Bazy danych

Wykład 1_1

Temat: Wprowadzenie do przedmiotu

Sławomir Świętoniowski

slawomir-swietoniowski@wp.pl

Plan wykładu

1. Wprowadzenie do baz danych.
2. Program wykładu BD.
3. Program laboratorium i projektu BD.
4. Wymagania i sposób zaliczania przedmiotu.

Wprowadzenie do baz danych

- Definicja bazy danych.
- Znaczenie baz danych w systemach informatycznych.
- Najważniejsze produkty bazodanowe na rynku.
- Podstawowe typy architektur baz danych.

2:Data in Table 'Pracownik' in 'Uczelnia_Wroclaw' on 'HOMESQL'

IdPracownika	Nazwisko	Imie	NIP	PESEL	Adres	Miejscowosc
1	Nowakowski	Andrzej	612-412-54-64	57121943212	ul. Świerkowa 6	Wrocław
2	Kowalski	Jan	663-654-76-87	72013142337	ul. Strzelecka 15/8	Wrocław
3	Janicki	Bogdan	432-543-654-6	49042343259	ul. Pastelowa 58/9	Wrocław
4	Marcinkowski	Piotr	789-098-23-45	76110309878	ul. Lakiernicza 98A	Wrocław
5	Andrzejewski	Grzegorz	980-432-23-12	52082898152	ul. Nobla 8/3	Wrocław
6	Piotrowski	Bartlomiej	780-678-66-11	65102189131	ul. Nadrzeczna 16/	Legnica
7	Bogdanska	Ewa	430-543-55-22	79031487924	ul. Wrocławska 23/	Legnica
8	Grzegorzewski	Pawel	250-532-99-90	76061898778	ul. Karkonoska 58	Wrocław
9	Styczen	Tomasz	610-220-96-52	68071678156	ul. Janowska 43/63	Legnica
10	Romanowski	Janusz	616-420-90-19	74090209835	ul. Góralska 43	Legnica

Co to jest baza danych?

Definicja 1

Baza danych (ang. *database*) – magazyn faktów z nałożoną wewnętrzną strukturą, reprezentujących pewien obszar analizy (ang. Universe Of Discourse, UOD).
(Beynon-Davies, 2003)

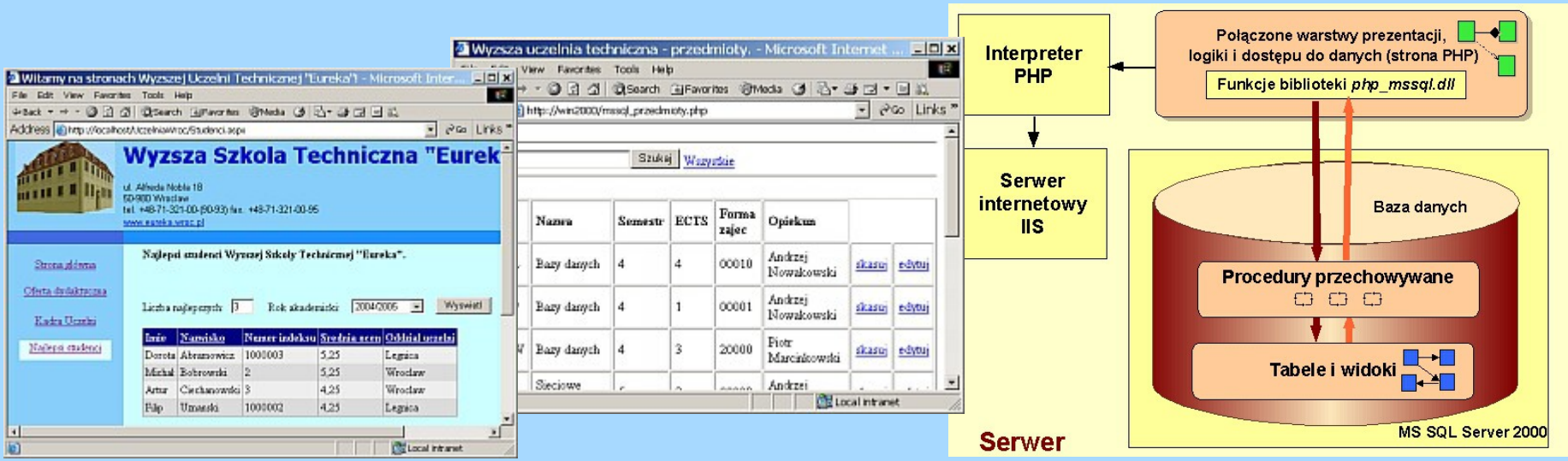
Definicja 2

Baza danych – zbiór danych trwałych, wykorzystywanych przez system aplikacji określonej organizacji (firmy, przedsiębiorstwa).

(Date, 2000)

Znaczenie baz danych

- Dane są zazwyczaj najcenniejszym zasobem firmy.
- Większość tworzonych obecnie systemów informatycznych to aplikacje klient-serwer. Ich integralną częścią jest **warstwa bazodanowa**, która ma kluczowe znaczenie dla funkcjonalności i wydajności systemów.
- Na rynku pracy przeważająca część stanowisk dla informatyków ma coś wspólnego z bazami danych (np. webmaster, administrator systemu, analityk / projektant, programista, wdrożeniowiec, serwisant).



Przykładowe DBMS na rynku

Klasa *Desktop*

- MS Access,
- MS SQL Server Express Edition,
- Oracle Express Edition
- SQLite.

Klasa *Enterprise*

- Oracle Enterprise Edition,
- MS SQL Server Enterprise,
- IBM DB2.

Klasa *Pośrednia*

- PostgreSQL,
- MySQL.

Typy architektur baz danych

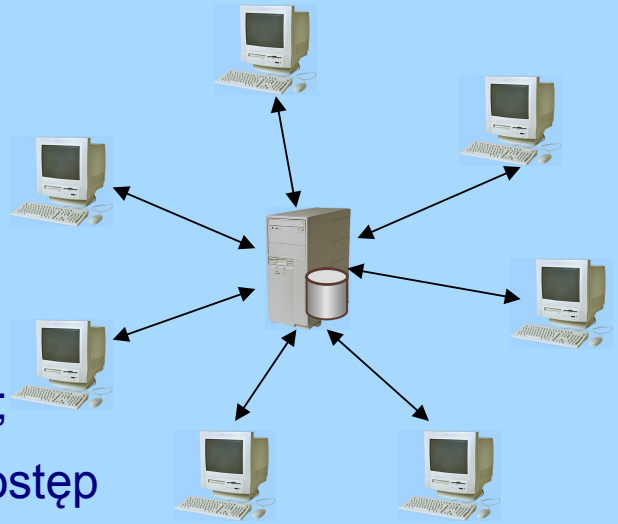
- **stacjonarne** (ang. *stand-alone databases)*:
  - jeden użytkownik,
  - prosta struktura bazy,
  - prosta aplikacja wykorzystująca bazę;
- **sieciowe** (ang. *network databases)*:
  - wielu użytkowników z jednoczesnym dostępem do bazy,
  - zwiększone wymogi odnośnie wydajności serwera i bezpieczeństwa danych;
- **rozproszone** (ang. *distributed databases)*:
  - wiele baz sieciowych (architektura homo- lub heterogeniczna),
  - replikacja danych,
  - zapytania i transakcje rozproszone.

Baza stacjonarna
(ang. *stand-alone database*)

- **w danym momencie korzysta z niej tylko jeden użytkownik** (jest to najważniejsza cecha),
- prosta struktura bazy,
- prosta aplikacja wykorzystująca bazę,
- przykład: baza do domowej ewidencji płyt CD

Sieciowa baza danych

Sieciowa baza danych (ang. network database) – baza, z której jednocześnie może korzystać wielu użytkowników.

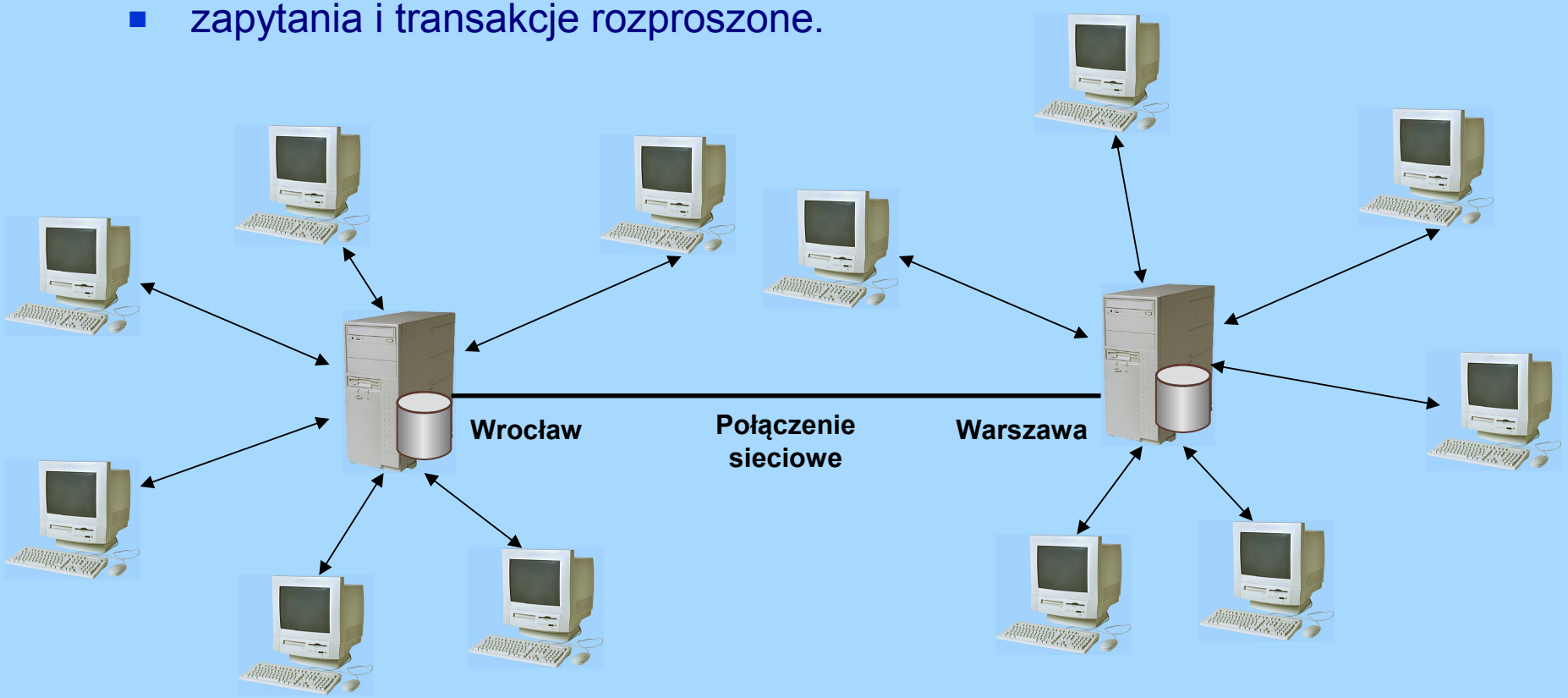


Typowe problemy:

- **integralność** (spójność) – dane mogą być modyfikowane przez wielu użytkowników;
- **wydajność** – wielodostęp silniej obciąża serwer;
- **bezpieczeństwo** – użytkownicy powinni mieć dostęp tylko do tych zasobów, do których mają uprawnienia;
- **zależność od warstwy sieciowej** – poprawność i efektywność wykorzystania bazy zależy od parametrów sieci komputerowej.

Baza rozproszona
(ang. *distributed database*)

- wiele połączonych ze sobą baz sieciowych (architektura homo- lub heterogeniczna),
- replikacja danych,
- zapytania i transakcje rozproszone.

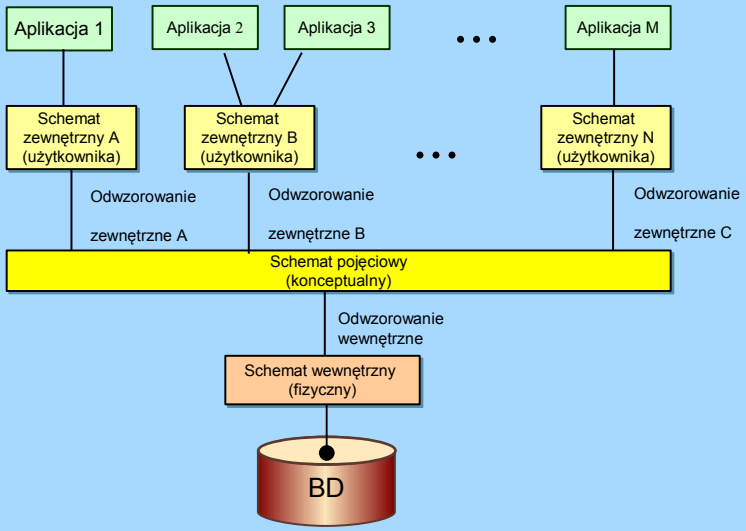


Program wykładu BD

1. Architektura systemu baz danych.
2. Relacyjny model danych.
3. Prosta metodologia tworzenia baz danych.
4. Język SQL i Transact-SQL.
5. Tworzenie schematu bazy danych.
6. Dodawanie, modyfikowanie, usuwanie i wyszukiwanie danych.
7. Obiekty kodu T-SQL przechowywane na serwerze.
8. Zaawansowane elementy języka T-SQL.

1. Architektura systemu baz danych

- System zarządzania bazą danych (DBMS) i jego składniki.



2.1. Relacyjny model danych

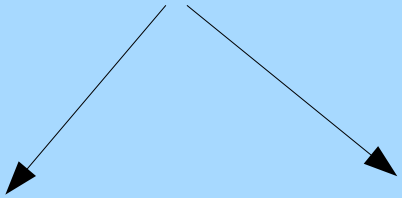
- Założenia i historia modelu relacyjnego.
- Model formalny.
- Algebra relacyjna i jej odniesienie do języka SQL.

2.2. Relacyjny model danych

- Klucze główne i obce.
- Integralność encji i referencyjna.
- Normalizacja bazy danych: 1NF, 2NF, 3NF.

Kursy (2 NF)

KodKursu	NrPrac	NazwiskoPrac	ImiePrac
INF407	234	Dudek	Damian
INF507	234	Dudek	Damian
INF517	345	Choroś	Kazimierz



Kursy (3 NF)

KodKursu	NrPrac
INF407	234
INF507	234
INF517	345

Kursy (3 NF)

NrPrac	NazwiskoPrac	ImiePrac
234	Dudek	Damian
345	Choroś	Kazimierz

3. Prosta metodologia tworzenia baz danych

- Analiza dziedziny.
- Tworzenie schematu bazy danych.
- Deklaratywne więzy integralności.
- Widoki i procedury przechowywane, realizujące operacje CRUD.
- Proceduralne więzy integralności – wyzwalacze.

4. Język SQL i T-SQL

- Historia i standardy.
- Podstawowe podzbiory funkcjonalne: DDL, DML, DIL, DCL.

5. Tworzenie schematu bazy danych

- Tabele, pola i ich typy.
- Relacje i ich własności.
- Deklaratywne więzy integralności: PK, FK, CHECK, DEFAULT, UNIQUE.

6. Dodawania, modyfikowanie, usuwanie i wyszukiwanie danych

- Polecenia INSERT, UPDATE, DELETE.
- Polecenie SELECT z klauzulami FROM, WHERE, GROUP BY, HAVING, ORDER BY.
- Złączenia tabel: INNER | OUTER | CROSS JOIN.

7. Obiekty kodu T-SQL przechowywane na serwerze

- Motywacja.
- Widoki, procedury przechowywane, funkcje i wyzwalacze.
- Tworzenie, parametryzacja i wykorzystanie procedur przechowywanych.

8. Zaawansowane elementy języka T-SQL

- Wbudowane procedury i funkcje *MS SQL Server 2008 R2*.
- Walidacja danych i zabezpieczenie kodu w procedurach.
- Dynamiczny kod T-SQL.

Organizacja kursu BD

- Wykłady nie są obowiązkowe, ale mogą bardzo pomóc w pozytywnym zaliczeniu kursu. Praktyka wskazuje, że studenci uczestniczący w wykładach mają zdecydowanie lepsze wyniki od tych, którzy je opuszczają.
- Materiały do przedmiotu są publikowane wyłącznie w systemie **Moodle** na stronie Uczelni: **http://e-learning.wsiz.wroc.pl** (wymagany jest klucz dostępu, podany na wykładzie).
- Kurs kończy się egzaminem: testem komputerowym w systemie Moodle.
- Do egzaminu mogą być dopuszczeni tylko ci studenci, którzy uzyskają wcześniej pozytywną ocenę z laboratorium.
- Ocena końcowa z wykładu wystawiana jest na podstawie oceny z egzaminu (60%) i z laboratorium (40%).

Literatura

1. BEYNON-DAVIES P., *Systemy baz danych – nowe wydanie*, WNT, Warszawa 2003.
2. MICROSOFT, *Books On-Line* – dokumentacja systemu *MS SQL Server*, Microsoft Corp. 1988 – 2000.
3. MSDN page: http://www.msdn.microsoft.com.
4. PANKOWSKI T., *Podstawy baz danych*, Wydawnictwo Naukowe PWN, Warszawa, 1992.
5. RANKINS R., JENSEN P., BERTUCCI P., *Microsoft SQL Server 2000. Księga eksperta.*, HELION, Gliwice 2003 (pozycja dostępna w bibliotece WSIZ „Copernicus").
6. WAYMIRE R., SAWTELL R., *MS SQL Server 2000 dla każdego*, HELION, Gliwice 2002 (pozycja dostępna w bibliotece WSIZ "Copernicus").

Bazy danych

Wykład 1_1

Dziękuję za uwagę !

Bazy danych

Wykład 1_2

Temat: Architektura systemu baz danych

Sławomir Świętoniowski

slawomir-swietoniowski@wp.pl

Plan wykładu

1. **Podstawowe typy baz danych.**
2. System zarządzania bazą danych (DBMS).
3. Prezentacja systemu MS SQL Server 2008 R2 Ent Edition.

Typy baz danych

- transakcyjne (produkcyjne, OLTP),
- analityczne (m.in. OLAP),
- informacyjne,
- internetowe,
- multimedialne.

Bazy transakcyjne
(produkcyjne, OLTP - *On-Line Transactional Processing*)

- służą do obsługi bieżących danych operacyjnych przedsiębiorstwa,
- dominują polecenia operowania na danych – CRUD (*create*, *read*, *update*, *delete*),
- jednoczesny dostęp wielu użytkowników,
- aplikacje o bardzo dużych wymaganiach odnośnie niezawodności i wydajności (ang. *mission-critical*),
- przykłady:
  - baza systemu dystrybucji energii elektrycznej,
  - baza sprzedaży hipermarketu.

Bazy analityczne
(wspomagające decyzje)

- hurtownie danych (składnice, magazyny) importowanych z baz OLTP,
- duży rozmiar (nawet rzędu setek TB – terabajtów),
- prawie w 100% do odczytu,
- specjalne, wielowymiarowe struktury danych,
- „wyrafinowane” zapytania, analizy OLAP, zestawienia, eksploracja danych (data mining),
- przykłady:
  - baza analiz sprzedaży sieci hipermarketów.

Bazy informacyjne

- służą do udostępniania informacji,
- znacznie częstsze wyszukiwanie,
  niż aktualizacja danych,
- wyszukiwanie pełnotekstowe,
- przykłady:
  - baza informacji o rozkładzie PKP,
  - baza - przewodnik po hotelach w Polsce.

Bazy internetowe

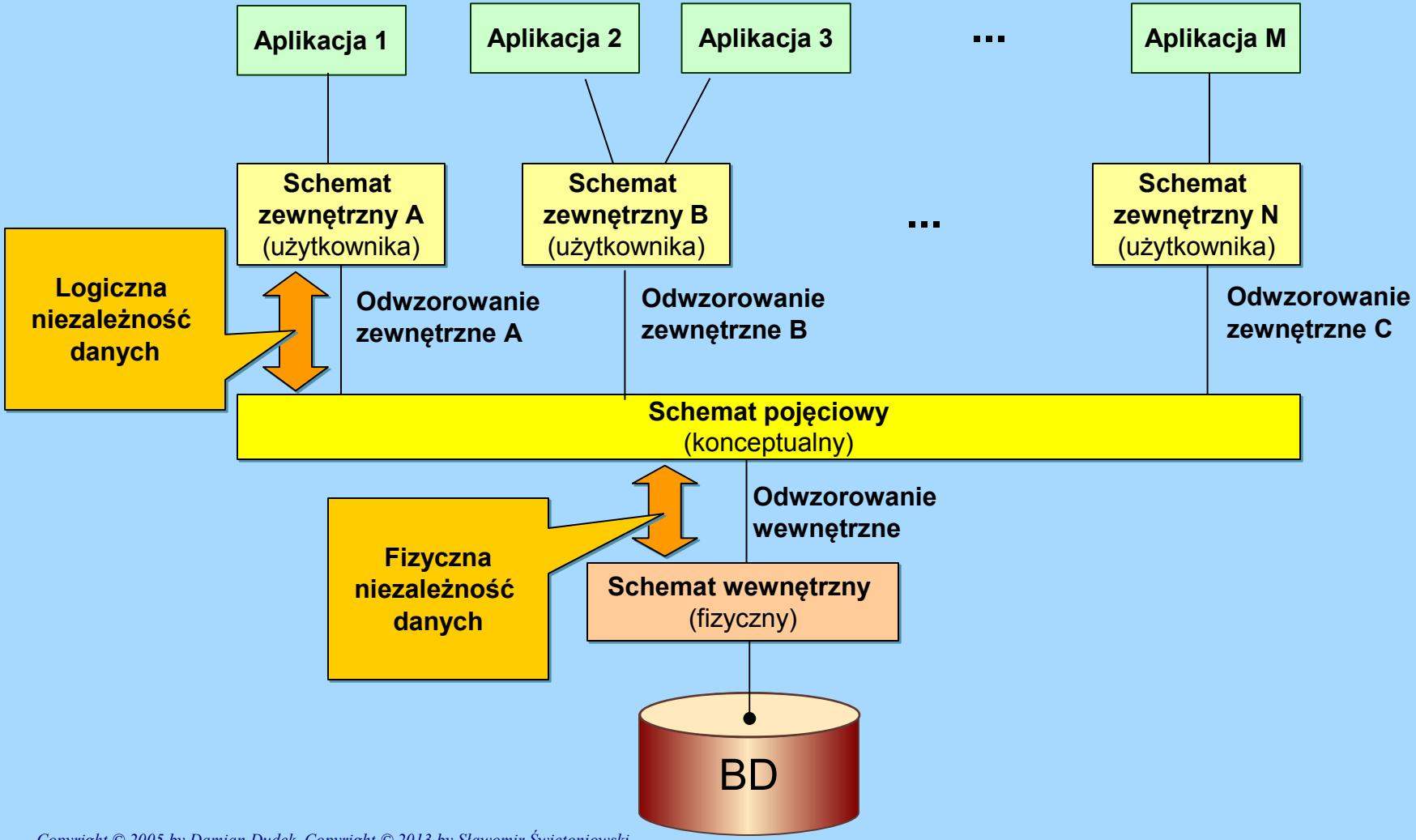
- służą do współpracy z aplikacjami i dynamicznymi stronami WWW,
- zbliżone strukturą do baz informacyjnych lub transakcyjnych – zależnie od zastosowania,
- potencjalnie bardzo duża liczba anonimowych użytkowników jednoczesnych,
- bardzo ważne opcje bezpieczeństwa danych,
- przykłady:
  - baza danych portalu e-biznesowego.

Bazy multimedialne

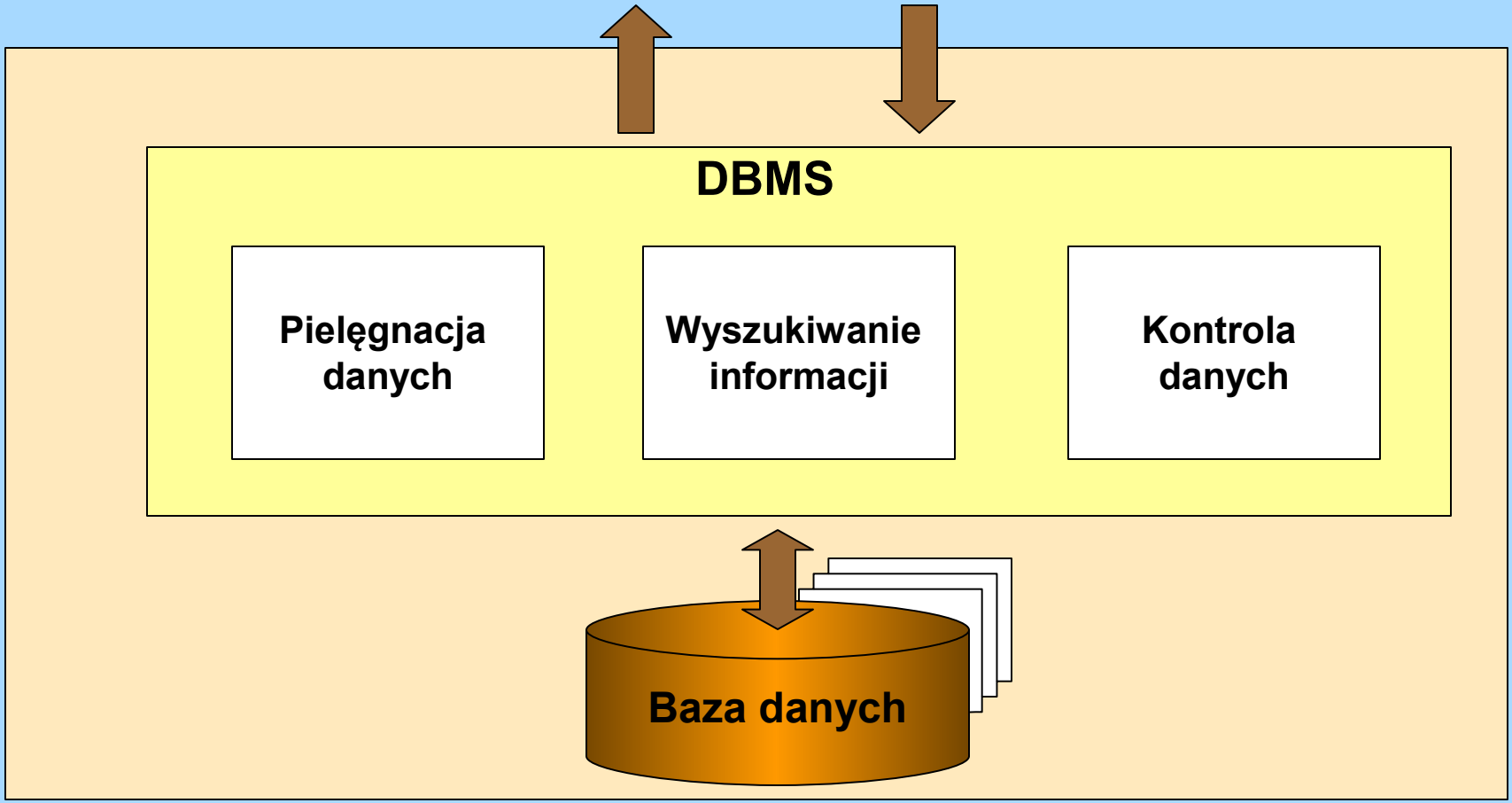
- obok prostych typów danych przechowują dźwięki, obrazy, filmy, animacje,
- potencjalnie bardzo duży rozmiar rekordu,
- specjalne metody wyszukiwania i udostępniania danych,
- przykłady:
  - system sprzedaży muzyki i filmów *on-line*.

Architektura ANSI/SPARC

American National Standards Institute - Standards Planning And Requirements Committee



System zarządzania bazą danych (DBMS)



Funkcje systemu DBMS

- funkcje CRUD,
- słownik danych,
- zarządzanie transakcjami,
- sterowanie współbieżnością,
- odtwarzanie bazy,
- uprawnienia,
- komunikacja pomiędzy różnymi warstwami,
- zapewnienie integralności danych,
- narzędzia do administrowania.

Budowa DBMS

Zestaw narzędzi DBMS

- konsola zarządzania serwerem,
- konsola procesora zapytań,
- monitor operacji na bazie danych,
- narzędzia optymalizacyjne,
- usługi importu i eksportu danych,

Narzędzia DBMS - przykład

Interfejs DBMS – język SQL

- **język definiowania danych** (ang. *data definition language*, DDL)
- **język operowania danymi** (ang. *data manipulation language*, DML)
- **język kontroli danych** (ang. *data control language*, DCL)
- **język integralności danych** (ang. *data integrity language*, DIL)

Język definiowania danych, DDL

Służy do tworzenia i modyfikowania struktury bazy danych: tabel i ograniczeń deklaratywnych, relacji, innych obiektów bazy danych (np. widoków).

Język operowania danymi, DML

Służy do implementacji tzw. operacji CRUD: wstawiania, odczytywania, modyfikowania i usuwania danych.

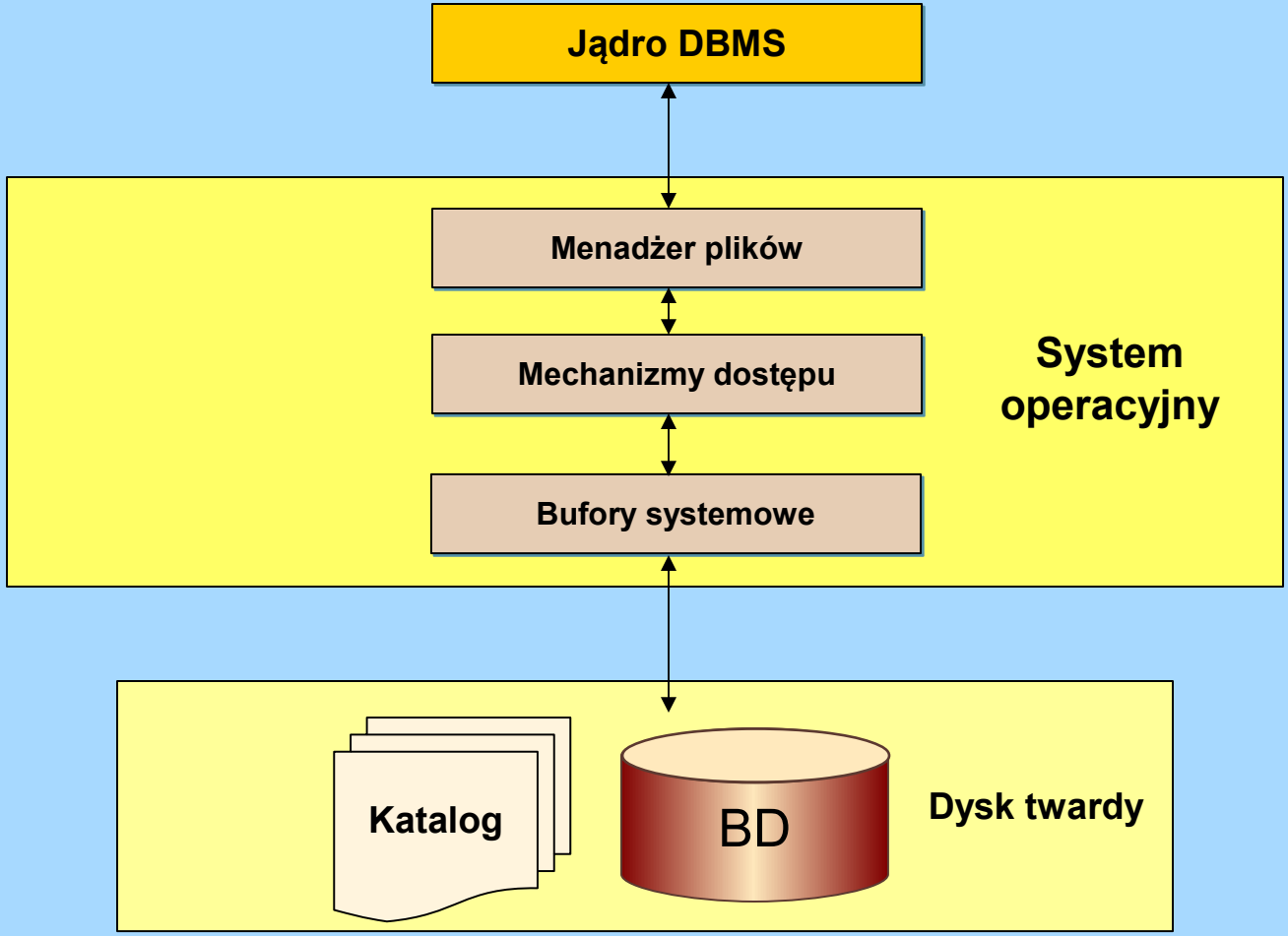
Język kontroli danych, DCL

Wyodrębniony podzbiór języka SQL, służący do definiowania systemu bezpieczeństwa: użytkowników, ról oraz ich uprawnień na serwerze i bazach danych.

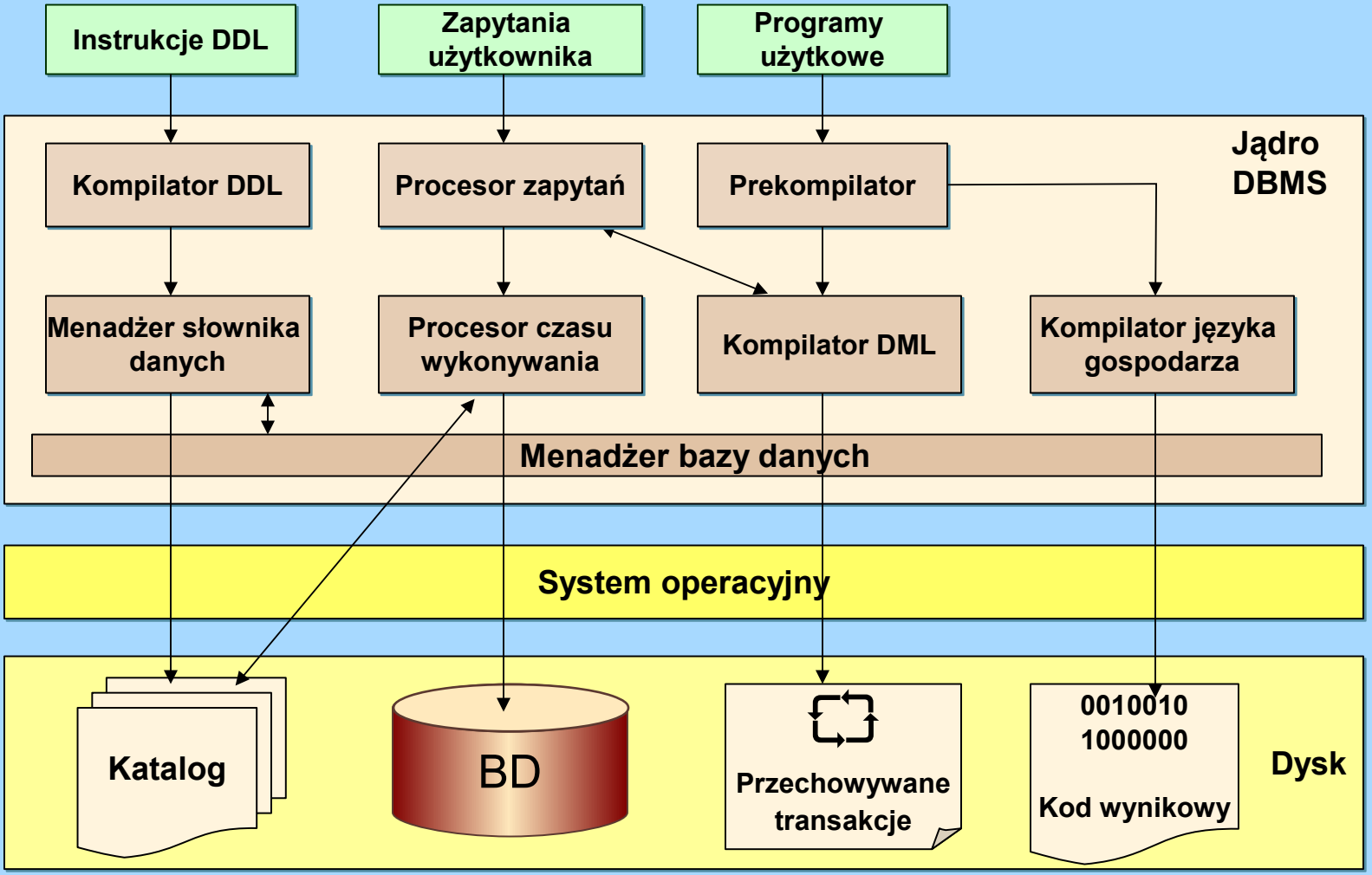
Język integralności danych, DIL

Wykorzystywany do definiowania ograniczeń deklaratywnych i reguł, które zapewniają integralność danych. Ponieważ podzbiór ten częściowo nachodzi na DDL, rzadko jest wyodrębniany.

Współpraca DBMS i systemu operacyjnego (OS)



Jądro DBMS



Plan wykładu

1. Podstawowe typy baz danych.

2. System zarządzania bazą danych (DBMS).

3. Prezentacja systemu MS SQL Server 2008 R2 Ent Edition.

Microsoft SQL Server

- System MS SQL Server jest obecnie znaczącym produktem bazodanowym na rynku, konkurującym z takimi systemami, jak: Oracle, Sybase, DB2.
- MS SQL Server spełnia większość reguł Codda lub pozwala na taką konfigurację, w której reguły te są spełniane.
- System charakteryzuje się dobrym stosunkiem wydajność/cena (np. mierzonym ceną jednej transakcji). Powszechnie uznawane zestawienia można zobaczyć na stronie: **www.tpc.org**.
- MS SQL Server jest dostępny wyłącznie na platformach: MS Windows.

Microsoft SQL Server - wersje

- **Enterprise Edition** – wersja najbardziej zaawansowana, kompletna, skalowalna i wydajna; przeznaczona do dużych środowisk produkcyjnych.
- **Standard Edition** – wersja mniej zaawansowana, skalowalna i wydajna od Enterprise Edition; przeznaczona dla małych i średnich firm.
- **Developer Edition** – wersja o funkcjonalności Enterprise, ale przeznaczona wyłącznie do celów testowych i programistycznych.
- **Express Edition –** wersja przeznaczona dla indywidualnego użytkownika, wymagającego lokalnej bazy danych, pracującego bez połączenia z siecią, także na komputerach przenośnych.

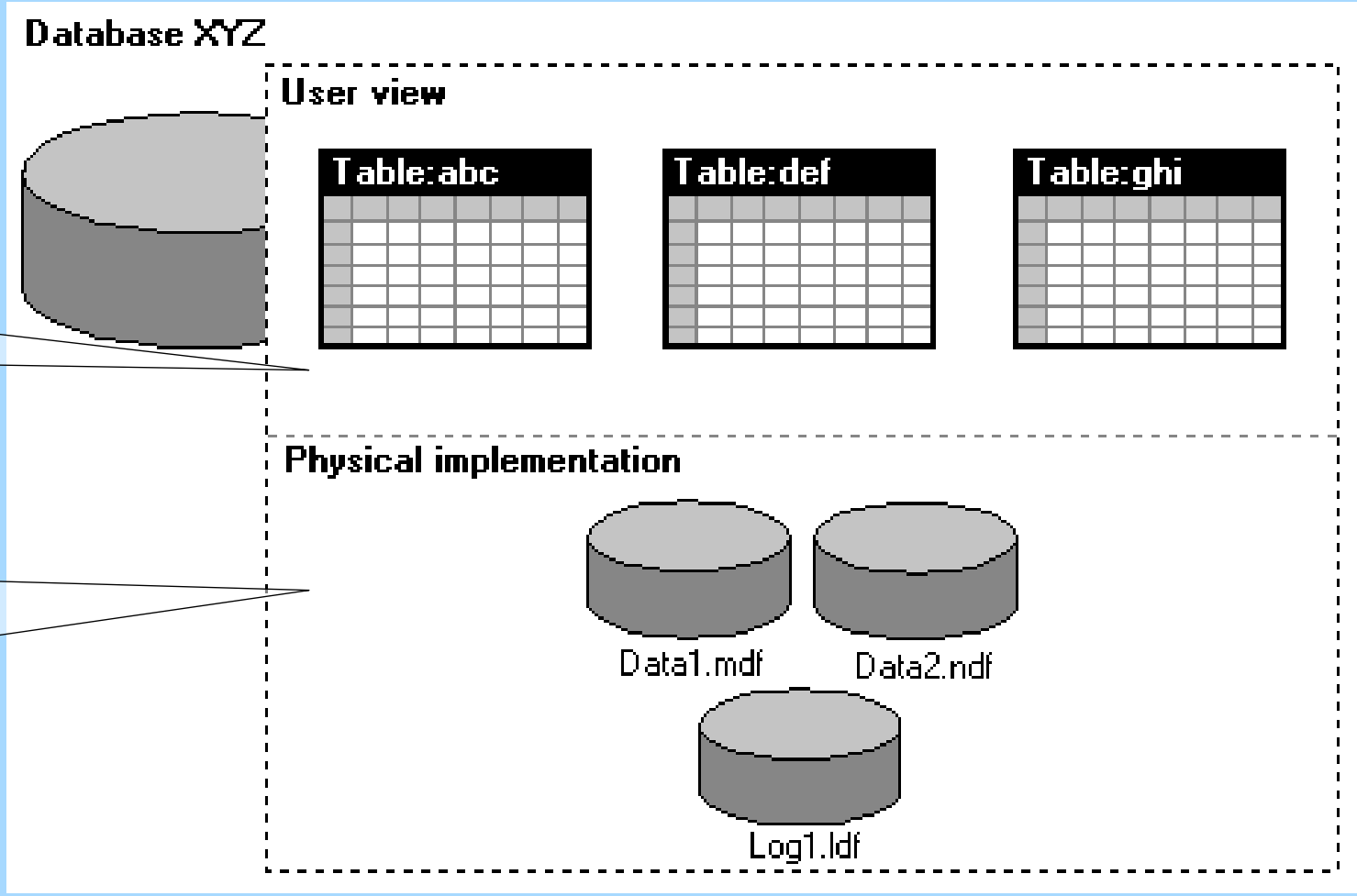
Microsoft SQL Server - cechy

- Wbudowany język Transact-SQL (T-SQL) jest zgodny ze standardem ANSI-92 (SQL2) i zawiera komplet funkcji DDL, DML, DIL i DCL.
- Wygodne oprogramowywanie bazy danych z poziomu interfejsu graficznego i języka SQL.
- Bardzo dobra integracja z architekturą klient-serwer opartą na rozwiązaniach Microsoft (np. MS .NET).
- Wydajna praca w środowisku heterogenicznym (współpraca z serwerami innymi, niż MS SQL Server).

Systemowe bazy danych

- **Master** – zawiera wszystkie informacje systemowe dotyczące SQL Servera: definicje obiektów, konta logowania, ustawienia konfiguracyjne, systemowe procedury przechowywane, informacje o istnieniu baz danych. Jeżeli baza *master* ulegnie uszkodzeniu, serwer nie uruchomi się (należy tworzyć jej kopie zapasowe).
- **Msdb** – przechowuje informacje dla SQL Server Agent, m.in.: zadania, alarmy, operacje automatycznego tworzenia kopii zapasowych.
- **Model** – szablon, na podstawie którego tworzone są wszystkie bazy danych użytkowników. Można ją modyfikować zgodnie z potrzebami projektowymi.
- **Tempdb –** przechowuje obiekty tymczasowe: tabele tymczasowe, tymczasowe procedury przechowywane i obiekty tworzone przez system.

Microsoft SQL Server – architektura bazy danych



Widok użytkownika: tabele.

Fizyczna implementacja bazy - przynajmniej dwa pliki: podstawowy (*.mdf) i dziennika transakcji (*.ldf).

Źródło: MICROSOFT, *Books On-Line* – dokumentacja elektroniczna systemu MS SQL Server 2000.

Pliki bazy danych

- **Plik podstawowy** (*.mdf) – wchodzi w skład każdej bazy danych; zawiera obiekty systemowe skopiowane z bazy *model* (tzw. katalog bazy danych) i obiekty użytkownika (np. tabele, widoki).
- **Pliki dodatkowe** (*.ndf) – tworzone są opcjonalnie przez użytkownika w celu zwiększenia wydajności lub bezpieczeństwa danych (np. poprzez umieszczenia na oddzielnych dyskach twardych; zawierają obiekty użytkownika bez obiektów systemowych.
- **Plik dziennika transakcji** (*.ldf) – wchodzi w skład każdej bazy; stanowi chronologiczny rejestr wszystkich transakcji, wykonywanych w bazie danych; pozwala na odtwarzanie danych w przypadku błędów lub awarii.

Składniki bazy danych

- **tabele** (tables) – podstawowe struktury danych, zawierające: pola, typy, wartości domyślne, klucze, ograniczenia, procedury wyzwalane;
- **indeksy** (indexes) – struktury pozwalające na przyśpieszenie dostępu do danych w tabelach;
- **widoki** (views) – tabele wirtualne, powstające w wyniku selekcji, projekcji lub złączenia tabel fizycznych;
- **procedury przechowywane** (stored procedures) – instrukcje SQL przechowywane na serwerze w formie prekompilowanej (pozwala to na ich szybsze wykonywanie);
- **role i użytkownicy** (roles, users) – pozwalają na zarządzanie bezpieczeństwem poprzez definiowanie dostępu do obiektów bazy.

Microsoft SQL Server – pokaz środowiska

- **SQL Server Management Studio:**
  - rejestrowanie nowego serwera SQL;
  - tworzenie nowej bazy danych;
  - tworzenie nowej tabeli, definiowanie pól i ograniczeń;
  - tworzenie relacji i diagramów;
  - tworzenie widoków;
  - tworzenie procedur przechowywanych.
  - tworzenie i uruchamianie zapytań;
  - plan wykonania;
  - uruchamianie procedur przechowywanych.
- **Business Intelligence Development Studio**.

Literatura

1. BEYNON-DAVIES P., *Systemy baz danych – nowe wydanie*, WNT, Warszawa 2003.
2. MICROSOFT, *Books On-Line* – dokumentacja systemu *MS SQL Server*, Microsoft Corp. 1988 – 2000.
3. MSDN page: http://www.msdn.microsoft.com.
4. PANKOWSKI T., *Podstawy baz danych*, Wydawnictwo Naukowe PWN, Warszawa, 1992.
5. RANKINS R., JENSEN P., BERTUCCI P., *Microsoft SQL Server 2000. Księga eksperta.*, HELION, Gliwice 2003 (pozycja dostępna w bibliotece WSIZ „Copernicus").
6. WAYMIRE R., SAWTELL R., *MS SQL Server 2000 dla każdego*, HELION, Gliwice 2002 (pozycja dostępna w bibliotece WSIZ "Copernicus").

Bazy danych

Wykład 1_2

Dziękuję za uwagę !

Bazy danych

Wykład 2_1

Temat: Relacyjny model danych

Sławomir Świętoniowski

slawomir-swietoniowski@wp.pl

Plan wykładu

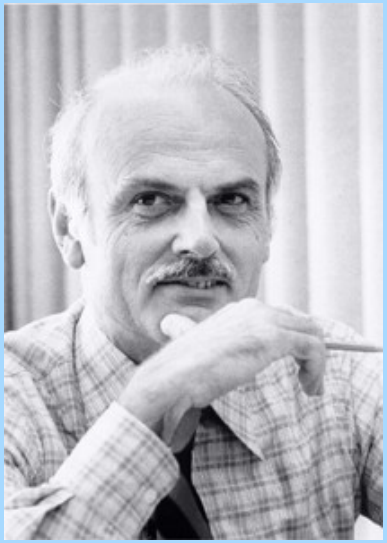
1. **Założenia relacyjnego modelu danych.**
2. Definicja podstawowych pojęć.
3. Algebra relacyjna i język SQL.
4. Reguły Codda dla systemów relacyjnych baz danych.

Model relacyjny - historia

Główny twórca:

Edgar F. Codd

(23 VIII 1923 - 18 IV 2003)



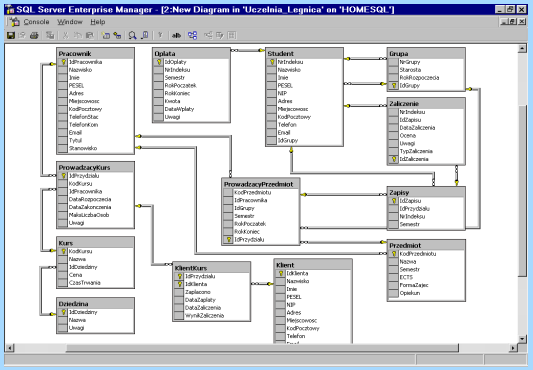
Źródło: Wikipedia (http://en.wikipedia.org).

Ważniejsze publikacje:

- "A Relational Model of Data for Large Shared Data Banks", 1970, CACM 13, No. 6.
- „Extending the Relational Database Model to Capture More Meaning", ACM Transactions on Database Systems, 1979, Vol. 4, No. 4, pp. 397-434.
- „Relational Database: A Practical Foundation for Productivity", Communications of ACM, 1982, Vol. 25, No. 2.
- „The Relational Model for Database Management: Version 2", Reading, Mass., Addison-Wesley, 1990.

Model relacyjny – podstawowe założenia

- Każda tabela w bazie danych ma jednoznaczną nazwę.
- Każda kolumna ma jednoznaczną nazwę w ramach jednej tabeli.
- Wszystkie wartości kolumny muszą być tego samego typu – zdefiniowane na tej samej dziedzinie
- Porządek kolumn w tabeli nie jest istotny.
- Każdy wiersz w tabeli musi być różny - powtórzenia nie są dozwolone.
- Porządek wierszy nie jest istotny.
- Każda wartość pola tabeli (na przecięciu kolumna/wiersz) powinna być atomowa – nie może być ciągiem, ani zbiorem.



IdPracownika	Nazwisko	Imie	NIP	PESEL	Adres	Miejscowosc
1	Nowakowski	Andrzej	612-412-54-64	57121943212	ul. Świerkowa 6	Wrocław
2	Kowalski	Jan	663-654-76-87	72013142337	ul. Strzelecka 15/8	Wrocław
3	Janicki	Bogdan	432-543-654-6	49042343259	ul. Pastelowa 58/9	Wrocław
4	Marcinkowski	Piotr	789-098-23-45	76110309878	ul. Lakiernicza 98A/	Wrocław
5	Andrzejewski	Grzegorz	980-432-23-12	52082898152	ul. Nobla 8/3	Wrocław
6	Piotrowski	Bartlomiej	780-678-66-11	65102189131	ul. Nadrzeczna 16/	Legnica
7	Bogdanska	Ewa	430-543-55-22	79031487924	ul. Wrocławska 23/	Legnica
8	Grzegorzewski	Pawel	250-532-99-90	76061898778	ul. Karkonoska 58	Wrocław
9	Styczen	Tomasz	610-220-96-52	68071678156	ul. Janowska 43/63	Legnica
10	Romanowski	Janusz	616-420-90-19	74090209835	ul. Góralska 43	Legnica

Plan wykładu

1. Założenia relacyjnego modelu danych.
2. **Definicja podstawowych pojęć.**
3. Algebra relacyjna i język SQL.
4. Reguły Codda dla systemów relacyjnych baz danych.

Def. 1 Relacja matematyczna

Niech dane będą zbiory $D_1, D_2, ..., D_n$.

Relacją matematyczną R nad tymi zbiorami nazywamy dowolny podzbiór iloczynu kartezjańskiego nad tymi zbiorami, tzn.

$R \subseteq D_1 \times D_2 \times ... \times D_n = \{(d_1, d_2, ..., d_n): d_i \in D_i, i=1,2, ...,n\}$.

Przykład:

R: > (relacja większości)

$D_1 = \{3, 4\}$

$D_2 = \{1, 2, 3\}$

$D_1 \times D_2 = \{(3, 1), (3, 2), (3, 3), (4, 1), (4, 2), (4, 3)\}$

$R \subseteq D_1 \times D_2 = \{(3, 1), (3, 2), (4, 1), (4, 2), (4, 3)\}$

Na podstawie: PANKOWSKI T., *Podstawy baz danych*, Wydawnictwo Naukowe PWN, Warszawa, 1992.

Def. 2. Wiersz

Zbiór atrybutów (kolumn): $U = \{A_1, A_2, ...\}$. Dla każdego $A \in U$, $DOM(A)$ jest zbiorem wartości **dziedziną** (domeną) atrybutu A.

Wierszem typu U nazywamy dowolną funkcję:

$$f: U \rightarrow \cup \{DOM(A): A \in U\}$$

taką, że dla dowolnego $A \in U$, $f(A) \in DOM(A)$.

Zbiór wszystkich wierszy typu U oznaczamy jako: $WIERSZ(U)$.

Na podstawie: PANKOWSKI T., *Podstawy baz danych*, Wydawnictwo Naukowe PWN, Warszawa, 1992.

Wiersz - przykład

Zbiór atrybutów: $U = \{imię, nazwisko, wiek\}$.

$DOM(imię) = \{Jan, Andrzej\}$

$DOM(nazwisko) = \{Nowak, Kowalski, Jabłoński\}$

$DOM(wiek) = \mathbb{N} \cap [1; 130]$

Przykładowy wiersz typu U:

$r(U) = \{(imię, Andrzej), (nazwisko, Nowak), (wiek, 22)\}$

$WIERSZ(U) = \{$
$\{(imię, Jan), (nazwisko, Nowak), (wiek, 1)\},$
$\{(imię, Jan), (nazwisko, Nowak), (wiek, 2)\},$
$...,$
$\{(imię, Andrzej), (nazwisko, Jabłoński), (wiek, 130)\}\ \}$

Def. 3. Relacja (tabela)

Relacją (ang. *relation*) typu U nazywamy dowolny, skończony podzbiór zbioru *WIERSZ* (U).

Zbiór wszystkich relacji (tabel) typu U oznaczamy: *REL* (U).

Oznaczenia:

- Relacje typu U: $R(U)$, $S(U)$, $T(U)$,... lub R, S, T, ...
- Wiersze typu U: $r(U)$, $s(U)$, $t(U)$,... lub r, s, t, ...
- Podzbiory U: X, Y, Z, ...
- Wiersz $r(U)$: $r\ (U) = \{(A_1, a_1), (A_2, a_2), ..., (A_n, a_n)\}$
  lub w uproszczeniu: $r\ (U) = (a_1, a_2, ..., a_n)$
  np. $r\ (U) = (Andrzej, Nowak, 22)$

Relacja (tabela) - przykład

Tabela: Osoby

Imię	Nazwisko	Wiek
Jan	Kowalski	18
Andrzej	Jabłoński	37
Andrzej	Nowak	25

$$WIERSZ\,(U) = \{$$
$$\{(imię,\ Jan),\ (nazwisko,\ Nowak),\ (wiek,\ 1)\},$$
$$\{(imię,\ Jan),\ (nazwisko,\ Nowak),\ (wiek,\ 2)\},$$
$$\ldots,$$
$$\{(imię,\ Andrzej),\ (nazwisko,\ Jabłoński),\ (wiek,\ 130)\}\ \}$$

Def. 4. Zależność funkcyjna

Niech dana będzie tabela $R(U)$ i niech $X, Y \subseteq U$ będą zbiorami atrybutów. Mówimy, że w R spełniona jest zależność funkcyjna $X \rightarrow Y$, jeśli dla wszystkich wierszy w relacji R wartości atrybutów ze zbioru Y zależą od wartości atrybutów ze zbioru X. Mówimy wówczas, że Y zależy funkcyjnie od X lub, że X determinuje funkcyjnie Y.

Tabela *R*

Nr_indeksu	Nazwisko	Przedmiot	Ocena
1000	Kowalski	Bazy danych	4.5
1000	Kowalski	Akademia CISCO	4.0
1003	Morawski	Bazy danych	5.0
1006	Nowak	Bazy danych	3.0
1006	Nowak	Akademia CISCO	4.5

W tabeli R występują poniższe zależności funkcyjne:

$Nr\_indeksu \rightarrow Nazwisko$ oraz $\{Nr\_indeksu, Przedmiot\} \rightarrow Ocena$

Klucze główne i obce

- **Klucz główny** (ang. *primary key*, PK):
  - zbiór atrybutów, który identyfikuje jednoznacznie wiersze tabeli;
  - w tabeli może być kilka kluczy kandydujących, spośród których wybieramy jeden klucz główny (np. w tabeli [Osoba] kluczami kandydującymi mogą być kolumny [NIP], [PESEL], [IdOsoby]).
- **Klucz obcy** (ang. *foreign key*, FK):
  - pozwala na łączenie danych z różnych tabel;
  - zbiór atrybutów w tabeli, który czerpie swoje wartości z tej samej dziedziny, co klucz główny tabeli powiązanej.

Integralność encji

- Każda tabela musi mieć klucz główny, który jednoznacznie identyfikuje wiersze tej tabeli.
- Klucz główny nie może zawierać wartości pustych (null).
- Zabronione są powtórzenia wierszy w ramach jednej tabeli.

Integralność referencyjna

- Klucz obcy może przyjmować jedną z dwóch wartości:
  - klucz główny z tabeli powiązanej;
  - wartość NULL (jeżeli nie koliduje to z innymi regułami integralności).
- Niedozwolone są wskazania poprzez klucz obcy na wiersz, który nie istnieje.
- Kaskadowa aktualizacja i usuwanie zależą od konkretnego zastosowania (np. jeśli usuwamy fakturę VAT z tabeli [Faktura], to usuwamy także wszystkie jej pozycje z powiązanej tabeli [Pozycja]; natomiast jeśli usuwamy grupę studentów z tabeli [Grupa], to raczej nie usuwamy z bazy wszystkich studentów z tej grupy, zapisanych w powiązanej tabeli [Student]).

Plan wykładu

1. Założenia relacyjnego modelu danych.
2. Definicja podstawowych pojęć.
3. **Algebra relacyjna i język SQL.**
4. Reguły Codda dla systemów relacyjnych baz danych.

Algebra relacyjna

Algebra relacyjna do zbiór operacji na tabelach i wierszach. Większość tych operacji ma bardzo duże znaczenie praktyczne i stanowi podstawę działania komend języka SQL. (Operacje o znaczeniu praktycznym są pogrubione).

- **projekcja**,
- **selekcja**,
- **złączenie**,
- **suma**,
- **różnica**,
- **przekrój**,
- *dopełnienie*,
- *podzielenie*.

Projekcja tabeli

Projekcja tabeli *R* jest to tabela *T*, w której są tylko wybrane kolumny z tabeli *R*.

Tabela *R*

Imie	Nazwisko	Wiek
Jan	Kowalski	18
Andrzej	Jabłoński	37
Andrzej	Nowak	25
Krzysztof	Nowak	25
Leszek	Morawski	59

```
-- Odpowiednik w języku SQL:
SELECT DISTINCT Nazwisko, Wiek
FROM R
```

Projekcja tabeli *R* na kolumny [Nazwisko, Wiek]

Nazwisko	Wiek
Kowalski	18
Jabłoński	37
Nowak	25
Morawski	59

Selekcja tabeli

Selekcja tabeli *R* jest to tabela *T*, w której są tylko wybrane wiersze z tabeli *R*, zgodnie z nałożonym warunkiem selekcji.

Tabela *R*

Imie	Nazwisko	Wiek
Jan	Kowalski	18
Andrzej	Jabłoński	37
Andrzej	Nowak	25
Krzysztof	Nowak	25
Leszek	Morawski	59

Selekcja tabeli *R* - warunek: Wiek > 30.

Imie	Nazwisko	Wiek
Andrzej	Jabłoński	37
Leszek	Morawski	59

```
-- Odpowiednik w języku SQL:
SELECT *
FROM R
WHERE Wiek > 30 -- Warunek selekcji.
```

Złączenie tabel

Złączeniem tabel *R* i *S* jest tabela *T*, w której wiersze mają połączone kolumny z tabel *R* i *S*. W tabeli *T* są tylko te wiersze, w których nastąpiło dopasowanie wartości określonych kolumn z obu tabel.

Tabela *R*

Imie	Nazwisko	KodPrzedmiotu
Jan	Kowalski	INF507
Andrzej	Jabłoński	INF517
Krzysztof	Nowak	INF517
Leszek	Morawski	INF517

N — 1

Tabela *S*

KodPrzedmiotu	Nazwa	ECTS
INF407	Bazy danych	4
INF507	Sieciowe bazy danych	4
INF517	Grafika komputerowa	5

Złączone tabele *R* i *S*

Imie	Nazwisko	KodPrzedmiotu	Nazwa	ECTS
Jan	Kowalski	INF507	Sieciowe bazy danych	4
Andrzej	Jabłoński	INF517	Grafika komputerowa	5
Krzysztof	Nowak	INF517	Grafika komputerowa	5
Leszek	Morawski	INF517	Grafika komputerowa	5

```
-- Odpowiednik w języku SQL:
SELECT Imie, Nazwisko, KodPrzedmiotu, Nazwa, ECTS
FROM R INNER JOIN S ON R.KodPrzedmiotu = S.KodPrzedmiotu
```

Suma tabel

Sumą tabel *R* i *S* jest tabela *T*, w której są połączone wiersze z tabel *R* i *S* (bez powtórzeń).

Tabela *R*

Imie	Nazwisko	Wiek
Jan	Kowalski	18
Andrzej	Jabłoński	37
Andrzej	Nowak	25

Tabela *S*

Imie	Nazwisko	Wiek
Krzysztof	Nowicki	23
Leszek	Morawski	59

Suma tabel *R* i *S*

Imie	Nazwisko	Wiek
Jan	Kowalski	18
Andrzej	Jabłoński	37
Andrzej	Nowak	25
Krzysztof	Nowicki	23
Leszek	Morawski	59

```
-- Odpowiednik w języku SQL:
SELECT *
FROM R
 UNION -- Operator sumy.
SELECT *
FROM S
```

Różnica tabel

Różnicą tabel *R* i *S* jest tabela *T*, w której są wiersze z tabeli R, które nie występują w tabeli *S*.

Tabela *R*

Imie	Nazwisko	Wiek
Jan	Kowalski	18
Andrzej	Jabłoński	37
Andrzej	Nowak	25

Tabela *S*

Imie	Nazwisko	Wiek
Andrzej	Nowak	25
Leszek	Morawski	59

Różnica tabel: *R – S*

Imie	Nazwisko	Wiek
Jan	Kowalski	18
Andrzej	Jabłoński	37

```
-- Odpowiednik w języku SQL:
SELECT *
FROM R
WHERE NOT EXISTS
    (SELECT *
     FROM S
     WHERE R.Imie = S.Imie AND
     R.Nazwisko = S.Nazwisko AND
     R.Wiek = S.Wiek)
```

Przekrój tabel

Przekrojem tabel *R* i *S* jest tabela *T*, której wiersze jednocześnie występują w tabeli R i w tabeli *S*.

Tabela *R*

Imie	Nazwisko	Wiek
Jan	Kowalski	18
Andrzej	Jabłoński	37
Andrzej	Nowak	25

Tabela *S*

Imie	Nazwisko	Wiek
Andrzej	Nowak	25
Leszek	Morawski	59

Przekrój tabel *R i S*

Imie	Nazwisko	Wiek
Andrzej	Nowak	25

```
-- Odpowiednik w języku SQL:
SELECT *
FROM R
WHERE EXISTS
    (SELECT *
     FROM S
     WHERE R.Imie = S.Imie AND
     R.Nazwisko = S.Nazwisko AND
     R.Wiek = S.Wiek)
```

Co to jest język SQL?

SQL – Structured Query Language

(wymowa: `*es-`kju-`el* lub `*si:kuel*):

- Najbardziej rozpowszechniony, ustandaryzowany język baz danych.
- Pozwala na:
  - **definiowanie struktury bazy danych** (pozbiór DDL – Data Definition Language), np. CREATE TABLE, ALTER VIEW, DROP PROCEDURE;
  - **operowanie na danych**: dodawanie, udostępnianie, modyfikowanie i usuwanie (pozdbiór DML – Data Manipulation Language), np. INSERT, SELECT, UPDATE, DELETE.
  - **zarządzanie dostępem do danych** (podzbiór DCL – Data Control Language), np. GRANT, DENY, REVOKE;
  - **definiowanie więzów integralności** (podzbiór DIL – Data Integrity Language), np. CREATE RULE.

Plan wykładu

1. Założenia relacyjnego modelu danych.
2. Definicja podstawowych pojęć.
3. Algebra relacyjna i język SQL.
4. **Reguły Codda dla systemów relacyjnych baz danych.**

Reguły Codda dla relacyjnych systemów DBMS

- W 1985 roku twórca relacyjnych baz danych Edgar F. Codd opublikował zestaw 12 reguł, które powinien spełniać relacyjny system zarządzania bazą danych – DBMS (ang. *Database Management System*).
- Reguły Codda zapewniają zgodność systemu DBMS z modelem relacyjnym, na którym jest on oparty, a przez to wpływają korzystnie na integralność danych w bazie i wydajność ich przetwarzania.
- Współczesne produkty DBMS zazwyczaj spełniają jedynie część tych reguł (w najlepszym przypadku około 10 z 12).
- Niektóre reguły są trudne do realizacji w rzeczywistych systemach baz danych (np. reguła 11 – niezależności od rozproszenia).

Reguła podstawowa

Każdy relacyjny system DBMS powinien być zdolny do zarządzania bazami danych wyłącznie poprzez swoje właściwości relacyjne.

Reguła 1 – Reguła informacyjna (ang. *The Information Rule*)

Wszystkie informacje w relacyjnej bazie danych są reprezentowane jawnie na poziomie logicznym w dokładnie jeden sposób – jako wartości pól w tabelach.

Reguła 2 – Reguła gwarantowanego dostępu (ang. *Guaranteed Access Rule*)

Do każdej danej (wartości atomowej) w relacyjnej bazie danych jest zagwarantowany jednoznaczny dostęp logiczny poprzez podanie kombinacji: nazwy tabeli, wartości klucza głównego i nazwy kolumny.

Reguła 3 – Systematyczna obsługa wartości NULL (ang. *Systematic Treatment of Null Values*)

Wartość pola tabeli może przyjmować wartość NULL (różną od ciągu pustego, ciągu spacji, zera i innej liczby), o ile nie jest to kolumna klucza głównego. Relacyjny DBMS powinien zapewniać systematyczną obsługę wartości NULL, które reprezentują brakujące informacje.

Reguła 4 – Dynamiczny katalog on-line oparty na modelu relacyjnym

(ang. *Dynamic On-Line Catalog Based on the Relational Model*)

Relacyjny DBMS musi zapewniać dostęp do struktury bazy danych w taki sam sposób, jak do zwykłych danych.

Komentarz: Jest to zazwyczaj realizowane poprzez przechowywanie definicji struktury w specjalnych tabelach systemowych (zwanych katalogiem).

Reguła 5 – Reguła ogólnego subjęzyka danych (ang. *Comprehensive Data Sublanguage Rule*)

Relacyjny DBMS musi obsługiwać przynajmniej jeden język o dobrze zdefiniowanej składni jako łańcuch znaków, który umożliwia:

- definiowanie danych,
- manipulowanie danymi,
- definiowanie więzów integralności,
- definiowanie uprawnień dostępu do danych,
- kontrolę transakcji.

Komentarz: Wszystkie komercyjne, relacyjne DBMS używają do tego języka SQL (ang. *Structured Query Language*).

Reguła 6 – Reguła modyfikacji perspektyw (ang. *View Updating Rule*)

Dane mogą być logicznie prezentowane użytkownikowi w różny sposób za pomocą widoków (perspektyw). Każdy widok powinien umożliwiać taki sam poziom dostępu do danych (np. INSERT, UPDATE, DELETE), jak tabele, na których jest on oparty.

Komentarz: W praktyce reguła ta jest trudna do spełnienia i nie jest w pełni realizowana przez żaden współczesny DBMS.

Reguła 7 – Wstawianie, aktualizacja i usuwanie na wysokim poziomie

(ang. *High-level Insert, Update, and Delete*)

Dane otrzymywane w wyniku zapytań do bazy relacyjnej mogą być złożone z wielu wierszy i wielu tabel.

Komentarz: Innymi słowy, operacje wstawiania, modyfikacji i usuwania powinny być dostępne dla każdego zbioru danych, który może być wyszukany, a nie tylko dla pojedynczego wiersza w jednej tabeli.

Reguła 8 – Fizyczna niezależność danych (ang. *Physical Data Independence*)

Programy użytkowe i operacje interfejsu powinny być fizycznie niezależne od jakichkolwiek zmian w sposobie przechowywania lub metodach dostępu (np. rozkład plików na dyskach, struktura indeksów).

Reguła 9 – Logiczna niezależność danych (ang. *Logical Data Independence*)

Sposób widzenia danych przez użytkownika powinien być niezależny od jakichkolwiek zmian w strukturze logicznej bazy danych (np. w strukturze tabel).

Komentarz: Jest to reguła szczególnie trudna do spełnienia, ponieważ w większości baz danych istnieje silny związek pomiędzy prezentowanym widokiem danych, a rzeczywistą strukturą tabel, na których ten widok jest oparty.

Reguła 10 – Niezależność integralności (ang. *Integrity Independence*)

Język relacyjnej bazy danych (np. SQL) powinien umożliwiać definiowanie więzów integralności – ograniczeń zachowujących bazę w stanie spójności – bez konieczności ich definiowania w programach użytkowych.

Komentarz: Ta własność nie jest realizowana we wszystkich produktach. Jednakże jako minimum wszystkie bazy zapewniają z poziomu SQL:

- integralność encji – klucz główny tabeli nie może przyjmować wartości NULL,
- integralność referencyjną – dla każdego klucza obcego musi istnieć odpowiedni klucz główny w tabeli powiązanej.

Reguła 11 – Niezależność od rozproszenia (ang. *Distribution Independence*)

Użytkownik nie powinien dostrzegać rozproszenia bazy danych – ich podziału na więcej, niż jedną lokację (serwer).

Komentarz: W rzeczywistych systemach reguła ta jest trudna do spełnienia (np. z powodu opóźnienia w przesyłaniu danych ze zdalnego serwera).

Reguła 12 – Reguła nieomijania reguł
(ang. *Nonsubversion Rule*)

W relacyjnym DBMS nie powinno być innej metody modyfikacji struktury bazy danych oprócz wbudowanego języka operowania na danych (np. SQL).

Komentarz: W wielu współczesnych systemach są dostępne narzędzia administracyjne, które umożliwiają bezpośrednią manipulację na strukturze bazy i przechowywanych danych (a więc niezgodnie z niniejszą regułą), choć w niektórych przypadkach narzędzia są jedynie graficzną nakładką na wbudowany język SQL.

Literatura

1. BEYNON-DAVIES P., *Systemy baz danych – nowe wydanie*, WNT, Warszawa 2003.
2. MICROSOFT, *Books On-Line* – dokumentacja systemu *MS SQL Server*, Microsoft Corp. 1988 – 2000.
3. PANKOWSKI T., *Podstawy baz danych*, Wydawnictwo Naukowe PWN, Warszawa, 1992.
4. RANKINS R., JENSEN P., BERTUCCI P., *Microsoft SQL Server 2000. Księga eksperta.*, HELION, Gliwice 2003 (książka dostępna w bibliotece WSIZ „Copernicus").

Bazy danych

Wykład 2_1

Dziękuję za uwagę !

Bazy danych

Wykład 2_2

Temat: Integralność relacyjnej bazy danych

Sławomir Świętoniowski

slawomir-swietoniowski@wp.pl

Plan wykładu

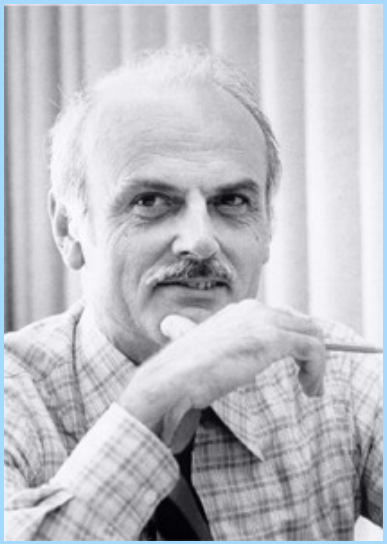
1. **Założenia relacyjnego modelu danych - przypomnienie.**
2. Definicja podstawowych pojęć.
3. Normalizacja bazy danych: 1NF, 2NF, 3NF.

Model relacyjny - historia

Główny twórca:

Edgar F. Codd

(23 VIII 1923 - 18 IV 2003)



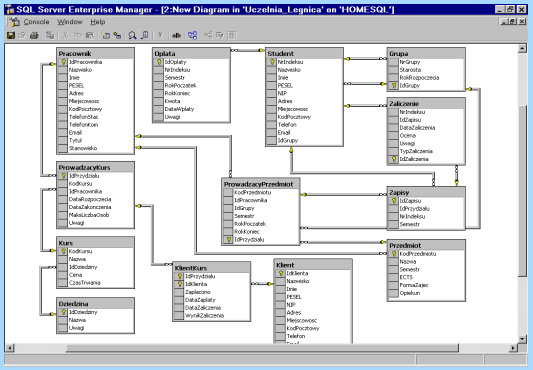
Źródło: Wikipedia (http://en.wikipedia.org).

Ważniejsze publikacje:

- "A Relational Model of Data for Large Shared Data Banks”, 1970, CACM 13, No. 6.
- „Extending the Relational Database Model to Capture More Meaning”, ACM Transactions on Database Systems, 1979, Vol. 4, No. 4, pp. 397-434.
- „Relational Database: A Practical Foundation for Productivity”, Communications of ACM, 1982, Vol. 25, No. 2.
- „The Relational Model for Database Management: Version 2”, Reading, Mass., Addison-Wesley, 1990.

Model relacyjny – podstawowe założenia

- Każda tabela w bazie danych ma jednoznaczną nazwę.
- Każda kolumna ma jednoznaczną nazwę w ramach jednej tabeli.
- Wszystkie wartości kolumny muszą być tego samego typu – zdefiniowane na tej samej dziedzinie
- Porządek kolumn w tabeli nie jest istotny.
- Każdy wiersz w tabeli musi być różny – powtórzenia nie są dozwolone.
- Porządek wierszy nie jest istotny.
- Każda wartość pola tabeli (na przecięciu kolumna/wiersz) powinna być atomowa – nie może być ciągiem, ani zbiorem.



IdPracownika	Nazwisko	Imie	NIP	PESEL	Adres	Miejscowosc
1	Nowakowski	Andrzej	612-412-54-64	57121943212	ul. Świerkowa 6	Wrocław
2	Kowalski	Jan	663-654-76-87	72013142337	ul. Strzelecka 15/8	Wrocław
3	Janicki	Bogdan	432-543-654-6	49042343259	ul. Pastelowa 58/9	Wrocław
4	Marcinkowski	Piotr	789-098-23-45	76110309878	ul. Lakiernicza 98A/	Wrocław
5	Andrzejewski	Grzegorz	980-432-23-12	52082898152	ul. Nobla 8/3	Wrocław
6	Piotrowski	Bartlomiej	780-678-66-11	65102189131	ul. Nadrzeczna 16/	Legnica
7	Bogdanska	Ewa	430-543-55-22	79031487924	ul. Wrocławska 23/	Legnica
8	Grzegorzewski	Pawel	250-532-99-90	76061898778	ul. Karkonoska 58	Wrocław
9	Styczen	Tomasz	610-220-96-52	68071678156	ul. Janowska 43/63	Legnica
10	Romanowski	Janusz	616-420-90-19	74090209835	ul. Góralska 43	Legnica

Plan wykładu

1. Założenia relacyjnego modelu danych - przypomnienie.
2. **Definicja podstawowych pojęć.**
3. Normalizacja bazy danych: 1NF, 2NF, 3NF.

Def. 1 Relacja matematyczna

Niech dane będą zbiory $D_1, D_2, ..., D_n$.

Relacją matematyczną R nad tymi zbiorami nazywamy dowolny podzbiór iloczynu kartezjańskiego nad tymi zbiorami, tzn.

$R \subseteq D_1 \times D_2 \times ... \times D_n = \{(d_1, d_2, ..., d_n): d_i \in D_i, i=1,2, ...,n\}$.

Przykład:

R: > (relacja większości)

$D_1 = \{3, 4\}$

$D_2 = \{1, 2, 3\}$

$D_1 \times D_2 = \{(3, 1), (3, 2), (3, 3), (4, 1), (4, 2), (4, 3)\}$

$R \subseteq D_1 \times D_2 = \{(3, 1), (3, 2), (4, 1), (4, 2), (4, 3)\}$

Na podstawie: PANKOWSKI T., *Podstawy baz danych*, Wydawnictwo Naukowe PWN, Warszawa, 1992.

Def. 2. Wiersz

Zbiór atrybutów (kolumn): $U = \{A_1, A_2, ...\}$. Dla każdego $A \in U$, $DOM(A)$ jest zbiorem wartości **dziedziną** (domeną) atrybutu A.

Wierszem typu U nazywamy dowolną funkcję:

$$f: U \rightarrow \cup \{DOM(A): A \in U\}$$

taką, że dla dowolnego $A \in U$, $f(A) \in DOM(A)$.

Zbiór wszystkich wierszy typu U oznaczamy jako: $WIERSZ(U)$.

Na podstawie: PANKOWSKI T., *Podstawy baz danych*, Wydawnictwo Naukowe PWN, Warszawa, 1992.

Wiersz - przykład

Zbiór atrybutów: $U = \{imię, nazwisko, wiek\}$.

$DOM(imię) = \{Jan, Andrzej\}$

$DOM(nazwisko) = \{Nowak, Kowalski, Jabłoński\}$

$DOM(wiek) = \mathbb{N} \cap [1; 130]$

Przykładowy wiersz typu U:

$r(U) = \{(imię, Andrzej), (nazwisko, Nowak), (wiek, 22)\}$

$WIERSZ(U) = \{$
$\{(imię, Jan), (nazwisko, Nowak), (wiek, 1)\},$
$\{(imię, Jan), (nazwisko, Nowak), (wiek, 2)\},$
$...,$
$\{(imię, Andrzej), (nazwisko, Jabłoński), (wiek, 130)\}\ \}$

Def. 3. Relacja (tabela)

Relacją (ang. *relation*) typu U nazywamy dowolny, skończony podzbiór zbioru *WIERSZ* (U).

Zbiór wszystkich relacji (tabel) typu U oznaczamy: *REL* (U).

Oznaczenia:

- Relacje typu U: $R(U)$, $S(U)$, $T(U)$,... lub R, S, T, ...
- Wiersze typu U: $r(U)$, $s(U)$, $t(U)$,... lub r, s, t, ...
- Podzbiory U: X, Y, Z, ...
- Wiersz $r(U)$: $r\ (U) = \{(A_1, a_1), (A_2, a_2), ..., (A_n, a_n)\}$
  lub w uproszczeniu: $r\ (U) = (a_1, a_2, ..., a_n)$
  np. $r\ (U)$ = (*Andrzej*, *Nowak*, 22)

Relacja (tabela) – przykład

Tabela: Osoby

Imię	Nazwisko	Wiek
Jan	Kowalski	18
Andrzej	Jabłoński	37
Andrzej	Nowak	25

$$WIERSZ\,(U) = \{$$
$$\{(imię,\ Jan),\ (nazwisko,\ Nowak),\ (wiek,\ 1)\},$$
$$\{(imię,\ Jan),\ (nazwisko,\ Nowak),\ (wiek,\ 2)\},$$
$$\ldots,$$
$$\{(imię,\ Andrzej),\ (nazwisko,\ Jabłoński),\ (wiek,\ 130)\}\ \}$$

Def. 4. Zależność funkcyjna

Niech dana będzie tabela $R(U)$ i niech $X, Y \subseteq U$ będą zbiorami atrybutów. Mówimy, że w R spełniona jest zależność funkcyjna $X \rightarrow Y$, jeśli dla wszystkich wierszy w relacji R wartości atrybutów ze zbioru Y zależą od wartości atrybutów ze zbioru X. Mówimy wówczas, że Y zależy funkcyjnie od X lub, że X determinuje funkcyjnie Y.

Tabela *R*

Nr_indeksu	Nazwisko	Przedmiot	Ocena
1000	Kowalski	Bazy danych	4.5
1000	Kowalski	Akademia CISCO	4.0
1003	Morawski	Bazy danych	5.0
1006	Nowak	Bazy danych	3.0
1006	Nowak	Akademia CISCO	4.5

W tabeli R występują poniższe zależności funkcyjne:

$Nr\_indeksu \rightarrow Nazwisko$ oraz $\{Nr\_indeksu, Przedmiot\} \rightarrow Ocena$

Klucze główne i obce

- **Klucz główny** (ang. *primary key*, PK):
  - zbiór atrybutów, który identyfikuje jednoznacznie wiersze tabeli;
  - w tabeli może być kilka kluczy kandydujących, spośród których wybieramy jeden klucz główny (np. w tabeli [Osoba] kluczami kandydującymi mogą być kolumny [NIP], [PESEL], [IdOsoby]).
- **Klucz obcy** (ang. *foreign key*, FK):
  - pozwala na łączenie danych z różnych tabel;
  - zbiór atrybutów w tabeli, który czerpie swoje wartości z tej samej dziedziny, co klucz główny tabeli powiązanej.

Integralność encji

- Każda tabela musi mieć klucz główny, który jednoznacznie identyfikuje wiersze tej tabeli.
- Klucz główny nie może zawierać wartości pustych (null).
- Zabronione są powtórzenia wierszy w ramach jednej tabeli.

Integralność referencyjna

- Klucz obcy może przyjmować jedną z dwóch wartości:
  - klucz główny z tabeli powiązanej;
  - wartość NULL (jeżeli nie koliduje to z innymi regułami integralności).
- Niedozwolone są wskazania poprzez klucz obcy na wiersz, który nie istnieje.
- Kaskadowa aktualizacja i usuwanie zależą od konkretnego zastosowania (np. jeśli usuwamy fakturę VAT z tabeli [Faktura], to usuwamy także wszystkie jej pozycje z powiązanej tabeli [Pozycja]; natomiast jeśli usuwamy grupę studentów z tabeli [Grupa], to raczej nie usuwamy z bazy wszystkich studentów z tej grupy, zapisanych w powiązanej tabeli [Student]).

Plan wykładu

1. Założenia relacyjnego modelu danych - przypomnienie.
2. Definicja podstawowych pojęć.
3. **Normalizacja bazy danych: 1NF, 2NF, 3NF.**

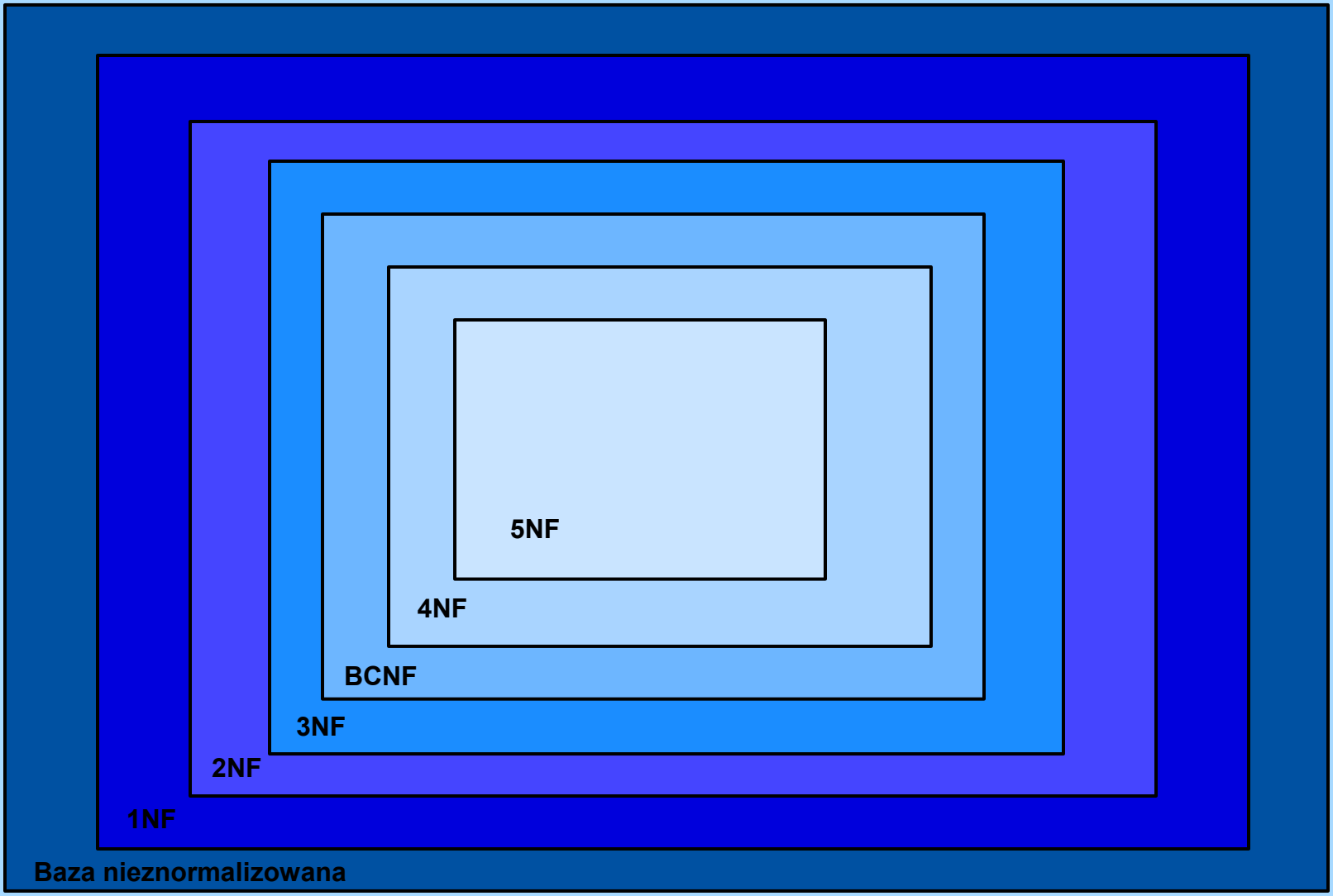
Normalizacja bazy danych

- Normalizacja bazy danych jest oparta na teorii modelu relacyjnego. Jest to technika projektowa, która polega na odpowiednim porządkowaniu schematu bazy tak, aby spełniał on poniższe wymagania;
  - dane przechowywane w bazie są zawsze w stanie spójności, a ich modyfikacja nie powoduje błędów;
  - wyszukiwanie i modyfikowanie danych jest wydajne;
  - struktura bazy jest przejrzysta i zrozumiała dla projektantów oraz programistów, co ułatwia administrowanie, utrzymanie i rozwijanie.
- Istnieją następujące postaci normalne: 1NF, 2NF, 3NF, BCNF, 4NF, 5NF. W praktyce zazwyczaj normalizujemy bazę do 2–3NF, wyższe postacie w większości przypadków nie są stosowane.
- Na ogólnym poziomie normalizacja sprowadza się do poniższych zasad:
  - każda tabela powinna mieć klucz, które jednoznacznie identyfikuje wiersze;
  - jedna tabela powinna przechowywać dane na temat jednej encji (klasy obiektów);
  - pola tabeli nie powinny przyjmować wartości NULL;
  - tabele nie powinny zawierać zduplikowanych kolumn, ani wartości danych.

Zalety i wady normalizacji

- Zalety normalizacji:
  - eliminuje anomalie przy modyfikacji danych, zapewniając ich spójność;
  - redukuje nadmiar danych (redundancję), przez co osiągana równowaga pomiędzy zajętością przestrzeni dyskowej i wydajnością;
  - przyśpiesza tworzenie indeksów, sortowanie i przeszukiwanie tabel na skutek zmiejszenia ich rozmiaru;
  - ułatwia zachowanie integralności referencyjnej;
  - prowadzi do struktury relacyjnej, która jest przejrzysta i zrozumiała;
  - zmniejsza liczbę kolumn w tabeli, dzięki czemu maleje rozmiar wierszy, więcej wierszy wchodzi na jedną stronę danych – a w konsekwencji przyśpieszane są operacje wejścia-wyjścia.
- Wady normalizacji:
  - zwiększa liczbę tabel w bazie, przez co wiele z nich musi być złączanych przy wyszukiwaniu lub modyfikacji danych; zmniejsza to wydajność bazy, gdyż złączenia są operacjami bardzo kosztownymi (rozwiązanie → denormalizacja).

Poziomy normalizacji bazy danych



Nieznormalizowany zbiór danych

Kursy_Oceny

KodKursu	NrPrac	NazwiskoPrac	ImiePrac	IndeksStud	NazwiskoStud	ImieStud	Ocena	TypOceny
INF407	234	Dudek	Damian	34698, 37798, 34888	Kowalski, Nowak, Bracki	Jan, Artur, Krzysztof	3.5; 4.5; 4.0	L1, L2, L1
INF507	234	Dudek	Damian	34698	Kowalski	Jan	4.0; 3.5	L1, L2
INF517	345	Choroś	Kazimierz	34668	Morawski	Andrzej	5.0	Egz

Problem: w kolumnach [IndeksStud], [NazwiskoStud], [ImieStud], [Ocena], [TypOceny] występuje wiele wartości, które sobie odpowiadają (np. student Jan Kowalski, numer indeksu 34698 za L1 z przedmiotu „INF407” uzyskał ocenę 3.5). Z tego powodu zapis w tabeli może nie być jednoznaczny, a wyszukiwanie danych jest bardzo utrudnione i nieefektywne.

Pierwsza postać normalna (1NF)

Tabela $R(U)$ jest w **pierwszej postaci normalnej** (ang. *first normal form*, 1NF), jeżeli dziedziny $DOM(A)$ wszystkich kolumn $A \in U$ są zbiorami wartości prostych, to znaczy nie są ani zbiorami, ani ciągami elementów należących do zbioru $\cup \{DOM(A): A \in U\}$.

Definicje przytoczone są na podstawie książki:
PANKOWSKI T., *Podstawy baz danych, Wydawnictwo Naukowe PWN, Warszawa, 1992,* str. 162 - 192.

Baza w 1NF – przykład

Kursy_Oceny (tabela nieznormalizowana)

KodKursu	NrPrac	NazwiskoPrac	ImiePrac	IndeksStud	NazwiskoStud	ImieStud	Ocena	TypOceny
INF407	234	Dudek	Damian	34698, 37798, 34888	Kowalski, Nowak, Bracki	Jan, Artur, Krzysztof	3.5; 4.5; 4.0	L1, L2, L1
INF507	234	Dudek	Damian	34698	Kowalski	Jan	4.0; 3.5	L1, L2
INF517	345	Choroś	Kazimierz	34668	Morawski	Andrzej	5.0	Egz

Kursy_Oceny (pierwsza postać normalna – 1NF)

KodKursu	NrPrac	NazwiskoPrac	ImiePrac	IndeksStud	NazwiskoStud	ImieStud	Ocena	TypOceny
INF407	234	Dudek	Damian	34698	Kowalski	Jan	3.5	L1
INF407	234	Dudek	Damian	37798	Nowak	Artur	4.5	L2
INF407	234	Dudek	Damian	34888	Bracki	Krzysztof	4.0	L1
INF507	234	Dudek	Damian	34698	Kowalski	Jan	4.0	L1
INF507	234	Dudek	Damian	34698	Kowalski	Jan	3.5	L2
INF517	345	Choroś	Kazimierz	34668	Morawski	Andrzej	5.0	Egz

Problemy z bazą w 1NF

Kursy_Oceny (pierwsza postać normalna – 1NF)

KodKursu	NrPrac	NazwiskoPrac	ImiePrac	IndeksStud	NazwiskoStud	ImieStud	Ocena	TypOceny
INF407	234	Dudek	Damian	34698	Kowalski	Jan	3.5	L1
INF407	234	Dudek	Damian	37798	Nowak	Artur	4.5	L2
INF407	234	Dudek	Damian	34888	Bracki	Krzysztof	4.0	L1
INF507	234	Dudek	Damian	34698	Kowalski	Jan	4.0	L1
INF507	234	Dudek	Damian	34698	Kowalski	Jan	3.5	L2
INF517	345	Choroś	Kazimierz	34668	Morawski	Andrzej	5.0	Egz

- **Anomalia usuwania:** jeśli usuniemy studenta o numerze indeksu 34668, utracimy informację o kursie "INF517" i jego wykładowcy.
- **Anomalia modyfikacji** (aktualizacji): jeśli zmienimy wykładowcę przedmiotu "INF407", to: (1) musimy aktualizować jednocześnie trzy kolumny [NrPrac], [NazwiskoPrac] i [ImiePrac]; (2) musimy modyfikować wiele wierszy, co może prowadzić do okresowego zablokowania tabeli albo do sprzeczności danych.
- **Anomalia wstawiania** (dołączania): nie możemy zapisać nowego studenta na dany przedmiot, dopóki nie będzie on miał przynajmniej jednej oceny z tego przedmiotu (chyba że wstawimy wartość NULL w kolumnach [Ocena] i [TypOceny]).

[Kursy_Oceny] w 1NF – zależności funkcyjne

Kursy_Oceny (pierwsza postać normalna – 1NF)

KodKursu	NrPrac	NazwiskoPrac	ImiePrac	IndeksStud	NazwiskoStud	ImieStud	Ocena	TypOceny
INF407	234	Dudek	Damian	34698	Kowalski	Jan	3.5	L1
INF407	234	Dudek	Damian	37798	Nowak	Artur	4.5	L2
INF407	234	Dudek	Damian	34888	Bracki	Krzysztof	4.0	L1
INF507	234	Dudek	Damian	34698	Kowalski	Jan	4.0	L1
INF507	234	Dudek	Damian	34698	Kowalski	Jan	3.5	L2
INF517	345	Choroś	Kazimierz	34668	Morawski	Andrzej	5.0	Egz

Klucz główny: K = {KodKursu, IndeksStud, TypOceny} (drugi kadydat: {NrPrac, IndeksStud, TypOceny})

Zależności funkcyjne:

K → NrPrac, K → NazwiskoPrac, K → ImiePrac, K → NazwiskoStud, K → ImieStud, K → Ocena,

KodKursu → NrPrac, KodKursu → NazwiskoPrac, KodKursu → ImiePrac,

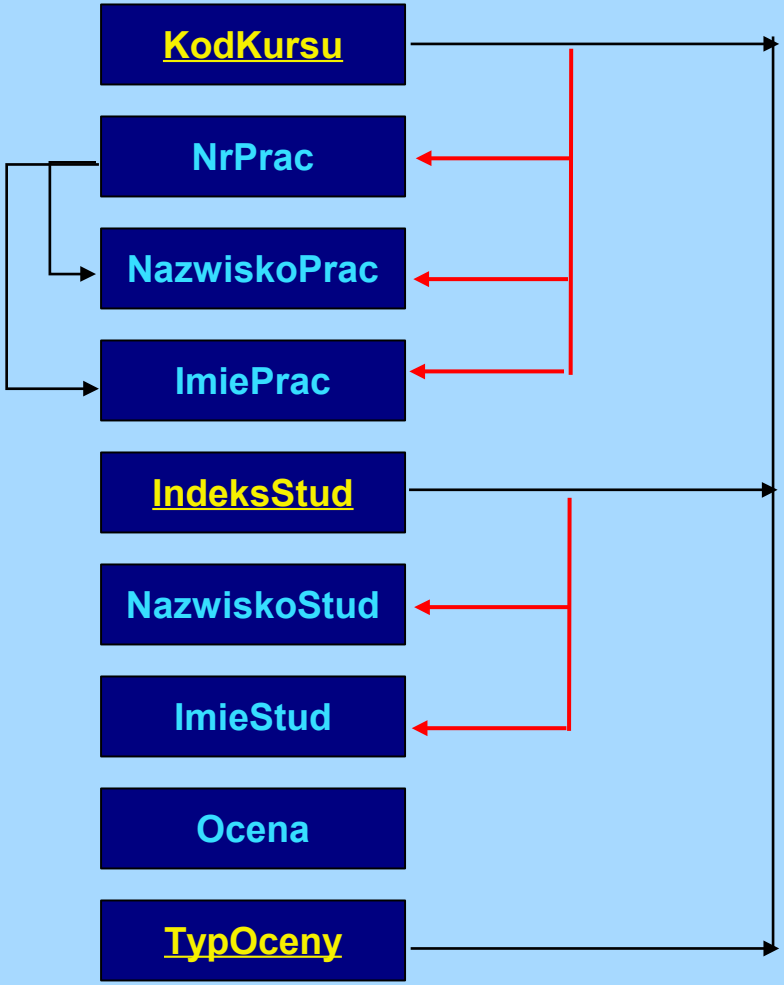
NrPrac → NazwiskoPrac, NrPrac → ImiePrac, IndeksStud → NazwiskoStud, IndeksStud → ImieStud

Źródła problemów:

kolumny: NrPrac, NazwiskoPrac i ImiePrac są zależne od części klucza - kolumny KodKursu;

kolumny NazwiskoStud i ImieStud są zależne od części klucza - kolumny IndeksStud.

[Kursy_Oceny] w 1NF – zależności funkcyjne



Cel: chcemy, aby każda kolumna niekluczowa był w pełni zależna od klucza głównego.

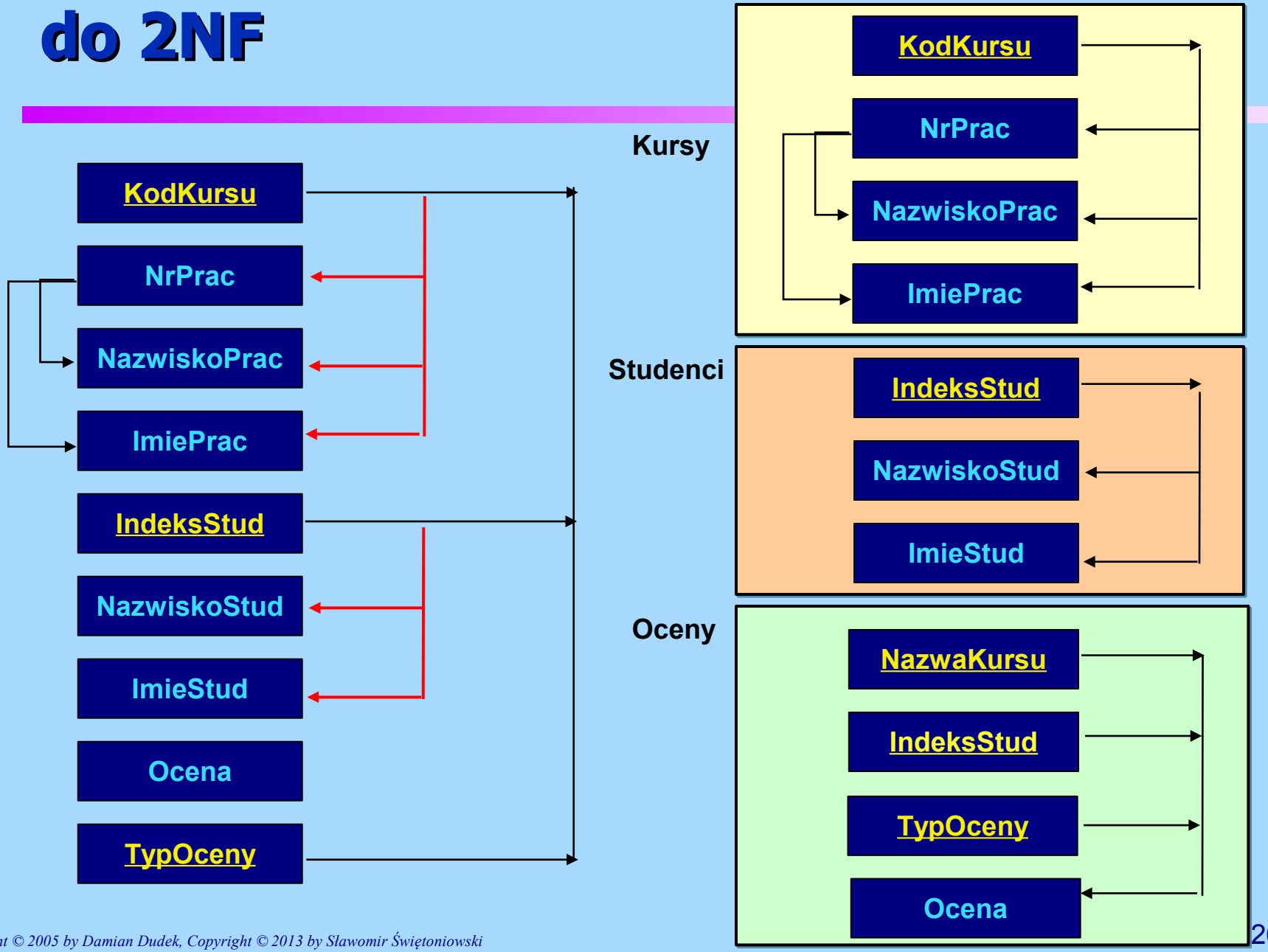
Druga postać normalna (2NF)

Tabela R jest w **drugiej postaci normalnej** (ang. *second normal form*, 2NF), jeżeli jest w 1NF i każda niekluczowa kolumna $A \in U$ jest w pełni zależna funkcyjnie od klucza głównego tej tabeli (i od każdego innego klucza kandydującego).

Normalizacja tabeli do 2NF:

- Przeprowadzamy rozkład (dekompozycję) tabeli względem niepełnych zależności funkcyjnych. W ten sposób dążymy, aby każda tabela odnosiła się do jednej klasy obiektów.
- Tabela jest już w 2NF, jeśli klucz główny jest jednoelementowy.
- Przeprowadzanie tabeli do 2NF nie jest procesem jednoznacznym, tzn. dla jednej tabeli może istnieć wiele równoważnych informacyjnie rozkładów w 2NF.

Normalizacja tabeli [Kursy_Oceny] do 2NF



Normalizacja tabeli [Kursy_Oceny] do 2NF

Kursy_Oceny (pierwsza postać normalna – 1NF)

KodKursu	NrPrac	NazwiskoPrac	ImiePrac	IndeksStud	NazwiskoStud	ImieStud	Ocena	TypOceny
INF407	234	Dudek	Damian	34698	Kowalski	Jan	3.5	L1
INF407	234	Dudek	Damian	37798	Nowak	Artur	4.5	L2
INF407	234	Dudek	Damian	34888	Bracki	Krzysztof	4.0	L1
INF507	234	Dudek	Damian	34698	Kowalski	Jan	4.0	L1
INF507	234	Dudek	Damian	34698	Kowalski	Jan	3.5	L2
INF517	345	Choroś	Kazimierz	34668	Morawski	Andrzej	5.0	Egz

Kursy (2 NF)

KodKursu	NrPrac	NazwiskoPrac	ImiePrac
INF407	234	Dudek	Damian
INF507	234	Dudek	Damian
INF517	345	Choroś	Kazimierz

Studenci (2NF i od razu 3NF)

IndeksStud	NazwiskoStud	ImieStud
34698	Kowalski	Jan
37798	Nowak	Artur
34888	Bracki	Krzysztof
34668	Morawski	Andrzej

Oceny (2NF i od razu 3NF)

KodKursu	IndeksStud	TypOceny	Ocena
INF407	34698	L1	3.5
INF407	37798	L2	4.5
INF407	34888	L1	4.0
INF507	34698	L1	4.0
INF507	34698	L2	3.5
INF517	34668	Egz	5.0

Problemy z bazą w 2NF

Kursy (2 NF)

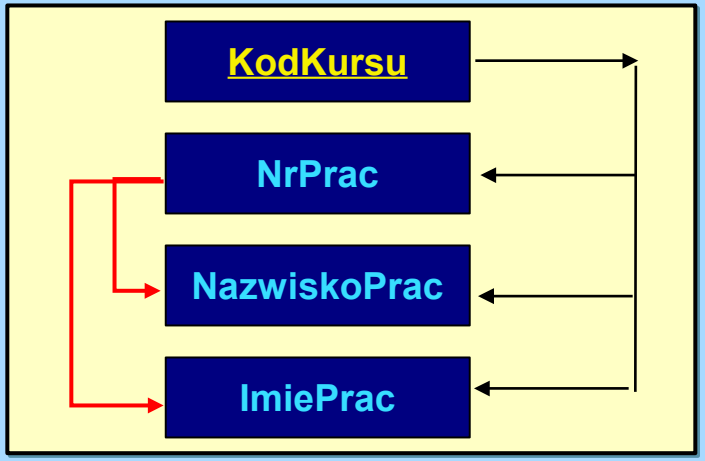
KodKursu	NrPrac	NazwiskoPrac	ImiePrac
INF407	234	Dudek	Damian
INF507	234	Dudek	Damian
INF517	345	Choroś	Kazimierz

- **Anomalia usuwania:** jeśli usuniemy przedmiot "INF517", utracimy informację o wykładowcy; podobnie, usunięcie wierszy dotyczących wykładowcy o numerze "234" powoduje utratę informacji o kursach "INF407" i "INF507".
- **Anomalia modyfikacji** (aktualizacji): zmiana numeru pracownika (pola "NrPrac") lub jego danych osobowych (pola "NazwiskoPrac" i "ImiePrac") wymaga dokonania aktualizacji w wielu wierszach; może to prowadzić do okresowego zablokowania tabeli albo do sprzeczności danych.
- **Anomalia wstawiania** (dołączania): nie można zapisać informacji o zatrudnionym pracowniku, jeśli nie prowadzi on przynajmniej jednego kursu.

Kursy 2NF – zależności funkcyjne

Kursy (2 NF)

KodKursu	NrPrac	NazwiskoPrac	ImiePrac
INF407	234	Dudek	Damian
INF507	234	Dudek	Damian
INF517	345	Choroś	Kazimierz



Klucz główny: K = {KodKursu}

Zależności funkcyjne:

KodKursu → NrPrac, KodKursu → NazwiskoPrac, KodKursu → ImiePrac,

NrPrac → NazwiskoPrac, NrPrac → ImiePrac

Źródło problemów:

kolumny: NazwiskoPrac i ImiePrac są zależne funkcyjnie od atrybutu niekluczowego NrPrac (zależność tranzytywna - inaczej przechodnia).

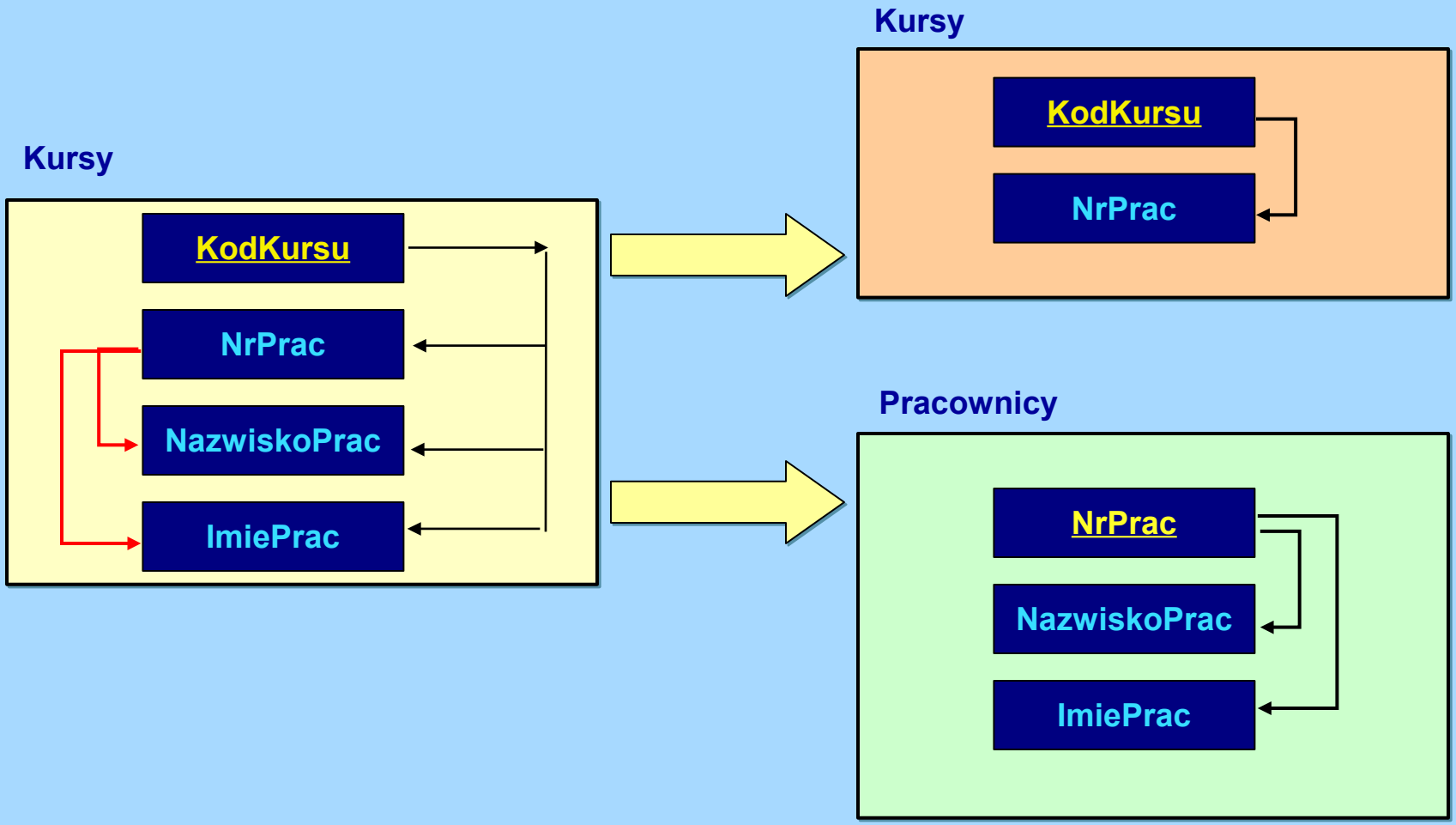
Trzecia postać normalna (3NF)

Tabela *R* jest w **trzeciej postaci normalnej** (ang. *third normal form*, 3NF), jeżeli jest w 2NF i żaden zbiór kolumn niekluczowych nie jest tranzytywnie (przechodnio) zależny od klucza głównego tabeli *R* (ani żadnego innego klucza kandydującego).

Normalizacja tabeli do 3NF:

- Przeprowadzamy rozkład (dekompozycję) relacji względem zależności tranzytywnych.
- Przeprowadzanie relacji z 2NF do 3NF nie jest procesem jednoznacznym, tzn. dla jednej relacji może istnieć wiele równoważnych informacyjnie rozkładów w 3NF.

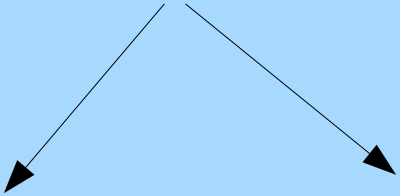
Normalizacja relacji [Kursy] do 3NF — rozkład (dekompozycja)



Normalizacja relacji [Kursy] do 3NF – rozkład (dekompozycja)

Kursy (2 NF)

KodKursu	NrPrac	NazwiskoPrac	ImiePrac
INF407	234	Dudek	Damian
INF507	234	Dudek	Damian
INF517	345	Choroś	Kazimierz



Kursy (3 NF)

KodKursu	NrPrac
INF407	234
INF507	234
INF517	345

Kursy (3 NF)

NrPrac	NazwiskoPrac	ImiePrac
234	Dudek	Damian
345	Choroś	Kazimierz

Literatura

1. BEYNON-DAVIES P., *Systemy baz danych – nowe wydanie*, WNT, Warszawa 2003.
2. MICROSOFT, *Books On-Line* – dokumentacja systemu *MS SQL Server*, Microsoft Corp. 1988 – 2000.
3. PANKOWSKI T., *Podstawy baz danych*, Wydawnictwo Naukowe PWN, Warszawa, 1992.
4. RANKINS R., JENSEN P., BERTUCCI P., *Microsoft SQL Server 2000. Księga eksperta.*, HELION, Gliwice 2003 (książka dostępna w bibliotece WSIZ „Copernicus").

Bazy danych

Wykład 2_2

Dziękuję za uwagę !

Bazy danych

Wykład 3

Temat: Metodologia tworzenia baz danych

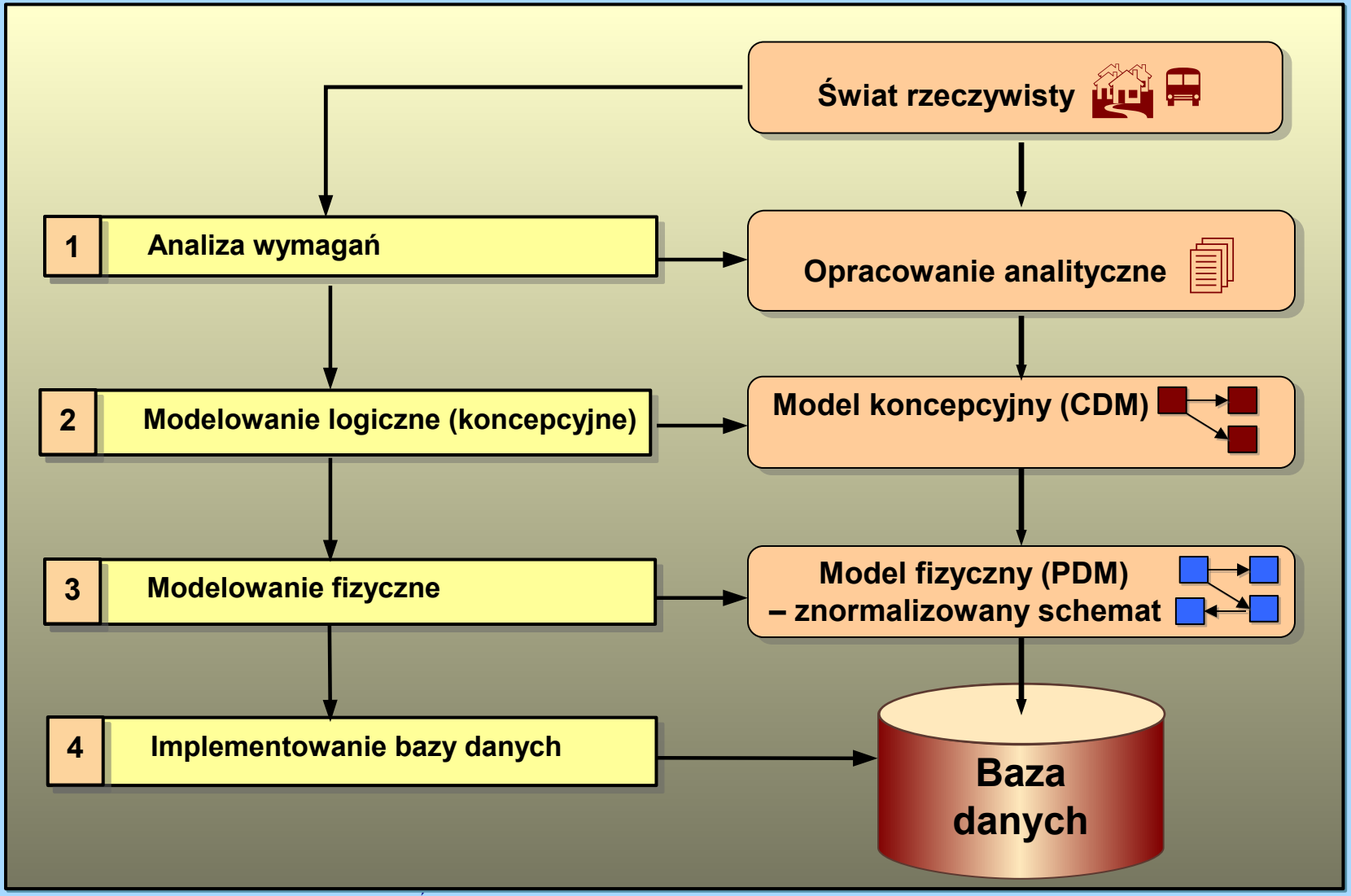
Sławomir Świętoniowski

slawomir-swietoniowski@wp.pl

Plan wykładu

1. **Etapy tworzenia bazy danych.**
2. Prosta metodologia tworzenia relacyjnych baz danych.
3. Przykład analizy dziedziny „Wyższa uczelnia techniczna”.

Etapy tworzenia bazy danych



1. Analiza wymagań

- Rozpoznanie systemu i zachodzących w nim procesów, które mają być modelowane.
- Określenie celu, zakresu i funkcji systemu.
- Zidentyfikowanie głównych grup użytkowników i ich sposobu korzystania z systemu.
- Analiza korzyści z nowego systemu, w porównaniu z obecnym rozwiązaniem.

Powstaje: **opracowanie analityczne** – dokument zawierający opis tekstowy oraz schematy (np. diagramy modelu procesów biznesowych BPM), których dobór zależy od przyjętej metodologii.

2. Modelowanie logiczne

- Definiowanie obiektów, ich atrybutów oraz relacji między obiektami.
- Modelowanie zachowania systemu (bez implementacji).
- Na tym poziomie celowo pomija się uwarunkowania implementacyjne (np. system bazodanowy, język programowania, platformę systemową), aby móc się skoncentrować na właściwym przygotowaniu koncepcyjnego modelu systemu.

Powstaje: **logiczny model danych** (ang. *conceptual data model*, CDM) – opisujący strukturę bazy (obiekty, powiązania między nimi) w sposób niezależny od docelowego systemu DBMS.

3. Modelowanie fizyczne

- Tworzenie schematu bazy danych dla konkretnego systemu bazodanowego.
- Normalizacja schematu bazy – standardowo do 2–3 NF.
- Analiza integralności danych – opis więzów deklaratywnych i proceduralnych.
- Analiza ilościowa, użycia i transakcji – optymalizacja schematu bazy.
- Analiza zabezpieczeń i kontroli – projekt systemu bezpieczeństwa.
- Projekt struktur przechowywania (np. rozkładu plików bazy na dyskach).
- Na tym poziomie rozstrzyga się wszystkie istotne kwestie techniczne (m.in. platforma systemowa, system bazodanowy, architektura aplikacji klient-serwer, język programowania dla poszczególnych warstw systemu).

Powstaje: **fizyczny model danych** (ang. *physical data model*, PDM) – stanowiący projekt bazy dostosowany do konkretnego systemu DBMS (np. Oracle) w celu uzyskania jak największej wydajności.

4. Implementacja bazy danych

Implementacja poniższych elementów (ręczna lub automatyczna):

- schemat bazy danych: tabele i relacje;
- więzy integralności:
  - wewnętrzne – deklaratywne (ograniczenia PK, FK, CHECK, UNIQUE, DEFAULT, reguły, wartości domyślne);
  - dodatkowe – proceduralne (np. wyzwalacze – ang. *triggers*);
- obiekty kodu SQL, przechowywane na serwerze:
  - procedury przechowywane dla transakcji i operacji CRUD (INSERT, SELECT, UPDATE, DELETE);
  - widoki;
- indeksy – na podstawie struktury bazy danych i sposobu jej wykorzystania;
- system bezpieczeństwa – konta, role, użytkownicy i ich uprawnienia.

Powstaje: baza danych (ang. *database*, DB) – realizacja modelu fizycznego w konkretnym środowisku technicznym, gotowa do wdrożenia, optymalizacji i strojenia (ang. *tuning*).

Plan wykładu

1. Etapy tworzenia bazy danych.
2. **Prosta metodologia tworzenia relacyjnych baz danych.**
3. Przykład analizy dziedziny „Wyższa uczelnia techniczna”.

Prosta metodologia tworzenia relacyjnych baz danych

1. **Analiza dziedziny**: rozpoznanie systemu i zachodzących w nim procesów, które mają być modelowane.
2. **Projekt i implementacja schematu bazy danych (2–3NF)**: tabele i relacje.
3. **Projekt i implementacja więzów integralności**:
   - **wewnętrznych – deklaratywnych** (ograniczenia PK, FK, CHECK, UNIQUE, DEFAULT, reguły, wartości domyślne);
   - **dodatkowych – proceduralnych** (wyzwalacze – ang. *triggers*).
4. **Tworzenie widoków i procedur przechowywanych dla operacji CRUD** – instrukcje INSERT, SELECT, UPDATE, DELETE; transakcje.
5. **Projekt i implementacja indeksów** – na podstawie struktury bazy danych i sposobu jej wykorzystania.
6. **Projekt i implementacja systemu bezpieczeństwa** – konta, role, użytkownicy i ich uprawnienia.
7. **Wdrożenie, konfiguracja zadań automatycznych i utrzymania bazy**.

Prosta metodologia – zastosowanie

- Przedstawiona metodologia może być wykorzystana do:
  - szybkiego tworzenia baz danych o małej i średniej złożoności;
  - tworzenia warstwy bazodanowej w aplikacji klient-serwer (zwłaszcza dwuwarstwowych – baza danych + interfejs wraz z logiką biznesową);
  - tworzenia niewielkich aplikacji, w których nie jest wymagana rozbudowana dokumentacja;
  - budowy systemów w warunkach braku zaawansowanych narzędzi do projektowania (np. opartych na języku UML).
- Metodologia ta nie jest zalecana, jeśli:
  - tworzony jest system o dużej złożoności (np. aplikacja wielowarstwowa klient – serwer z dużą liczbą współpracujących ze sobą obiektów w poszczególnych warstwach lub z bardzo zaawansowanymi algorytmami);
  - dysponujemy dobrymi narzędziami do modelowania (np. PowerDesigner);
  - integrujemy lub rozbudowujemy wiele istniejących systemów baz danych (szczególnie heterogenicznych – z różnymi systemami DBMS).

Plan wykładu

1. Etapy tworzenia bazy danych.
2. Prosta metodologia tworzenia relacyjnych baz danych.
3. **Przykład analizy dziedziny „Wyższa uczelnia techniczna”.**

Przykład dziedziny modelowania Wyższa uczelnia techniczna

Skrócony opis dziedziny.

- Uczelnia prowadzi studia inżynierskie z zakresu informatyki. Na program nauczania składają się przedmioty, przy czym każdy może mieć kilka form dydaktycznych (np. wykład, laboratorium).
- Do każdego przedmiotu przyporządkowywany jest co najwyżej jeden pracownik naukowo-dydaktyczny, który odpowiada za program nauczania i prowadzi nadzór merytoryczny nad jego wykonaniem.
- Uczelnia zatrudnia pracowników:
  - administracyjnych (pracują w dziekanacie, kwesturze itp.);
  - technicznych (np. konserwatorzy, administratorzy sieci i sprzętu komputerowego);
  - naukowo-dydaktycznych (prowadzą zajęcia ze studentami, piszą publikacje itp.).
- Każdy student ma przyporządkowywany niepowtarzalny numer indeksu z danego zakresu liczbowego.
- Student wpłaca uczelni miesięczne opłaty, uprawniające do studiowania.

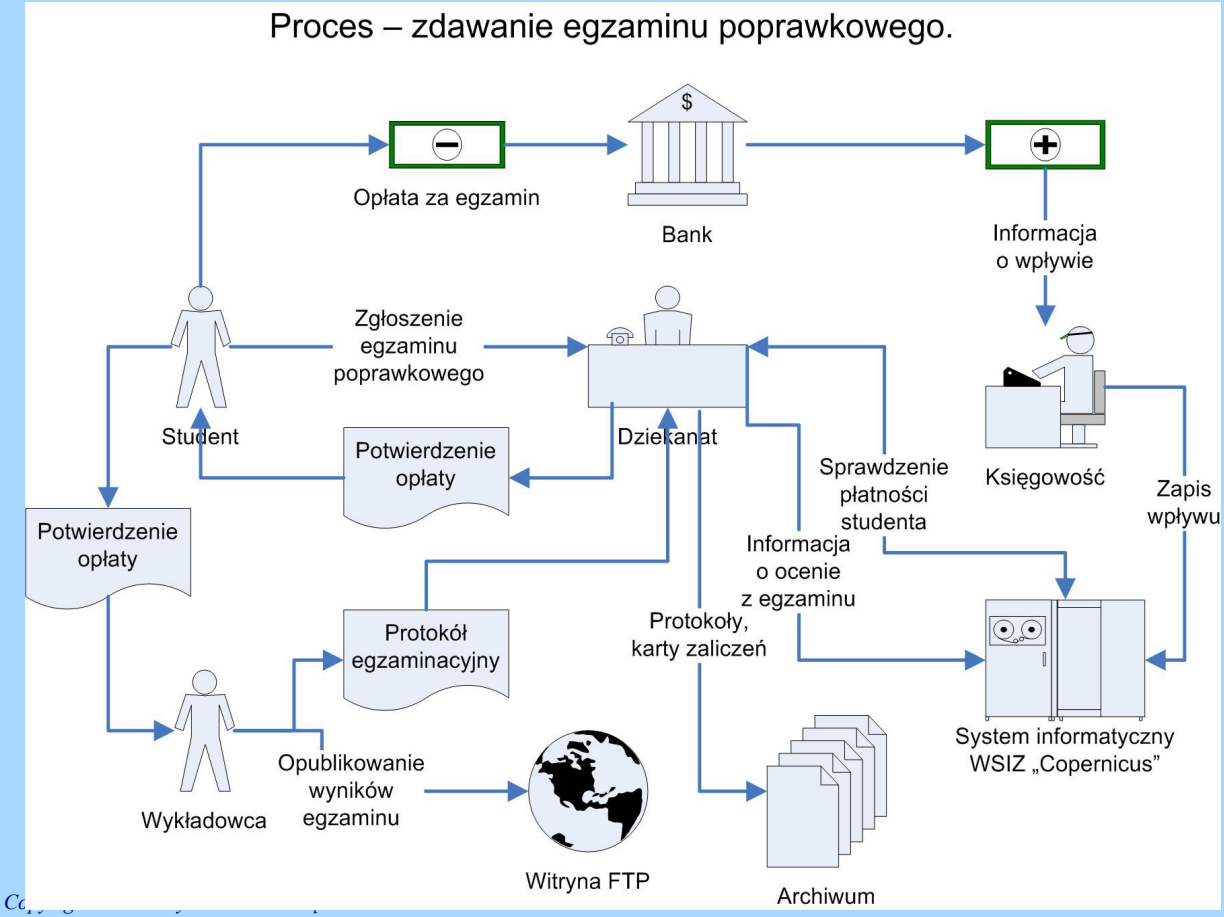
Przykład dziedziny modelowania Wyższa uczelnia techniczna

Skrócony opis dziedziny – ciąg dalszy.

- Studenci jednego roku są dzieleni na grupy, których nazwy są unikalne w skali jednego semestru. Jeden student może jednocześnie należeć do jednej grupy; grupa może zawierać od 15 do 30 studentów.
- Każda grupa studentów może mieć co najwyżej jednego studenta, który ją reprezentuje, nazywanego „starostą”.
- Przedmioty są realizowane przez określonych pracowników naukowo-dydaktycznych, w danym roku akademickim i semestrze, w danej grupie studentów lub indywidualnie (tylko przedmiot „Praca dyplomowa”).
- Student jest zapisywany na zajęcia z określonego przedmiotu – poprzez grupę (przedmioty grupowe) albo indywidualnie („Praca dyplomowa” lub inny przedmiot, realizowany w ramach indywidualnego toku studiów).
- Z każdego przedmiotu student w danym semestrze uzyskuje jedno lub więcej zaliczeń (1 – 3 terminy dla egzaminów i 1 – 2 terminy dla innych zaliczeń).

Schemat działania systemu

Bardzo cennym uzupełnieniem słownego opisu działania systemu są diagramy (np. BPM, WFD), za pomocą których można czytelnie przedstawić np. obiekty, relacje i procesy zachodzące w danej organizacji (poniżej przykład WFD).



Wyodrębnienie klas obiektów

Analizując słowny opis dziedziny, szukamy pojęć, obiektów, znaczeń, które dają się wyodrębnić z modelowanej rzeczywistości.

- **Przedmiot** (Kod, Nazwa, Semestr, ECTS);
- **Pracownik** (NrPracownika, Imie, Nazwisko, NIP, PESEL, Ulica, NumerDomu, NumerMieszkania, KodPocztowy, Miejscowosc, TelefonStac, TelefonKom, Email, Tytul, Stanowisko, DataZatrudnienia, OkresZatrudnienia, TypZatrudnienia, Wynagrodzenie);
- **Student** (NrIndeksu, Nazwisko, Imie, NIP, PESEL, Ulica, NumerDomu, NumerMieszkania, KodPocztowy, Miejscowosc, TelefonStac, TelefonKom, Email, DataZapisania, DataWypisania);
- **Grupa** (NrGrupy, RokAkademicki);
- **Zajęcia** (FormaZajec, LiczbaGodzin, WymaganiaWstepne, Opis);
- **Zaliczenie** (DataZaliczenia, Ocena, RodzajZaliczenia, Termin);
- **Opłata** (DataWpłaty, Wpłacający, Miesiąc, Rok, Kwota).

Powiązania między obiektami

- **Pracownik – Przedmiot –** dany pracownik prowadzi dany przedmiot w danej grupie, w określonym semestrze i roku akademickim.
- **Student – Opłata** – pozwala na ewidencjonowanie opłat studenta.
- **Student – Grupa** – wyraża przynależność studenta do grupy.
- **Student – Pracownik – Przedmiot** – student zapisuje się na dany przedmiot do określonego prowadzącego;
- **Student – Przedmiot – Zaliczenie –** po zapisaniu się na dany przedmiot, student uzyskuje z niego zaliczenie (może być kilka prób);

Funkcje systemu – opis ogólny

- **Pracownik, Grupa, Przedmiot, Student, Opłata**
  – dodawanie, edycja, usuwanie.
- **Przypisanie studenta do grupy**.
- **Przypisanie pracownika do prowadzenia przedmiotu** w określonej grupie studentów.
- **Zapisanie studenta na przedmiot** do określonego prowadzącego (dotyczy przedmiotu „Praca dyplomowa”, który jest realizowany indywidualnie, a nie przez całą grupę).
- **Wpis zaliczenia studenta** z danego przedmiotu.

Wybrane funkcje systemu – specyfikacja

- **Przypisanie studenta do grupy:**
  1. Jeśli nie jest zapisany, zapisanie studenta do bazy.
  2. Jeśli nie istnieje grupa, dodanie nowej grupy.
  3. Zapisanie studenta do grupy.
- **Przypisanie pracownika do prowadzenia przedmiotu w określonej grupie studentów**:
  1. Jeśli nie istnieje pracownik, zapisanie pracownika do bazy.
  2. Jeśli nie istnieje przedmiot, dodanie nowego przedmiotu.
  3. Jeśli nie istnieje grupa, dodanie nowej grupy.
  4. Przypisanie pracownika do danego przedmiotu.
- **Zapisanie studenta na przedmiot do określonego prowadzącego:**
  1. Jeśli żaden pracownik nie został przypisany do wybranego przedmiotu
     → Proces: przypisanie pracownika do przedmiotu.
  2. Jeśli nie ma studenta w bazie, zapisanie studenta do bazy.
  3. Jeśli student zaliczył wymagane przedmioty z poprzedniego semestru i uiścił opłatę za nowy semestr, to zapisz studenta na przedmiot.
  4. W przeciwnym razie, odmowa zapisu.

Grupy użytkowników

W systemie wyróżniamy 5 grup użytkowników.

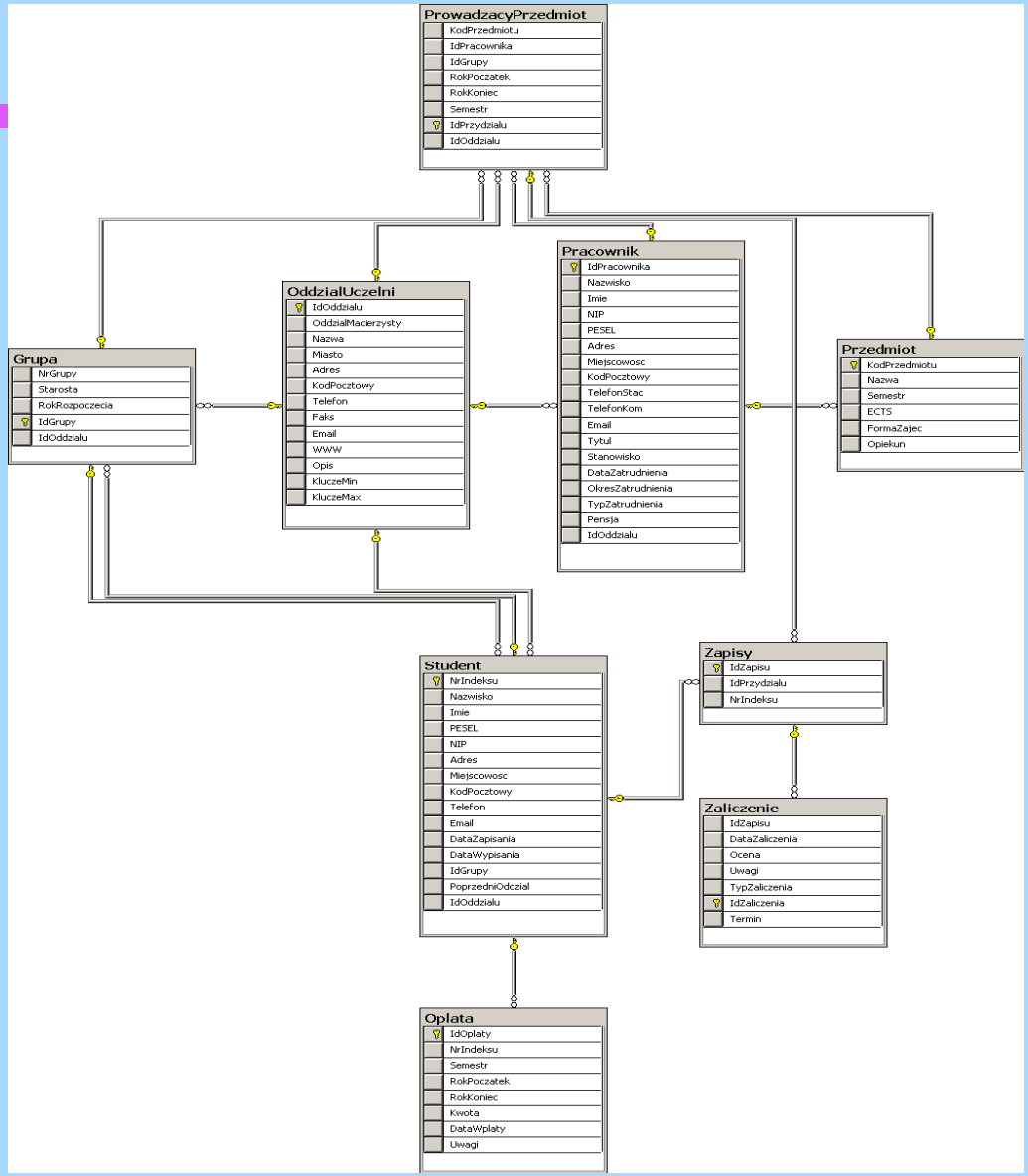
- **Administratorzy** – nieograniczone prawa operowania na danych i modyfikacji ustawień systemu. Do tej grupy należą administratorzy systemu informatycznego oraz kierownictwo uczelni.
- **Pracownicy naukowo-dydaktyczni** – mogą: przeglądać i częściowo edytować dane prowadzonych przez nich przedmiotów, zmieniać swoje dane osobowe i zarządzać zaliczeniami do momentu ich zatwierdzenia, odczytywać płatności za zaliczenia poprawkowe (tylko dla swoich studentów). Pozostałe obiekty – tylko do odczytu.
- **Pracownicy administracyjni** – mogą dodawać, odczytywać i modyfikować dane dotyczące pracowników, studentów, zajęć i opłat studentów. Mogą w pełni zarządzać grupami. Pozostałe obiekty – tylko do odczytu;
- **Studenci** – mogą przeglądać dane przedmiotów, grup i zajęć. Mogą także odczytać: ograniczone dane pracowników (tytuł, imię, nazwisko, stanowisko), swoje dane osobowe (+ prawo edycji telefonu i adresu e-mail), swoje zaliczenia i opłaty.
- **Goście** – na przykład nie zalogowani użytkownicy internetowi. Mogą wyłącznie czytać dane przedmiotów i ograniczone dane pracowników (tytuł, imię, nazwisko, stanowisko).

Uprawnienia na poziomie obiektów

Poniżej przedstawiona jest specyfikacja uprawnień grup użytkowników na poziomie wyodrębnionych obiektów, w formie tzw. **macierzy CRUD** (Create, Read, Update, Delete). Gwiazdka (*) wskazuje, że określone uprawnienia są ograniczone, zgodnie z podanym wcześniej opisem słownym.

	Administratorzy	PracownicyND	PracownicyADM	Studenci	Goscie
Przedmiot	CRUD	–RU*–	–R—	–R—	–R—
Pracownik	CRUD	–RU*–	CRU–	–R*—	–R*—
Student	CRUD	–R—	CRU–	–RU*–	———
Grupa	CRUD	–R—	CRUD	–R—	———
Zajecia	CRUD	–R—	CRU–	–R—	———
Zaliczenie	CRUD	CRUD*	–R—	–R*—	———
Oplata	CRUD	–R*—	CRU–	–R*—	———

Schemat bazy



Zrzut ekranowy przedstawia bazę danych „Wyższa uczelnia techniczna”, wersja 1.1.

Literatura

1. BEYNON-DAVIES P., *Systemy baz danych – nowe wydanie*, WNT, Warszawa 2003.
2. MICROSOFT, *Books On-Line* – dokumentacja systemu *MS SQL Server*, Microsoft Corp. 1988 – 2000.
3. PANKOWSKI T., *Podstawy baz danych*, Wydawnictwo Naukowe PWN, Warszawa, 1992.
4. RANKINS R., JENSEN P., BERTUCCI P., *Microsoft SQL Server 2000. Księga eksperta.*, HELION, Gliwice 2003 (książka dostępna w bibliotece WSIZ „Copernicus").

Bazy danych

Wykład 5

Dziękuję za uwagę !

Damian Dudek

Bazy danych

Wykład 4_1

Temat: Język SQL i T-SQL – wprowadzenie

Sławomir Świętoniowski

slawomir-swietoniowski@wp.pl

Plan wykładu

1. **Standard języka SQL i jego historia.**
2. Podzbiory funkcjonalne języka SQL.
3. Język Transact-SQL w MS SQL Server 2008 R2.

Co to jest język SQL?

SQL – Structured Query Language

(wymowa: `*es-`kju-`el* lub `*si:kuel*):

- Najbardziej rozpowszechniony, ustandaryzowany język baz danych.
- Pozwala na:
  - **definiowanie struktury bazy danych** (pozbiór DDL – Data Definition Language), np. CREATE TABLE, ALTER VIEW, DROP PROCEDURE;
  - **operowanie na danych**: dodawanie, udostępnianie, modyfikowanie i usuwanie (pozdbiór DML – Data Manipulation Language), np. INSERT, SELECT, UPDATE, DELETE.
  - **zarządzanie dostępem do danych** (podzbiór DCL – Data Control Language), np. GRANT, DENY, REVOKE;
  - **definiowanie więzów integralności** (podzbiór DIL – Data Integrity Language), np. CREATE RULE.

SQL – historia (1)

1970–1980	Edgar F. Codd (IBM Research Laboratory, CA, USA) opublikował teorię relacyjnego modelu danych, który stał się podstawą relacyjnych baz danych. Spowodowalo do liczne badania naukowe i rozwój wielu produktów DBMS.
1974	Pojawił się język relacyjny *Structured English Query Language* – SEQUEL (twórca: Donald Chamberline, IBM). Prototypowa implementacja nosiła nazwę SEQUEL-XRM.
1976–1977	Została opracowana poprawiona wersja języka, nazwana SEQUEL/2 (SQL). Na jej podstawie IBM opracował ulepszony język *System R*, wdrożony u wielu klientów firmy IBM.

SQL – historia (2)

1980–1983	IBM zaprezentował nowe wersje produktów SQL: SQL/DS dla środowiska VSE (1981 r.) i środowiska VM (1982 r.) oraz DB2 dla środowiska MVS (1983 r.).
1977–1990	Po sukcesie i akceptacji produktów IBM, inne firmy zaczęły rozwijać własne produkty bazodanowe, oparte na modelu relacyjnym i języku SQL: ORACLE (1977 r.), DG/SQL (Data General Corporation, 1984 r.), SYBASE (Sybase Inc., 1986 r.), INGRES (Relational Technology Inc., 1981-1985 r.), IDM (Britton-Lee Inc, 1982-1985 r.).
1982	Ustanowienie pierwszego standardu języka SQL przez ANSI (American National Standards Institute).

SQL – historia (3)

1987	Akceptacja standardu ANSI przez ISO (International Organisation for Standarization) jako standardu międzynarodowego: „SQL-86” (lub SQL1).
1989	Zatwierdzenie rozszerzonego standardu SQL IEF (Integrity Enhancement Feature): „SQL-89” (SQL2).
1992	Rozszerzony standard międzynarodowy: „ISO/IEC 9075”, zwany powszechnie jako „SQL-92” (lub SQL3). Ta wersja jest najczęściej implementowana w produktach komercyjnych.

SQL – historia (4)

1999/2003	Standard ANSI SQL:2003 (wersja rozwojowa SQL:1999). Dodano nowe typy danych, w tym obsługę XML, pojawiło się nowe polecenie MERGE i jeszcze kilka innych elementów (SQL4/SQL5).
2008	Standard ANSI SQL:2008. Dalsza aktualizacja języka SQL (SQL6).
2011	Standard ANSI SQL:2011. Najaktualniejsza wersja standardu (SQL7).

Standard SQL – korzyści i wady

Zalety:

- redukcja kosztów szkoleń,
- przenośność aplikacji,
- długowieczność aplikacji,
- ułatwienie komunikacji międzysystemowej,
- ułatwienie użytkownikowi wyboru systemu DBMS.

Wady:

- ograniczenie kreatywności,
- język SQL jest niedoskonały jako język relacyjny,
- standard SQL ma wiele braków, m.in. nie określa wielu elementów potrzebnych w praktyce (np. dane typu logicznego, nazewnictwo obiektów).

Plan wykładu

1. Standard języka SQL i jego historia.
2. **Podzbiory funkcjonalne języka SQL.**
3. Język Transact-SQL w MS SQL Server 2008 R2.

Podzbiory języka SQL

- **Język definiowania danych** (ang. *data definition language*, DDL).
- **Język operowania danymi** (ang. *data manipulation language*, DML).
- **Język kontroli danych** (ang. *data control language*, DCL).
- **Język integralności danych** (ang. *data integrity language*, DIL).

Język definiowania danych, DDL

Służy do tworzenia i modyfikowania struktury bazy danych: tabel i ograniczeń deklaratywnych, relacji, innych obiektów bazy danych (np. widoków).

Język operowania danymi, DML

Służy do implementacji tzw. operacji CRUD: wstawiania, odczytywania, modyfikowania i usuwania danych.

Język kontroli danych, DCL

Wyodrębniony podzbiór języka SQL, służący do definiowania systemu bezpieczeństwa: użytkowników, ról oraz ich uprawnień na serwerze i bazach danych.

Język integralności danych, DIL

Wykorzystywany do definiowania ograniczeń deklaratywnych i reguł, które zapewniają integralność danych. Ponieważ podzbiór ten częściowo nachodzi na DDL, rzadko jest wyodrębniany.

Plan wykładu

1. Standard języka SQL i jego historia.
2. Podzbiory funkcjonalne języka SQL.
3. **Język Transact-SQL w MS SQL Server 2008 R2.**

Język SQL w MS SQL Server 2008 R2

Transact-SQL (T-SQL) **–** transakcyjny język SQL, wbudowany w systemie bazodanowym Microsoft SQL Server 2008 R2:

- jest zgodny ze standardem ANSI-92 na poziomie podstawowym (np. jeśli chodzi o polecenia INSERT, SELECT, UPDATE, DELETE);
- udostępnia pewne funkcje ANSI-99, a także jego rozszerzenia np.:
  - programy przechowywane na serwerze (procedury przechowywane, wyzwalacze, funkcje użytkownika);
  - dodatkowe typy danych (także definiowane przez użytkownika) np. table;
  - role w bazach danych;
  - wartości domyślne pól tabel;
  - duży zbiór funkcji wbudowanych (np. GETDATE( )).
- jest podstawą implementacji ogromnie licznych, wbudowanych procedur systemowych (np. *sp_helpdb*), które umożliwiają wykonywanie wielu zaawansowanych czynności administracyjnych i programistycznych z poziomu konsoli SQL (bez narzędzi wizualnych, kreatorów itp.).

Transact-SQL – przykład

Język T-SQL pozwala na pisanie zaawansowanych programów w obrębie warstwy bazodanowej, dzięki złożonym konstrukcjom kodu strukturalnego (przykład poniżej). Brakuje z nim natomiast elementów typowo obiektowych.

```
IF EXISTS -- Sprawdzenie, czy ten sam student nie jest zapisywany drugi raz.
        (SELECT *
         FROM Student
         WHERE PESEL = @PESEL)
BEGIN
        RAISERROR('W bazie jest już student o tym numerze PESEL!', 16, 1)
        RETURN
END

DECLARE @IndeksMin int
DECLARE @IndeksMax int
-- Pobranie dopuszczalnego zakresu numeru indeksu.
SELECT @IndeksMin = KluczeMin, @IndeksMax = KluczeMax
FROM OddzialUczelni
WHERE OddzialMacierzysty = 1

SET ANSI_NULLS ON
IF (@NrIndeksu IS NOT NULL) -- Jeśli numer indeksu został podany jako parametr.
BEGIN
        -- Sprawdzenie, czy numer mieści się w zakresie ustalonym dla tego oddziału.
        IF (@NrIndeksu NOT BETWEEN @IndeksMin AND @IndeksMax)
        BEGIN
                RAISERROR('Podany numer indeksu jest bledny!', 16, 1)
                RETURN
        END
        ELSE
                INSERT Student (NrIndeksu, Nazwisko, Imie, PESEL, NIP, Adres, Miejscowosc,
                KodPocztowy, Telefon, Email, DataZapisania, DataWypisania, IdGrupy,
```

Literatura

1. BEYNON-DAVIES P., *Systemy baz danych – nowe wydanie*, WNT, Warszawa 2003.
2. DATE C. J., DARWEN H., *SQL. Omówienie standardu języka*, WNT, Warszawa 2000 (książka dostępna w bibliotece WSIZ "Copernicus").
3. MICROSOFT, *Books On-Line* – dokumentacja systemu *MS SQL Server*, Microsoft Corp. 1988 – 2000.
4. RANKINS R., JENSEN P., BERTUCCI P., *Microsoft SQL Server 2000. Księga eksperta.*, HELION, Gliwice 2003 (książka dostępna w bibliotece WSIZ „Copernicus").
5. Strona MSDN: http://msdn.microsoft.com.
6. WAYMIRE R., SAWTELL R., *MS SQL Server 2000 dla każdego*, HELION, Gliwice 2002 (książka dostępna w bibliotece WSIZ "Copernicus").

Bazy danych

Wykład 4_1

Dziękuję za uwagę !

Bazy danych

Wykład 4_2

Temat: Tworzenie schematu bazy w T-SQL

Sławomir Świętoniowski

slawomir-swietoniowski@wp.pl

Plan wykładu

1. **Tworzenie i modyfikowanie bazy danych.**
2. Typy danych w MS SQL Server 2008 R2.
3. Tworzenie, modyfikowanie i usuwanie tabel.

Prosta metodologia tworzenia relacyjnych baz danych

1. **Analiza dziedziny**: rozpoznanie systemu i zachodzących w nim procesów, które mają być modelowane.
2. **Projekt i implementacja schematu bazy danych (2–3NF)**: tabele i relacje.
3. **Projekt i implementacja więzów integralności**:
   - **wewnętrznych – deklaratywnych** (ograniczenia PK, FK, CHECK, UNIQUE, DEFAULT, reguły, wartości domyślne);
   - **dodatkowych – proceduralnych** (wyzwalacze – ang. *triggers*).
4. **Tworzenie widoków i procedur przechowywanych dla operacji CRUD** – instrukcje INSERT, SELECT, UPDATE, DELETE; transakcje.
5. **Projekt i implementacja indeksów** – na podstawie struktury bazy danych i sposobu jej wykorzystania.
6. **Projekt i implementacja systemu bezpieczeństwa** – konta, role, użytkownicy i ich uprawnienia.
7. **Wdrożenie, konfiguracja zadań automatycznych i utrzymania bazy**.

T-SQL: Tworzenie nowej bazy Składnia

```
CREATE DATABASE database_name -- nazwa bazy
    [ ON                                  -- wskazuje, że pliki danych są podawane jawnie
    [ < filespec > [ ,...n ] ]            -- pliki danych w grupie “Primary”
    [ , < filegroup > [ ,...n ] ]         -- grupy plików; n - wskazuje, że może być ich wiele
    ]
    [ LOG ON { < filespec > [ ,...n ] } ]      -- definicja plików dziennika transakcji
    [ COLLATE collation_name ] -- porządek sortowania i porównywania znaków
    [ FOR LOAD | FOR ATTACH ]             -- opcja dbo use only | przyłączenia bazy
< filespec > ::=
[ PRIMARY ]                               -- definicja grupy plików “Primary”
    ( [ NAME = logical_file_name , ]      -- logiczna nazwa pliku danych
    FILENAME = 'os_file_name’             -- fizyczna nazwa pliku wraz ze ścieżką
    [ , SIZE = size ]                     -- początkowy rozmiar pliku
    [ , MAXSIZE = { max_size | UNLIMITED } ]      -- rozmiar maks. | nieograniczony
    [ , FILEGROWTH = growth_increment ] ) [ ,...n ] -- przyrost rozmiaru pliku
< filegroup > ::=                         -- definicja grupy plików
FILEGROUP filegroup_name < filespec > [ ,...n ]   -- specyfikacja nazw grupy i plików
```

T-SQL: Tworzenie nowej bazy Przykłady (1)

```
-- Proste tworzenie bazy [Uczelnia_Wroclaw] z domyślnymi parametrami
-- (m.in. ścieżki przechowywania plików, ich rozmiar początkowy i przyrost).
CREATE DATABASE Uczelnia_Wroclaw
GO  -- Komenda “GO” rozpoznawana przez m.in. Query Analyzer, sygnalizująca
    -- koniec ciągu instrukcji SQL, które powinny być przesłane do serwera.

-- Usunięcie bazy [Uczelnia_Wroclaw].
-- Aby móc wykonać to polecenie, usuwana baza nie może być używana.
-- (Klient nie może mieć połączenia w kontekście usuwanej bazy,
-- lecz należy uruchomić najpierw np. USE master).

-- Zmiana bieżącej bazy na [master].
USE master
GO

-- Usunięcie bazy.
DROP DATABASE Uczelnia_Wroclaw
GO
```

T-SQL: Tworzenie nowej bazy Przykłady (2)

```
-- Utworzenie bazy z podaniem parametrów.
CREATE DATABASE Uczelnia_Wroclaw
ON PRIMARY
( NAME = Uczelnia_Wroclaw_dat,
 FILENAME = 'd:\Uczelnia_Wroclaw.mdf', -- Lokalizacja głównego pliku bazy.
 SIZE = 20MB, -- Rozmiar początkowy.
 -- Ponieważ nie ma podanego parametru MAXSIZE, plik może rosnąć bez ograniczeń.
FILEGROWTH = 15% ), -- Procentowy rozrost bazy.

LOG ON
( NAME = Uczelnia_Wroclaw_log,
 FILENAME = 'd:\Uczelnia_Wroclaw.ldf', -- Lokalizacja pliku dziennika transakcji.
 SIZE = 10MB,
 FILEGROWTH = 10MB )
-- Porządek sortowania i porównywania znaków: polski, strona kodowa CP1250,
-- bez rozróżniania małych i wielkich liter (CI - case insensitive), z rozróżnianiem
-- znaków narodowych przy sortowaniu, np. A,Ą,B,C,Ć,... (AS - accent sensitive).
COLLATE SQL_Polish_CP1250_CI_AS
```

Właściwości bazy danych MS SQL

Wybrane właściwości:

- **Auto update statistics** – automatyczna aktualizacja statystyk (np. do optymalizacji indeksów);
- **Auto close –** baza danych jest zamykana po wyjściu z niej ostatniego użytkownika;
- **Auto shrink** – pliki bazy automatycznie zmniejszają swoje rozmiary przy kasowaniu danych;
- **Torn page detection –** SQL Server wykrywa niekompletne operacje wejścia – wyjścia.

Możemy je sprawdzać za pomocą:

- **Enterprise Manager –** Database Properties – Options;
- **funkcji „DATABASEPROPERTYEX”**;
- **procedur systemowych „sp_dboption” i „sp_helpdb”**.

T-SQL: Modyfikowanie bazy Składnia

```
ALTER DATABASE database                          -- nazwa bazy danych
    { ADD FILE < filespec > [ ,...n ]            -- dodawanie nowego pliku danych
    [ TO FILEGROUP filegroup_name ]              -- grupa, do której mają być dodane pliki
    | ADD LOG FILE < filespec > [ ,...n ]        -- dodawanie nowego pliku dziennika
    | REMOVE FILE logical_file_name              -- usuwanie pliku poprzez nazwę logiczną
    | ADD FILEGROUP filegroup_name               -- dodawanie grupy plików
    | REMOVE FILEGROUP filegroup_name            -- usuwanie grupy plików
    | MODIFY FILE < filespec >                   -- modyfikowanie pliku
    | MODIFY NAME = new_dbname                   -- zmiana nazwy bazy danych
    | MODIFY FILEGROUP filegroup_name            -- zmiana grupy plików
    {filegroup_property | NAME = new_filegroup_name }
    | SET < optionspec > [ ,...n ] [ WITH < termination > ]  -- ustawianie opcji bazy
    | COLLATE < collation_name >                 -- ustawianie porządku sortowania
    }
```

T-SQL: Modyfikowanie bazy Przykłady (1)

- **Zmiana rozmiaru pliku bazy:**

```
ALTER DATABASE Uczelnia_Wroclaw
MODIFY FILE
    ( NAME = Uczelnia_Wroclaw_dat,
    SIZE = 30MB)
GO
```

- **Dodawanie nowego pliku do bazy:**

```
ALTER DATABASE Uczelnia_Wroclaw
ADD FILE
    (NAME = Uczelnia_Student_dat,
    FILENAME = 'c:\data\Uczelnia_Student.ndf',
    SIZE = 50MB,
    MAXSIZE = 100MB,
    FILEGROWTH = 5MB)
GO
```

T-SQL: Modyfikowanie bazy Przykłady (2)

- **Ustawianie właściwości bazy danych:**

```
-- Wyłączenie automatycznego, okresowego zmniejszania plików bazy.
ALTER DATABASE Uczelnia_Wroclaw
SET AUTO_SHRINK OFF
GO
```

Plan wykładu

1. Tworzenie i modyfikowanie bazy danych.
2. **Typy danych w MS SQL Server 2008 R2.**
3. Tworzenie, modyfikowanie i usuwanie tabel.

Typy danych ANSI-92 a T-SQL

SQL ANSI-92	MS Transact-SQL
binary varying	varbinary
char varying	varchar
character	char lub char(1)
character(n)	char(n)
character varying(n)	varchar(n)
dec	decimal
double precision	float
float[(n)], n = 1-7	real
float[(n)], n = 8-15	float
integer	int
national character(n)	nchar(n)
national char(n)	nchar(n)
national character varying(n)	nvarchar(n)
national char varying(n)	nvarchar(n)
national text	ntext/nvarchar(max)
rowversion	timestamp

Typy danych w T-SQL (1)

- typy znakowe:
  - char [(n)] — 0 – 8000 B (8000 znaków)
  - varchar [(n)] — 0 – 8000 B (8000 znaków)
  - Text/varchar(max) — 0 – 2 GB
- typy znakowe, obsługujące Unicode:
  - nchar [(n)] — 0 – 8000 B (4000 znaków)
  - nvarchar [(n)] — 0 – 8000 B (4000 znaków)
  - Ntext/nvarchar(max) — 0 – 2 GB
- typy binarne (ciąg bitów):
  - binary [(n)] — 0 – 8000 B
  - varbinary [(n)] — 0 – 8000 B

Typy danych w T-SQL (2)

- typy związane z datą i czasem:
  - datetime (01.01.1753 - 31.12.9999; precyzja 0.03s) 8 B
  - smalldatetime (01.01.1990 - 06.06.2079; precyzja 60s) 4 B
  - date/time/datetime2 od SQL 2008
- typy numeryczne, precyzyjne:
  - decimal [(p[,s])] 2 – 17 B
  - numeric[(p[,s])] 2 – 17 B
- typy numeryczne, przybliżone:
  - float [(n)] 8 B
  - real 4 B

Typy danych w T-SQL (3)

- typy całkowite:
  - int (od -2 147 483 648 do 2 147 483 647) 4 B
  - bigint (od -9223372036854775808 do 9223372036854775807) 8 B
  - smallint (od -32 768 do 32 767) 2 B
  - tinyint (od 0 do 255) 1 B
- typy monetarne:
  - money (od -922337203685477,5808 do 922337203685477,5807) 8 B
  - smallmoney (od -214 748,3648 do 214 748,3647) 4 B
- typy do przechowywania obrazów:
  - Image/varbinary(max) 0 – 2 GB

Typy danych w T-SQL (4)

- typ XML – od SQL 2005
- typy geograficzne – od SQL 2008:
  - geometry
  - geography
- typ hierarchiczny – od SQL 2008:
  - hierarchyid

Typy danych w T-SQL (4)

- typy globalnych indentyfikatorów:
  - uniqueidentifier — 16 B
- typy specjalne:
  - bit (wartości: 1, 0, NULL) — 1 B
  - cursor — 0 – 8 B
  - timestamp — 8 B
  - sysname — 256 B
  - table
  - sql_variant — 0 – 8016 B
    (zmienne innych typów z wyjątkiem:
    text, ntext, image, timestamp, sql_variant)
  - typy tablicowe od SQL 2008.

Plan wykładu

1. Tworzenie i modyfikowanie bazy danych.
2. Typy danych w MS SQL Server.
3. **Tworzenie, modyfikowanie i usuwanie tabel.**

T-SQL: Tworzenie nowej tabeli Składnia (1)

```
CREATE TABLE
    [ database_name.[ owner ] . | owner. ] table_name -- nazwa bazy, właściciela
    ( { < column_definition >     -- specyfikacja kolumny
    | column_name AS computed_column_expression  -- nazwa kolumny
    | < table_constraint > ::= [ CONSTRAINT constraint_name ] }
| [ { PRIMARY KEY | UNIQUE } [ ,...n ] )  -- określenie klucza głównego
    [ ON { filegroup | DEFAULT } ]
    [ TEXTIMAGE_ON { filegroup | DEFAULT } ]

< column_definition > ::= { column_name data_type }  -- definicja kolumny
    [ COLLATE < collation_name > ]         -- porządek sortowania pola znakowego
    [ [ DEFAULT constant_expression ]  -- wartość domyślna
    | [ IDENTITY [ ( seed , increment ) -- pole typu autonumerycznego; “seed” -
    [ NOT FOR REPLICATION ] ] ]]     -- wartość początkowa; “increment” - przyrost
    [ ROWGUIDCOL]
    [ < column_constraint > ] [ ...n ] -- c.d.n.
```

T-SQL: Tworzenie nowej tabeli Składnia (2)

```
...
< column_constraint > ::= [ CONSTRAINT constraint_name ]  -- ograniczenie
    { [ NULL | NOT NULL ]       -- dopuszczalna lub niedopuszczalna wartość pusta
    | [ { PRIMARY KEY | UNIQUE }   -- klucz główny
    [ CLUSTERED | NONCLUSTERED ]  -- indeks zgrupowany / niezgrupowany
    [ WITH FILLFACTOR = fillfactor ]            -- współczynnik wypełnienia indeksu
    [ON {filegroup | DEFAULT} ] ]
    ]
    | [ [ FOREIGN KEY ]          -- klucz obcy
    REFERENCES ref_table [ ( ref_column ) ]   -- tabela powiązana
    [ ON DELETE { CASCADE | NO ACTION } ] -- kaskadowe kasowanie
    [ ON UPDATE { CASCADE | NO ACTION } ] -- kaskadowa aktualizacja
    [ NOT FOR REPLICATION ] ]
    | CHECK [ NOT FOR REPLICATION ]  -- ograniczenie typu “Check” - “sprawdź”
    ( logical_expression )
    }

(...) -- Instrukcja “CREATE TABLE” ma więcej opcji, które tutaj pomijamy.
```

T-SQL: Tworzenie nowej tabeli Definiowanie kolumn

Podajemy:

- nazwę tabeli (unikalną w bazie);
- nazwy i typy kolumny (unikalne w tabeli);
- dopuszczalność wartości pustych (NULL albo NOT NULL).

```
CREATE TABLE Przedmiot
(
KodPrzedmiotu char (7) NOT NULL,
Nazwa char (128) NOT NULL,
Semestr tinyint NOT NULL,
ECTS tinyint NOT NULL,
FormaZajec char (10) NOT NULL,
Opiekun int NULL
)
```

T-SQL: Tworzenie nowej tabeli Pola autonumeryczne

Właściwość IDENTITY (autoinkrementacja):

- stosujemy do tworzenia kluczy autonumerycznych;
- określamy wartość początkową („seed”) i przyrost („increment”).

```
CREATE TABLE Zaliczenie
(
IdZaliczenia int IDENTITY (1, 1) NOT NULL,
IdZapisu int NOT NULL,
DataZaliczenia datetime NOT NULL,
Ocena decimal(18, 1) NOT NULL,
Uwagi char (256) NULL,
TypZaliczenia char (32) NOT NULL,
Termin tinyint NOT NULL
)
```

T-SQL: Modyfikacja tabeli Składnia

```
ALTER TABLE table                -- "table" - nazwa tabeli
    { [ ALTER COLUMN column_name    -- zmiana kolumny o podanej nazwie
    { new_data_type [ ( precision [ , scale ] ) ] -- nowy typ danych
    [ COLLATE < collation_name > ]       -- nowy porządek sortowania
    [ NULL | NOT NULL ]          -- nowe określenie dopuszczalności wartości NULL
    | {ADD | DROP } ROWGUIDCOL } ]
    | ADD                        -- dodanie nowej kolumny
    { [ < column_definition > ]
    | column_name AS computed_column_expression
    } [ ,...n ]
    | [ WITH CHECK | WITH NOCHECK ] ADD
    { < table_constraint > } [ ,...n ]
    | DROP                       -- usunięcie ograniczenia, kolumny lub innych elementów
    { [ CONSTRAINT ] constraint_name
    | COLUMN column } [ ,...n ]
    | { CHECK | NOCHECK } CONSTRAINT
    { ALL | constraint_name [ ,...n ] }
    | { ENABLE | DISABLE } TRIGGER -- włączenie (wyłączenie) procedury wyzwalanej
    { ALL | trigger_name [ ,...n ] } }
(...) -- W dalszej części występują analogiczne elementy, jak przy CREATE TABLE.
```

T-SQL: Modyfikacja tabeli Przykłady

```
-- Zmiana typu danych kolumny.
ALTER TABLE Pracownik
ALTER COLUMN Nazwisko nvarchar (50)
GO

-- Dodawanie nowej kolumny do tabeli.
ALTER TABLE Pracownik
ADD COLUMN DataUrodzenia datetime NULL
GO

-- Usuwanie kolumny z tabeli.
ALTER TABLE Pracownik
DROP COLUMN Pensja
GO
```

T-SQL: Usuwanie tabeli

- Składnia:

```
DROP TABLE table_name
```

- Przykład:

```
DROP TABLE Pracownik
```

- **Uwaga:** tabeli nie można usunąć, jeżeli zawiera ona wiersze wskazywane przez klucze obce w innych tabelach (integralność referencyjna, zapewniana automatycznie przez DBMS).
  W przeciwnym razie w innych tabelach mogłyby powstać puste wskazania – do nieistniejących wierszy.

Literatura

1. BEYNON-DAVIES P., *Systemy baz danych – nowe wydanie*, WNT, Warszawa 2003.
2. DATE C. J., DARWEN H., *SQL. Omówienie standardu języka*, WNT, Warszawa 2000 (książka dostępna w bibliotece WSIZ "Copernicus").
3. MICROSOFT, *Books On-Line* – dokumentacja systemu *MS SQL Server*, Microsoft Corp. 1988 – 2000.
4. RANKINS R., JENSEN P., BERTUCCI P., *Microsoft SQL Server 2000. Księga eksperta.*, HELION, Gliwice 2003 (książka dostępna w bibliotece WSIZ „Copernicus").
5. Strona MSDN: http://msdn.microsoft.com.
6. WAYMIRE R., SAWTELL R., *MS SQL Server 2000 dla każdego*, HELION, Gliwice 2002 (książka dostępna w bibliotece WSIZ "Copernicus").

Bazy danych

Wykład 4_2

Dziękuję za uwagę !

Bazy danych

Wykład 4_3

Temat: Deklaratywne więzy integralności w języku T-SQL

Sławomir Świętoniowski

slawomir-swietoniowski@wp.pl

Plan wykładu

1. **Integralność relacyjnej bazy danych.**
2. Więzy integralności w MS SQL Server 2000.

Więzy integralności w bazie danych

- Baza danych jest w stanie **integralności** (spójności), jeśli dane w niej przechowywane są prawidłowe, to znaczy zgodnie z przyjętymi założeniami nie zawierają błędów.
- W transakcyjnych bazach danych zachowanie integralności jest wymogiem absolutnie podstawowym, nadrzędnym wobec wydajności.
- Ze względu na sposób realizacji, rozróżniamy dwa rodzaje integralności:
  - **integralność wewnętrzna** – więzy definiowane na poziomie tabel (szybkie i zalecane, ale posiadające ograniczone możliwości).
  - **integralność dodatkowa** – więzy definiowane zewnętrznie, czyli poza tabelami (duże możliwości, ale zazwyczaj wolniejsze działanie) za pomocą procedur przechowywanych i wyzwalaczy.

Rodzaje więzów integralności

Ze względu na cel stosowania, reguły integralności w bazie danych można podzielić na cztery kategorie.

- **Integralność encji** (spójność jednostkowa) – zapewnia jednoznaczność odwołania do wierszy zapisanych w tabeli, ponieważ każdy wierszy musi się różnić od pozostałych. Własność tę zapewnia klucz główny tabeli (PRIMARY KEY).
- **Integralność dziedziny** – sprawia, że w danej kolumnie wprowadzane są tylko wartości dopuszczalne, zgodne z założeniami systemu. Jest ona realizowana przez definicję typu danych pola, własności NULL / NOT NULL, ograniczenia CHECK, DEFAULT, FOREIGN KEY oraz reguły.
- **Integralność referencyjna** (spójność odwoławcza) – gwarantuje, że jeżeli w danej tabeli jest odwołanie do wierszy w innej tabeli, wiersze te rzeczywiście tam istnieją, a relacja jest zgodna z założeniami systemu.
- **Integralność zdefiniowana przez użytkownika** – reguły, ograniczenia, zasady działania systemu, które nie należą do żadnej z powyższych kategorii. Najczęściej są to więzy skomplikowane, realizowane proceduralnie w samej bazie, albo w innych warstwach aplikacji.

Klucze główne i obce

- **Klucz główny** (ang. *primary key*, PK):
  - zbiór atrybutów, który identyfikuje jednoznacznie wiersze tabeli;
  - w tabeli może być kilka kluczy kandydujących, spośród których wybieramy jeden klucz główny (np. w tabeli [Osoba] kluczami kandydującymi mogą być kolumny [NIP], [PESEL], [IdOsoby]).
- **Klucz obcy** (ang. *foreign key*, FK):
  - pozwala na łączenie danych z różnych tabel;
  - zbiór atrybutów w tabeli, który czerpie swoje wartości z tej samej dziedziny, co klucz główny tabeli powiązanej.

Integralność encji

- Każda tabela musi mieć klucz główny, który jednoznacznie identyfikuje wiersze tej tabeli.
- Klucz główny nie może zawierać wartości pustych (NULL).
- Zabronione są powtórzenia wierszy w ramach jednej tabeli.

Integralność referencyjna

- Klucz obcy może przyjmować jedną z dwóch wartości:
  - klucz główny z tabeli powiązanej;
  - wartość NULL (jeżeli nie koliduje to z innymi regułami integralności).
- Niedozwolone są wskazania poprzez klucz obcy na wiersz, który nie istnieje.
- Kaskadowa aktualizacja i usuwanie zależą od konkretnego zastosowania (np. jeśli usuwamy fakturę VAT z tabeli [Faktura], to usuwamy także wszystkie jej pozycje z powiązanej tabeli [Pozycja]; natomiast jeśli usuwamy grupę studentów z tabeli [Grupa], to raczej nie usuwamy z bazy wszystkich studentów z tej grupy, zapisanych w powiązanej tabeli [Student]).

Plan wykładu

1. Integralność relacyjnej bazy danych.

2. Więzy integralności w MS SQL Server 2008 R2.

Więzy integralności w MS SQL Server

- **Więzy deklaratywne** (preferowane) - więzy statyczne, stosunkowo łatwe do tworzenia i zarządzania, nie obciążają zbytnio systemu**:**
  - ograniczenia (ang. *constraints*);
  - reguły (ang. *rules*);
  - wartości domyślne (ang. *default values*).
- **Metody proceduralne** (wspomagające) - więzy dynamiczne, pozwalające na oprogramowanie skomplikowanych zależności i reguł biznesowych; obciążają system bardziej, niż deklaracje**:**
  - procedury przechowywane (ang. *stored procedures*);
  - procedury wyzwalane, wyzwalacze (ang. *triggers*);
  - *kod innych warstw aplikacji* (poza warstwą bazy danych).

Ograniczenia (Constraints)

Ograniczenia są podstawowym środkiem zachowania spójności danych.
Rodzaje ograniczeń:

- PRIMARY KEY – klucz główny;
- FOREIGN KEY – klucz obcy;
- UNIQUE – pole bez powtórzeń wartości;
- CHECK – sprawdzenie warunku;
- DEFAULT – wartość domyślna pola.

Aktualny stan ograniczeń w tabeli możemy sprawdzać za pomocą procedury systemowej „**sp_helpconstraint**” (np. sp_helpconstraint Student).

PRIMARY KEY – klucz główny

- Ograniczenie PRIMARY KEY służy do zachowania integralności encji poprzez definiowanie klucza głównego tabeli (bez powtórzeń wartości i bez wartości NULL).
- Klucz powinien być jak najkrótszy (najlepiej pojedyncza kolumna). Jeżeli konieczne jest tworzenie klucza z wielu kolumn, należy rozważyć dodanie klucza zastępczego (Id_kolumny).
- Domyślnie na kolumnie (kolumnach) klucza głównego PRIMARY KEY system DBMS tworzy indeks zgrupowany (CLUSTERED INDEX), który przyśpiesza wyszukiwanie wierszy w tabeli.

PRIMARY KEY – przykład

```
-- Definicja tabeli bez klucza.
CREATE TABLE Przedmiot
(
    KodPrzedmiotu char (7) NOT NULL,
    Nazwa char (128) NOT NULL,
    Semestr tinyint NOT NULL,
    ECTS tinyint NOT NULL,
    FormaZajec char (10) NOT NULL,
    Opiekun int NULL
)
GO
-- Oddzielna definicja klucza głównego – rozwiązanie zwiększające
-- elastyczność skryptu T-SQL.
ALTER TABLE Przedmiot ADD
    CONSTRAINT PK_Przedmiot PRIMARY KEY CLUSTERED (KodPrzedmiotu)
GO
```

FOREIGN KEY – klucz obcy

- Ograniczenie FOREIGN KEY zapewnia integralność referencyjną i umożliwia definiowanie schematu relacyjnego (powiązań pomiędzy tabelami).
- Kolumna z ograniczeniem FOREIGN KEY wskazuje na kolumnę w innej tabeli lub inną kolumnę w tej samej tabeli. Obie kolumny muszą być tego samego typu.
- Kolumna, na którą wskazuje FOREIGN KEY musi mieć ograniczenie PRIMARY KEY lub UNIQUE.

FOREIGN KEY – przykłady (1)

- Definiowanie klucza obcego w poleceniu CREATE TABLE.

```
CREATE TABLE Przedmiot
(
    KodPrzedmiotu char (7) NOT NULL,
    Nazwa char (128) NOT NULL,
    Semestr tinyint NOT NULL,
    ECTS tinyint NOT NULL,
    FormaZajec char (10) NOT NULL,
    Opiekun int CONSTRAINT FK_Przedmiot_Opiekun
    REFERENCES Pracownik (NrPracownika) NULL
)
GO
```

FOREIGN KEY – przykłady (2)

- Dodawanie klucza obcego za pomocą polecenia ALTER TABLE.

```
-- Definicja tabeli bez klucza głównego i obcego.
CREATE TABLE Przedmiot
(
    KodPrzedmiotu char (7) NOT NULL,
    Nazwa char (128) NOT NULL,
    Semestr tinyint NOT NULL,
    ECTS tinyint NOT NULL,
    FormaZajec char (10) NOT NULL,
    Opiekun int NULL
)
GO
-- Oddzielna definicja klucza obcego – rozwiązanie zwiększające elastyczność skryptu.
ALTER TABLE Przedmiot ADD
    CONSTRAINT FK_Przedmiot_Opiekun FOREIGN KEY (Opiekun)
    REFERENCES Pracownik (NrPracownika)
GO
```

Przekazywanie integralności referencyjnej

- Służy do wymuszenie kaskadowego aktualizowania i kasowania wierszy w tabeli powiązanej (wskazywanej przez FOREIGN KEY) przy odpowiedniej zmianie wiersza w tabeli macierzystej.
- Do przekazywania integralności służą klauzule:
  - ON UPDATE - przy aktualizacji;
  - ON DELETE - przy kasowaniu.
- Do powyższych klauzul dodajemy słowa:
  - CASCADE - wymusza kaskadowe aktualizowanie lub kasowanie;
  - NO ACTION - wyłącza przekazywanie zmian.
- Należy dobrze przemyśleć użycie tych opcji, zgodnie z zależnościami w dziedzinie, którą modelujemy (np. jeżeli w hurtowni usuwamy fakturę VAT, to powinniśmy usunąć także wszystkie jej pozycje; natomiast jeżeli w szkole usuwamy grupę (np. z powodu zbyt małej liczebności), to niekoniecznie usuwamy automatycznie wszystkich studentów, którzy do niej należeli).

Przekazywanie integralności referencyjnej – przykład

- Definiowanie przekazywania zmian za pomocą ALTER TABLE.

```
ALTER TABLE Oplata
ADD CONSTRAINT FK_Oplata_Student FOREIGN KEY (NrIndeksu)
REFERENCES Student (NrIndeksu)
    ON UPDATE CASCADE
    ON DELETE CASCADE
GO
```

UNIQUE – pole bez powtórzeń

- Ograniczenie UNIQUE jest podobne do PRIMARY KEY, lecz dopuszcza wystąpienie jednej wartości NULL. Zaleca się jednak ustawienie NOT NULL (ponieważ powyższe zachowanie jest nieprzewidywalne).
- UNIQUE stosujemy wówczas, gdy wymagamy, aby dane pole nie zawierało powtarzających się wartości (np. NIP, Nr_faktury).
- Nałożenie ograniczenia UNIQUE na określone kolumny powoduje utworzenie na nich unikalnego indeksu niezgrupowanego (unique, nonclustered index).

UNIQUE – przykłady

```
-- Kolumny [NIP] i [PESEL] nie mogą zawierać powtórzonych wartości.
ALTER TABLE Pracownik ADD
CONSTRAINT UN_Pracownik_NIP UNIQUE (NIP)
GO

ALTER TABLE Pracownik ADD
CONSTRAINT UN_Pracownik_PESEL UNIQUE (PESEL)
GO
```

CHECK – sprawdzenie warunku

- Ograniczenie CHECK określa, jakie wartości są dopuszczalne w danej kolumnie.
- Ograniczenie CHECK jest realizowane za pomocą wyrażenia logicznego, które musi przyjąć wartość TRUE, aby była możliwa modyfikacja danych w określonym polu.
- Wyrażenie sprawdzane w ograniczeniu CHECK może się odnosić do kolumn w tej samej tabeli albo do funkcji bez parametrów. Nie można odwoływać się do pól w innych tabelach.

CHECK – przykłady

```
-- Wymuszenie masek wprowadzania dla numeru PESEL i kodu pocztowego.
ALTER TABLE Pracownik ADD
CONSTRAINT CK_Pracownik_PESEL CHECK
    (PESEL LIKE '[0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9]'),
CONSTRAINT CK_Pracownik_KodPocztowy CHECK
    (KodPocztowy LIKE '[0-9][0-9]-[0-9][0-9][0-9]')
GO

-- Dopuszczalny zakres numeru semestru na studiach od 1 do 7 włącznie.
ALTER TABLE Przedmiot ADD
CONSTRAINT CK_Przedmiot_Semestr CHECK (Semestr BETWEEN 1 AND 7)
GO
```

Wartości domyślne – DEFAULTS

- Wartości domyślne określają, jaką wartość ma przyjmować dana kolumna w tabeli, jeżeli nie zostanie jawnie wprowadzona.
- Wartościami domyślymi mogą być: stałe, wbudowane funkcje oraz wyrażenia arytmetyczne.
- Rodzaje wartości domyślnych (identyczne pod względem funkcjonalności, ale różnie implementowane):
  - **deklaratywne** – definiowane jako ograniczenie DEFAULT w tabeli,
  - **związywane** – tworzone jako oddzielne obiekty w bazie danych.
- Zastosowanie wartości domyślnych:
  - wypełnianie kolumn (np. wartością „nieznany”), które normalnie miałyby wartość NULL;
  - wprowadzanie wartości najczęściej używanych (np. domyślna miejscowość zamieszkania pracownika – „Wrocław”);
  - generowanie danych przez funkcje systemowe (np. GETDATE( )).

Deklaratywne wartości domyślne

Jako rodzaj ograniczeń są implementowane za pomocą poleceń CREATE TABLE oraz ALTER TABLE.

```
ALTER TABLE Student ADD
-- Domyślna wartość numeru indeksu.
CONSTRAINT DF_Student_NrIndeksu DEFAULT (1) FOR NrIndeksu,
-- Data systemowa jako wartość domyślna pola.
CONSTRAINT DF_Student_DataZapisania DEFAULT (GETDATE( ))
FOR DataZapisania,
-- Ustalona data wskazująca na niewykorzystane pole (zamiast NULL).
CONSTRAINT DF_Student_DataWypisania DEFAULT ('01-01-1900')
FOR DataWypisania
GO

-- Usunięcie ograniczenia DEFAULT.
ALTER TABLE Student DROP CONSTRAINT DF_Student_NrIndeksu
GO
```

Literatura

1. BEYNON-DAVIES P., *Systemy baz danych – nowe wydanie*, WNT, Warszawa 2003.
2. DATE C. J., DARWEN H., *SQL. Omówienie standardu języka*, WNT, Warszawa 2000 (książka dostępna w bibliotece WSIZ "Copernicus").
3. MICROSOFT, *Books On-Line* – dokumentacja systemu *MS SQL Server*, Microsoft Corp. 1988 – 2000.
4. RANKINS R., JENSEN P., BERTUCCI P., *Microsoft SQL Server 2000. Księga eksperta.*, HELION, Gliwice 2003 (książka dostępna w bibliotece WSIZ „Copernicus").
5. Strona MSDN: http://msdn.microsoft.com.
6. WAYMIRE R., SAWTELL R., *MS SQL Server 2000 dla każdego*, HELION, Gliwice 2002 (książka dostępna w bibliotece WSIZ "Copernicus").

Bazy danych

Wykład 4_3

Dziękuję za uwagę !

Bazy danych

Wykład 5_1

Temat: Dodawanie, modyfikowanie i usuwanie danych w języku T-SQL

Sławomir Świętoniowski

Plan wykładu

1. **Operacje CRUD w bazie transakcyjnej.**
2. Wstawianie danych.
3. Modyfikowanie danych.
4. Usuwanie danych.

Język operowania na danych (DML)

CRUD (Create, Read, Update, Delete) to podstawowa grupa operacji modyfikujących dane w bazie transakcyjnej (OLTP). Jest ona realizowana w języku SQL za pomocą poleceń (odpowiednio):

- **INSERT** – wstawianie wierszy do tabeli;
- **SELECT** – wybieranie (wyszukiwanie) wierszy z tabeli;
- **UPDATE** – aktualizacja wierszy w tabeli;
- **DELETE** – kasowanie wierszy.

Instrukcja INSERT – wstawianie

Instrukcja INSERT dodaje nowy wiersz do tabeli.

Składnia (najważniejsze elementy):

```
INSERT [ INTO]
    { table_name WITH ( < table_hint_limited > [ ...n ] )
            | view_name
            | rowset_function_limited
    }

    { [ ( column_list ) ]
            { VALUES
                    ( { DEFAULT | NULL | expression } [ ,...n] )
                    | derived_table
                    | execute_statement
            }
    }
```

Instrukcja INSERT – przykłady

```
-- Dodawanie jednego wiersza do tabeli [Przedmiot].
INSERT Przedmiot
(KodPrzedmiotu, Nazwa, Semestr, ECTS, FormaZajec, Opiekun)
VALUES ('INF407W', 'Bazy danych', 4, 4, '20000', 1)


-- Wstawianie wielu wierszy z zastąpieniem klauzuli "VALUES"
-- przez zapytanie "SELECT": „Wstaw to tabeli [Przedmiot_Archiwum]
-- wszystkie przedmioty zawierające w nazwie słowo „baz".
INSERT Przedmiot_Archiwum
(KodPrzedmiotu, Nazwa, Semestr, ECTS, FormaZajec, Opiekun)
SELECT KodPrzedmiotu, Nazwa, Semestr, ECTS, FormaZajec, Opiekun
FROM Przedmiot
WHERE Nazwa LIKE '%' + 'baz' + '%'
```

Kolejność wstawiania danych

- Ze względu na relacje między tabelami (ograniczenia FK) wstawianie wierszy nie może odbywać się w dowolnej kolejności, lecz tak, aby zachowana była integralność referencyjna.
- Najpierw wstawiane są wiersze do tabel, do których są odwołania z innych tabel (przez FK). Odwrotna kolejność powoduje błędy.
- Właściwy porządek wstawiania dla bazy [Uczelnia_Wroclaw]. Jeśli na jednym poziomie jest więcej, niż jedna tabela, oznacza to, że są one wzajemnie niezależne i można do nich wstawiać wiersze naprzemiennie.

  1) Tabele: [Pracownik], [Grupa];

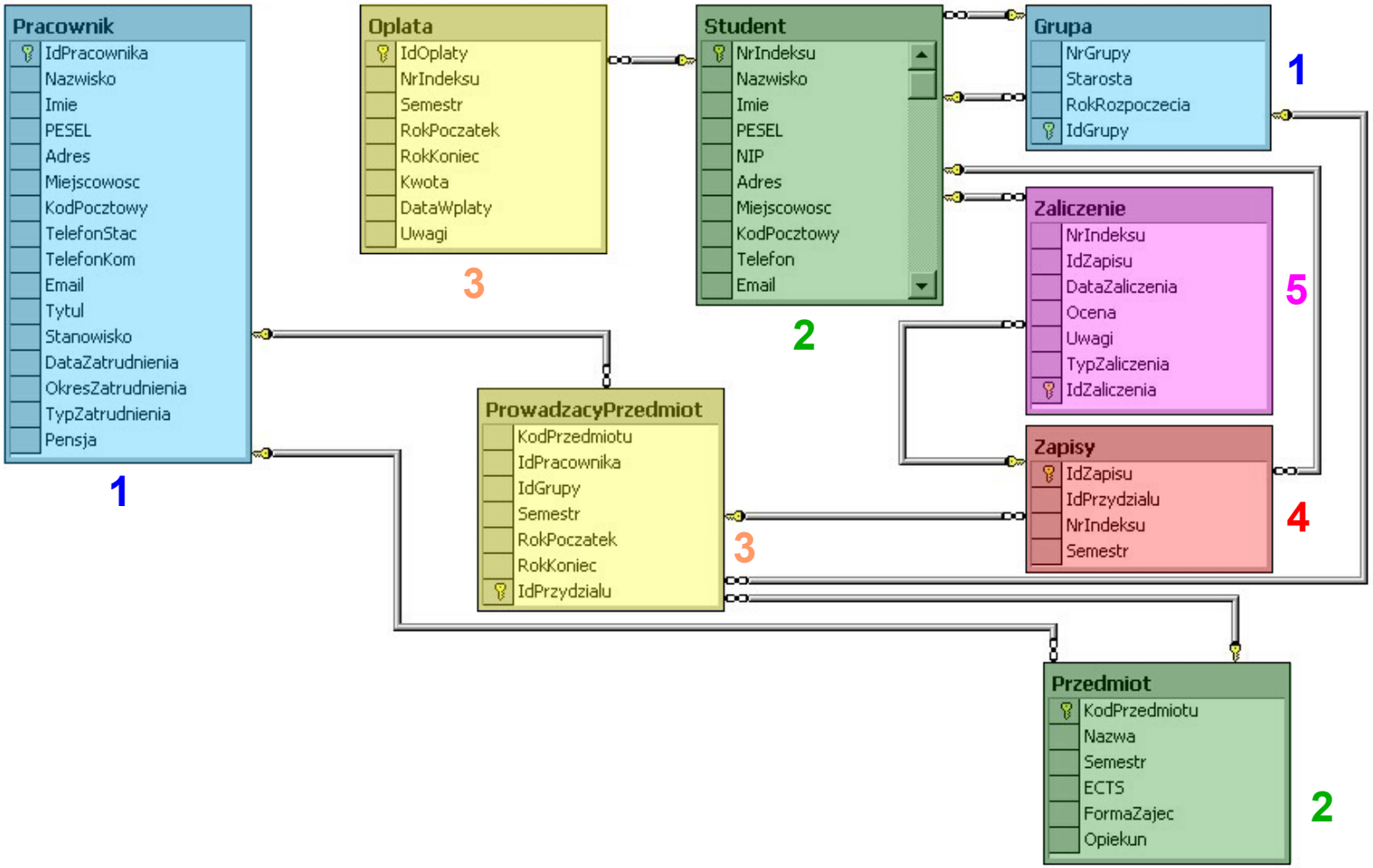
  2) Tabele: [Student], [Przedmiot];

  3) Tabele: [Oplata], [ProwadzacyPrzedmiot];

  4) Tabela: [Zapisy];

  5) Tabela: [Zaliczenie].

Kolejność wstawiania danych



Zrzut ekranowy przedstawia bazę danych „Wyższa uczelnia techniczna”, wersja 1.0.

Instrukcja UPDATE – modyfikacja

Instrukcja UPDATE służy do aktualizacji wartości kolumn w tabeli. Składnia (najważniejsze elementy):

```
UPDATE
    {
            table_name WITH ( < table_hint_limited > [ ...n ] )
            | view_name
            | rowset_function_limited
    }
    SET
    { column_name = { expression | DEFAULT | NULL }
            | @variable = expression
            | @variable = column = expression } [ ,...n ]

    { { [ FROM { < table_source > } [ ,...n ] ]

    [ WHERE < search_condition > ] }
```

Instrukcja UPDATE – przykłady

```
-- Ustanowienie starosty (student z [NrIndeksu] = 3) w grupie '5DB inf'.
UPDATE Grupa
SET Starosta = 3
WHERE NrGrupy = '5DB inf'
GO

-- Modyfikacja kilku pól w jednej instrukcji UPDATE.
UPDATE Pracownik
SET    Adres = 'ul. Nizinna 84/9',
       TelefonStac = '+48-76-790-18-90',
       Stanowisko = 'wykładowca'
WHERE IdPracownika = 10
GO
```

DELETE, TRUNCATE, DROP TABLE

W języku T-SQL dane z tabeli można usunąć na trzy sposoby.

- **DELETE** – wiersze są usuwane z tabeli z możliwością określenia warunku (opcjonalna klauzula WHERE):
  - operacja jest rejestrowana w dzienniku transakcji;
  - licznik pola autonumerycznego (IDENTITY) pozostaje niezmieniony;
- **TRUNCATE TABLE** – usuwa wszystkie wiersze z tabeli, dając wynik podobny do instrukcji DELETE bez klauzuli WHERE, przy czym:
  - strony przechowujące dane są zwalniane bez rejestrowania operacji w dzienniku transakcji (**czyli bezpowrotnie**);
  - operacja jest znacznie szybsza od DELETE, jeśli trzeba usunąć wszystkie wiersze;
  - licznik pola autonumerycznego (IDENTITY) jest ustawiany na wartość początkową (SEED).
- **DROP TABLE** – trwale usuwa ze schematu bazy danych określoną tabelę, a wraz z nią wszystkie jej dane (rozwiązanie niezalecane po wdrożeniu bazy).

Instrukcja DELETE – usuwanie

Instrukcja DELETE służy do kasowania wierszy z tabeli.
Sama tabela pozostaje w schemacie bazy danych, nawet jeśli jest pusta.

Składnia (najważniejsze elementy):

```
DELETE
    [ FROM ]
    { table_name WITH ( < table_hint_limited > [ ...n ] )
    | view_name
    | rowset_function_limited
    }

    [ FROM { < table_source > } [ ,...n ] ]

[ WHERE < search_condition > ]
```

DELETE i TRUNCATE TABLE Przykłady

```
-- Kasowanie wszystkich wierszy z tabeli [Zaliczenie].
DELETE Zaliczenie
GO

-- Usuwanie zaliczeń wystawionych w lutym 2013 r.
DELETE Zaliczenie
WHERE DataZaliczenia BETWEEN '2013-02-01' AND '2013-02-28'
GO

-- Szybkie usuwanie wszystkich wierszy w tabeli [Zaliczenie] z ustawieniem
-- licznika IDENTITY na wartość początkową (SEED).
TRUNCATE TABLE Zaliczenie
GO
```

Kolejność usuwania danych

- Podobnie, jak w przypadku instrukcji INSERT, kasowanie danych w bazie musi się odbywać w określonej kolejności, ze względu na relacje między tabelami (ograniczenia FK).
- Właściwy porządek usuwania wierszy z tabel jest odwrotny do tego, który był przy wstawianiu.
- W bazie [Uczelnia_Wroclaw]:

  1) Tabela: [Zaliczenie];

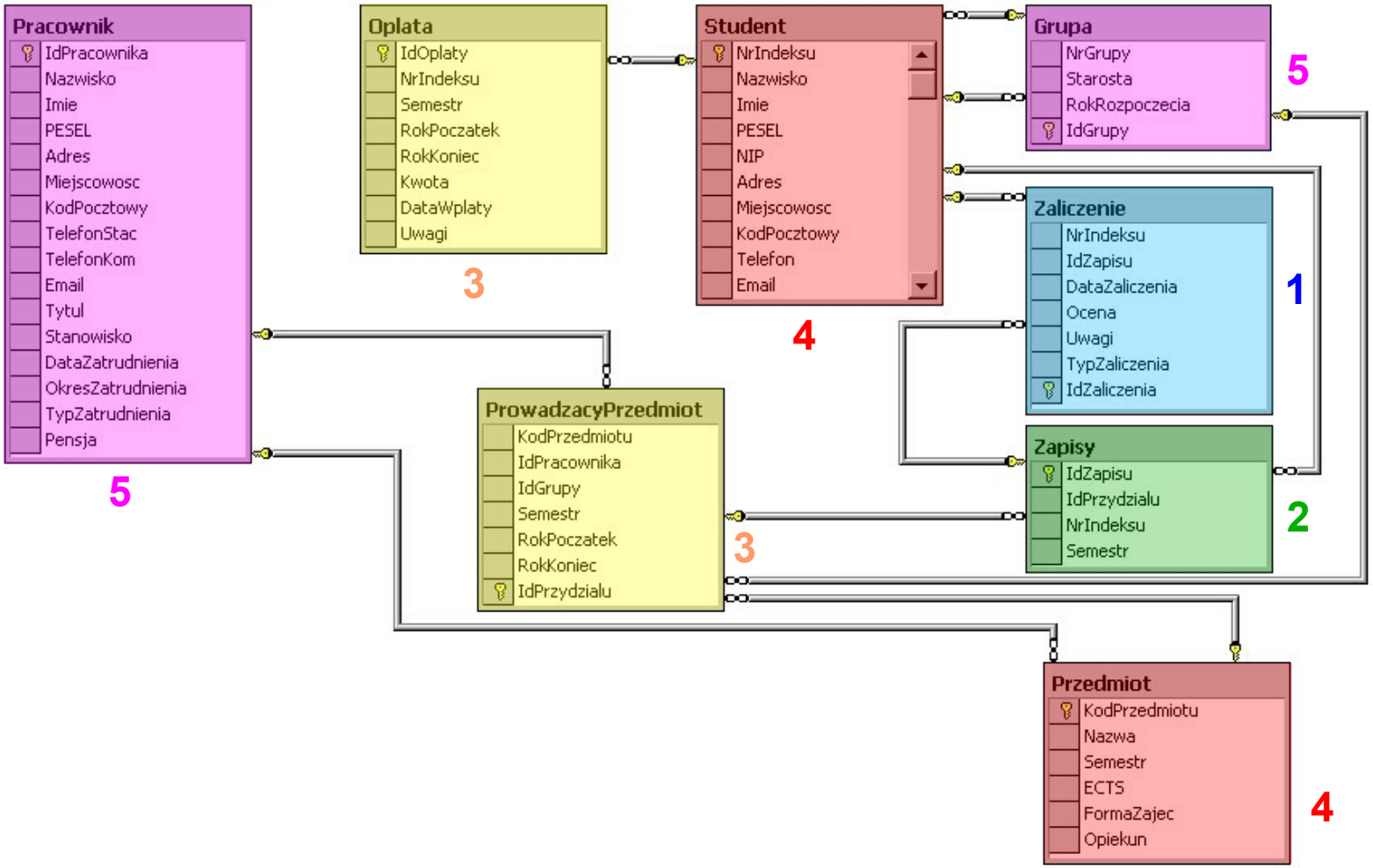
  2) Tabela: [Zapisy];

  3) Tabele: [Oplata], [ProwadzacyPrzedmiot];

  4) Tabele: [Student], [Przedmiot];

  5) Tabele: [Pracownik], [Grupa].

Kolejność usuwania danych



Zrzut ekranowy przedstawia bazę danych „Wyższa uczelnia techniczna”, wersja 1.0.

Literatura

1. BEYNON-DAVIES P., *Systemy baz danych – nowe wydanie*, WNT, Warszawa 2003.
2. DATE C. J., DARWEN H., *SQL. Omówienie standardu języka*, WNT, Warszawa 2000 (książka dostępna w bibliotece WSIZ "Copernicus").
3. MICROSOFT, *Books On-Line* – dokumentacja systemu *MS SQL Server*, Microsoft Corp. 1988 – 2000.
4. RANKINS R., JENSEN P., BERTUCCI P., *Microsoft SQL Server 2000. Księga eksperta.*, HELION, Gliwice 2003 (książka dostępna w bibliotece WSIZ „Copernicus").
5. Strona MSDN: http://msdn.microsoft.com.
6. WAYMIRE R., SAWTELL R., *MS SQL Server 2000 dla każdego*, HELION, Gliwice 2002 (książka dostępna w bibliotece WSIZ "Copernicus").

Bazy danych

Wykład 5_1

Dziękuję za uwagę !

Bazy danych

Wykład 5_2

Temat: Obiekty kodu T-SQL przechowywane na serwerze

Sławomir Świętoniowski

Plan wykładu

1. **Widoki.**
2. Procedury przechowywane.

Widoki (Views)

Widok (ang. *view*, nazywany także „perspektywą”) – jest zapytaniem SQL, przechowywanym w bazie danych, w postaci obiektu. Stanowi on swego rodzaju „okno na dane”, znajdujące się w bazie.

- Cele stosowania widoków:
  - **dostarczenie użytkownikowi tylko takiej informacji, która jest dla niego przydatna i zrozumiała** (np. dane zagregowane lub złączone z wielu tabel, nie zawierające identyfikatorów autonumerycznych);
  - **zwiększenie wydajności** - serwer przechowuje kompilowany plan wykonania dla widoku (nieindeksowanego), dzięki czemu jego uruchamianie jest szybsze od wywoływania zwykłego zapytania SQL;
  - **kontrola dostępu do danych** – różni użytkownicy mogą mieć prawo uruchamiania różnych widoków w ramach tej samej bazy danych.
- Widoki nie zajmują miejsca na dysku, gdyż nie są fizycznie przechowywane w bazie, lecz tworzone na bieżąco w formie tabel wirtualnych.
- Za pomocą widoku można nie tylko wyświetlać dane, ale także wstawiać je do tabel, modyfikować i usuwać (z pewnymi ograniczeniami).

Zasady tworzenia widoków

- Widok jest zawsze oparty na poleceniu SELECT, które wybiera dane z jednej lub kilku tabel (albo widoków). Polecenie to podajemy po słowie AS.
- Wyrażenie SELECT w definicji widoku nie może zawierać:
  - odwołania do tabeli tymczasowej lub zmiennej tabelarycznej;
  - odwołania do więcej niż 1024 kolumn;
  - klauzuli ORDER BY, chyba że wyrażenie SELECT zawiera także klauzulę TOP;
  - słowa kluczowego INTO;
  - klauzul COMPUTE i COMPUTE BY.
- Widoki muszą być tworzone w bieżącej bazie danych. W poleceniu CREATE VIEW nie można określić nazwy bazy.

Tworzenie i modyfikowanie widoku – składnia

Polecenia CREATE VIEW i ALTER VIEW są praktycznie identyczne, ponieważ każda modyfikacja widoku polega na zamianie jego dotychczasowej zawartości na nową, określoną w poleceniu ALTER VIEW.

CREATE / ALTER VIEW [< database_name > .] [< owner > .] view_name
[(column [,...n])]
[WITH < view_attribute > [,...n]]
AS
select_statement
[WITH CHECK OPTION]

< view_attribute > ::=

{ ENCRYPTION | SCHEMABINDING | VIEW_METADATA }

T-SQL: Tworzenie widoku Przykłady

```
-- Prosty widok wyświetlający przedmioty (bez opiekunów).
CREATE VIEW V_Przedmiot_dane
AS -- Niektóre z wybieranych pól mają zmienioną nazwę – nagłówek kolumny.
SELECT KodPrzedmiotu AS [Kod przedmiotu], Nazwa,
       FormaZajec AS [Forma zajec]
FROM Przedmiot
GO

-- Modyfikacja widoku polega na podaniu nowej zawartości jego kodu T-SQL.
ALTER VIEW V_Przedmiot_dane
AS -- Dodatkowo mają być wyświetlane pola [Semestr] i [ECTS].
SELECT KodPrzedmiotu AS [Kod przedmiotu], Nazwa, Semestr, ECTS,
       FormaZajec AS [Forma zajec]
FROM Przedmiot
GO
```

Zwracanie danych z widoku

- Widok może być traktowany jak wirtualna tabela, której wiersze nie są zapisywane trwale w bazie. Zwrócenie danych z widoku odbywa się przez polecenie "SELECT...".
- Jeżeli chcemy wyświetlić tylko niektóre kolumny, ich nazwy muszą być zgodne z nazwami kolumn zdefiniowanymi w widoku, a nie w tabelach źródłowych.

```
SELECT [Kod przedmiotu], Nazwa
FROM V_Przedmiot_dane
GO
```

Wstawianie danych przez widok

Za pomocą widoku można nie tylko wyświetlać dane, ale też wstawiać je trwale do bazy. Jest to możliwe, jeżeli widok zawiera wszystkie pola wymagane w tabeli (czyli NOT NULL).

```
-- W widoku [V_Przedmiot_dane] są wszystkie pola obowiązkowe, nie ma zaś
-- pola [Opiekun] z dopuszczalną wartością NULL. Możemy zatem wykorzystać
-- ten widok do wstawiania danych.
INSERT V_Przedmiot_dane
-- Ważne jest, aby nazwy pól były zgodne z definicją widoku, a nie docelowej tabeli.
([Kod przedmiotu], Nazwa, Semestr, ECTS, [Forma zajec])
VALUES
('INF708W', 'Windows 2', 7, 4, '20000')
GO
```

Modyfikowanie i usuwanie danych przez widok

Widok może także służyć do modyfikowania i usuwania wierszy na identycznych zasadach, jak w przypadku zwykłych tabel.

```
-- Modyfikowanie danych przez widok.
UPDATE V_Przedmiot_dane
SET ECTS = 5
WHERE [Kod przedmiotu] = 'INF708W'
GO

-- Usuwanie danych przez widok.
DELETE V_Przedmiot_dane
WHERE Semestr = 7
GO
```

Plan wykładu

1. Widoki.

2. Procedury przechowywane.

Procedury przechowywane

Procedura przechowywana (ang. *stored procedure*) – jest to jedno lub więcej poleceń języka SQL, przechowywanym w bazie danych w postaci wykonywalnego obiektu.

Cechy procedur przechowywanych:

- można je wywoływać interakcyjnie z poziomu aplikacji klienta, a także z innej procedury przechowywanej lub wyzwalanej;
- procedury mogą pobierać i zwracać parametry, co zwiększa ich użyteczność i elastyczność (np. w porównaniu z widokami);
- mogą zwracać zbiór wyników (np. wierszy z tabeli) i kod wyjścia;
- procedury są przechowywane na serwerze w formie skompilowanej – razem z planem wykonania (*execution plan*).

Sposoby zwracania danych z procedury przechowywanej

- Procedura przechowywana może zwracać dane na cztery sposoby:
  - zbiór wierszy (record set), będący wynikiem wykonania polecenia SELECT;
  - parametry wyjściowe (OUTPUT), zwracające wartości (np. integer, char) albo wskaźnik na kursor (zmienna typu cursor);
  - kod powrotu (return code, return value) – zawsze jako liczba całkowita; może służyć do wychwytywania i obsługi błędów;
  - globalny kursor, do którego można się odwoływać poza procedurą.

Procedury przechowywane Zalety (1)

- **gwarancja integralności danych:**
  - kodowanie logiki aplikacji w pojedynczym miejscu, co ułatwia zarządzanie regułami biznesowymi i zachowanie ich jednolitości;
  - jeśli aplikacja klienta może zmieniać dane wyłącznie za pośrednictwem procedur, jest mniejsze ryzyko wprowadzenia modyfikacji naruszających spójność danych;
  - ograniczenie błędów programistycznych przy tworzeniu aplikacji wielowarstwowych – łatwiejsze wykonywanie złożonych operacji, przekazywanie mniejszej ilości informacji pomiędzy warstwami;
  - zautomatyzowanie złożonych transakcji o dużym znaczeniu dla integralności danych;
- **modularny charakter programowania** – kod SQL podzielony jest na mniejsze fragmenty, które są łatwiejsze do zarządzania;
- **łatwiejsze utrzymanie i modyfikacja aplikacji wielowarstwowej** – przejrzyste oddzielenie bazy danych od warstwy dostępu do danych;

Procedury przechowywane Zalety (2)

- **wzrost bezpieczeństwa danych** – łatwiejsza ochrona przed atakami (np. przed tzw. *wstrzyknięciem kodu SQL – ang. SQL injection* – zob. poz. nr 1 w spisie literatury);
- **ograniczenie ruchu w sieci** – procedury (nawet te złożone z bardzo wielu poleceń SQL) są wywoływane zdalnie za pomocą pojedynczego polecenia z parametrami;
- **wzrost wydajności przy powtarzających się wywołaniach** – procedury na serwerze są optymalizowane (kompilowane przy pierwszym uruchomieniu i przechowywane potem razem z gotowym planem wykonania);
- **ograniczenie dostępu do tabel tylko do zdefiniowanych czynności** – aplikacja klienta może mieć uprawnienia tylko do procedur przechowywanych, a nie bezpośrednio do tabel.

Procedury przechowywane Wady

- **Transact-SQL jest językiem o ograniczonym możliwościach** – może on być niewystarczający do oprogramowania bardzo złożonych operacji; sytuacja ta ulega jednak zmianie, do implementacji procedur przechowywanych, oprócz standardowego T-SQL, można wykorzystywać także wysoko-poziomowe, obiektowe języki platformy MS .NET (np. C#, VB);
- **gorsza integracja ze środowiskami programistycznymi** – nie wszystkie narzędzia do tworzenia aplikacji wielowarstwowych pozwalają na kontrolę wersji i usuwanie błędów (ang. *debugging*) na poziomie procedur przechowywanych (chociaż np. *MS Visual Studio .NET* ma tę możliwość);
- **mała przenośność** – mimo istnienia specyfikacji tworzenia procedur przechowywanych w standardzie ANSI SQL 99, przez większość producentów systemów baz danych nie jest ona w pełni przestrzegana; dlatego kod procedur przechowywanych najczęściej nie daje się bezpośrednio przenosić np. z bazy MS SQL Server do Oracle.

T-SQL: Tworzenie i modyfikowanie procedury – składnia

Podobnie, jak w przypadku widoków, składnia polecenia tworzenia (CREATE) i modyfikowania (ALTER) procedury przechowywanej jest identyczna, ponieważ każda modyfikacja procedury polega na zamianie jego dotychczasowej zawartości na nową, określoną w poleceniu ALTER PROCEDURE.

```
CREATE / ALTER PROC [ EDURE ] procedure_name [ ; number ]
    [ { @parameter data_type }
    [ VARYING ] [ = default ] [ OUTPUT ]
    ] [ ,...n ]

    [ WITH  -- Opcja WITH ENCRYPTION służy do zaszyfrowania kodu T-SQL.
    { RECOMPILE | ENCRYPTION | RECOMPILE , ENCRYPTION } ]

    [ FOR REPLICATION ]

    AS sql_statement [ ...n ]
```

Tworzenie procedury przechowywanej – przykład

```
-- Dodawanie nowego przedmiotu.
CREATE PROC Przedmiot_wstawianie -- Unikalna nazwa procedury.
@Par_KodPrzedmiotu char(7), -- Lista parametrów.
@Par_Nazwa char(128),
@Par_Semestr tinyint,
@Par_ECTS tinyint = 4, -- Wartość domyślna parametru.
@Par_FormaZajec char(10),
@Par_Opiekun int
AS -- Po słowie „AS” następuje ciało procedury – właściwy ciąg poleceń T-SQL.
INSERT Przedmiot
(KodPrzedmiotu, Nazwa, Semestr, ECTS, FormaZajec, Opiekun)
VALUES -- Zamiast stałych wykorzystywane są wartości parametrów procedury.
(@Par_KodPrzedmiotu, @Par_Nazwa, @Par_Semestr, @Par_ECTS,
@Par_FormaZajec, @Par_Opiekun)
GO
```

Przykład wywołania procedury przechowywanej

```
-- Do wywołania procedury używamy słowa EXECUTE (albo EXEC), po którym
-- podajemy parametry i ich wartości. Pomaga to uniknąć pomyłek.
EXECUTE Przedmiot_wstawianie
@Par_KodPrzedmiotu = 'INF407W',
@Par_Nazwa = 'Bazy danych',
@Par_Semestr = 4,
@Par_ECTS = DEFAULT, -- Wymuszenie wartości domyślnej parametru.
@Par_FormaZajec = '20000',
@Par_Opiekun = 1
GO
-- Można także stosować skróconą składnię – podajemy tylko wartości parametrów,
-- koniecznie w takiej kolejności, w jakiej są zadeklarowane w definicji procedury.
-- Taki zapis jest krótszy i wygodniejszy, ale bardziej podatny na błędy.
EXEC Przedmiot_wstawianie 'INF507W', 'Sieciowe bazy danych', 5, 4, '10000', 1
GO
```

Procedura z parametrami wyjściowymi – przykład

```
CREATE PROC Przedmiot_edycja -- Procedura zmienia dane przedmiotu.
@Par_KodPrzedmiotu char(7), -- Parametr służący do identyfikacji przedmiotu.
@Par_Nazwa char(128), -- Pozostałe parametry przekazują nowe wartości pól.
@Par_Semestr tinyint,     -- Konwencja nazewnicza parametrów wejściowych
@Par_ECTS tinyint,        -- „@Par_<nazwa>”, a wyjściowych – „@Out_<nazwa>”.
@Par_FormaZajec char(10),
@Par_Opiekun int,
@Out_LiczbaWierszy int OUTPUT -- Parametr wyjściowy.
AS
UPDATE Przedmiot
SET   Nazwa = @Par_Nazwa, Semestr = @Par_Semestr, ECTS = @Par_ECTS,
      FormaZajec = @Par_FormaZajec, Opiekun = @Par_Opiekun
 WHERE KodPrzedmiotu = @Par_KodPrzedmiotu
 -- Wartość parametru wyjściowego to liczba zmodyfikowanych wierszy.
 SELECT @Out_LiczbaWierszy = @@ROWCOUNT
 GO
```

Przykład wywołania procedury z parametrami wyjściowymi

```
-- Wyłączamy wyświetlanie standardowego komunikatu "N row(s) affected".
-- To ustawienie normalnie obowiązuje dla bieżącego połączenia.
SET NOCOUNT ON
-- Deklarujemy zmienną do pobrania wartości parametru wyjściowego.
DECLARE @Var_Wiersze int
-- Wywołanie procedury. Po zmiennej, podanej jako parametr wyjściowy,
-- musi być umieszczone słowo "OUTPUT".
EXEC    Przedmiot_edycja 'INF407W', 'Bazy danych', 5, 4, '20000', 1,
        @Var_Wiersze OUTPUT
-- Wyświetlenie komunikatu tekstowego. Funkcja CONVERT dokonuje konwersji
-- z jednego typu danych na inny (tutaj z int do varchar(3)).
PRINT ('Zmieniono: ' + CONVERT(varchar(3), @Var_Wiersze) + ' wiersz(y).')
-- Przywracamy wyświetlanie komunikatu "N row(s) affected".
SET NOCOUNT OFF
GO
```

Literatura

1. ANLEY C., *Advanced SQL Injection In SQL Server Applications*, Next Generation Security Software Ltd (http://www.ngssoftware.com), 2002.
2. BEYNON-DAVIES P., *Systemy baz danych – nowe wydanie*, WNT, Warszawa 2003.
3. DATE C. J., DARWEN H., *SQL. Omówienie standardu języka*, WNT, Warszawa 2000 (książka dostępna w bibliotece WSIZ "Copernicus").
4. MICROSOFT, *Books On-Line* – dokumentacja systemu *MS SQL Server*, Microsoft Corp. 1988 – 2000.
5. RANKINS R., JENSEN P., BERTUCCI P., *Microsoft SQL Server 2000. Księga eksperta.*, HELION, Gliwice 2003 (książka dostępna w bibliotece WSIZ „Copernicus").
6. Strona ASP.NET: http://www.asp.net.
7. Strona MSDN: http://msdn.microsoft.com.
8. Strona PHP: http://www.php.net.
9. WAYMIRE R., SAWTELL R., *MS SQL Server 2000 dla każdego*, HELION, Gliwice 2002 (książka dostępna w bibliotece WSIZ "Copernicus").

Bazy danych

Wykład 5_2

Dziękuję za uwagę !

Bazy danych

Wykład 5_3

Temat: Wyszukiwanie danych w języku T-SQL

Sławomir Świętoniowski

Plan wykładu

1. **Polecenie SELECT.**
2. Złączenia tabel.
3. Podzapytania.

Instrukcja SELECT: wyszukiwanie danych

Instrukcja SELECT zwraca (domyślnie) wszystkie wiersze spełniające warunek wyszukiwania. Składnia – ważniejsze elementy:

```
SELECT [ ALL | DISTINCT ] [ TOP [ PERCENT ] [ WITH TIES ] columns
    [ AS column_heading ]
    [ INTO new_table_name ]
    FROM table_source  [ AS table_alias ]
    [ INNER | {{ LEFT | RIGHT | FULL } [ OUTER ] }  | [ CROSS ] ] | JOIN
    table2
    [ [ AS ] table_alias ]
    [ WHERE search_condition ]
    [ GROUP BY [ ALL group_by_expression ] ]
    [ HAVING search_condition ]
    [ ORDER BY order_expression [ ASC | DESC ] ]
```

Instrukcja SELECT Parametry (1)

- **ALL** – zwrócone mogą być wiersze z powtórzeniami wartości (ustawienie domyślne);
- **DISTINCT** – zwracane są wyłącznie różne wiersze;
- **TOP n [PERCENT] –** zwracane jest *n* pierwszych wierszy (pierwsze *n* procent wierszy);
- **WITH TIES** – zwrócone może być więcej, niż *n* pierwszych wierszy, jeśli mają one wartość sortowanego pola taką samą, jak ostatni, *n-ty* wiersz; wymagana jest klauzula ORDER BY;
- **columns** – kolumny, które mają być zwrócone w wyniku; symbol “*” zwraca wszystkie kolumny;
- **AS column_heading** – przy wyświetlaniu wyników nazwa kolumny jest zmieniona na „column heading”;
- **INTO** – tworzona jest nowa tabela, do której kopiowane są wyniki zapytania;
- **FROM** – określna tabele (jedną lub więcej), z których mają być zwrócone wiersze;

Instrukcja SELECT Parametry (2)

- **INNER | {LEFT, RIGHT, FULL} OUTER | CROSS JOIN** – złączenie z inną tabelą – odpowienio: wewnętrzne, zewnętrzne (lewo- i prawostronne, pełne), krzyżowe;
- **WHERE** – określa wyrażenie ograniczające zbiór zwracanych wierszy;
- **search_condition** – wyrażenie, które musi być spełnione, aby dany wiersz był zwrócony.
- **GROUP BY** – określa kolumny (lub wyrażenia nieagregujące), względem których mają być grupowane wiersze w wyniku;
- **HAVING** – klauzula definiująca agregację lub warunek zapytania przy grupowaniu;
- **ORDER BY –** definiuje kolumny, według których ma być sortowany wynikowy zbiór wierszy;
- **ASC | DESC** – porządek sortowania – odpowiednio: rosnący (domyślnie) albo malejący.

SELECT: klauzula WHERE

Klauzula WHERE pozwala na zdefiniowanie warunku, który ogranicza wynikowy zbiór wierszy. Dopuszczalne są następujące operatory:

- **=, < > (różny), <, >, >=, <=** – zwyczajne operatory porównania;
- **BETWEEN wyrażenie1 AND wyrażenie2** – operator “pomiędzy” (łącznie z końcami przedziału);
- **IN | NOT IN (element1, element2, ..., elementN)** – wartość kolumny musi (albo nie może) należeć do podanego zbioru;
- **LIKE wyrażenie** – pozwala na porównania przybliżone; można je definiować za pomocą znaków wieloznacznych (ang. *wildcards*):
  - “%”d – dowolny znak lub cyfra;
  - “-”d – dowolny pojedynczy znak;
  - “[ ]” - dowolny znak umieszczony w nawiasach.

SELECT...FROM...WHERE Przykład

```
-- Proste wyszukiwanie przedmiotów, które zawierają w nazwie podany wzorzec.
CREATE PROC Przedmiot_szukanie
@Par_Nazwa varchar(128) = ' ' -- Wzorzec, który musi wystąpić w nazwie przedmiotu.
AS
BEGIN  -- Kolumny [KodPrzedmiotu] i [Forma] mają zmienione nagłówki przez AS.
SELECT KodPrzedmiotu AS [Kod], Nazwa, Semestr, ECTS,
            FormaZajec AS [Forma]
FROM Przedmiot
WHERE Nazwa LIKE '%' + @Par_Nazwa + '%'
ORDER BY KodPrzedmiotu ASC -- Sortowanie rosnące według pola [KodPrzedmiotu].
END
GO

-- Przykładowe wywołanie.
EXEC Przedmiot_szukanie 'sieci'
```

SELECT: kolumny wyliczeniowe

W instrukcji SELECT można podawać, oprócz nazw pól, także wyrażenia obliczane na podstawie pól i funkcji.

```
-- Zapytanie: "zwróć tytuł, imię i nazwisko pracowników – jako pojedyncze pole,
-- posortowane po nazwisku".
SELECT Tytul + Imie + Nazwisko AS [Pracownik]
FROM Pracownik
ORDER BY Nazwisko ASC
GO

-- Zapytanie: "zwróć numer indeksu, imię, nazwisko studenta i liczbę miesięcy,
-- które upłynęły od daty jego zapisania (staż)".
-- Staż jest obliczany na podstawie pola [DataZapisania] i bieżącej daty systemowej.
SELECT NrIndeksu, Imie, Nazwisko,
DATEDIFF(Month, DataZapisania, GETDATE( )) AS [Staz na uczelni]
FROM Student
ORDER BY [Staz na uczelni] DESC, Nazwisko ASC
GO
```

Klauzule GROUP BY i HAVING

- Klauzula GROUP BY umożliwia utworzenie podsumowań dla wybranych grup tabeli.
- Jeżeli wraz z GROUP BY ma być użyta klauzula WHERE, to musi być ona umieszczona przed grupowaniem. Klauzula WHERE nie może zawierać funkcji agregujących.
- Każde pole wymienione na liście SELECT, nie będące wyrażeniem agregującym, musi być umieszczone także w klauzuli GROUP BY, albo listy pól po SELECT i GROUP BY muszą być identyczne.
- Zalecane jest używanie wraz z GROUP BY klauzuli ORDER BY, aby grupowane wiersze zostały zwrócone w podanej kolejności. W przeciwnym razie kolejność ta nie jest określona.
- Klauzula HAVING pozwala na ograniczenie liczby grupowanych wierszy na podstawie podanego warunku, który może zawierać funkcje agregujące.

GROUP BY i HAVING Przykład

```
-- Procedura zliczająca studentów, mieszkających w poszczególnych miejscowościach.
-- Pod uwagę brane są tylko te miejscowości, których nazwa zawiera podany wzorzec
-- i w których mieszka co najmniej zadana liczba studentów.
CREATE PROCEDURE Student_zlicz_miejsc
@Par_Miejscowosc varchar(40) = ' ', -- Wzorzec nazwy miejscowości.
@Par_MinStud int = 0 -- Minimalna liczba studentów.
AS
BEGIN
SELECT Miejscowosc, COUNT(*) AS [Liczba studentow]
FROM Student
WHERE Miejscowosc LIKE '%' + @Par_Miejscowosc + '%'
GROUP BY Miejscowosc
HAVING COUNT(*) >= @Par_MinStud  -- Warunek minimalnej liczby studentów.
ORDER BY [Liczba studentow] DESC -- Sortowanie malejące po liczbie studentów.
END
GO
```

Plan wykładu

1. Polecenie SELECT.
2. **Złączenia tabel.**
3. Podzapytania.

Złączenia tabel

Złączenia służą do przedstawiania połączonych danych z więcej, niż jednej tabeli**.**

- Połączenie opiera się na porównaniu wartości kolumn w obu tabelach, które często opowiadają relacji kluczy głównych i obcych.
- Wyróżniane są następujące rodzaje złączeń:
  - wewnętrzne (INNER JOIN);
  - własne (SELF-JOIN) – odmiana złączenia wewnętrznego;
  - zewnętrzne (LEFT | RIGHT | FULL OUTER JOIN);
  - krzyżowe (CROSS JOIN).
- Są dwa rodzaje składni określającej złączenie:
  - warunek złączenia w klauzuli WHERE (składnia stara, niezalecana);
  - warunek złączenia w klauzuli FROM (nowa, zalecana składnia, zgodna ze standardem ANSI-92).

Złączenie tabel jako operacja algebry relacyjnej

Złączeniem (wewnętrznym) tabel *R* i *S* jest tabela *T*, w której wiersze mają połączone kolumny z tabel *R* i *S*. W tabeli *T* są tylko te wiersze, w których nastąpiło dopasowanie wartości określonych kolumn z obu tabel.

Tabela *R*

Imie	Nazwisko	KodPrzedmiotu
Jan	Kowalski	INF507
Andrzej	Jabłoński	INF517
Krzysztof	Nowak	INF517
Leszek	Morawski	INF517

N — 1

Tabela *S*

KodPrzedmiotu	Nazwa	ECTS
INF407	Bazy danych	4
INF507	Sieciowe bazy danych	4
INF517	Grafika komputerowa	5

Złączone tabele *R* i *S*

Imie	Nazwisko	KodPrzedmiotu	Nazwa	ECTS
Jan	Kowalski	INF507	Sieciowe bazy danych	4
Andrzej	Jabłoński	INF517	Grafika komputerowa	5
Krzysztof	Nowak	INF517	Grafika komputerowa	5
Leszek	Morawski	INF517	Grafika komputerowa	5

```
-- Odpowiednik w języku SQL (nowa składnia zgodna z ANSI-92):
SELECT Imie, Nazwisko, KodPrzedmiotu, Nazwa, ECTS
FROM R INNER JOIN S ON R.KodPrzedmiotu = S.KodPrzedmiotu
```

Złączenia tabel
Składnia nowa i stara

-- Oba poniższe zapytania zwracają identyczny wynik:
-- „zwróć dane przedmiotów i nazwiska ich opiekunów merytorycznych”.
-- Nowa składnia, zgodna z ANSI-92 (zalecana).
-- Warunek złączenia jest podany w klazuli FROM.

```
SELECT KodPrzedmiotu AS [Kod], Nazwa,  Nazwisko AS [Opiekun merytoryczny]
FROM Przedmiot INNER JOIN Pracownik
    ON Przedmiot.Opiekun = Pracownik.IdPracownika
```

-- Składnia stara (niezalecana).
-- Warunek złączenia jest podany w klazuli WHERE.

```
SELECT KodPrzedmiotu AS [Kod], Nazwa, Nazwisko AS [Opiekun merytoryczny]
FROM Przedmiot, Pracownik
WHERE Przedmiot.Opiekun = Pracownik.IdPracownika
```

Złączenia tabel
Zalety nowej składni (ANSI)

- Przejrzystość i zrozumiałość – zwłaszcza przy złączeniach wielu tabel jednocześnie.
- Zapewnienie podwójnych złączeń zewnętrznych, których nie obsługuje stara składnia.
- Możliwość użycia poleceń dla optymalizatora zapytań, określających sposób złączenia tabel. Opcja ta nie jest dostępna w starej składni.
- W kolejnych wersjach MS SQL Server stara składnia może nie być obsługiwana.

Złączenie wewnętrzne: INNER JOIN

Jest to najczęściej stosowane złączenie. Wiersze zwracane w wyniku złączenia wewnętrznego mają taką samą wartość określonej kolumny w obu łączonych tabelach.

```
-- Zapytanie: "zwróć studentów z podanej grupy".
CREATE PROCEDURE Student_Grupa_szukanie
@Par_NrGrupy varchar(7) = ' '
AS
BEGIN
SELECT NrIndeksu AS [Numer indeksu], Imie, Nazwisko, NrGrupy AS [Grupa]
FROM Student INNER JOIN Grupa -- Nazwy tabel i rodzaj ich złączenia.
   ON Student.IdGrupy = Grupa.IdGrupy -- Opis kolumn łączących.
-- Nie wszystkie kolumny łączące nie muszą występować na liście SELECT.
WHERE NrGrupy LIKE '%' + @Par_NrGrupy + '%'
ORDER BY NrGrupy ASC, Nazwisko ASC
END
```

Złączenie własne: SELF JOIN

Złączenie własne jest odmianą złączenia wewnętrznego, w której dana tabela jest łączona z samą sobą.

```
-- Zapytanie: "zwróć studentów, którzy mają takie samo nazwisko".
CREATE PROCEDURE Student_Nazwisko
@Par_Nazwisko varchar(7) = ' '
AS
BEGIN
SELECT stud1.NrIndeksu, stud1.Imie, stud1.Nazwisko,
stud2.NrIndeksu, stud2.Imie, stud2.Nazwisko, stud1.Miejscowosc
FROM Student stud1 INNER JOIN Student stud2 -- Wykorzystywane są aliasy.
    ON stud1.Nazwisko = stud2.Nazwisko -- Dopasowywane kolumny.
WHERE (stud1.Nazwisko LIKE '%' + @Par_Nazwisko + '%')
AND   (stud1.NrIndeksu < stud2.NrIndeksu) -- Warunek zapobiegający powtórzeniom.
ORDER BY stud1.Nazwisko ASC
END
```

Złączenie zewnętrzne: OUTER JOIN

Złączenie zewnętrzne zwraca wszystkie wiersze z tabeli, która została określona jako zewnętrzna (LEFT OUTER, RIGHT OUTER, FULL OUTER), nawet jeśli wartości łączonych kolumn nie są równe.

- Wiersze tabeli zewnętrznej, dla których nie istnieje dopasowanej wartości z tabeli wewnętrznej, zawierają w kolumnie łączonej wartość NULL.
- Złączenie lewostronne (LEFT OUTER JOIN) i prawostronne (RIGHT OUTER JOIN) nie różnią się między sobą poza kolejnością specyfikacji tabeli zewnętrznej i wewnętrznej.
- Złączenie zewnętrzne pełne (FULL OUTER JOIN) zwraca wszystkie odpowiadające sobie wiersze z łączonych tabel, a następnie wiersze, dla których dopasowanie nie zachodzi.

Złączenie zewnętrzne: OUTER JOIN – przykład

```
-- Zwracanie grup i ich starostów (jeśli są ustanowieni).
CREATE PROC Grupa_szukanie
@Par_RokRozpoczecia smallint = 1990 -- Rok rozpoczęcia studiów przez grupę.
AS
BEGIN
-- Imię i nazwisko starosty grupy są sklejane w pojedynczy ciąg znaków
-- po prawostronnym obcięciu spacji przez funkcję RTRIM( ).
SELECT NrGrupy AS [Nazwa grupy], RokRozpoczecia AS [Rok rozpoczecia],
RTRIM (Imie) + ' ' + RTRIM (Nazwisko) AS [Starosta], NrIndeksu AS [Nr indeksu]
-- Lewostronne złączenie zewnętrzne zwraca wszystkie grupy spełniające warunek.
-- WHERE, nawet jeśli nie mają one starostów (wówczas są to pola NULL).
FROM Grupa LEFT OUTER JOIN Student
      ON Grupa.Starosta = Student.NrIndeksu
WHERE RokRozpoczecia >= @Par_RokRozpoczecia
END
```

Złączenie krzyżowe: CROSS JOIN

Wynikiem złączenia krzyżowego są wiersze, zawierające wszystkie możliwe kombinacje wartości określonych kolumn (iloczyn kartezjański tabel).
Liczba zwracanych wierszy może być bardzo duża.

```
-- Złączenie krzyżowe.
-- Zapytanie: "wypisz wszystkie kombinacje imion i nazwisk studentów".
CREATE PROCEDURE Student_Imie_Nazwisko
AS
BEGIN
SELECT DISTINCT stud1.Imie, stud2.Nazwisko
FROM Student stud1 CROSS JOIN Student stud2
ORDER BY stud2.Nazwisko, stud1.Imie
END
GO
```

Plan wykładu

1. Polecenie SELECT.
2. Złączenia tabel.
3. **Podzapytania.**

Podzapytania

Podzapytanie jest zapytaniem SQL, które występuje wewnątrz innego zapytania i może być użyte w miejscu wyrażenia.

- Wynikiem podzapytania może być pojedynczy wiersz lub kolumna tabeli.
- Wyniki podzapytania i złączenia są często takie same. Złączenie jest jednak bardziej wydajne niż podzapytanie. Wyjątkiem jest sytuacja, w której mają być usunięte powtarzające się wiersze (DISTINCT). Wówczas podzapytanie użyte z NOT EXISTS działa szybciej.
- Podzapytania są konieczne w przypadku zapytań, które:
  - zawierają warunki nieistnienia lub niewchodzenia w skład (NOT EXISTS, NOT IN);
  - wymagają użycia funkcji agregującej w klauzuli WHERE.

Podzapytania – przykłady (1)

```
-- Poniższe zapytanie można realizować alternatywnie za pomocą złączeń
-- (jest ono tak zaimplementowane w procedurze Student_Grupa_szukanie).
-- Zapytanie: "zwróć studentów z podanej grupy".

CREATE PROCEDURE Student_Grupa_podzapytanie
@Par_NrGrupy varchar(7) = ''
AS
BEGIN
SELECT NrIndeksu AS [Numer indeksu], Imie, Nazwisko
FROM Student
WHERE IdGrupy IN (SELECT IdGrupy
                  FROM Grupa
                  WHERE NrGrupy LIKE '%' + @Par_NrGrupy + '%')
ORDER BY Nazwisko ASC, Imie ASC
END
GO
```

Podzapytania – przykłady (2)

```
-- Poniższego zapytania NIE MOŻNA realizować alternatywnie
-- za pomocą złączeń, ponieważ odwołuje się do warunków nieistnienia.
-- Zapytanie: "zwróć studentów, którzy nie wnieśli żadnej opłaty od podanej daty".

CREATE PROCEDURE Student_Oplata_podzapytanie
@Par_DataWplaty datetime
AS
BEGIN
SELECT DISTINCT NrIndeksu AS [Numer indeksu], Imie, Nazwisko
FROM Student
WHERE NrIndeksu NOT IN    (SELECT NrIndeksu
                           FROM Oplata
                           WHERE DataWplaty >= @Par_DataWplaty)
ORDER BY Nazwisko ASC, Imie ASC
END
GO
```

Podzapytania – przykłady (3)

```
-- Poniższego zapytania NIE MOŻNA realizować alternatywnie
-- za pomocą złączeń, ponieważ wykorzystuje ono funkcję agregującą
-- w klauzuli WHERE.
-- Zapytanie: "zwróć pracowników, których wynagrodzenie jest niższe
--                  od średniej płacy wszystkich pracowników".

CREATE PROCEDURE Pracownik_Placa_podzapytanie
AS
BEGIN
SELECT Tytul, Imie, Nazwisko, Pensja
FROM Pracownik
WHERE Pensja <    (SELECT AVG(Pensja)
                   FROM Pracownik)
ORDER BY Pensja ASC, Nazwisko ASC, Imie ASC
END
GO
```

Podzapytania jako tabele pochodne

W zapytaniu można zastosować tabelę pochodną, która jest zapytaniem zagnieżdżonym, zawartym w klauzuli FROM.

- Do tabeli pochodnej można odwoływać się przez alias i można traktować ją jak normalną tabelę lub jak dynamiczny widok.
- Podzapytanie może zawierać agegacje (np. COUNT), grupowania (GROUP BY) i sumy wierszy (UNION).

Podzapytania jako tabele pochodne – przykład

```
-- Zapytanie:
-- "wypisz wszystkie kombinacje imion i nazwisk studentów i pracowników".

CREATE PROCEDURE Student_Prac_podzapytanie
AS
BEGIN
SELECT DISTINCT stud1.Imie, stud2.Nazwisko
FROM    (SELECT Imie FROM Student
          UNION
         SELECT Imie FROM Pracownik) stud1
CROSS JOIN
          (SELECT Nazwisko FROM Student
           UNION
           SELECT Nazwisko FROM Pracownik) stud2
ORDER BY stud2.Nazwisko, stud1.Imie
END
GO
```

Literatura

1. BEYNON-DAVIES P., *Systemy baz danych – nowe wydanie*, WNT, Warszawa 2003.
2. DATE C. J., DARWEN H., *SQL. Omówienie standardu języka*, WNT, Warszawa 2000 (książka dostępna w bibliotece WSIZ "Copernicus").
3. MICROSOFT, *Books On-Line* – dokumentacja systemu *MS SQL Server*, Microsoft Corp. 1988 – 2000.
4. RANKINS R., JENSEN P., BERTUCCI P., *Microsoft SQL Server 2000. Księga eksperta.*, HELION, Gliwice 2003 (książka dostępna w bibliotece WSIZ „Copernicus").
5. Strona MSDN: http://msdn.microsoft.com.
6. WAYMIRE R., SAWTELL R., *MS SQL Server 2000 dla każdego*, HELION, Gliwice 2002 (książka dostępna w bibliotece WSIZ "Copernicus").

Bazy danych

Wykład 5_3

Dziękuję za uwagę !