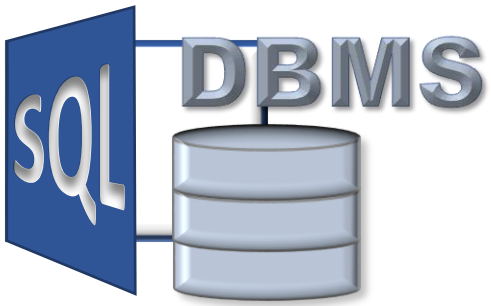
Bazy danych



Podzapytania i widoki

Operacje na strukturach i danych (DDL)

Manipulowanie danymi (DML)

Wykład 03 – podsumowanie

Złączenia

Złączenie wewnętrzne **INNER JOIN** – złączenie oparte o zależności pomiędzy wybranymi kolumnami łączonych źródeł danych

```
SELECT atrybuty
FROM źródło1 INNER JOIN źródło2 ON warunek
```

Złączenie zewnętrzne **OUTER JOIN** – złączenie którego wynikiem są wszystkie krotki (wiersze) jednego źródła danych i odpowiadające im krotki drugiego źródła

```
SELECT atrybuty
FROM źródło1 LEFT|RIGHT|FULL OUTER JOIN źródło2 ON warunek
```

Obliczenia wykonywane na pojedynczych krotkach

```
SELECT atrybuty, wyrażenie AS nazwa FROM źródła_danych;
```

Agregacja danych (operacje na grupach krotek)

```
SELECT atrybuty, funkcje FROM źródła_danych
GROUP BY atrybuty
HAVING warunek;
```

Przykładowe dane

P(PESEL, NazwiskoP, ImięP, PWZ)

PESEL	*NazwiskoP*	*ImięP*	*PWZ*
82012090803	Kowalski	Jan	1234567
70052000234	Nowak	Andrzej	1234567
85102303225	Wiśniewska	Anna	7654321
93041020146	Kowalczyk	Julia	7654321

L(PWZ, NazwiskoL, ImęL)

PWZ	*NazwiskoL*	*ImięL*
1234567	Adamska	Alicja
7654321	Wójcik	Zofia

B(ICD9, Kategoria, Cena)

ICD9	*Kategoria*	*Cena*
A14	Leukocyty	25
A15	Glukoza	40
C09	Erytrogram	37
C19	Hemoglobina	28
C41	Limfocyty T	45

S(PESEL, ICD9, Data, PWZ)

PESEL	**ICD9**	**Data**	*PWZ*
70052000234	A14	15.01.21	1234567
70052000234	A15	15.01.21	1234567
82012090803	A14	18.01.21	7654321
82012090803	C09	18.01.21	7654321
82012090803	C19	18.01.21	7654321
70052000234	C09	18.01.21	1234567
82012090803	A14	11.02.21	1234567
70052000234	C41	25.02.21	1234567

Podzapytania niezależne

Podzapytanie niezależne jest zapytaniem języka SQL, którego wynik jest zapisywany w pamięci (tworzy wirtualny zbór danych), a jego wartości są przekazywane do zapytania głównego. Podzapytanie niezależne zawsze może być wykonane samodzielnie bez konieczności uruchamiania zapytania głównego.

Przykładowa składnia

```
SELECT atrybuty, (podzapytanie1) AS nazwa
FROM źródła_danych, (podzapytanie2) AS nazwa
WHERE atrybut operator (podzapytanie3);
```

Podstawowe zasady konstruowania podzapytań

- podzapytanie w klauzuli SELECT musi zwracać wartość skalarną,
- podzapytanie w klauzuli FROM możne tworzyć dowolny zbiór,
- dozwolony wynik podzapytania w warunku (ON, WHERE, HAVING) jest zależny od zastosowanego operatora (patrz *s.8*) .

Uwaga: podzapytanie niezależne jest wykonywane tylko raz przed wykonaniem zapytania głównego.

Podzapytania niezależne – przykład I

Lista badań, których cena jest większa od średniej.

Uwaga: przed sprawdzeniem warunku WHERE należy obliczyć średnią cenę badań.

```
SELECT * FROM B
WHERE Cena > (SELECT AVG(Cena) FROM B)
```

ICD9	*Kategoria*	*Cena*
A14	Leukocyty	25
A15	Glukoza	40
C09	Erytrogram	37
C19	Hemoglobina	28
C41	Limfocyty T	45

Podzapytanie

```
SELECT AVG(Cena)
FROM B
```

(25+40+37+28+45) / 5 = 35.00

```
SELECT * FROM B
WHERE Cena > 35
```

Zapytanie główne

ICD9	*Kategoria*	*Cena*
A15	Glukoza	40
C09	Erytrogram	37
C41	Limfocyty T	45

Podzapytania niezależne – przykład II

Lista badań, których cena jest większa od średniej z informacją o różnicy pomiędzy ceną jednostkową i średnią.

```
SELECT ICD9, (Cena - (SELECT AVG(Cena) FROM B) ) AS Różnica
FROM B
WHERE Cena > (SELECT AVG(Cena) FROM B)
```

ICD9	*Kategoria*	*Cena*
A14	Leukocyty	25
A15	Glukoza	40
C09	Erytrogram	37
C19	Hemoglobina	28
C41	Limfocyty T	45

Podzapytanie

```
SELECT AVG(Cena)
FROM B
```

(25+40+37+28+45) / 5 = 35.00

```
SELECT ICD9,(Cena-35) AS Różnica
FROM B
WHERE Cena > 35
```

Zapytanie główne

ICD9	*Różnica*
A15	5
C09	2
C41	10

Podzapytania niezależne – przykład III

Lista pacjentów, którzy otrzymali skierowanie na badania w lutym 2021.

```
SELECT ImieP, NazwiskoP, Data, ICD9
FROM P INNER JOIN
      (SELECT PESEL, ICD9, Data FROM S
       WHERE Data BETWEEN '2021-02-01' AND '2021-02-28') AS Sk
    ON P.PESEL = Sk.PESEL;
```

podzapytanie

PESEL	*ICD9*	*Data*
82012090803	A14	11.02.21
70052000234	C41	25.02.21

złączenie z tabelą P

ImięP	*NazwiskoP*	*Data*	*ICD9*
Jan	Kowalski	11.02.21	A14
Andrzej	Nowak	25.02.21	C41

Uwaga: identyczny wynik daje zapytanie

```
SELECT ImieP, NazwiskoP, Data, ICD9
FROM P INNER JOIN S ON P.PESEL = S.PESEL
WHERE Data BETWEEN '2021-02-01' AND '2021-02-28';
```

ma ono jednak większy koszt wykonania, ponieważ pierwotna tabela S ma większą liczbę krotek, co prowadzi do większej liczby składników iloczynu skalarnego.

Operatory IN, ANY, ALL, EXISTS

Operatory `IN`, `ANY`, `ALL`, `EXISTS` działają na wektorach wartości.

Wektor wartości to zbiór elementów skalarnych opisanych jednym atrybutem, czyli wynik zapytania generującego odpowiedź z jedną kolumną.

Działanie operatorów

- **`IN`** sprawdza przynależność elementu do zbioru

```
atrybut IN (zbiór)
```

- **`ANY`** porównuje wartość elementu z każdym elementem zbioru, wyrażenie jest prawdziwe gdy jeden z wyników porównania jest prawdziwy

```
atrybut <operator> ANY (zbiór)
```

- **`ALL`** porównuje wartość elementu z każdym elementem zbioru, wyrażenie jest prawdziwe gdy każdy z wyników porównania jest prawdziwy

```
atrybut <operator> ALL (zbiór)
```

- **`EXISTS/NOT EXISTS`** sprawdza czy zbiór zawiera elementy lub jest pusty

```
EXISTS (zbiór)
```

Uwaga: wykonanie bezpośrednie operacji porównania (operatory =, >, >=, <, <=, <>) na wektorze wartości powoduje błąd, konieczne jest użycie operatorów **`ANY/ALL`**.

Operator IN – przykład

Lista lekarzy, który wystawili skierowania na badanie A14.

Bez użycia podzapytania (wynik z powtórzeniami)

```
SELECT ImięL, NazwiskoL FROM L INNER JOIN S ON L.PWZ=S.PWZ
WHERE ICD9 = 'A14'
```

ImięL	*NazwiskoL*
Alicja	Adamska
Zofia	Wójcik
Alicja	Adamska

Alicja Adamska wystawiła dwa Skierowania na badanie A14

Z wykorzystaniem podzapytania (wynik prawidłowy, bez powtórzeń)

```
SELECT ImieL, NazwiskoL FROM L
WHERE PWZ IN (SELECT PWZ FROM S WHERE ICD9='A14')
```

wynik podzapytania

PWZ
1234567
7654321
1234567

ImięL	*NazwiskoL*
Alicja	Adamska
Zofia	Wójcik

Każdy lekarz występuje tylko raz w tabeli L

Operatory ANY i ALL – przykłady

Lista lekarzy, który wystawili skierowania na badanie A14 (rozwiązanie alternatywne)

```
SELECT ImieL, NazwiskoL FROM L
WHERE PWZ = ANY(SELECT PWZ FROM S WHERE ICD9='A14');
```

wynik podzapytania

PWZ
1234567
7654321
1234567

Lekarze, których PWZ był równy jednemu (ANY) z elementów zbioru

ImięL	*NazwiskoL*
Alicja	Adamska
Zofia	Wójcik

Lista pacjentów, który nigdy nie byli skierowani na badania (rozwiązanie alternatywne, patrz *s.3-11*)

```
SELECT ImieP, NazwiskoP FROM P
WHERE PESEL <> ALL(SELECT PESEL FROM S);
```

wynik podzapytania

PESEL
70052000234
70052000234
82012090803
82012090803
82012090803
70052000234
82012090803
70052000234

Pacjenci, których PESEL był różny od wszystkich (ALL) elementów zbioru

ImięP	*NazwiskoP*
Anna	Wiśniewska
Julia	Kowalczyk

Zagnieżdżanie podzapytań

Zagnieżdżanie to technika polegające na umieszczeniu jednej operacji wewnątrz innej (w tym przypadku podzapytania wewnątrz innego podzapytania).

Zagnieżdżanie podzapytań może być zastosowane w dowolnym bloku kwerendy, liczba poziomów zagłębienia zależy od własności DBMS. Podzapytania zagnieżdżone wykonywane są kolejno zaczynając od poziomu najniższego.

Przykładowa składnia

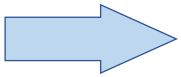
```
SELECT atrybuty, (SELECT atrybuty
                  FROM źródła_danych
                  WHERE podzapytanie zagnieżdżone) AS nazwa
FROM źródła_danych, (podzapytanie2) AS nazwa
WHERE atrybut operator (SELECT atrybuty
                        FROM źródła_danych
                        WHERE podzapytanie zagnieżdżone);
```

Zagnieżdżanie podzapytań – przykład

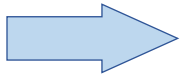
PESELE pacjentów i daty skierowań na badania, których cena jest większa od średniej (patrz przykłady na *s.5* i *6*)

```
SELECT PESEL, S.ICD9, Data
FROM S INNER JOIN
     (SELECT ICD9 FROM B
      WHERE Cena > (SELECT AVG(Cena) FROM B)) AS Lista
      ON S.ICD9 = Lista.ICD9;
```

ICD9	*Kategoria*	*Cena*
A14	Leukocyty	25
A15	Glukoza	40
C09	Erytrogram	37
C19	Hemoglobina	28
C41	Limfocyty T	45

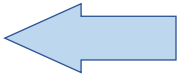


```
SELECT AVG(Cena)
FROM B
```

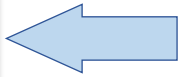


```
SELECT ICD9 FROM B
WHERE Cena > 35
```

ICD9
A15
C09
C41



```
SELECT PESEL, S.ICD9, Data
FROM S INNER JOIN (A15,C09,C41)
ON S.ICD9 = Lista.ICD9
```



PESEL	*ICD9*	*Data*
70052000234	A15	15.01.21
82012090803	C09	18.01.21
70052000234	C09	18.01.21
70052000234	C41	25.02.21

Podzapytania skorelowane

Podzapytanie skorelowane jest wykonywane dla każdego wiersza w zapytaniu głównym. Jest bezpośrednio związane z zapytaniem głównym poprzez wartości atrybutów, które są przekazywane do podzapytania i wykorzystywane do wyznaczenia rezultatu. Podzapytanie skorelowane nie może być wykonane samodzielnie i zawsze wymaga uruchomienia zapytania głównego.

Przykładowa składnia

```
SELECT atrybuty, (podzapytanie1) AS nazwa
FROM źródła_danych, (podzapytanie2) AS nazwa
WHERE atrybut operator (podzapytanie3);
```

Uwaga 1: w celu zapisu jednoznacznych odwołań skorelowanych źródłom danych występującym w zapytaniu głównym oraz podzapytaniu należy nadawać unikalne aliasy.

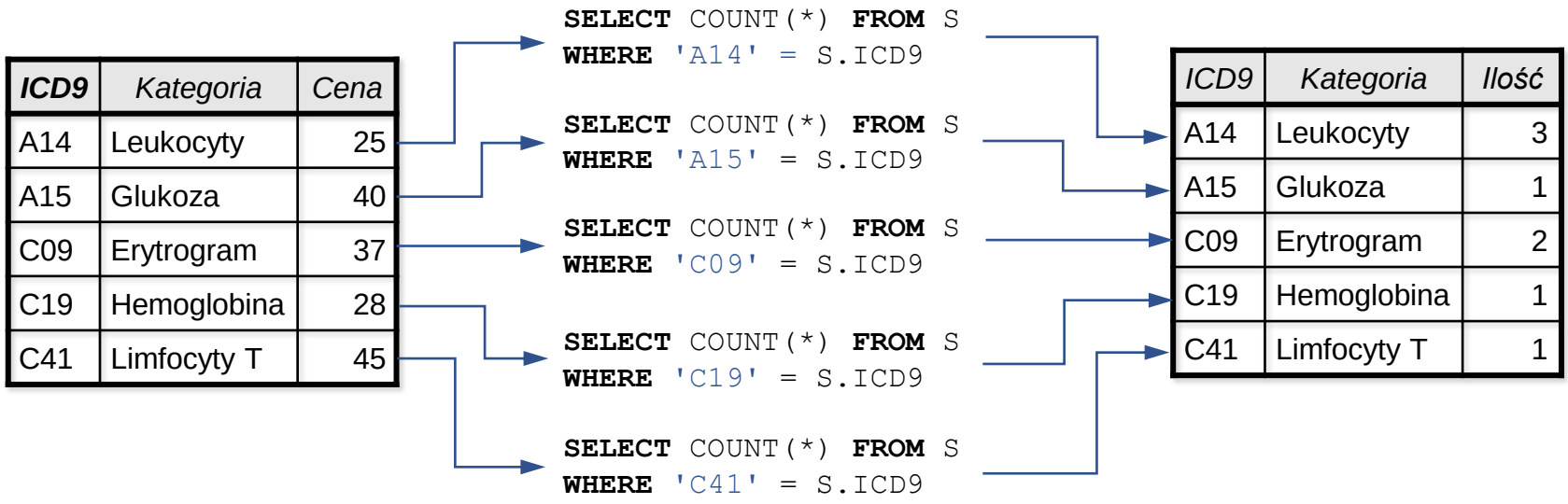
Uwaga 2: podzapytanie skorelowane jest wykonywane dla każdego wiersza zapytania głównego, stąd instrukcja SQL zawierające tego typu podzapytania może być czasochłonna (konieczność wielokrotnego wyznaczenia wyniku podzapytania).

Uwaga 3: podzapytanie skorelowane często może być wyeliminowanie poprzez zastosowanie agregacji i/lub odpowiedniego złączenia, co zazwyczaj prowadzi do instrukcji o większej efektywności.

Podzapytania skorelowane – przykład I

Liczba skierowań na poszczególne badania

```
SELECT ICD9, Kategoria,
       (SELECT COUNT(*) FROM S
        WHERE B.ICD9 = S.ICD9 ) AS Ilość
FROM B;
```



ICD9	Kategoria	Cena
A14	Leukocyty	25
A15	Glukoza	40
C09	Erytrogram	37
C19	Hemoglobina	28
C41	Limfocyty T	45

ICD9	Kategoria	Ilość
A14	Leukocyty	3
A15	Glukoza	1
C09	Erytrogram	2
C19	Hemoglobina	1
C41	Limfocyty T	1

Uwaga: Identyczny wynik daje zapytanie

```
SELECT B.ICD9, Kategoria, COUNT(*) AS Ilość
FROM B LEFT OUTER JOIN S ON B.ICD9 = S.ICD9
GROUP BY B.ICD9, Kategoria;
```

Podzapytania skorelowane – przykład II

Lista badań, których sumaryczny koszt przekroczył 40PLN

```
SELECT ICD9, Kategoria
FROM B AS Main
WHERE (SELECT SUM(Cena) AS Koszt
       FROM B AS Sub INNER JOIN S ON Sub.ICD9 = S.ICD9
       WHERE Sub.ICD9 = Main.ICD9                      ) > 40;
```

Podzapytanie wyznacza sumaryczną wartość badań dla każdego wiersza tabeli B (źródło danych w zapytaniu głównym), warunek WHERE wybiera badania, których koszt nie przekroczył 40PLN.

Uwaga 1: Korelacja wymaga porównania wartości ICD9 z tabeli B będącej źródłem danych w zapytaniu głównym oraz tabeli B będącej źródłem danych w podzapytaniu. W celu uzyskania jednoznacznego odwołania konieczne jest nadanie odpowiednich aliasów (w przykładzie odpowiednio Main i Sub).

Uwaga 2: Identyczny wynik daje zapytanie

```
SELECT B.ICD9, Kategoria
FROM B INNER JOIN S ON B.ICD9 = S.ICD9
GROUP BY B.ICD9, Kategoria
HAVING SUM(Cena) > 40;
```

Widoki (perspektywy)

Widok jest wirtualną tabelą utworzoną na podstawie wyniku kwerendy (polecenie **SELECT**). Jest traktowana jak zwykła tabela, jednak nie przechowuje danych, a każdorazowe odwołanie powoduje uruchomienie zapytania na podstawie którego została utworzona.

Tworzenie/modyfikacja widoku

```
CREATE [OR REPLACE] VIEW <nazwa_widoku> AS
  SELECT atrybuty FROM tabele
  WHERE warunek;
```

Odczyt danych z widoku

```
SELECT * FROM <nazwa_widoku>;
```

Uwaga 1: w kwerendach definiujących postać widoków można wykorzystać wszystkie omówione elementy składni instrukcji **SELECT**.

Uwaga 2: każdy odczyt danych powoduje ponowne wykonanie zapytania tworzącego widok (widoki aktualizują zbiór związanych danych przy każdym odwołaniu).

Widoki – przykłady

Widok z danymi pacjentów lekarza o PWZ 1234567

```
CREATE VIEW pacjenci_lekarza_1234567 AS
  SELECT * FROM P WHERE PWZ='1234567';
```

Widok z pełnymi danymi pacjentów (z imieniem i nazwiskiem lekarza)

```
CREATE VIEW pełne_dane_pacjentów AS
  SELECT PESEL, NazwiskoP, ImieP, L.PWZ, NazwiskoL, ImieL
  FROM P INNER JOIN L ON P.PWZ=L.PWZ;
```

Widok z liczbą pacjentów poszczególnych lekarzy

```
CREATE VIEW liczba_pacjentów AS
  SELECT L.PWZ, COUNT(*) FROM P INNER JOIN L ON P.PWZ=L.PWZ
  GROUP BY PWZ;
```

Odczyt danych z widoków

```
SELECT * FROM pacjenci_lekarza;
SELECT * FROM pełne_dane_pacjentów;
SELECT * FROM liczba_pacjentów;
```

Widoki jako źródła danych

Widoki mogą być używane jako źródła danych w kwerendach SQL. Wykonanie kwerendy wymusza odczyt danych z widoku, więc w takim zastosowaniu widok stanowi podzapytanie niezależnym (patrz s.4).

Przykładowa składnia

```
SELECT atrybuty, (SELECT * FROM nazwa widoku1) AS nazwa1
FROM źródła_danych, nazwa_widoku2 AS nazwa2
WHERE atrybut operator (SELECT * FROM nazwa widoku3);
```

Podstawowe zasady wykorzystania widoków (analogicznie do podzapytań)

- widok w klauzuli SELECT musi zwracać wartość skalarną,
- widok w klauzuli FROM możne tworzyć dowolny zbiór danych,
- dozwolony wynik widoku w warunku (ON, WHERE, HAVING) jest zależny od zastosowanego operatora (patrz *s.8*) .

Uwaga: każdy widok może być używany samodzielnie, więc może odwoływać się do danych zapytania nadrzędnego (głównego), stąd tworzenie podzapytań skorelowanych za pomocą widoków nie jest możliwe.

Widoki jako źródła danych – przykłady

Badania z ceną większą od średniej (s.5)

```
CREATE VIEW średnia_cena_badania AS
  SELECT AVG(Cena) FROM B;

SELECT * FROM B WHERE Cena > (SELECT * FROM średnia_cena_badania);
```

Badania z ceną większą od średniej z informacją o różnicy (s.6)

```
SELECT ICD9, (Cena - (SELECT * FROM średnia_cena_badania)) AS Różnica
FROM B WHERE Cena > (SELECT * FROM średnia_cena_badania)
```

Pacjenci, którzy otrzymali skierowanie na badania w lutym 2021 (s.7)

```
CREATE VIEW skierowania_luty_2021 AS
  SELECT PESEL, ICD9, Data
  FROM S WHERE Data BETWEEN '2021-02-01' AND '2021-02-28';

SELECT ImieP, NazwiskoP, Data, ICD9
FROM P INNER JOIN skierowania_luty_2021 AS Sk ON P.PESEL = Sk.PESEL;
```

Tworzenie i usuwanie bazy danych

Wyświetlenie istniejących baz danych

```
SHOW DATABASES;
```

Tworzenie nowej bazy danych

```
CREATE DATABASE <nazwa>;
```

Usuwanie istniejącej bazy danych

```
DROP DATABASE <nazwa>;
```

Wybór bazy danych

```
USE <nazwa>;
```

Kopia bazy danych

```
BACKUP DATABASE <nazwa> TO DISK = <nazwa_pliku>
[WITH DIFFERENTIAL];
```

Opcjonalna klauzula **WITH DIFFERENTIAL** oznacza wykonanie kopii różnicowej, uwzględniającej wyłącznie elementy, które uległy zmianie od ostatniej kopii bazy.

Uwaga: <nazwa> powinna być dozwolonym i unikalnym identyfikatorem (polecenie **CREATE**) lub nazwą istniejącej bazy danych (polecenia **DROP**, **USE** i **BACKUP**).

Tworzenie nowej tabeli

```
CREATE TABLE <nazwa> (
   <atrybut1> typ_danych więzy_spójności,
   <atrybut2> typ_danych więzy_spójności,
   …
   <atrybutN> typ_danych więzy_spójności
);
```

Tworzenie tabeli na podstawie wyniku kwerendy

```
CREATE TABLE <nazwa> AS
   SELECT atrybuty
   FROM tabela
   WHERE … ;
```

Uwaga: `tabela` musi być nazwą istniejącego źródła (źródeł) danych. Jeżeli źródło znajduje się w innej niż bieżąca bazie danych można odwołać się do niego używając konstrukcji *nazwa_bazy.nzawa_źródła*.

Typy danych (MySQL)

Typy znakowe

- `CHAR(N)` – ciąg znaków o stałej długości do 255 znaków,
- `VARCHAR(N)` – ciąg znaków o zmiennej długości do 65535 znaków.

Typy całkowite

- `TINYINT [UNSIGNED]` – liczba całkowita -127..128 (0..255),
- `SMALL [UNSIGNED]` – liczba całkowita -32768..32767 (0..65535),
- `INT [UNSIGNED]` – liczba całkowita −2147483648..2147483647 (0..4294967295).

Typy rzeczywiste

- `DECIMAL(N,D)` – liczba rzeczywista stałoprzecinkowa, N cyfr, D cyfr po przecinku,
- `FLOAT` – rzeczywista zmiennoprzecinkowa, pojedyncza prec. −1,4e+38 .. 3,4e+38,
- `DOUBLE` – rzeczywista zmiennoprzecinkowa, podwójna prec. −1,8+308 .. −1,8+308.

Typy daty i czasu

- `DATE` – data w formacie `rrrr-mm-dd`, 1000-01-01..9999-12-31,
- `TIME` – czas w formacie `gg:mm:ss`, -838:59:59..838:59:59
- `DATETIME` – data i czas w formacie `rrrr-mm-dd gg:mm:ss`.

Więzy spójności (ograniczenia)

Więzy spójności specyfikują reguły określające poprawność danych poprzez nałożenie dodatkowych warunków na wartości, które mogą być wprowadzone do bazy. Są sprawdzane przed wykonaniem dowolnej operacji na danych (dodawanie, modyfikacja, usuwanie) i przerywają ją w przypadku wykrycia naruszenia reguł.

Ograniczenia w SQL

- `NOT NULL` – zabroniona wartość `NULL`, wartość musi być wprowadzona,
- `UNIQUE` – wartość atrybutu musi być unikalna,
- `PRIMARY KEY` – klucz główny (`NOT NULL+UNIQUE`), wartość jednoznacznie identyfikuje krotki tabeli,
- `FOREIGN KEY` – klucz obcy, dozwolone wartości, które stanowią klucz główny w innej tabeli,
- `CHECK` – wartość atrybutu musi spełniać warunek (dozwolone kilka warunków),
- `DEFAULT` – domyślna wartość atrybutu.

PRIMARY KEY, FOREIGN KEY, DEFAULT, CHECK (MySQL)

Składnia ograniczeń PRIMARY KEY

```
CREATE TABLE <nazwa> (
   <atr1> typ PRIMARY KEY,
   <atr2> typ, …
   <atrN> typ
);
```

```
CREATE TABLE <nazwa> (
   <atr1> typ, …
   <atrN> typ,
   PRIMARY KEY (lista_atrybutów)
);
```

Składnia ograniczeń FOREIGN KEY

```
CREATE TABLE <nazwa> (
   <atr1> typ PRIMARY KEY,
   <atr2> typ, …
   <atrN> typ,
   FOREIGN KEY (atrybut) REFERENCES tabela(atrybut)
);
```

Składnia ograniczeń CHECK i DEFAULT

```
<atrybut> typ_danych CHECK (warunek)
```

```
<atrybut> typ_danych DEFAULT (wartość | wyrażenie)
```

CREATE TABLE – przykład I

Przykład 1 – nowa baza danych „test” i tabela „Lekarz”

```
CREATE DATABASE test;

USE test;

CREATE TABLE Lekarz (
     PWZ CHAR(7) PRIMARY KEY CHECK (LENGTH(PWZ)=7) ,
     NazwiskoL VARCHAR(25) NOT NULL,
     ImieL VARCHAR(25)
);
```

Przykład 2 – tabela „Lekarz” jako kopia istniejącej tabeli „L”

```
CREATE TABLE Lekarz AS
   SELECT *
   FROM moja_baza.L;
```

Uwaga: „moja_baza” jest przykładową nazwą istniejącej i dostępnej dla danego użytkownika bazy danych, która zawiera tabelę „L”.

CREATE TABLE – przykład II

Tabela „Pacjent” z ograniczeniami

```
USE test;

CREATE TABLE Pacjent (
    PESEL CHAR(11) CHECK (LENGTH(PESEL)=11),
    NazwiskoP VARCHAR(25) NOT NULL,
    ImieP VARCHAR(25) NOT NULL,
    PWZ CHAR(7),
    DataZapisu DATE DEFAULT (CURRENT_DATE()),
    PRIMARY KEY (PESEL),
    FOREIGN KEY (PWZ) REFERENCES Lekarz(PWZ)
);
```

Uwaga 1: baza musi zawierać tabelę „Lekarz”, której kluczem głównym jest „PWZ”.

Uwaga 2: w przykładzie wykorzystano funkcje dostępne w MySQL: `LENGTH` (określa długość łańcucha znakowego) oraz `CURRENT_DATE` (zwraca bieżącą datę).

Modyfikacja struktury tabeli

Dodawanie kolumny

```
ALTER TABLE <nazwa_tabeli>
ADD <atrybut> typ_danych;
```

Usuwanie kolumny

```
ALTER TABLE <nazwa_tabeli>
DROP COLUMN atrybut;
```

Modyfikacja kolumny (MySQL)

```
ALTER TABLE <nazwa_tabeli>
MODYFY COLUMN <atrybut> typ_danych;
```

Uwaga: postać instrukcji modyfikującej kolumnę zależy od serwera bazy danych: **MODYFY COLUMN** w MySQL i starszych wersjach Oracle, **MODYFY** w nowych wersjach Oracle, **ALTER COLUMN** w SQL Server.

Modyfikacja struktury tabeli – przykład

```
USE test;

ALTER TABLE Pacjent                    <-- dodanie pola DataUrodzenia
ADD DataUrodzenia DATETIME;                do tabeli Pacjent

ALTER TABLE Pacjent                    <-- zmiana typu pola
MODIFY COLUMN DataUrodzenia DATE;          z DATETIME na DATE

ALTER TABLE Pacjent                    <-- usunięcie pola
DROP COLUMN DataUrodzenia;                 DataUrodzenia z tabeli
```

Uwaga: Instrukcja **ALTER TABLE** pozwala również na dodawanie, modyfikację i usuwanie ograniczeń (*s.21*). Składnia operacji jest zależna od typu ograniczenia.

Wstawianie krotek

```
INSERT INTO <nazwa_tabeli> (atrybut1, atrybut2,…,atrybutN)
VALUES (wartość1-1, wartość1-2, …,wartość1-N),
       (wartość2-1, wartość2-2, …,wartość2-N),…;
```

Uwaga: nazwy atrybutów mogą być pominięte, jeżeli instrukcja wstawia wartości do wszystkich kolumn i wartości są wymienione w kolejności zgodnej z definicją w tabeli.

Aktualizacja krotek

```
UPDATE <nazwa_tabeli>
SET atrybut1 = wartość1, atrybut2 = wartość2,…
WHERE warunek;
```

Usuwanie krotek

```
DELETE FROM <nazwa_tabeli>
WHERE warunek;
```

Uwaga: klauzula **WHERE** w instrukcjach **UPDATE** i **DELETE** jest opcjonalna, brak oznacza aktualizację/usunięcie wszystkich krotek tabeli (składnia *warunku* jak w **SELECT**).

Dane lekarzy

```
INSERT INTO Lekarz(PWZ, NazwiskoL, ImieL)
VALUES('1234567', 'Adamska', 'Alicja'),
      ('7654321', 'Wójcik', 'Zofia');
```

Dane pacjentów

```
INSERT INTO Pacjent(PESEL, NazwiskoP, ImieP, PWZ, DataZapisu)
VALUES
 ('70052000234', 'Nowak', 'Andrzej', '1234567', '2021-01-10'),
 ('82012090803', 'Kowalski', 'Jan', '1234567', '2021-01-15'),
 ('85102303225', 'Wiśniewska', 'Anna', '7654321', '2021-02-11');
```

Nowy pacjent z domyślną datą zapisu (*s.24*, ograniczenie **DEFAULT**)

```
INSERT INTO Pacjent(PESEL, NazwiskoP, ImieP, PWZ)
VALUES ('93041020146', 'Kowalczyk', 'Julia', '7654321');
```