
5 rows in set (0.00 sec)

mysql>
```

Transakcje

- W systemie baz danych z wieloma użytkownikami transakcja jest niepodzielną, spójną, izolowaną i trwałą (ACID-atomic, consistent, isolatable, and durable) procedurą realizującą dostęp do danych
- Podczas realizacji transakcji możliwe jest wykorzystanie zmiennych oraz blokowanie dostępu do danych
- Transakcje mogą być wycofane

Transakcje składnia

```
START (BEGIN) TRANSACTION;
polecenia
COMMIT – zatwierdzenie transakcji
lub
ROLLBACK – wycofanie transakcji
```


Transakcje w MS SQL

- Użycie tabeli w trakcie transakcji blokuje dostęp do tabeli
- Polecenie ROLLBACK kasuje działania wykonane podczas transakcji
- Polecenie COMMIT zatwierdza działania

IdProduktu	IdAsortymentu	CechaTekstowa1	CechaTekstowa2
2944	25	5	2048

```
BEGIN TRANSACTION;
UPDATE     tch_ListaProduktow
SET              CechaTekstowa1 = 4
WHERE      (IdProduktu = 2944);
ROLLBACK;
```

IdProduktu	IdAsortymentu	CechaTekstowa1	CechaTekstowa2
2944	25	5	2048

```
BEGIN TRANSACTION;
UPDATE     tch_ListaProduktow
SET              CechaTekstowa1 = 4
WHERE      (IdProduktu = 2944);
COMMIT;
```

IdProduktu	IdAsortymentu	CechaTekstowa1	CechaTekstowa2
2944	25	4	2048

Transakcje w MySQL

- Potwierdzanie lub wycofywanie transakcji możliwe jest tylko w przypadku użycia mechanizmu obsługi pamięci InnoDB

Procedury składowane

- Specyficzny język umożliwiający wykonywanie wielu poleceń SQL oraz wprowadzający dynamikę


```
CREATE PROCEDURE procedura
@Asortyment char(255),
@Cecha char(255)
 AS
SELECT dbo.ds_Asortymenty.NazwaAsortymentu,
dbo.ds_SlownikCech.WartoscSlownikowa, dbo.tch_ListaProduktow.Wymiar1,
                    dbo.tch_ListaProduktow.Wymiar2,
dbo.tch_ListaProduktow.Wymiar3, dbo.ds_SlownikCech.IdSlownikaCech,
                     dbo.tch_ListaProduktow.IdProduktu
FROM         dbo.tch_ListaProduktow INNER JOIN
                    dbo.ds_SlownikCech ON
dbo.tch_ListaProduktow.CechaTekstowa1 = dbo.ds_SlownikCech.IdSlownikaCech
INNER JOIN
                    dbo.ds_Asortymenty ON
dbo.tch_ListaProduktow.IdAsortymentu = dbo.ds_Asortymenty.IdAsortymentu
WHERE ds_Asortymenty.NazwaAsortymentu=@asortyment  AND
dbo.ds_SlownikCech.WartoscSlownikowa=@Cecha
GO
```



```
procedura 'Rura profilowa', 'eliptyczna'
```

	NazwaAsortymentu	WartoscSlownikowa	Wymiar1	Wymiar2	Wymiar3	IdSlownikaCech	IdProduktu
1	Rura profilowa	eliptyczna	37.0	21.0	2.5	4	2939
2	Rura profilowa	eliptyczna	34.0	17.5	1.0	4	2940
3	Rura profilowa	eliptyczna	34.0	17.5	1.2	4	2941
4	Rura profilowa	eliptyczna	42.0	34.0	1.2	4	6927
5	Rura profilowa	eliptyczna	44.0	32.0	1.2	4	6928
6	Rura profilowa	eliptyczna	56.0	30.0	1.2	4	6929
7	Rura profilowa	eliptyczna	65.0	30.0	1.2	4	6930
8	Rura profilowa	eliptyczna	34.0	17.5	1.0	4	6978
9	Rura profilowa	eliptyczna	34.0	17.5	1.2	4	6979
10	Rura profilowa	eliptyczna	31.0	17.5	1.0	4	6980
11	Rura profilowa	eliptyczna	37.0	21.0	2.5	4	6981
12	Rura profilowa	eliptyczna	44.0	32.0	1.2	4	6982

Wyzwalacze w SQL 99

- wyzwalacz (ang. trigger) jest specjalnym rodzajem składowanej procedury, która jest uruchamiana automatycznie podczas wykonywania instrukcji modyfikacji danych
- wyzwalacz jest powiązany z konkretną instrukcją modyfikacji danych (INSERT, UPDATE, DELETE)
- wyzwalacz jest uruchamiany przed modyfikacją, po lub zamiast modyfikacji

Wyzwalacze SQL 99

```
CREATE TRIGGER nazwa_wyzwalacza
{ BEFORE | AFTER } { [DELETE] | [NSERT] [UPDATE]
[OF kolumna [,...n]}
ON nazwa_tabeli
[REFERENCING {OLD [ROW] [AS] nazwa_starej_krotki |
NEW [ROW] [AS] nazwa_nowej_krotki OLD TABLE [AS]
nazwa_starej_tabeli | NEW TABLE [AS]
nazwa_nowej_tabeli}]
{FOR EACH {ROW | STATEMENT}]
[WHEN (warunki)]
blok_kodu
```

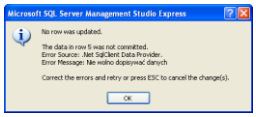

Wyzwalacze SQL 99

```
CREATE TRIGGER dopisanie_rekordu
INSTEAD OF INSERT SymbolTowaru, NazwaTowaru ON
Towar
REFERENCING
      OLD AS StaraKrotka
      NEW AS NowaKrotka
      OLD_TABLE Stare
WHEN NowaKrotka.NazwaTowaru IN
       Stare;
       UPDATE Towar
       SET NazwaTowaru = StaraKrotka.NazwaTowaru;
```

Wyzwalacz w MS SQL

```
USE test
    DROP TRIGGER uwaga
GO
CREATE TRIGGER uwaga
ON Przedmioty
AFTER INSERT, UPDATE
AS RAISERROR ('Nie wolno dopisywaæ
danych', 16, 10)
GO
```


Wyzwalacz w MS SQL



Archiwacja w MS SQL

```
CREATE TRIGGER archiwacja ON ds_asortymenty
FOR DELETE
AS
INSERT INTO ds_asortymenty_arch
SELECT *
FROM deleted
```


Stan wyjściowy

ds-asortymenty

IdAsortymentu	IdZrodla	NazwaAsortymentu	IdJednostkiMiary
1	0	Krąg kupny	5
2	1	Taśma po perforacji	5
3	1	Krąg przycięty na wymiar	5
4	0	Lupa	5
5	1	Lupa przycięta na wymiar	5
6	9	R.wysokostop.z/s ciagniona na zimno	5
7	9	Rura b/s	5
8	9	Rura b/s do pracy w podwyższ.temp.	5
9	9	Rura b/s kotłowa	5
10	9	Rura b/s ogolnego przeznaczenia	5
11	9	Rura b/s okreslonego zastosowania	5
12	9	Rura b/s okretowa	5
13	9	Rura b/s precyzyjna	5
14	9	Rura b/s wysokostop.austen.-ferryt.	5
15	9	Rura b/s wysokostop.ciagn.na zimno	5
16	9	Rura b/s wysokostop.walcow.na zimno	5
17	9	Rura b/s wysokostopowa	5
18	9	Rura b/s ze st.żaroodp.i na zb.cis.	5
19	9	Rura b/s ze stali żaroodpornych	5

Efekt usuwania

```
DELETE FROM ds_asortymenty WHERE IdAsortymentu<4
```

ds-asortymenty

IdAsortymentu	IdZrodla	NazwaAsortymentu	IdJednostkiMiary
4	0	Lupa	5
5	1	Lupa przycięta na wymiar	5
6	9	R.wysokostop.z/s ciagniona na zimno	5
7	9	Rura b/s	5
8	9	Rura b/s do pracy w podwyższ.temp.	5
9	9	Rura b/s kotłowa	5
10	9	Rura b/s ogolnego przeznaczenia	5
11	9	Rura b/s okreslonego zastosowania	5
12	9	Rura b/s okretowa	5
13	9	Rura b/s precyzyjna	5

ds_asortymenty_arch

IdAsortymentu	IdZrodla	NazwaAsortymentu	IdJednostkiMiary
1	0	Krąg kupny	5
2	1	Taśma po perforacji	5
3	1	Krąg przycięty na wymiar	5

Inne narzędzia proceduralne

- języki czwartej generacji 4GL
- graficzne interfejsy użytkownika GUI (Graphical User Interface)
- interfejsy typu QBE (Qery By Example)
- interfejsy języka naturalnego
- osadzony SQL
- interfejsy programów użytkowych API (application programming interface)

Języki 4GL

- generatory oparte na formularzach
- wysokiego poziomu języki proceduralne
- przykłady:
  - » PL/SQL firmy ORACLE
  - » Progress 4GL


```
/* h-CustSample.p- shows a few things about the Progress 4GL */
DEFINE VARIABLE cMonthList AS CHARACTER NO-UNDO
INIT "JAN,FEB,MAR,APR,MAY,JUN,JUL,AUG,SEP,OCT,NOV,DEC".
DEFINE VARIABLE iDays AS INTEGER NO-UNDO. /* used in calcDays proc */
/* First display each Customer from New Hampshire: */
FOR EACH Customer WHERE state = "NH" BY City:
DISPLAY CustNum NAME City.
/* Show the Orders unless the Credit Limit is less than
twice the balance. */
IF CreditLimit < 2 * Balance THEN
DISPLAY "Credit Ratio:" CreditLimit / Balance .
ELSE FOR EACH Order OF Customer:
DISPLAY OrderNum LABEL "Order"
OrderDate ShipDate FORMAT "99/99/99" WITH CENTERED.
/* Show the month as a three-letter abbreviation, along with the
number of days since the order was shipped. */
IF ShipDate NE ? THEN
DISPLAY ENTRY(MONTH(ShipDate), cMonthList) LABEL "Month".
RUN calcDays (INPUT ShipDate, OUTPUT iDays).
DISPLAY iDays LABEL "Days" FORMAT "ZZZ9".
END.
END.
PROCEDURE calcDays:
/* This calculates the number of days since the Order was shipped. */
DEFINE INPUT PARAMETER pdaShip AS DATE NO-UNDO.
DEFINE OUTPUT PARAMETER piDays AS INTEGER NO-UNDO.
piDays = IF pdaShip = ? THEN 0
ELSE TODAY - pdaShip.
END PROCEDURE.
```