

informatyka+

Algorytmika i programowanie

Bazy danych

Multimedia, grafika i technologie internetowe

Sieci komputerowe

Tendencje w rozwoju informatyki i jej zastosowań

informatyka+

Wszechnica Informatyczna: Bazy danych

Podstawy projektowania
i implementacji baz danych

Andrzej Ptasznik

Człowiek – najlepsza inwestycja

Człowiek – najlepsza inwestycja



KAPITAŁ LUDZKI
NARODOWA STRATEGIA SPÓJNOŚCI



**WARSZAWSKA
WYŻSZA SZKOŁA
INFORMATYKI**

UNIA EUROPEJSKA
EUROPEJSKI
FUNDUSZ SPOŁECZNY



Projekt współfinansowany ze środków Unii Europejskiej w ramach Europejskiego Funduszu Społecznego.



KAPITAŁ LUDZKI
NARODOWA STRATEGIA SPÓJNOŚCI



**WARSZAWSKA
WYŻSZA SZKOŁA
INFORMATYKI**

UNIA EUROPEJSKA
EUROPEJSKI
FUNDUSZ SPOŁECZNY



Projekt współfinansowany ze środków Unii Europejskiej w ramach Europejskiego Funduszu Społecznego.

Podstawy projektowania i implementacji baz danych





Rodzaj zajęć: Wszechnica Informatyczna

Tytuł: Podstawy projektowania i implementacji baz danych

Autor: mgr inż. Andrzej Ptasznik

Redaktor merytoryczny: prof. dr hab. Maciej M Sysło

Zeszyt dydaktyczny opracowany w ramach projektu edukacyjnego **Informatyka+** – ponadregionalny program rozwijania kompetencji uczniów szkół ponadgimnazjalnych w zakresie technologii informacyjno-komunikacyjnych (ICT).

www.informatykaplus.edu.pl

kontakt@informatykaplus.edu.pl

Wydawca: Warszawska Wyższa Szkoła Informatyki

ul. Lewartowskiego 17, 00-169 Warszawa

www.wysi.edu.pl

rektorat@wysi.edu.pl

Skład: Recontra Studio Graficzne

Warszawa 2010

Copyright © Warszawska Wyższa Szkoła Informatyki 2010

Publikacja nie jest przeznaczona do sprzedaży.



KAPITAŁ LUDZKI
NARODOWA STRATEGIA SPÓJNOŚCI



WARSZAWSKA
WYŻSZA SZKOŁA
INFORMATYKI

UNIA EUROPEJSKA
EUROPEJSKI
FUNDUSZ SPOŁECZNY



Projekt współfinansowany ze środków Unii Europejskiej w ramach Europejskiego Funduszu Społecznego.

Podstawy projektowania i implementacji baz danych



Andrzej Ptasznik

Warszawska Wyższa Szkoła Informatyki

aptaszni@wwsi.edu.pl

Streszczenie

W ramach tego kursu uczniowie zapoznają się z MS SQL Server 2008. Poznają podstawy projektowania relacyjnych baz danych i wykonają instalację instancji SQL Server 2008 Express. Praktycznie wykonają implementację zaprojektowanej bazy danych. Nauczą się, jak określać i definiować reguły poprawności dla wybranych danych. W kolejnych zadaniach wyjaśnione zostaną problemy definiowania widoków i ich wykorzystanie. Nauczą się pisanie zapytań do bazy danych. Omówione zostaną indeksy i ich znaczenie dla wydajności w procesie pobierania danych. Dodatkowo omówione zostaną sposoby tworzenia kopii zapasowej bazy danych i odtwarzania bazy z jej kopii. Problemy omawiane w trakcie kursu oraz realizowane ćwiczenia powinny umożliwić uczestnikom samodzielne tworzenie prostych baz danych.

Spis treści

1. Wprowadzenie	
1.1. Wstęp	5
1.2. Podstawowe pojęcia związane z bazami danych	5
2. Projektowanie bazy danych	5
3. Technologia MS SQL Server 2008	8
4. Środowisko SQL Server Management Studio	15
5. Implementacja zaprojektowanej bazy danych	16
6. Podstawy języka SQL	26
6.1. Wprowadzenie do języka SQL	26
6.2. Język opisu danych (DDL).....	27
6.3. Język manipulacji danymi (DML).....	28
7. Zapytania do baz danych SELECT	29
8. Inne obiekty bazy danych	39
8.1. Widoki.....	39
8.2. Procedury składowane.....	42
8.3. Funkcje składowane.....	43
8.4. Wyzwalacze.....	44
Literatura	45



1. WPROWADZENIE

1.1. WSTĘP

Historia relacyjnego modelu danych rozpoczęła się w roku 1970 wraz z publikacją E.F. Codda *A Relational Model of Data for Large Shared Data Banks*. (pol. *Relacyjny model danych dla dużych współdzielonych banków danych*). Ten artykuł wzbudził duże zainteresowanie, ponieważ przedstawiał możliwości realizacji i praktyczne zastosowania komercyjnego systemu baz danych. Praca ta stworzyła teoretyczne podstawy budowania baz danych w oparciu o relacyjne podejście do jej modelowania. Opublikowana teoria bardzo szybko zainteresowała potencjalnych twórców i producentów systemów zarządzania bazami danych. Droga od teorii do praktyki bywa często długa i wyboista ale w tym przypadku, ze względu na pilne potrzeby rynku, przebiegała dość szybko i sprawnie. W ramach naszego kursu zapoznamy się z Systemem Zarządzania Relacyjnymi Bazami Danych MS SQL Server 2008.

1.2. PODSTAWOWE DEFINICJE ZWIĄZANE Z BAZAMI DANYCH

Wprowadzimy na początku kilka definicji wyjaśniających podstawowe pojęcia, którymi będziemy się posługiwali w dalszej części:

- **Dane** to liczby, znaki, symbole (i cokolwiek innego) zapisane w celu ich przetwarzania.
- **Informacja** to taki czynnik, któremu człowiek może przypisać określony sens (znaczenie), aby móc ją wykorzystywać do różnych celów.
- **Wiedza** to, zgodnie z prastarą definicją Platona, *ogół wiarygodnych informacji o rzeczywistości wraz z umiejętnością ich wykorzystywania*.

Można teraz uporządkować te pojęcia w kontekście baz danych i sposobu ich wykorzystania. Dane zbieramy i zapisujemy, by na ich podstawie uzyskiwać informacje, które będą stawały się wiedzą, gdy uzupełnimy je o sposoby i możliwości ich praktycznego wykorzystania. Nie trzeba chyba udowadniać tezy, że współczesne społeczeństwo, społeczeństwo informacyjne, opiera swoje działania i podstawy rozwoju na wiedzy, która jest między innymi pozyskiwana z baz danych.

Zgodnie z powyższymi definicjami, dane to prawie wszystko co może być zapisane i dlatego jednym z podstawowych zadań, żeby zbiór danych mógł stać się bazą danych, jest zapewnienie odpowiedniego sposobu uporządkowania. Bazy danych mogą gromadzić gigantyczne ilości danych zapewniając właściwy sposób ich uporządkowania. Teraz możemy spróbować zdefiniować pojęcie bazy danych:

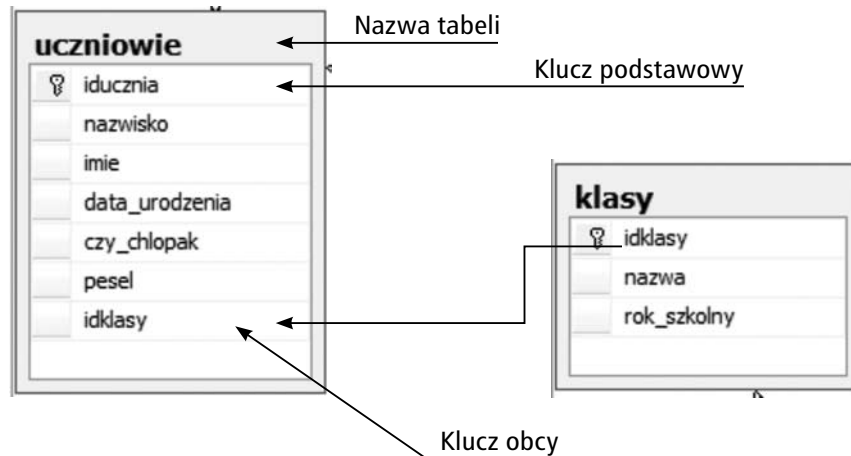
Baza danych to zbiór danych zapisanych w ściśle określony sposób w strukturach odpowiadających założonemu modelowi danych.

2. PROJEKTOWANIE BAZY DANYCH

Jedyną strukturą danych w modelu relacyjnym jest **tabela**, ale jedna tabela może zawierać dane tylko na określony temat (dane o uczniu, dane o planowanych wizytach lekarskich itp.). Nie można więc utożsamiać tabeli z bazą danych, ponieważ baza danych jest pojęciem szerszym a tabele są elementami składowymi bazy danych. To, jakie tabele będzie zawierała baza danych, jest określane na etapie jej **projektowania**. W trakcie projektowania bazy danych za pomocą dwuwymiarowych tabel opisujemy wybrany fragment rzeczywistości (bank, szkoła, kolekcja płyt). Teraz omówimy niektóre aspekty i zasady projektowania baz danych, zdając sobie sprawę, że proces projektowania baz jest daleko bardziej złożony.

Zanim przystąpimy do projektowania musi zostać określona dziedzina, dla której tworzymy bazę danych. Zakładamy, że chcemy zaprojektować bazę danych umożliwiającą rejestrowanie ocen wystawianych uczniom, czyli pewien fragment szkolnej rzeczywistości. Jednym z podstawowych zadań, na etapie projektowania bazy danych, jest określenie podstawowych obiektów występujących w dziedzinie, dla której te bazę projektujemy. W naszym przypadku dość szybko dojdziemy do wniosku, że nasza baza powinna opisywać uczniów oraz podstawową jednostkę organizacyjną szkoły czyli klasy. Poniżej przedstawiamy propozycję tabel opisujących te dwa podstawowe elementy.





Rysunek 1.
Przykładowe tabele

Wyjaśnimy teraz niektóre elementy zaproponowanego rozwiązania.

- Każda tabela musi mieć przypisaną nazwę i nazwa ta powinna określać rodzaj danych, jaki planujemy w tej tabeli przechowywać.
- Zgodnie z cechami modelu relacyjnego, każda tabela musi zawierać klucz podstawowy. Dla zaproponowanych tabel kluczami podstawowymi są kolumny o nazwach **iducznia** i **idklasy** – na pierwszy rzut oka nazwy te wydają się dziwne, ale odpowiada to pewnej przyjętej praktyce polegającej na tym, że projektując tabele ustala się tzw. **sztuczny klucz podstawowy**. W naszym przypadku kolumna **iducznia** (identyfikator ucznia) będzie zawierała unikatowe numery przypisywane uczniom, którzy zostaną zapisani w tabeli. W bazach danych istnieją mechanizmy, które automatycznie generują unikatowe numery dla tak zdefiniowanych kluczy.
- W tabeli **uczniowie** znajduje się kolumna o nazwie **idklasy** (na rys. 1 nazwana kluczem obcym). Wymaga to wyjaśnienia tym bardziej, że dotykamy istoty projektowania relacyjnych baz danych. Sprawą oczywistą jest to, że każdy uczeń jest przypisany do określonej klasy. Klasy, jako takie, są zapisywane w oddzielnej tabeli o nazwie **klasy**, w której kluczem podstawowym jest kolumna **idklasy**. Fakt umieszczenia w tabeli **Uczniowie** klucza obcego (czyli kolumny **idklasy**, która w innej tabeli pełni rolę klucza podstawowego) tworzy związek (powiązanie) pomiędzy tabelami **Uczniowie** i **Klasy**. Spróbujmy wyjaśnić dlaczego w ten sposób projektuje się elementy relacyjnych baz danych.
 - Faktem jest, że każdy uczeń jest przypisany do określonej klasy i teoretycznie moglibyśmy umieścić w tabeli **uczniowie** dodatkowe kolumny opisujące tę klasę (nazwę klasy i rok szkolny), ale w tej sytuacji dla uczniów tej samej klasy dane takie musiałyby być powtórzone, czyli to samo byłoby zapisywane w tabeli wielokrotnie. Sytuacja taka sprzyja powstawaniu błędów i niejednoznaczności, których przyczyną mogą być zwykłe błędy (literówki) na etapie zapisywania danych do tabeli.
 - Zapisując dane o klasach w osobnej tabeli zapewniamy, że dana klasa opisana jest tylko raz.
 - Umieszczenie w tabeli **uczniowie** klucza obcego **idklasy** zapewnia powiązanie danych o uczniu z danymi o klasie do której został dowiązany.
 - Jeżeli w pewnym wierszu tabeli **uczniowie** mamy zapisane dane ucznia i przykładowo w kolumnie **idklasy** zapisana jest liczba 5, to taki zapis interpretujemy w ten sposób: uczeń związany jest z klasą (zapisaną w tabeli **klasy**) o wartości klucza 5. Ponieważ **idklasy** w tabeli **klasy** jest kluczem podstawowym to mamy gwarancję, że nasza przykładowa wartość 5, odpowiada dokładnie jednemu wierszowi tabeli **Klasy** zawierającemu interesujący nas opis.

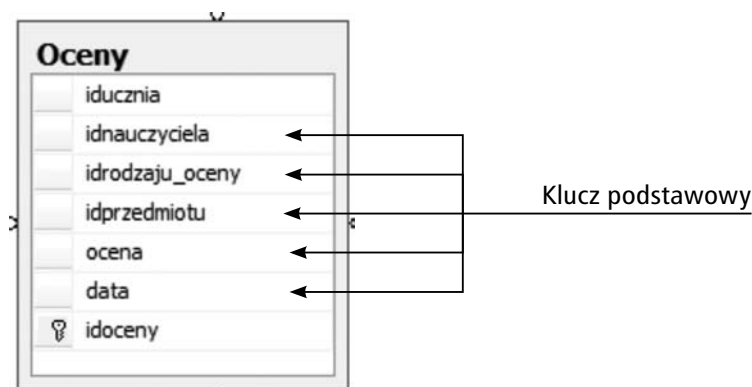
Kontynuujemy nasz projekt i kolejnym elementem może być tabela opisująca nauczycieli, ponieważ nie można wyobrazić sobie procesu wystawiania ocen bez wiedzy o nauczycielu, który taką ocenę wystawił. Proponowana tabela **nauczyciele** nie wprowadza żadnych nowych elementów do rozważań.



Rysunek 2.
Schematy tabel

Kilka słów wyjaśnienia przy propozycji kolejnych tabel w naszym projekcie. Zaproponowane tabele o nazwach **przedmioty** i **rodzaje_ocen** są tak zwanymi **tabelami słownikowymi**, czyli takimi, które będą przechowywać zbiory pewnych pojęć. Cel, dla którego projektujemy tego typu tabele wydaje się oczywisty – będziemy wykorzystywać klucze podstawowe z tych tabel jako klucze obce w innych tabelach, zawierających informacje o przedmiocie lub rodzaju wystawionej oceny i. podobnie jak we wcześniej opisywanym przypadku (**uczniowie** i **klasy**), dzięki takiemu podejściu zapewnimy jednoznaczność zapisywanych danych.

Na koniec tabela, która jest najważniejsza z punktu widzenia zaplanowanej bazy danych, czyli tabela w której będziemy przechowywali dane opisujące wystawione oceny.



Rysunek 3.
Tabela Oceny

W tabeli **Oceny** są przechowywane dane o pewnych zdarzeniach – wystawionych ocenach. Należy się spodziewać, że ta tabela będzie centralnym punktem naszej bazy danych. Zawartość tej tabeli można opisać następująco:

Pewien uczeń (**iducznia**) od jakiegoś nauczyciela (**idnauczyciela**) otrzymał pewien rodzaj oceny (**idrodzaju_oceny**) z pewnego przedmiotu (**idprzedmiotu**) o wartości oceny (**ocena**) wystawionej pewnego dnia (**data**). Dodatkowo, w tabeli **Oceny** jest kolumna **idoceny**, czyli sztuczny klucz podstawowy, który już poznaliśmy.

Na zakończenie tej części rozważań przedstawmy w całości nasz projekt bazy danych – patrz rys. 4.

BAZA DANYCH „ELEKTRONICZNY DZIENNIK OCEN”

Możemy uznać, że tak zaprojektowana baza danych będzie spełniać rolę miejsca, w którym gromadzone będą dane o wystawianych ocenach. Nic nie stoi na przeszkodzie, aby gromadzić w niej dane o wszystkich ocenach wystawianych w danej szkole. Rodzi się jednak pytanie – a jeśli chcielibyśmy zapisywać w takiej bazie danych oceny wystawiane w różnych szkołach. Czy taki projekt bazy byłby wystarczający? Jak należałoby zmodyfikować ten projekt, aby umożliwić zapisywanie ocen wystawianych w różnych szkołach?

Realizacja bazy ocen nie jest wystarczającym doświadczeniem, by móc uznać, że jesteśmy już projektantami baz danych. Problemy związane z projektowaniem baz są dalece bardziej złożone i wymagają oprócz wiedzy bardzo dużego doświadczenia. Zrobiliśmy tylko pierwszy krok.



Rysunek 4. Przykładowy schemat bazy danych

Ćwiczenie 1. Wykonanie własnego projektu bazy danych

W ramach ćwiczenia każdy uczestnik przygotuje projekt bazy danych składający się z około 5 tabel. Dziedzina problemu dowolna. Po wykonaniu projektu należy skonsultować go z prowadzącym ćwiczenia. Projekt powinien zawierać:

- Nazwy tabel
- Nazwy kolumn i określenie typu wartości
- Zaznaczenie kluczy podstawowych i obcych

W dalszej części kursu każdy uczestnik będzie implementował bazę danych na podstawie swojego projektu.

3. TECHNOLOGIA MS SQL SERVER 2008

ELEMENTY TECHNOLOGII MS SQL SERVER 2008

Dotychczasowe rozważania prowadziliśmy w oderwaniu od technologii, czyli od sposób realizacji. Koncentrowaliśmy się na teorii, a teraz przyszła pora na praktykę. W tym celu zdefiniujemy nowe pojęcie – **Systemem Zarządzania Bazami Danych (SZBD)** nazywamy specjalistyczne oprogramowanie umożliwiające tworzenie baz danych oraz ich eksploatację.

Wydaje się oczywiste, że tworzenie i działanie baz danych musi być wspierane przez specjalistyczne oprogramowanie, które powinno umożliwiać realizację pewnych zadań:

- definiowanie obiektów bazy danych,
- manipulowanie danymi,
- generowanie zapytań,
- zapewnienie spójności i integralności danych.

Zadania te brzmią bardzo ogólnie, obejmują jednak większość potrzeb w zakresie tworzenia i eksploatacji baz danych. Dla przybliżenia pojęcia SZBD można podać kilka nazw handlowych, pod jakimi te produkty można spotkać na rynku i w zastosowaniach: MS SQL Server 2008, Oracle, MySQL, Access, DB2 i wiele, wiele innych mniej lub bardziej popularnych.

Jednym z najważniejszych zadań stojących przed SZBD jest zapewnienie **spójności i integralności danych**, czyli dostarczenie mechanizmów zapewniających przestrzeganie określonych reguł przez dane. SZBD dostarczają mechanizmów służących do zapewnienia spójności i integralności danych, czyli mówiąc innymi słowami, do zapewnienia logicznej poprawności danych zapisanych w bazie. Podstawowe mechanizmy realizujące te zadania to:

- deklaracja typu,
- definicje kluczy,
- reguły poprawności dla kolumny,
- reguły poprawności dla wiersza,
- reguły integralności referencyjnej

W ramach kursu będziemy wykorzystywać technologię MS SQL Server 2008 Express. Jest to jeden z najpopularniejszych serwerów baz danych. Edycja SQL Server Express, z której będziemy korzystać, jest wersją darmową z możliwością wykorzystania jej w celach komercyjnych. Technologia SQL Server 2008 zawiera następujące podsystemy:

- Serwer bazy danych (*Database Engine*) – podsystem odpowiedzialny za zarządzanie bazami danych (definiowanie, eksploatacja i administracja baz danych).
- Serwer raportowania (*Reporting Services*) – podsystem umożliwiający zarządzanie procesem tworzenia i dystrybucji raportów generowanych na podstawie danych z różnych źródeł (bazy danych, pliki Excel, pliki tekstowe, dokumenty XML).
- Serwer usług analitycznych (*Analysis Services*) – podsystem wspomagający organizację hurtowni danych, wielowymiarowych kostek analitycznych, tworzenie pulpitów menadżerskich oraz realizację algorytmów wyszukiwania złożonych zależności (*Data Mining*).
- Serwer usług integracyjnych (*Integration Services*) – podsystem realizujący zadania integracji danych, polegające, w dużym uproszczeniu, na pobieraniu danych z pewnych źródeł danych, poddanie ich procesowi przetwarzania (sprawdzanie poprawności, eliminowanie błędów itp.) a następnie zapisanie przetworzonych danych w docelowej lokalizacji. Zadania te są określane w teorii jako platforma ET&L (Extract, Transform and Load).

W ramach kursu będziemy wykorzystywać jedynie serwer baz danych, który zawiera również wiele dodatkowych technologii:

- Usługi asynchronicznego przetwarzania (*Service Broker*) – umożliwiają realizację asynchronicznego przetwarzania z wykorzystaniem kolejek.
- Usługi replikacji danych – umożliwiają konfigurowanie zadań związanych z odtwarzaniem części zasobów bazy danych w innych lokalizacjach.
- Usługi wyszukiwania pełno tekstowego – umożliwiają wyszukiwanie fragmentów tekstu niezależnie od ich lokalizacji w bazie danych (klasyczne zapytanie SQL wymagają określenia tabel, z których dane są pobierane).

Wymienione zostały niektóre elementy technologii MS SQL Server 2008 co i tak pokazuje, że jest to bardzo rozległy i złożony system umożliwiający realizację bardzo różnych zadań związanych z bazami danych. Nasz kurs należy traktować jako pierwszy krok w złożony i bardzo ciekawy świat technologii MS SQL Server 2008.

Ćwiczenie 2. Instalacja instancji MS SQL Server 2008 Express.

W ramach ćwiczenia zostanie omówiony proces instalacji SQL Server 2008 oraz każdy uczestnik kursu wykona instalację serwera na swoim komputerze.

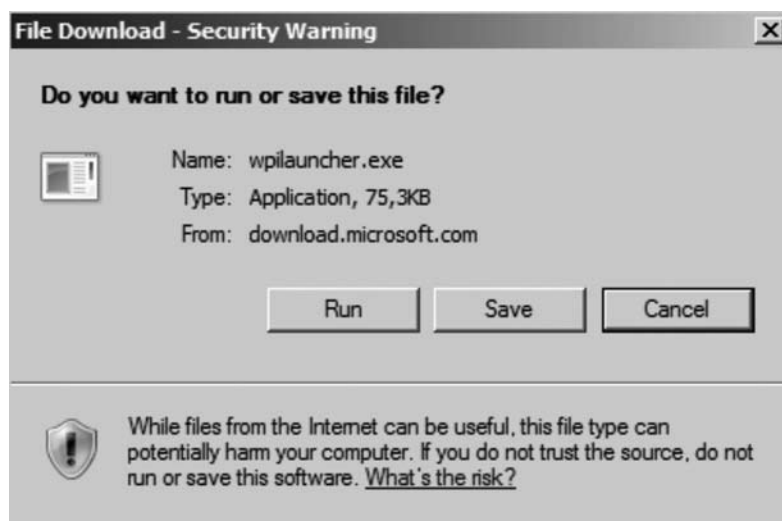
Obok komercyjnych wersji, SQL Server występuje także w darmowym wariantcie zwanym SQL Server 2008 Express. Jego instalacja jest jednak, podobnie jak pozostałych wersji, dość złożona i można popełnić przy niej wiele błędów. Aby się tego ustrzec i móc jak najszybciej zacząć wykorzystywać możliwości oferowane przez SQL Server 2008, podamy w miarę prosty i łatwy sposób zainstalowania tego systemu.



PRZYGOTOWANIE DO INSTALACJI I INSTALACJA

Aby zainstalować SQL Server 2008 Express wraz z narzędziami do zarządzania (SQL Server Management Studio) w możliwie prosty sposób, można ściągnąć ze strony WWW producenta pakiet Microsoft Web Platform Installer – <http://www.microsoft.com/web/downloads/platform.aspx> .

Jest to bardzo wygodne narzędzie nie tylko do instalowania systemu SQL Server 2008 Express, ale także wielu innych komponentów stosowanych do budowania rozwiązań opartych na .NET Framework 3.5 SP1. Zaczniemy więc od pobrania pakietu Microsoft Web Platform Installer. Można od razu wybrać uruchomienie pakietu bezpośrednio po ściągnięciu – patrz rys. 5.



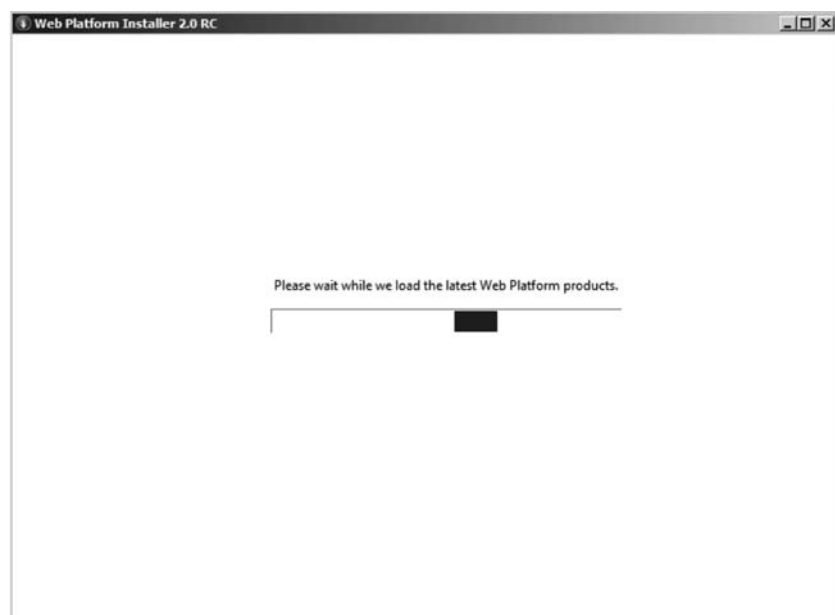
Rysunek 5. Uruchomieniu procesu pobierania pakietu Microsoft Web Platform Installer

Kolejnym krokiem jest wyrażenie zgody na instalację programu pochodzącego od określonego dostawcy (Microsoft) – rys. 6.



Rysunek 6. Uruchomienie instalacji pakietu

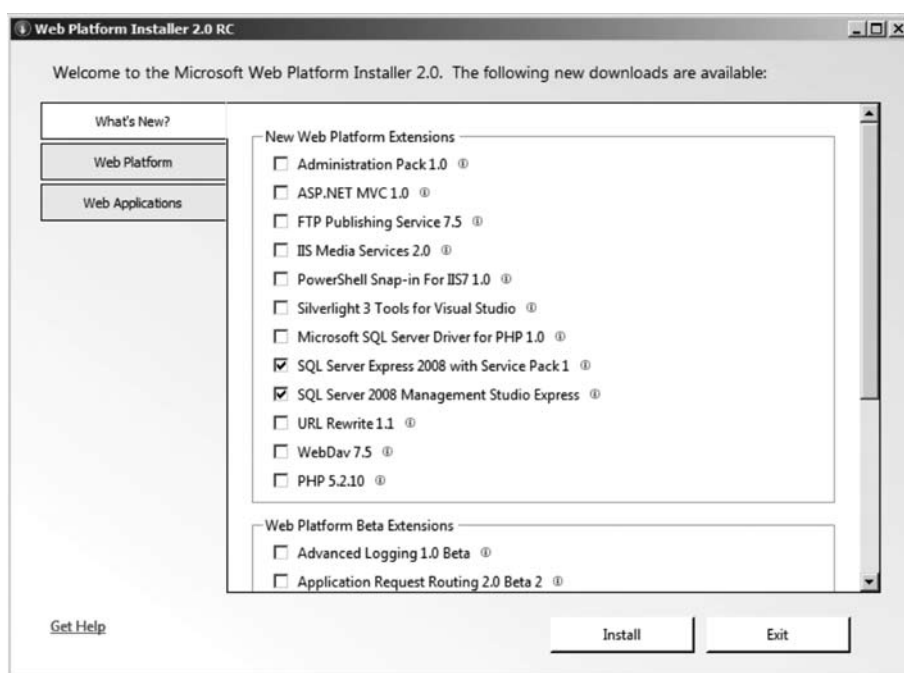
Po chwili pakiet zostaje zainstalowany i rozpoczyna się sprawdzanie najnowszych dostępnych wersji oprogramowania, które można instalować za jego pomocą. Ten proces będzie wykonywany przy każdym kolejnym uruchomieniu tego narzędzia i dzięki temu gwarantuje nam, że instalujemy zawsze aktualne oprogramowanie.



Rysunek 7.

Postęp procesu instalacji

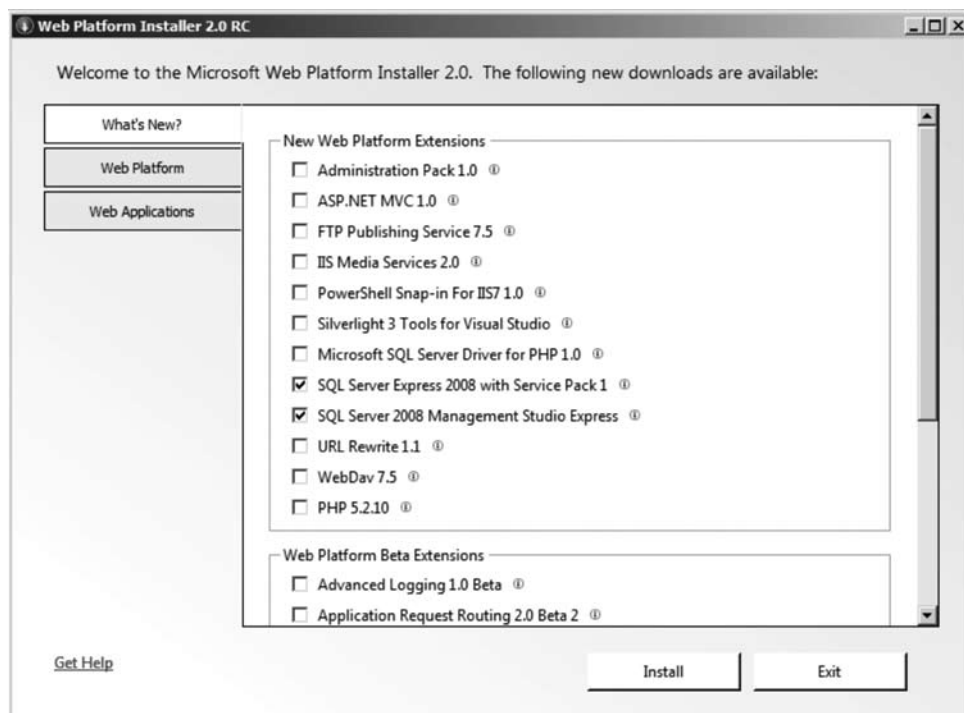
Gdy tylko ten proces się zakończy – rys. 7, będziemy mogli przejrzeć listę dostępnych komponentów oraz wybrać te, które mają zostać zainstalowane. Dla naszych potrzeb są to: SQL Server Express 2008 oraz SQL Server Management Studio Express – rys. 8.



Rysunek 8.

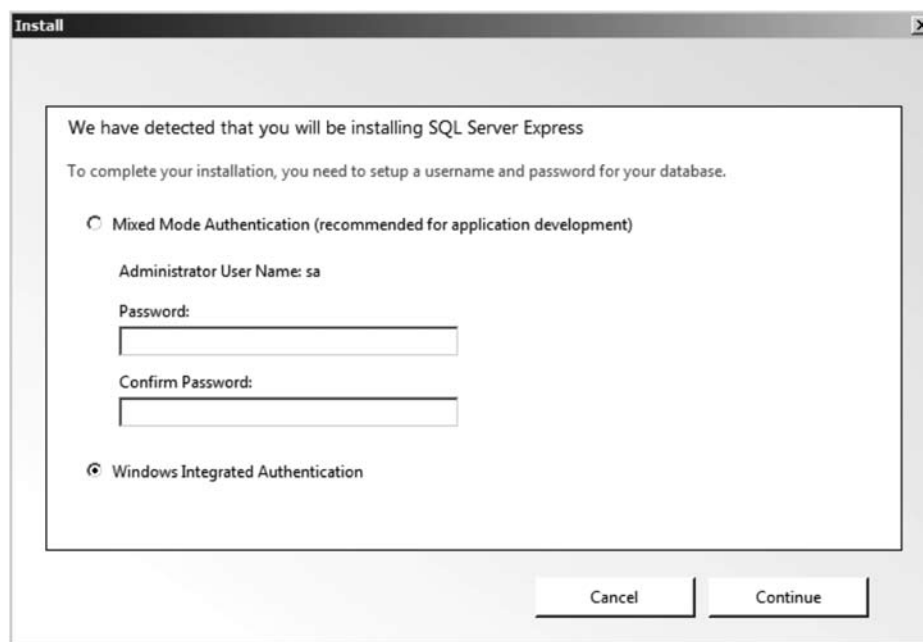
Wybór elementów do instalacji

Po zaakceptowaniu wyboru przez kliknięcie przycisku Install zostaniemy poinformowani, jakie komponenty zostaną pobrane z Internetu i jaka jest ich wielkość. Nie należy się dziwić, że lista ta może zawierać więcej opcji niż zaznaczyliśmy – będą to tzw. **zależności** czyli **komponenty**, bez których działanie wybranych przez nas opcji nie jest możliwe. Zawartość tej listy może się zmieniać w zależności od konfiguracji komputera i zainstalowanych wcześniej programów – patrz rys. 9.



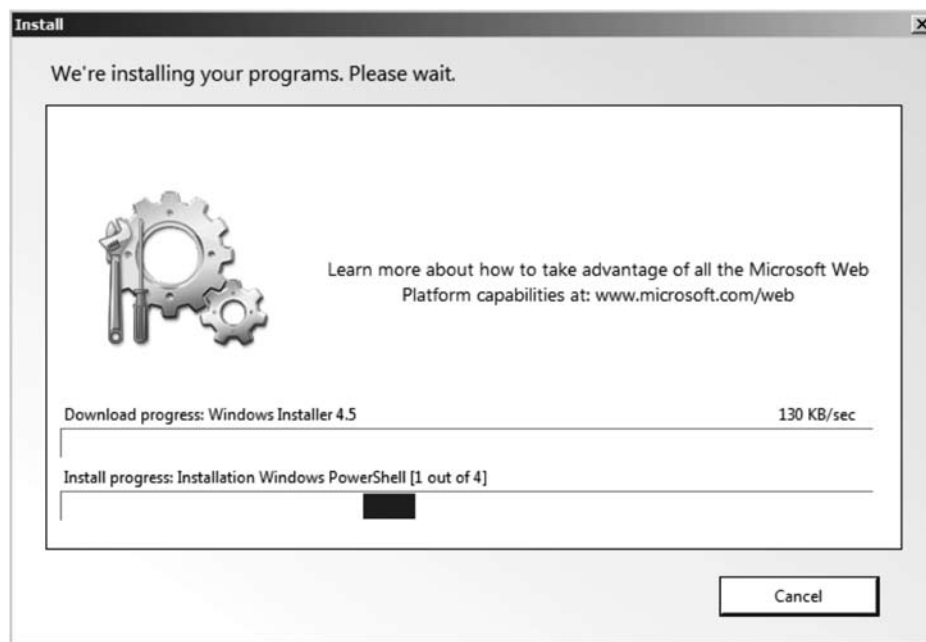
Rysunek 9.
Potwierdzenie zgody na warunki licencji oprogramowania

Zgadając się na warunki licencji przechodzimy do dalszego etapu instalacji – skonfigurowania trybu uwierzytelniania dla SQL Servera. Mamy do wyboru dwie opcje: Mixed Mode oraz Windows Integrated Security. Wybierzmy tę drugą (patrz rys. 10) – nasz serwer będzie przy uwierzytelnianiu polegał na mechanizmach systemu Windows – jeśli ktoś zaloguje się do komputera i nadamy mu uprawnienia do korzystania z serwera baz danych, to uzyska taki dostęp. Domyślnie taki dostęp będzie miał użytkownik, na którego koncie instalujemy oprogramowanie. W trybie Mixed Mode możliwe będzie dodatkowo tworzenie loginów i haseł (na poziomie SQL Server), które także będą umożliwiały uzyskiwanie dostępu do baz danych.



Rysunek 10.
Wybór trybu uwierzytelniania

Na tym kończy się konfigurowanie naszej instancji SQL Servera. Web Platform Installer rozpoczyna teraz pobieranie poszczególnych komponentów z Internetu i instalowanie ich (rys. 11).

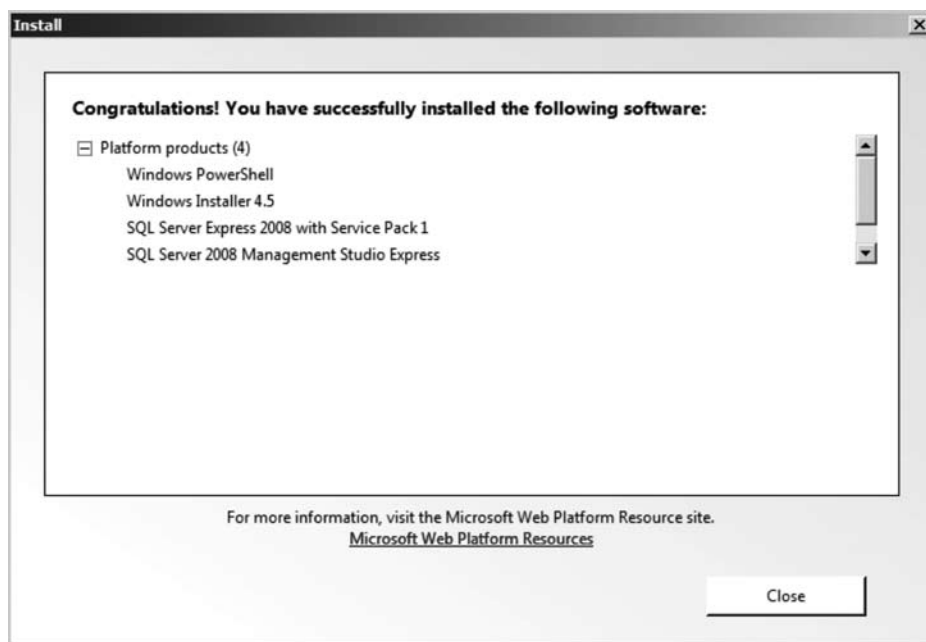


Rysunek 11.

Postęp procesu instalacji komponentów

W trakcie tego procesu może się zdarzyć, że któryś z instalowanych komponentów będzie wymagał restartu systemu. Należy się wówczas na to zgodzić, a po restarcie i zalogowaniu się proces instalacji będzie kontynuowany automatycznie.

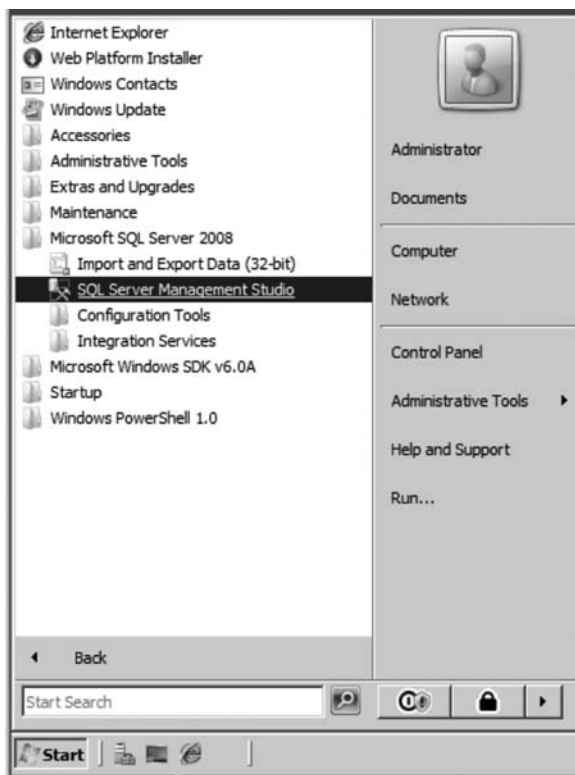
Instalowane będą kolejne komponenty, aż do ostatniego. Po zainstalowaniu wszystkich wybranych przez nas komponentów wyświetlone zostanie podsumowanie instalacji (rys. 12). Po zapoznaniu się z nim możemy zakończyć proces instalacji klikając przycisk Close.



Rysunek 12.

Zakończenie procesu instalacji

Teraz możemy zweryfikować poprawność instalacji praktycznie. Sama usługa SQL Servera (Database Engine) jest skonfigurowana tak, aby uruchamiała się automatycznie wraz ze startem systemu operacyjnego. W menu Start powinna pojawić się grupa Microsoft SQL Server 2008, zawierająca m.in. skrót do narzędzia SQL Server management Studio. Spróbujmy je uruchomić (rys. 13).



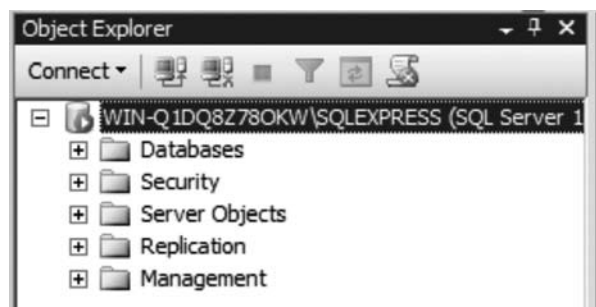
Rysunek 13.
Uruchamianie SQL Server Management Studio

Jeśli wszystko poszło dobrze, powinno uruchomić się właśnie zainstalowane narzędzie i zapytać o serwer baz danych, z którym ma się połączyć, przy czym domyślnie wskazany będzie nasz zainstalowany SQL Server 2008 Express oraz wybrany będzie tryb uwierzytelniania Windows Authentication (rys. 14).



Rysunek 14.
Okno logowania do SQL Server

Po nawiązaniu połączenia w oknie Object Explorera powinniśmy zobaczyć drzewo, za którego pomocą można zarządzać bazami danych jak i samym SQL Serverem (rys. 15).



Rysunek 15.
Object Explorer w SQL Server Management Studio

PROBLEMY PRZY INSTALACJI

Może się zdarzyć, że instalacja przebiegnie tak łatwo, jak przedstawiliśmy to powyżej. Typowym problem, na który możemy natrafić podczas instalacji, jest wykrzyk niezgodności któregoś z instalowanych komponentów z już zainstalowanymi na komputerze. W takim przypadku zwykle wystarcza odinstalowanie istniejącej (starszej) wersji i ponowne uruchomienie programu Web Platform Installer. Należałoby także wspomnieć, że SQL Server Express 2008 nie może być zainstalowany razem z istniejącym SQL Server Express 2005. W takim przypadku trzeba ten wcześniejszy program odinstalować przed rozpoczęciem instalacji wersji 2008.

W przypadku pojawienia się innych błędów przy instalacji nie należy wpadać w panikę. Lepiej poszukać w Internecie informacji, czy ktoś się już spotkał z takim błędem i czy ewentualnie znalazł jego rozwiązanie. Zwykle odnalezienie takich wskazówek nie jest zbyt trudne ani czasochłonne, więc w większości przypadków instalacja kończy się sukcesem.

4. ŚRODOWISKO SQL SERVER MANAGEMENT STUDIO

Ćwiczenie 3. Zapoznanie ze środowiskiem SQL Management Studio.

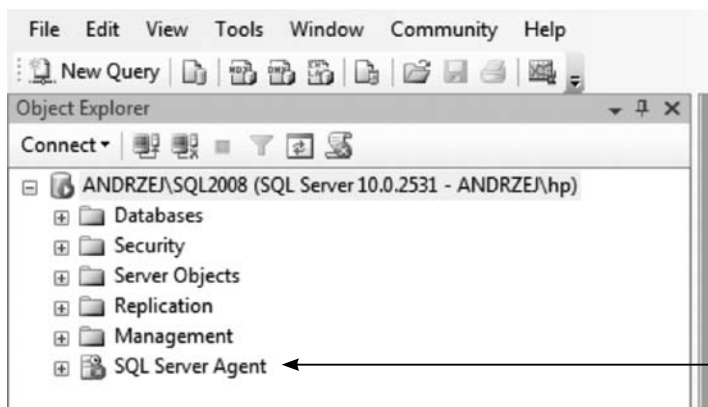
Wspólnie z prowadzącym kurs rozpoznajemy środowisko SZBD MS SQL Server 2008. Korzystanie z tego systemu umożliwia specjalne oprogramowanie SQL Server Management Studio.

1. Uruchamiamy SQL Server Management Studio (lokalizację programu poda prowadzący kurs).
2. Po uruchomieniu programu przechodzimy do logowania się do SQL Servera, na ekranie pojawi się okienko do logowania, w którym wpisujemy w pola wartości takie, jak podane na rys. 16 (w polu Server name wybieramy nazwę serwera, który został zainstalowany w ramach ćwic. 2).



Rysunek 16.
Okienko logowania do SQL Server

3. Po poprawnym wpisaniu powyższych wartości uruchomione zostanie oprogramowanie i pojawia się okna SQL Server Management Studio (rys. 17).
4. Wspólnie z prowadzącym kurs poznajemy wybrane elementy środowiska SQL Servera.



W oknie Object Explorer uzyskujemy dostęp do zarządzania obiektami zdefiniowanymi w SQL Server

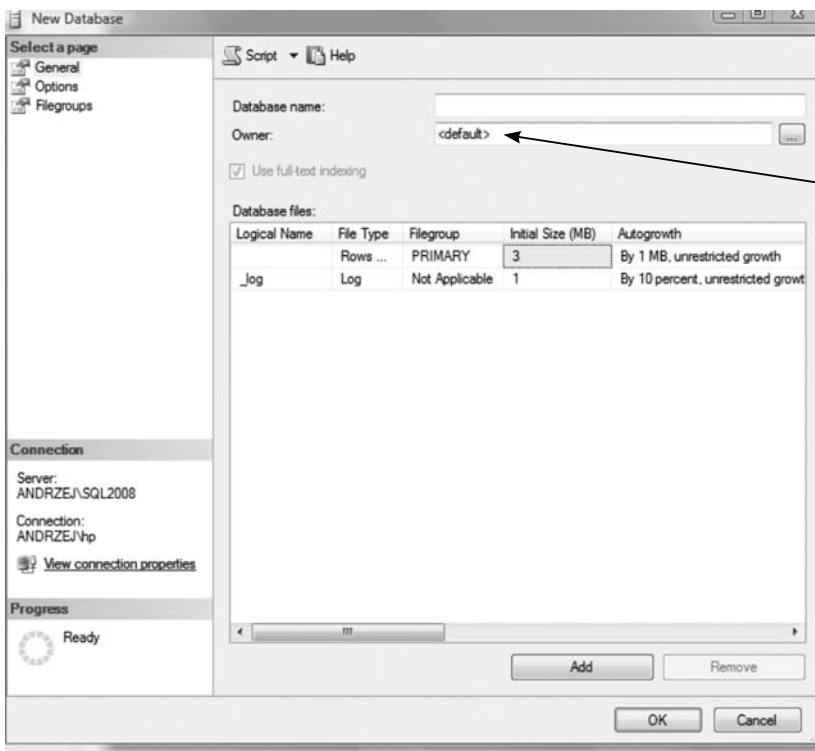
Rysunek 17.
Widok Object Explorer

5. IMPLEMENTACJA ZAPROJEKTOWANEJ BAZY DANYCH

Ćwiczenie 4. Tworzenie bazy danych.

W trakcie tego ćwiczenia każdy uczestnik warsztatów utworzy swoją pierwszą bazę danych, którą zaprojektował w ramach ćwicz. 1. W tym celu wykonujemy poniższe czynności.

1. W oknie Object Explorer wybieramy folder Databases.
2. Po kliknięciu prawym klawiszem myszy pojawia się menu, z którego wybieramy opcję New Database.
3. Po wybraniu opcji New Databases pojawia się okno definiowania bazy danych (rys. 18).



W okno Database name wpisujemy nazwę którą chcemy

Rysunek 18.
Okno definiowania bazy danych

Przez proces tworzenia nowej bazy danych poprowadzi prowadzący kurs. Jak widać, zainicjowanie tworzenia nowej bazy danych nie jest zbyt złożone (w standardowej wersji). Po utworzeniu nowej bazy możemy przystąpić do definiowania tabel.

Ćwiczenie 5. Definiowanie tabel.

W tabelach relacyjnych są przechowywane dane różnego typu (liczby, teksty, znaki, daty ...). Z cech modelu relacyjnego wynika, że każda kolumna w tabeli musi mieć określony typ przechowywanych danych. Deklaracja typu jest pierwszym sposobem zapewnienia poprawności danych – w ujęciu matematycznym jest to określenie dziedziny wartości dla kolumny. SZBD udostępniają zbiór typów, które mogą być wykorzystane w definicji kolumn.

PRZYKŁADOWE TYPY DANYCH W SQL SERVER 2008

Dla danych znakowych

- **char(n)** – ciąg n znaków o stałej długości, np. jeżeli kolumna ma określony typ **char(25)** a wpisujemy słowo kot, to i tak zostanie ono zapisane za pomocą 25 znaków, uzupełnione spacjami;
- **varchar(n)** – ciąg n znaków o zmiennej długości, np. jeżeli kolumna ma określony typ **varchar(25)** i wpisujemy słowo kot, to zostanie ono zapisane za pomocą 3 znaków;
- **varchar(max)** – ciąg znaków o zmiennej długości do 2 GB;

W tym miejscu można spróbować odpowiedzieć na następujące pytanie: Skoro typ **char** w porównaniu z **varchar** wykorzystuje więcej pamięci do zapisywania danych (uzupełnianie spacjami), to jakie korzyści możemy osiągnąć w przypadku wykorzystania typu **char**?

Dla danych liczbowych – liczby całkowite

- **tinyint** – liczba całkowita z zakresu [0, 255], przechowywana w jednym bajcie;
- **smallint** – liczba całkowita z zakresu [–32768, 32767], przechowywana na dwóch bajtach;
- **int** – liczba całkowita z zakresu [–2147483648, 2147483647], przechowywana na czterech 4 bajtach;
- **bigint** – liczba całkowita z zakresu [–9223372036854775808, 9223372036854775807], przechowywana na ośmiu bajtach.

Dla danych liczbowych – liczby z ułamkiem

- **real, float** – do zapisywania liczb zmiennopozycyjnych;
- **decimal, numeric** – do zapisywania liczb zmiennopozycyjnych o określonej precyzji;
- **money** – do zapisywania liczb wyrażających kwoty pieniężne.

Dla danych– daty i czasu

- **date** – do zapisywania dat, np. 2009-08-22;
- **time** – do zapisywania czasu, np. 19:22:07.2345644;
- **datetime** – do zapisywania łącznie daty i czasu, np. 2009-08-22 19:22:07.2345644.

Typy różne

- **bit** – do zapisywania wartości logicznych (true, false lub 0,1);
- **varbinary(n)** – do zapisywania danych binarnych o długości n bajtów;
- **varbinary(max)** – do zapisywania danych binarnych o długości do 2 GB (np. obrazy, dźwięki)
- **Timestamp** – specjalny znacznik który automatycznie zmienia swoją wartość przy modyfikacji wiersza
- **XML** – do zapisywania dokumentów w języku XML o długości do 2 GB

Nie wymieniliśmy wszystkich dostępnych typów danych, widać jednak, że jest dość duży zbiór typów danych i od decyzji projektanta zależy właściwy ich wybór.

Po utworzeniu nowej bazy danych jest ona pusta, czyli nie zawiera tabel (za wyjątkiem tabel systemowych). W ramach tego ćwiczenia, w zainicjowanej w ćwiczeniu 4 bazie danych, zdefiniowane zostaną tabele, które uczestnicy kursu zaprojektowali w ramach ćwiczenia 1. W celu utworzenia nowej tabeli wykonujemy następujące czynności:



1. W oknie Object Explorer rozwijamy folder Databases.
2. Po pojawieniu się listy dostępnych baz danych, wybieramy bazę utworzoną w ćwic. 4 – pojawi się lista elementów, które można definiować (rys. 19).



Rysunek 19.

Lista elementów bazy danych

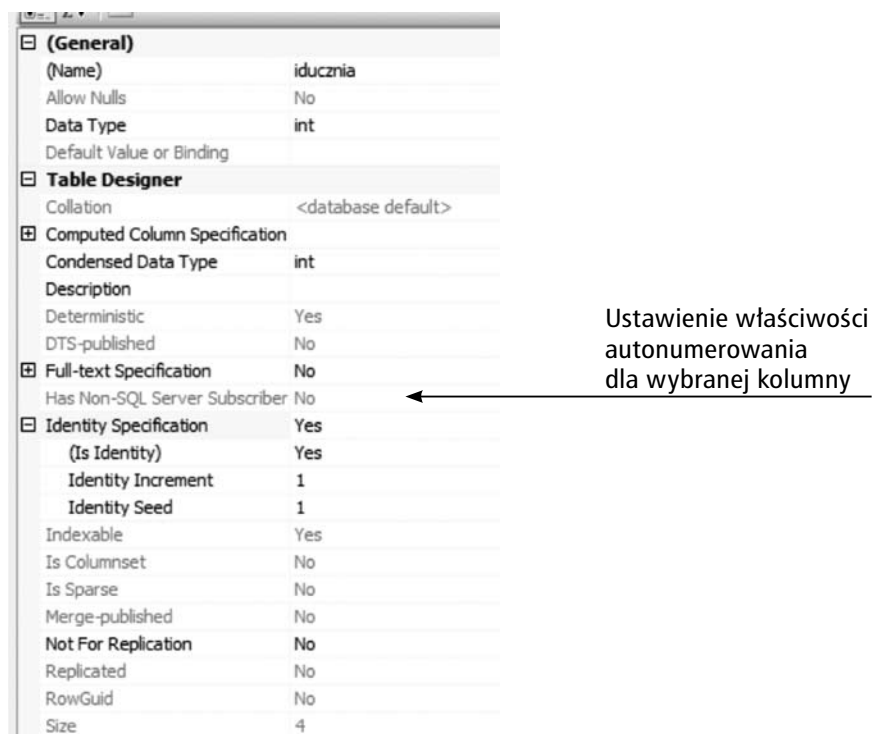
3. Prowadzący wyjaśni krótko widoczne na rys. 19 elementy.
4. Po kliknięciu prawym klawiszem myszy na folderze Tables, pojawi się menu, z którego wybieramy opcję New Table.
5. Pojawi się okno wspomagające proces definiowania tabeli. Utworzenie tabeli polega głównie na zdefiniowaniu poszczególnych kolumn (rys. 20).



Rysunek 20.

Podstawowe elementy definiowane dla kolumny

6. Dla wybranej kolumny możemy definiować dodatkowe właściwości widoczne w oknie Column Properties (rys. 21) – przykładowo dla kolumn klucza podstawowego najczęściej ustawia się właściwość autonumerowania (każdy nowy wiersz ma automatycznie określoną, unikatową wartość).



Rysunek 21.

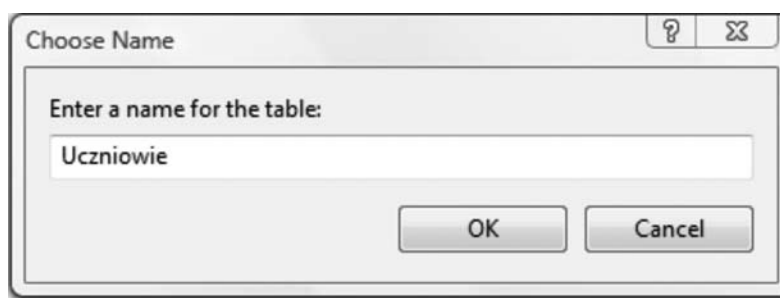
Okno właściwości kolumny

7. Po zakończeniu tworzenia tabeli, okno będzie wyglądało jak na rys. 22 (lista kolumn zależy od definiwanej tabeli).

	Column Name	Data Type	Allow Nul
<input checked="" type="checkbox"/>	iducznia	int	<input type="checkbox"/>
<input type="checkbox"/>	Nazwisko	varchar(50)	<input type="checkbox"/>
<input type="checkbox"/>	Imie	varchar(50)	<input type="checkbox"/>
<input type="checkbox"/>	Pesel	char(11)	<input type="checkbox"/>
<input type="checkbox"/>	dataUrodzenia	date	<input type="checkbox"/>
<input type="checkbox"/>			<input type="checkbox"/>

Rysunek 22.
Okno projektu tabeli

8. Po zakończeniu definiowania kolumn zamykamy okno i nadajemy nazwę tabeli (rys. 23).



Rysunek 23.
Okno wyboru nazwy dla projektowanej tabeli

Opisane czynności wykonujemy będą dla każdej tabeli zaprojektowanej w ramach ćwic. 1.

Ćwiczenie 7. Definiowanie reguł poprawności CHECK.

Omawiana wcześniej deklaracja typu określa dziedzinę wartości dla kolumny, ale często jest to dziedzina zbyt szeroka, czyli nie wszystkie wartości z tak określonej dziedziny możemy uznać za poprawne z punktu widzenia zawartości analizowanej kolumny. Dodatkowym utrudnieniem mogą być także zależności logiczne pomiędzy danymi zapisanymi w różnych kolumnach jednego wiersza. Jeżeli jesteśmy w stanie zdefiniować takie zależności, to możemy skorzystać z mechanizmu, dostarczanego przez SZBD, służącego do definiowania reguł poprawności dla kolumny lub wiersza. Problemy związane z regułami poprawności przeanalizujemy na przykładzie kolumny **pesel** z pokazanej na rysunku 24 przykładowej tabeli.

uczniowie	
<input checked="" type="checkbox"/>	iducznia
<input type="checkbox"/>	nazwisko
<input type="checkbox"/>	imie
<input type="checkbox"/>	data_urodzenia
<input type="checkbox"/>	czy_chlopak
<input type="checkbox"/>	pesel
<input type="checkbox"/>	idklasy

Rysunek 24.
Tabela Uczniowie



Założmy, że dla kolumny *pesel* został zdefiniowany typ danych *char(11)*. Wydaje się, że jest to definicja poprawna, ale rodzi się wiele problemów związanych z zapewnieniem poprawności danych zapisywanych w tej kolumnie, czyli musimy wymusić, żeby każda wartość zapisana w tej kolumnie była poprawnym numerem pesel.

Z punktu widzenia deklaracji typu *char(11)*, poprawnymi wartościami są wszystkie ciągi znakowe o długości nie przekraczającej 11 znaków i w tym momencie widzimy, jak daleko jeszcze do pełnej poprawności. Gdybyśmy poprzestali na takim zdefiniowaniu tej kolumny, to za poprawne dane mogłyby uchodzić nawet tak bezsensowne dane, jak: Ala ma kota, W45991AS, brak Pesel – każdy z tych przykładowych ciągów znakowych jest poprawny, ponieważ nie ma więcej niż 11 znaków.

Zadanie 1. Ograniczyć definicję typu tak, żeby jako poprawne były traktowane tylko ciągi składające się z dokładnie jedenastu znaków.

Można w tym celu skorzystać z mechanizmu definiowania ograniczeń. Ograniczenie dziedziny wartości sprowadza się do zdefiniowania wyrażenia logicznego, które, jeśli jest spełnione, uznaje dane za poprawne. W naszym przypadku takie wyrażenie mogłoby mieć następującą postać:

len(pesel) = 11

Funkcja o nazwie *len*, jako parametr ma ciąg znaków (w naszym przypadku zawartość kolumny *pesel*), a jej wynikiem jest liczba znaków, z których składa się przekazany parametr. W tym miejscu uznajemy, że potrafimy zdefiniować taką regułę i ... to dopiero początek długiej drogi, ponieważ w myśl tej definicji za poprawne zapisy uznane zostaną następujące ciągi znakowe: *aswedfcxsdr* i *a234543234j*, bo zawierają dokładnie 11 znaków.

Zadanie 2. Zapewnić, że dane w kolumnie *pesel* składają się z dokładnie 11 cyfr.

W tej sytuacji zadanie sprowadza się do uściślenia wcześniejszego wyrażenia. Posłużymy się funkcją *isnumeric*, która zwraca 1, jeżeli przekazany jako parametr ciąg znaków daje się poprawnie zamienić na liczbę lub wartość 0 gdy jest to niemożliwe. Nasze wyrażenie logiczne mogłoby mieć teraz następującą postać:

len(pesel) = 11 and isnumeric(pesel)=1

Nasza definicja staje się coraz bardziej złożona, ale po jej określeniu to SZBD będzie sprawdzał i wymuszał poprawność zapisywanych danych. Napracowaliśmy się dużo i ... nagle ołśnienie, że przecież pierwszych 6 cyfr w numerze Pesel także nie może być dowolnych ale musi odpowiadać dacie urodzenia, tym bardziej, że w naszej tabeli obok kolumny *pesel* jest także kolumna *data_urodzenia* i niedopuszczalne byłoby zaniedbanie synchronizacji między tymi danymi.

Zadanie 3. Zapewnić, żeby numer Pesel był zgodny z zapisaną w tym samym wierszu datą urodzenia.

W tym celu rozbudujemy wyrażenie logiczne o kolejny składnik, który będzie odpowiedzialny za sprawdzenie zgodności numeru pesel z datą urodzenia. Po modyfikacji wyrażenie logiczne przyjmie następującą postać:

len(pesel) = 11 and isnumeric(pesel)=1 and convert(varchar(8),DataUrodzenia,12) =substring(pesel,1,6)

Wykorzystane zostały funkcje *convert* i *substring* do zdefiniowania wyrażenia określającego poprawność odwzorowania daty urodzenia w numerze pesel. Na zajęciach sprawdzimy opis i działanie tych funkcji. W tym miejscu chcielibyśmy zwrócić uwagę, że po raz pierwszy wyrażenie odwołuje się do różnych kolumn



tabeli (oczywiście ma to sens w obrębie jednego konkretnego wiersza). Moglibyśmy pewnie już spocząć na laurach, gdyby nie jeszcze jedna informacja mówiąca o tym, że dziesiąta cyfra numeru Pesel także nie jest przypadkowa, gdyż jest to cyfra określająca płeć (parzysta kobiety, nieparzysta mężczyźni) a pamiętamy, że w naszej tabeli jest kolumna o dziwnej nazwie `czy_chlopak`, w której zapisujemy dane określające płeć.

Zadanie 4. Zapewnić sprawdzenie poprawności oznaczenia płci w numerze Pesel.

W tej sytuacji została nam, mamy nadzieję ostatnia modyfikacji naszej reguły poprawności do następującej postaci:

```
len(pesel) = 11 and isnumeric(pesel)=1 and  
convert(varchar(8),DataUrodzenia,12) =substring(pesel,1,6) and  
cast(substring(pesel,10,1) as int) %2=CzyChlopak
```

Proszę przeanalizować postać wyrażenia określającego regułę powiązania zawartości kolumny CzyChlopak (zadeklarowanej jako typ bit) z parzystością dziesiątej cyfry numeru Pesel.

Niezależnie od ostatecznej postaci wyrażenia wymuszającego poprawność, w trakcie omawiania problemów związanych z poprawnością numeru Pesel, wykazaliśmy, że zagadnienia zapewnienia spójności i integralności danych są złożone, ale jednocześnie bardzo istotne, gdyż tylko wymuszenie poprawności danych może gwarantować przydatność całej bazy danych.

Należy dodać, że nie wyczerpaliśmy w omawianym przykładzie wszystkich aspektów problemu, gdyż poprawność numeru Pesel jest także weryfikowana przez sumę kontrolną (ostatnia cyfra numeru Pesel musi być zgodna z wynikiem obliczeń specjalnego algorytmu) oraz dla dat urodzenia z XIX i XXI wieku reguła odwzorowania też jest inna (dodajemy do numeru miesiąca 20 lub 40). Definiowanie reguł poprawności dla danych jest bardzo ważnym elementem implementacji bazy danych.

W celu zdefiniowania reguły poprawności wykonujemy następujące czynności:

1. Zdefiniować przykładowa tabelę pokazana na rysunku 25.

	Column Name	Data Type	Allow Nulls
🔑	iducznia	int	<input type="checkbox"/>
	Nazwisko	varchar(50)	<input type="checkbox"/>
	Imie	varchar(50)	<input type="checkbox"/>
	DataUrodzenia	date	<input type="checkbox"/>
	CzyChlopak	bit	<input type="checkbox"/>
	Pesel	varchar(11)	<input type="checkbox"/>

Rysunek 25.

Przykładowa tabela

2. Kliknąć przycisk Manage Check Constraints



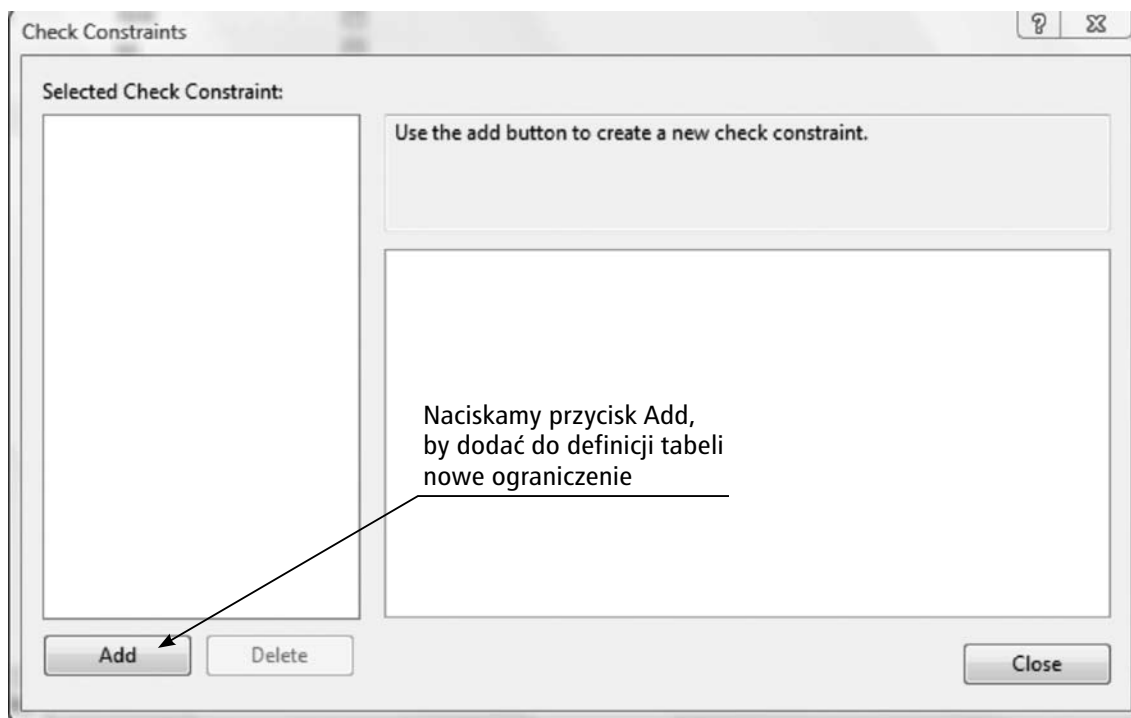
Przycisk Manage Check Constraints

Rysunek 26.

Przyciski Table Designer

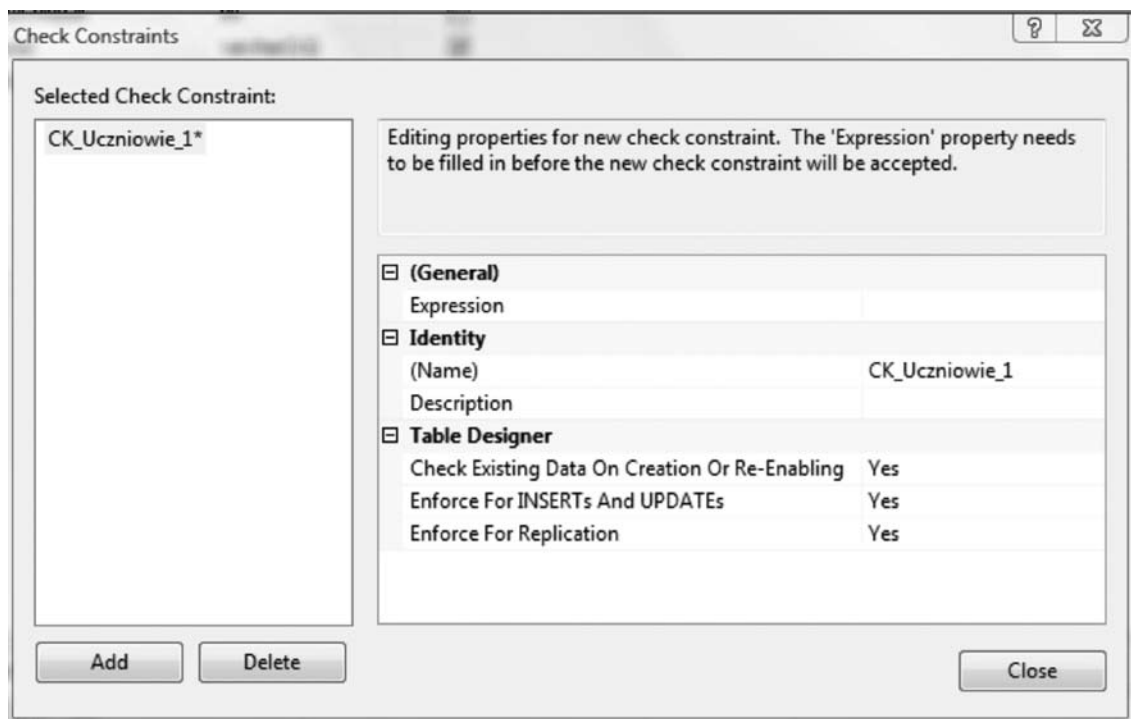
Pojawi się okno umożliwiające definiowanie reguły poprawności rys. 27).





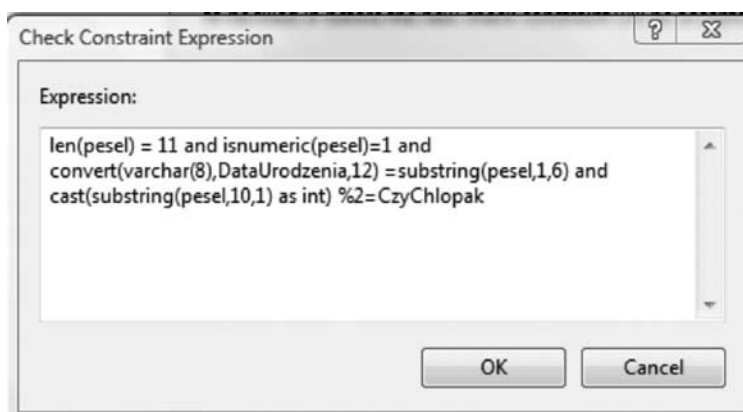
Rysunek 27.
Okno definiowania reguł typu Check

3. Po wciśnięciu przycisku Add możemy przystąpić do definiowania reguły – pojawi się okno umożliwiające określenie nazwy definiowanego ograniczenia oraz wpisanie wyrażenia będącego tą regułą (rys. 28).



Rysunek 28.
Okno definiowania reguły Check po dodaniu nowej reguły

4. Po uaktywnieniu okienka Expression, przechodzimy do wpisania wyrażenia (rys. 29).



Rysunek 29.

Okno do wpisywania wyrażenia reguły Check

5. Po zatwierdzeniu reguła zostaje zapisana w bazie danych.
6. Wprowadzamy przykładowe dane do zdefiniowanej tabeli i sprawdzamy działanie reguły.
7. Dla własnego projektu bazy danych proszę zdefiniować przykładowe reguły poprawności (po konsultacji z prowadzącym kurs)

Ćwiczenie 8. Definiowanie reguł integralności referencyjnej.

Przy projektowaniu bazy danych standardową praktyką jest doprowadzenie struktury bazy do tzw. **trzeciej postaci normalnej**. Prowadzi to do powstania większej liczby mniejszych (zawierających mniej kolumn) tabel. Żeby zapewnić spójność danych w takiej sytuacji potrzebny jest specjalny mechanizm, który po zdefiniowaniu reguł powiązań między tabelami będzie pilnował ich spójności. Właśnie do tego celu stworzony został jeden z rodzajów ograniczeń (*constraints*) – **klucz obcy**. Za jego pomocą możemy w wygodny sposób definiować reguły spójności danych pomiędzy tabelami. O tym jest mowa w dalszej części zajęć.

Przy tworzeniu ograniczeń typu klucz obcy należy wspomnieć o kaskadowych więzach integralności referencyjnej. Ich istota polega na zdefiniowaniu, w jaki sposób mają się zachować wiersze (i kolumny) klucza obcego w reakcji na usunięcie (lub zmodyfikowanie) wiersza (lub wartości klucza podstawowego).

Można wybrać jeden z czterech wariantów takiej reakcji dla każdej z dwóch opcji (ON DELETE i ON UPDATE):

- No Action – jest to domyślny wariant, który nie powoduje podjęcia jakichkolwiek działań w związku z usunięciem czy zmodyfikowaniem wiersza, do którego odwołuje się klucz obcy
- Cascade – jak sama nazwa wskazuje, powoduje kaskadowe usunięcie wierszy odwołujących się poprzez klucz obcy do usuwanego wiersza, lub modyfikację wartości kolumn klucza obcego w przypadku zmodyfikowania wartości klucza podstawowego, do którego się on odwołuje. Jest to opcja z jednej strony bardzo wygodna, gdyż zwalnia nas z „ręcznego” usuwania wierszy skojarzonych przez klucz obcy. Z drugiej strony zaś, przy odrobinie pecha można wyczyścić niechcący sporą część danych jednym niewinnym poleceniem usunięcia jednego wiersza z tabeli.
- Set Null – polega na tym, że w przypadku usunięcia wiersza, do którego odwoływał się klucz obcy, kolumnom tegoż klucza jest przypisywana wartość null. Żeby opcja ta zadziałała, kolumny klucza obcego muszą dopuszczać wartość null.
- Set Default – działanie zbliżone do Set null, z tą różnicą, że kolumny klucza obcego są ustawiane na wartość domyślną. Żeby opcja ta zadziałała, kolumny klucza muszą mieć określoną wartość domyślną lub dopuszczać wartość null (na którą będą ustawione w przypadku braku wartości domyślnej)

Warto wspomnieć o jeszcze kilku cechach kluczy obcych. Po pierwsze, klucz obcy nie musi być pojedynczą kolumną. Można go zdefiniować na kilku kolumnach, lecz trzeba wtedy pamiętać, że jeżeli te kolumny dopuszczają wartość null, a którakolwiek z nich będzie miała tę wartość, to pozostałe nie będą sprawdzane pod kątem zgodności z regułą klucza.




Klucz obcy nie musi odwoływać się do kolumny lub kolumn z innej tabeli. W pewnych przypadkach tworzy się klucze obce odwołujące się do kolumn w tej samej tabeli. Mówimy wtedy o samozłączeniu. Takie rozwiązanie bywa stosowane na przykład do budowania danych o strukturze hierarchicznej. Przykładowo weźmy tabelę pracownicy, która ma kolumny:

- PracownikID (klucz podstawowy),
- Nazwisko
- Imie
- kierownikID (klucz obcy, dopuszczalna wartość null)

Tak prosta struktura tabeli umożliwi zdefiniowanie hierarchii pracowników poprzez określenie, kto jest czym kierownikiem za pomocą wartości kolumny KierownikID, która jest kluczem obcym odwołującym się do kolumny z tej samej tabeli. W takiej hierarchii „szef wszystkich szefów” będzie miał wartość null w kolumnie kierownikID:) Rozwiązania oparte o taki schemat są bardzo łatwe w implementacji, lecz nieco uciążliwe w obsłudze, szczególnie gdy próbujemy się poruszać po hierarchii w poziomie.

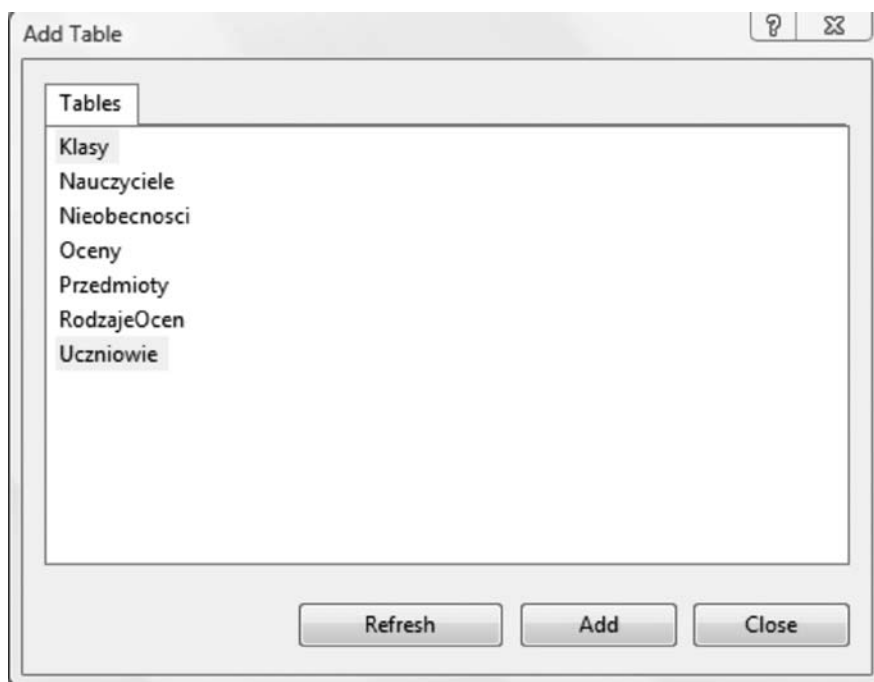
Przystąpimy teraz do praktycznego definiowania reguł integralności referencyjnej. W tym celu wykonujemy następujące czynności:

1. Definiujemy tabelę o nazwie Klasy, jak pokazano na rys. 30.

	Column Name	Data Type	Allow Nulls
	idklasy	int	<input type="checkbox"/>
	Nazwa	varchar(50)	<input type="checkbox"/>
	RokSzkolny	varchar(50)	<input type="checkbox"/>

Rysunek 30.
Schemat tabeli Klasy

2. W tabeli Uczniowie, zdefiniowanej w ramach ćwiczenia 5, dodajemy kolumnę o nazwie idklasy (typ danych integer)
3. Tworzymy diagram bazy danych – w oknie Object Explorer wybieramy folder Database Diagrams, z menu otrzymanego po kliknięciu prawym klawiszem myszy, wybieramy New Database Diagram . Pojawi się okno z listą dostępnych w bazie danych tabel, tak jak pokazano na rys. 31.



Rysunek 31.
Lista tabel w bazie danych

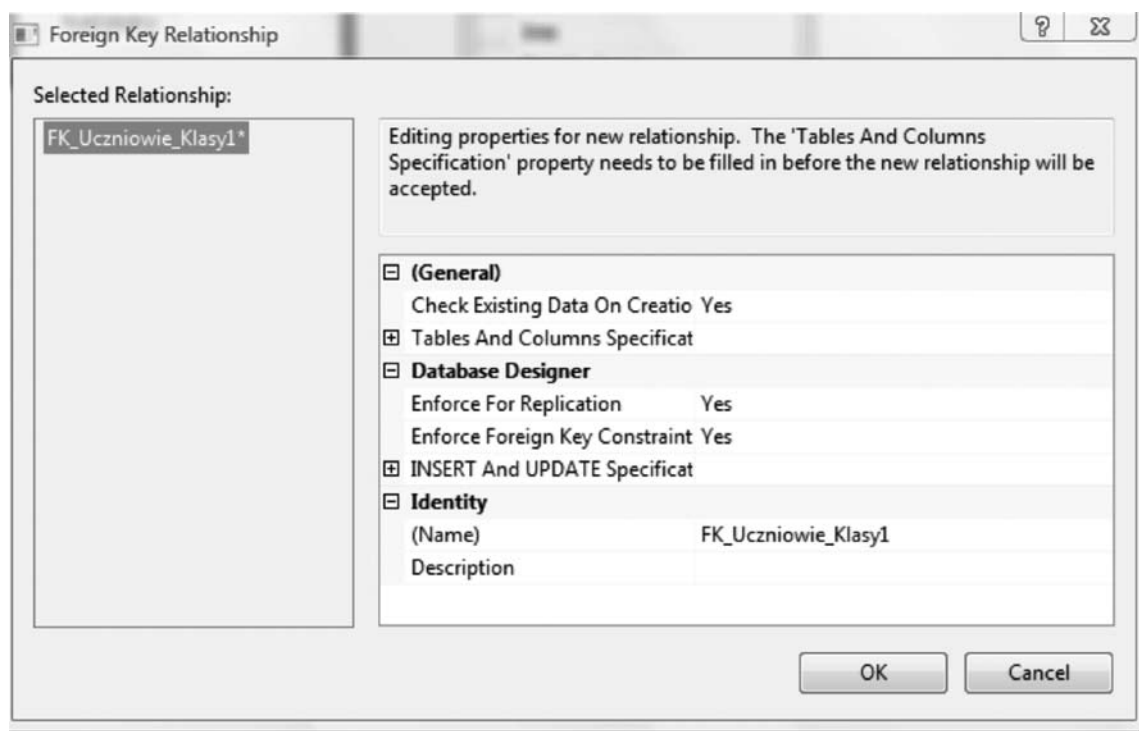
- Wybieramy tabelę Klasy i Uczniowie po czym wciskamy przycisk Add – pojawi się okno projektanta diagramu bazy danych, jak pokazano na rysunku 32.



Rysunek 32.

Okno definiowania diagramu bazy danych

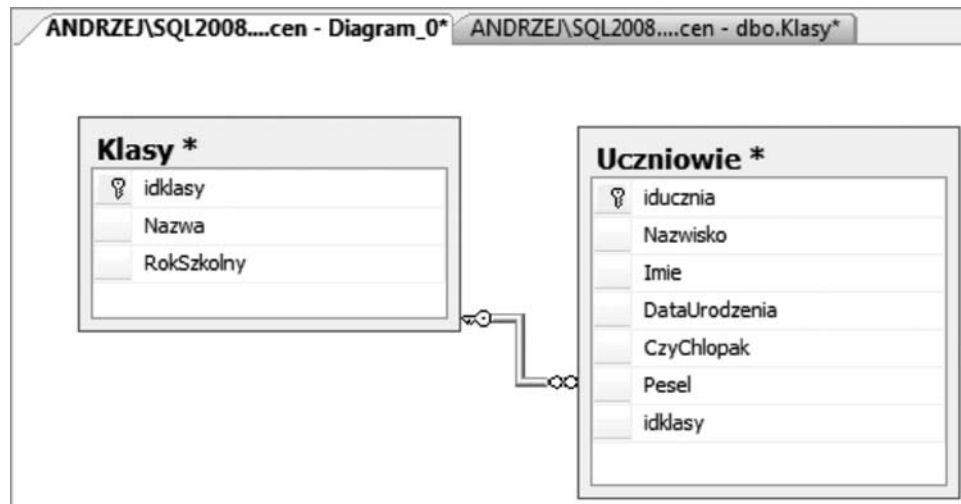
- Przeciągamy kolumnę idklasy z tabeli Klasy do kolumny idklasy w tabeli Uczniowie – pojawi się okno definiowania reguły integralności referencyjnej jak pokazano na rys. 33.



Rysunek 33.

Okno definiowania właściwości reguły klucza obcego

- Po zatwierdzeniu otrzymamy obraz połączonych tabel – jak pokazano na rys. 34.



Rysunek 34.
 Obraz połączonych tabel w oknie diagramu

Po zamknięciu okna (pojawi się żądania nadania nazwy tworzonemu diagramowi) reguła zostaje zdefiniowana w bazie danych.

7. Sprawdzamy działanie reguły integralności referencyjnej w trakcie wprowadzania do omawianych tabel przykładowych danych.

6. PODSTAWY JĘZYKA SQL

6.1. WPROWADZENIE DO JĘZYKA SQL

Historia relacyjnego modelu danych rozpoczęła się w roku 1970 wraz z publikacją E.F.Codda *A Relational Model of Data for Large Shared Data Banks*. (pol. *Relacyjny model danych dla dużych współdzielonych banków danych*). Ten artykuł wzbudził duże zainteresowanie, ponieważ przedstawiał możliwości realizacji i praktyczne zastosowania komercyjnego systemu baz danych. Praca ta stworzyła teoretyczne podstawy budowania baz danych w oparciu o relacyjne podejście do jej modelowania. Opublikowana teoria bardzo szybko zainteresowała potencjalnych twórców i producentów systemów zarządzania bazami danych. Droga od teorii do praktyki bywa często długa i wyboista ale w tym przypadku, ze względu na pilne potrzeby rynku, przebiegała dość szybko i sprawnie. Jednym z podstawowych wyzwań było opracowanie sposobu komunikowania się i korzystania z relacyjnych baz danych, czyli opracowanie specjalnego języka programowania.

W ramach prac nad językiem do obsługi baz danych, prowadzonych w firmie IBM, w roku 1974 powstał język SEQUEL (ang. *Structured English Query Language* – Stukturalny Angielski Język Zapytań), który następnie został rozwinięty i nazwany SQL (ang. *Structured Query Language* – Strukturalny Język Zapytań) – zmiana nazwy wynikała ze sporu prawnego o zastrzeżoną nazwę SEQUEL. Pod koniec lat 70-tych XX wieku, firma ORACLE wypuściła na rynek pierwszy komercyjny system zarządzania bazami danych oparty o SQL. W latach 80. i 90. ubiegłego wieku, trwał burzliwy rozwój baz danych opartych na modelu relacyjnym i języku SQL. Ponieważ wielu producentów zaczęło tworzyć rozwiązania baz danych oparte o model relacyjny i język SQL powstawało ryzyko, że u różnych producentów język SQL będzie rozwijał się inaczej. Rozwiązaniem tego problemu było zdefiniowanie standardów języka SQL przez organizację ISO (ang. *International Organization for Standardization*) i ANSI (ang. *American National Standards Institute*). Definiowanie standardu należy traktować jako wytyczne dla producentów systemów, w jakim kierunku rozwijać kolejne opracowania języka SQL, jakie nowe elementy mogą zostać wprowadzone do języka i jak system baz danych powinien realizować operacje związane z definiowaniem baz danych i ich eksploatacją. Obecnie język SQL jest powszechnie stosowanym językiem komunikacji z bazami danych w systemach opartych o relacyjny model danych.

Krótką historia standardów języka SQL:

- 1986: pierwszy standard SQL (SQL-86),
- 1989: następny standard SQL (SQL-89),
- 1992: kolejna, wzbogacona wersja standardu (SQL-92 lub SQL 2),
- 1999: wdrożenie następnej wersji standardu rozszerzonego o pewne cechy obiektowości (SQL 3),
- 2003: Kolejne rozszerzenie standardu (m.in. włączenie do standardu języka XML) SQL 4.

Język SQL jest nadal rozwijany i trudno przewidzieć, jakie kierunki rozwoju zostaną wybrane, zaś odpowiedzi na to pytanie dostarczać będą kolejne wersje standardu.

Język SQL jest zaliczamy do tak zwanych **języków deklaratywnych** (języki czwartej generacji 4GL) zorientowanych na wynik. W językach tego typu użytkownik definiuje, jaki efekt końcowy chce osiągnąć w wyniku działania wybranego polecenia bez określania, w jaki sposób należy to wykonać. O tym „jak” zrealizować polecenie języka SQL decyduje System Zarządzania Bazami Danych, który po otrzymaniu polecenia do wykonania realizuje czynności związane z jego realizacją (analiza składni, optymalizacja, opracowanie planu wykonania polecenia i realizacja przygotowanego planu). Polecenia języka SQL nie zawierają instrukcji sterujących wykonaniem programu, takich jak instrukcje warunkowe czy instrukcje pętli, gdyż użytkownik nie musi określać sposobu wykonania danego polecenia. Logika działania języka SQL w kontekście bazy danych jest oparta na algebrze relacji, czyli działa na zbiorach danych. Cechą charakterystyczną jest wykorzystywanie logiki trójwartościowej, czyli takiej, w której poza wartościami logicznymi **true** (prawda) i **false** (fałsz) występuje także wartość **unknown** (nieznana), reprezentowana przez wartość **null**.

Przetwarzanie poleceń w języku SQL jest realizowane w trybie interpretacji, czyli wysłanie polecenia do Systemu Zarządzania Bazami Danych uruchamia proces jego walidacji, optymalizacji i generowania planu wykonania. Dla poprawy wydajności, serwery baz danych przechowują możliwie długo utworzone plany wykonania, żeby przy kolejnym wywołaniu takiego samego polecenia skorzystać z przygotowanego planu bez konieczności wykonywania czynności przygotowawczych.

Praca z wykorzystaniem SQL może być realizowana na kilka sposobów:

- poprzez interaktywne zadawanie pytań do bazy (monitor),
- budowanie skryptów (zbioru wsadowo wykonywanych zapytań w SQL),
- osadzanie kodu (pojedynczych zapytań i całych procedur) SQL w innych językach programowania (na poziomie aplikacji),
- procedur składowanych (na poziome bazy danych).

Polecenia języka SQL można podzielić na trzy główne grupy:

- Język Definiowania Danych – **DDL** (ang. *Data Definition Language*)
- Język Manipulacji Danymi – **DML** (ang. *Data Manipulation Language*)
- Język Kontroli Danych – **DCL** (ang. *Data Control Language*)

W swojej podstawowej postaci język SQL nie zawiera wielu poleceń, ale każde z tych poleceń ma swoją złożoną składnię. Wszystkie działania na relacyjnej bazie danych można wykonać korzystając z poleceń języka SQL.

6.2. JĘZYK OPISU DANYCH (DDL)

Część języka umożliwiającą definiowanie struktur bazy danych (tabele, widoki, procedury, indeksy i inne obiekty bazy danych) oraz ich modyfikację. W skład DDL wchodzi następujące polecenia: **CREATE** (zdefiniuj obiekt bazy danych), **DROP** (usuń obiekt z bazy danych) i **ALTER** (zmień definicję istniejącego obiektu). Poniżej przedstawiamy kilka przykładowych poleceń SQL z krótkim objaśnieniem zasad ich działania.

- Definiowanie bazy danych:
CREATE DATABASE NaszaBaza – czyli utwórz nową bazę danych o nazwie NaszaBaza. Zwróćmy uwagę, że w tym poleceniu nie ma mowy o tym, jak tworzy się tę bazę. .
- Definiowanie przykładowej tabeli:
CREATE TABLE Uczniowie



```
(
    IdUcznia          int IDENTITY(1,1) NOT NULL,
    Nazwisko         varchar(50) NOT NULL,
    Imie             varchar(50) NOT NULL,
    DataUrodzenia    date NOT NULL,
    CzyChlopak      bit NOT NULL,
    Pesel           varchar(11) NULL,
    CONSTRAINT PK_uczniowie PRIMARY KEY CLUSTERED
        (IdUcznia ASC)
)
```

Aby zrozumieć to polecenie, sformulujemy zdanie w języku potocznym opisujące czynności, jakie serwer baz danych ma wykonać w odpowiedzi na to polecenie.

Zdefiniować nową tabelę o nazwie *Uczniowie*, która składa się z następujących kolumn:

- *IdUcznia* – kolumna typu *integer* z automatycznym ustalaniem wartości i niedopuszczalną wartością *null*,
- *Nazwisko* – kolumna typu *varchar(50)* z niedopuszczalną wartością *null*,
- *Imie* – kolumna typu *varchar(50)* z niedopuszczalną wartością *null*,
- *DataUrodzenia* – kolumna typu *date* z niedopuszczalną wartością *null*,
- *CzyChlopak* – kolumna typu *bit* (typ logiczny) z niedopuszczalną wartością *null*,
- *Pesel* – kolumna typu *varchar(11)* z dopuszczalną wartością *null*,

Dodatkowo definiujemy ograniczenie uznające kolumnę *idUcznia* za klucz podstawowy tabeli.”

Porównując postać polecenia i jego interpretację wyrażoną w języku potocznym, można zauważyć, że polecenie języka SQL w precyzyjny sposób określiło, jak ma wyglądać tabela, którą chcemy zdefiniować, brak jest w nim natomiast jakichkolwiek wskazówek, jak to ma być wykonane.

6.3. JĘZYK MANIPULACJI DANYMI (DML)

Część języka SQL umożliwiająca dokonywanie operacji na danych, takich jak: wstawianie wiersza do tabeli, modyfikowanie istniejących wierszy, usuwanie wierszy oraz pobieranie danych (zapytania). W skład DML wchodzi polecenia: **INSERT** (wstaw wiersze do tabeli), **UPDATE** (zmodyfikuj wiersze w tabeli), **DELETE** (usuń wiersze z tabeli) i **SELECT** (pobierz dane) oraz nowo wprowadzone do standardu polecenie **MERGE** (wykonaj aktualizację zbiorczą). Poniżej przedstawimy kilka przykładowych poleceń języka DML.

- Wstawianie wierszy do tabeli:

```
INSERT INTO Uczniowie (Nazwisko, Imie, DataUrodzenia, CzyChlopak, Pesel)
VALUES('Kot', 'Jan', '1991-07-12','true', '91071276538')
```

```
INSERT INTO Uczniowie (Nazwisko, Imie, DataUrodzenia, CzyChlopak, Pesel)
VALUES('Nowak', 'Janina', '1991-03-22','false', '91032267549')
```

```
INSERT INTO Uczniowie (Nazwisko, Imie, DataUrodzenia, CzyChlopak)
VALUES('Sarnowski', 'Piotr', '1991-12-12','true')
```

Składnię polecenia **INSERT**, na podstawie podanych przykładów można łatwo zinterpretować następująco: *Wstaw do tabeli o nazwie Uczniowie do kolumn wymienionych po nazwie tabeli wartości zawarte w klauzuli VALUES*

- Pobieranie części zawartości tabeli Uczniowie:

```
SELECT Nazwisko, Imie, Pesel
FROM Uczniowie
WHERE CzyChlopak=true
ORDER BY nazwisko
```



Tym razem zaprezentowany został przykład zapytania **SELECT**, które można opisać językiem potocznym: *Pobierz i dostarcz tabelę zawierającą nazwisko, imię i numer Pesel – dane pobrać z tabeli o nazwie Uczniowie (FROM). W wyniku zapytania mają znaleźć się tylko chłopcy (WHERE). Wynik zapytania uporządkować alfabetycznie według nazwiska (ORDER BY).*

Ćwiczenie 10. Definiowanie tabeli za pomocą polecenia SQL.

W ramach ćwiczenia zdefiniujemy nową tabelę o nazwie **Nauczyciele** za pomocą polecenia **CREATE TABLE** języka SQL. Strukturę definiowanej tabeli pokazano na rys. 35.

Column Name	Data Type	Allow Nulls
idnauczyciela	int	<input type="checkbox"/>
Nazwisko	varchar(50)	<input type="checkbox"/>
Imie	varchar(50)	<input type="checkbox"/>
DataUrodzenia	date	<input type="checkbox"/>
Nip	char(13)	<input type="checkbox"/>
Tytul	varchar(32)	<input type="checkbox"/>

Rysunek 35.

Schemat tabeli **Nauczyciele**

Poniżej przedstawiono polecenie SQL definiujące tabelę **Uczniowie** i korzystając z tego przykładu tworzymy polecenie definiujące tabelę pokazaną na rys. 35.

```
CREATE TABLE [dbo].[Uczniowie](
    [iducznia] [int] IDENTITY(1,1) NOT NULL,
    [Nazwisko] [varchar](50) NOT NULL,
    [Imie] [varchar](50) NOT NULL,
    [DataUrodzenia] [date] NULL,
    [CzyChlopak] [bit] NOT NULL,
    [Pesel] [varchar](11) NULL,
    [idklasy] [int] NOT NULL,
    CONSTRAINT [PK_uczniowie] PRIMARY KEY CLUSTERED
    (
        [iducznia] ASC
    ) ON [PRIMARY]
) ON [PRIMARY]
```

Możliwość pisania polecenia uzyskujemy poprzez otwarcie okna edycji za pomocą przycisku **New Query**. Napisane polecenie uruchamiamy naciskając klawisza **F5**.

Ćwiczenie 11. Wprowadzanie danych do tabel.

W ramach tego ćwiczenie wprowadzamy dane do przykładowych tabel wykorzystując polecenia **INSERT**, modyfikujemy wybrane dane wykorzystując polecenie **UPDATE** i usuwamy wybrane dane wykorzystując Polecenie **DELETE**. Składnie wymienionych poleceń poznajemy wykorzystując **HELP**.

7. ZAPYTANIA DO BAZ DANYCH

POLECENIE SELECT JĘZYKA SQL

Do realizacji zapytań, język SQL udostępnia polecenie **SELECT**. Polecenie to ma dość złożoną składnię – poniżej przedstawiamy jej uproszczoną wersję:

```
SELECT [TOP n] lista_kolumn
FROM lista_tabel
WHERE warunki_selekcji
GROUP BY lista_kolumn_grupowania
HAVING warunek_selekcji
ORDER BY lista_kolumn_porzadkowania
```

gdzie:

SELECT – polecenie języka SQL używane do realizacji zapytań do bazy danych,

TOP n – ogranicza liczbę wierszy zapytania do n wierszy,

lista_kolumn – określenie, jakie kolumny i w jakiej postaci mają się znaleźć w wyniku zapytania,

FROM – klauzula polecenia **SELECT**, w której określamy, jakie tabele i w jaki sposób połączone biorą udział w realizacji zapytania,

lista_tabel – określenie, które tabele i jak połączone biorą udział w realizacji zapytania,

WHERE – klauzula polecenia **SELECT**, służąca do określenia warunków selekcji,

warunek_selekcji – wyrażenie logiczne określające, jakie wiersze powinny znaleźć się w tabeli wynikowej,

GROUP BY – klauzula polecenia **SELECT**, definiująca sposób grupowania (wykorzystywana z funkcjami agregującymi, które będą omawiane w dalszej części wykładu),

lista_kolumn_grupowania – określenie kolumn, według których jest realizowana operacja grupowania,

HAVING – klauzula polecenia **SELECT**, tak zwany opóźniony warunek selekcji (wykorzystywany najczęściej z funkcjami agregującymi),

ORDER BY – klauzula polecenia **SELECT**, w której określamy sposób uporządkowania wyników zapytania

lista_kolumn_porzadkowania – określenie kolumn, według których należy uporządkować wynik zapytania.

Jak widać z powyższego opisu, polecenie **SELECT** nie ma zbyt wielu dodatkowych elementów składni, ale jak zobaczymy w dalszej części zajęć, można przy pomocy pozornie niewielu elementów wyrazić bardzo złożone zapytania.

W tej części zajęć skoncentrujemy się na formułowaniu zapytań kierowanych do jednej tabeli.

Najprostszą postacią polecenia **SELECT** jest żądanie pobrania wszystkich danych z wybranej tabeli:

```
SELECT *
FROM Uczniowie
```

Jako listę kolumn po nazwie poleceniu **SELECT** występuje znak * (gwiazdka), który należy interpretować jako wszystkie dostępne kolumny z tabeli, której nazwa występuje w klauzuli **FROM**. Przykładowy wynik takiego zapytania może mieć postać jak na rys. 36.

iducznia	Nazwisko	Imie	DataUrodzenia	CzyChlopak	Pesel	idklasy
1	Kotek	Katarzyna	1992-03-12	0	92031275446	2
2	Piesek	Jan	1992-05-15	1	92051587746	1
3	Lisek	Kasia	1992-02-22	0	92022277654	2
4	Kurka	Jola	1992-06-02	0	92060288788	2
5	Gąska	Wacek	1991-03-11	1	91031199123	1
6	Krówka	Rysio	1992-05-15	1	92051577646	1
7	Zebra	Wojtek	1993-03-13	1	93030399846	1
8	Gazela	Basia	1992-11-11	0	92111177446	2

Rysunek 36.

Wynik zapytania ogólnego

Podstawą zapytań kierowanych do jednej tabeli jest realizacja operacji projekcji i selekcji, dzięki którym możemy wybrać dowolny fragment tabeli wyjściowej.

Operację projekcji w zapytaniach **SELECT** realizujemy poprzez wymienienie listy kolumn, które powinny znaleźć się w tabeli wynikowej.

```
SELECT Nazwisko, Imie, Pesel, CzyChlopak
FROM Uczniowie
```

To zapytanie wybiera cztery wymienione kolumny z tabeli Uczniowie (patrz rys. 37).

Nazwisko	Imie	Pesel	CzyChlopak
Kotek	Katarzyna	92031275446	0
Piesek	Jan	92051587746	1
Lisek	Kasia	92022277654	0
Kurka	Jola	92060288788	0
Gąska	Wacek	91031199123	1
Krówka	Rysio	92051577646	1

Rysunek 37.

Wynik selekcji wybranych kolumn

Do przedstawionego wyżej zapytania dodamy teraz **warunek selekcji**:

```
SELECT Nazwisko, Imie, Pesel, CzyChlopak
FROM Uczniowie
WHERE CzyChlopak=1
```

Tak sformułowane zapytanie jest połączeniem operacji projekcji i selekcji, jego wynik jest na rys. 38.

Nazwisko	Imie	Pesel	CzyChlopak
Piesek	Jan	92051587746	1
Gąska	Wacek	91031199123	1
Krówka	Rysio	92051577646	1

Rysunek 38.

Wynik projekcji i selekcji

Często chcemy uzyskać wynik zapytania uporządkowany według zadanego kryterium. W poleceniu **SELECT**, porządkowanie wyniku zapytania można uzyskać za pomocą dołączonej do zapytania klauzuli **ORDER BY**. Przedstawiona na rys. 39 tabela jest wynikiem następującego zapytania, które poleca uporządkować listę uczniów według klasy, a w obrębie danej klasy – alfabetycznie według nazwiska.

```
SELECT Nazwisko, Imie, Pesel, Idklasy
FROM Uczniowie
WHERE Idklasy=1 OR Idklasy=2
ORDER BY Idklasy ASC, Nazwisko DESC
```

W klauzuli **ORDER BY** wymienione zostały dwie kolumny co należy interpretować następująco: *uporządkuj według Idlasy a w obrębie wierszy o tej samej wartości Idklasy uporządkuj według nazwiska*. Dodatkowo użyto słowa kluczowe **ASC** i **DESC** określające rodzaj uporządkowania:

ASC (*ascending* – rosnąco)
DESC (*descending* – malejąco).

Nazwisko	Imie	Pesel	Idklasy
Zebra	Wojtek	93030399846	1
Sarenka	Rysio	92121278766	1
Piesek	Jan	92051587746	1
Krówka	Rysio	92051577646	1
Gąska	Wacek	91031199123	1
Stokrotka	Rysio	93121278766	2
Ryba	Jan	93051587746	2
Różyczka	Basia	93111177446	2
Płotka	Wojtek	93030399846	2
Okoń	Rysio	93051577646	2

Rysunek 39.

Wynik zapytania uporządkowany po dwóch kolumnach

W dotychczas przedstawionych przykładach, w kolumnach tabeli wynikowej były przedstawiane dane pobrane bezpośrednio z tabeli, czyli w takiej postaci, w jakiej zostały zapisane. W zapytaniach możemy przekształcać pobrane dane do innej postaci w zależności od naszych potrzeb. W kolejnym przykładzie przekształcimy dane zapisane w kolumnie CzyChłopak do postaci bardziej czytelnej:

```
SELECT Nazwisko, Imie, Pesel,
CASE CzyChłopak
WHEN 1 THEN 'Mężczyzna'
ELSE 'Kobieta'
END as Płeć
FROM Uczniowie
WHERE Idklasy=2
```

Przykładowy wynik tego zapytania zawiera kolumnę o nazwie Płeć, w której są wyświetlane wartości tekstowe Kobieta lub Mężczyzna, pomimo tego, że takie dane nie są zapisane w tabeli Uczniowie.

Wykorzystane w zapytaniu wyrażenie, zaczynające się od słowa **CASE** należy interpretować następująco: *W zależności od wartości w kolumnie CzyChłopak (CASE CzyChłopak); jeżeli wartość kolumny CzyChłopak jest równa 1, to zwróć tekst Mężczyzna (WHEN 1 THEN 'Mężczyzna'), a w przeciwnym wypadku zwróć tekst Kobieta (ELSE 'Kobieta') utworzoną kolumnę nazwij Płeć (AS Płeć)*

Przykładowy wynik tego zapytania może mieć postać jak na rys. 40.

Nazwisko	Imie	Pesel	Płeć
Kotek	Katarzyna	92031275446	Kobieta
Lisek	Kasia	92022277654	Kobieta
Kurka	Jola	92060288788	Kobieta
Gazela	Basia	92111177446	Kobieta
Konik	Kasia	93031275446	Kobieta
Ryba	Jan	93051587746	Mężczyzna
Kura	Kasia	93022277654	Kobieta
Łoś	Jola	93060288788	Kobieta
Miś	Wacek	93031199123	Mężczyzna

Rysunek 40.

Wynik zapytania z nową kolumną Płeć

POLECENIE SELECT – ŁĄCZENIE TABEL

Do tej pory w zapytaniu odwoływaliśmy się do jednej tabeli, a teraz zajmiemy się zapytaniami, których tabele wynikowe będą zawierać dane z wielu tabel. Nie jest to o wiele trudniejsze. Zmieni się jedynie to, że w klauzuli **FROM** należy opisać sposób połączenia tabel, które będą brały udział w zapytaniu. Na przykład:

```
SELECT Uczniowie.* , Klasy.*
FROM Uczniowie JOIN Klasy
ON Uczniowie.Idklasy=Klasy.Idklasy
```

Sens tego zapytania można opisać w następujący sposób: *Wybrać (SELECT) wszystkie kolumny z tabeli Uczniowie (Uczniowie.*) oraz wszystkie kolumny z tabeli Klasy (Klasy.*), pobieraj dane z tabeli Uczniowie połączonej z tabelą Klasy (FROM Uczniowie JOIN Klasy), warunkiem połączenia jest równość wartości Idklasy w obu tabelach, czyli klucz obcy ma być równy kluczowi podstawowemu (ON Uczniowie.Idklasy=Klasy.Idklasy).* Przykładowy wynik takiego jest pokazany na rys. 41.

iducznia	Nazwisko	Imie	DataUrodzenia	CzyChlopak	Pesel	idklasy	idklasy	Nazwa	RokSzkolny
2	Piesek	Jan	1992-05-15	1	92051587746	1	1	Ia	2008/2009
5	Gąska	Wacek	1991-03-11	1	91031199123	1	1	Ia	2008/2009
6	Krówka	Rysio	1992-05-15	1	92051577646	1	1	Ia	2008/2009
7	Zebra	Wojtek	1993-03-13	1	93030399846	1	1	Ia	2008/2009
9	Sarenka	Rysio	1992-12-12	0	92121278766	1	1	Ia	2008/2009
1	Kotek	Katarzyna	1992-03-12	0	92031275446	2	2	Ila	2008/2009
3	Lisek	Kasia	1992-02-22	0	92022277654	2	2	Ila	2008/2009
4	Kurka	Jola	1992-06-02	0	92060288788	2	2	Ila	2008/2009

Rysunek 41.

Wynik połączenia dwóch tabel

Przykładowe zapytanie z użyciem omówionych do tej pory operacji:

```
SELECT Uczniowie.Nazwisko, Uczniowie.Imie,
CASE CzyChlopak
WHEN 1 THEN 'Mężczyzna'
ELSE 'Kobieta'
END as Płeć,
Klasy.Nazwa, Klasy.RokSzkolny
FROM Uczniowie JOIN Klasy ON Uczniowie.Idklasy=Klasy.Idklasy
WHERE YEAR(Uczniowie.DataUrodzenia)=1992
ORDER BY Płeć, Nazwisko DESC
```

Przykładowy wynik takiego zapytania jest pokazany na rys. 42.

Nazwisko	Imie	Płeć	Nazwa	RokSzkolny
Sarenka	Rysio	Kobieta	Ia	2008/2009
Orka	Kasia	Kobieta	Ilc	2008/2009
Lisek	Kasia	Kobieta	Ila	2008/2009
Kurka	Jola	Kobieta	Ila	2008/2009
Kotek	Katarzyna	Kobieta	Ila	2008/2009
Gazela	Basia	Kobieta	Ila	2008/2009
Foka	Jola	Kobieta	Ilc	2008/2009
Antylopa	Kasia	Kobieta	Ilc	2008/2009
Rekin	Jan	Mężczyzna	Ilc	2008/2009
Piesek	Jan	Mężczyzna	Ia	2008/2009
Krówka	Rysio	Mężczyzna	Ia	2008/2009

Rysunek 42.

Wynik bardziej złożonego zapytania



I jeszcze jeden przykład. Chcemy napisać zapytanie, które przygotuje wykaz uczniów (nazwisko i imię) oraz dane nauczyciela (nazwisko i imię oraz stopień zawodowy), który wystawił ocenę i datę wystawienia oceny tym uczniom, którzy w roku 2009 otrzymali z fizyki ocenę 5, wynik uporządkować malejąco według daty wystawienia oceny.

```
SELECT Uczniowie.Nazwisko+' '+Uczniowie.Imie AS Uczeń,
       Nauczyciele.Nazwisko+' ' Nauczyciele.Imie AS Nauczyciel,
       Oceny.DataWystawienia, Oceny.Ocena
FROM Uczniowie JOIN Oceny ON Uczniowie.Iducznia=Oceny.IdUcznia
      JOIN Nauczyciele ON Nauczyciele.IdNauczyciela=Oceny.IdNauczyciela
      JOIN Przedmioty ON Oceny.Idprzedmiotu=Przedmioty.Idprzedmiotu
WHERE YEAR(DataWystawienia)=2009 AND Ocena=5 AND Przedmioty.Nazwa='Fizyka'
```

W tym zapytaniu bez większego problemu zostały połączone cztery tabele. Przykładowy wynik tego zapytania jest przedstawiony na rys. 43.

Uczeń	Nauczyciel	DataWystawienia	Ocena
Piesek Jan	Lew Wojciech	2009-01-05	5.00
Piesek Jan	Lew Wojciech	2009-02-09	5.00
Kotek Katarzyna	Lew Wojciech	2009-02-07	5.00
Konik Kasia	Pantera Saba	2009-01-27	5.00
Łoś Jola	Gepard Hala	2009-01-30	5.00
Konik Kasia	Gepard Hala	2009-01-27	5.00
Sarenka Rysio	Gepard Hala	2009-01-24	5.00
Okoń Rysio	Lew Wojciech	2009-01-08	5.00

Rysunek 43.

Wynik złożonego zapytania, w którym zostały połączone cztery tabele

Omawiając przykłady łączenia tabel koncentrowaliśmy się na podstawowej operacji, opartej na **złączeniu wewnętrznym** (ang. *inner join*), które powoduje że tylko wiersze, które spełniają warunek łączenia, znajdują się w tabeli wynikowej. W poprzednim przykładzie, ci uczniowie, którzy w roku 2009 nie otrzymali oceny 5 z fizyki, nie pojawią się w wyniku zapytania. W przypadku stosowania tak zwanego **połączenia zewnętrznego** (ang. *outer join*) będziemy mogli zapewnić występowanie w tabeli wynikowej nawet tych wierszy z wybranej tabeli, które nie spełniają warunku połączenia. Zademonstrujemy to na następującym przykładzie:

```
SELECT Uczniowie.Nazwisko, Uczniowie.Imie, Oceny.DataWystawienia, Ocena
FROM Uczniowie LEFT OUTER JOIN Oceny
      ON Uczniowie.iducznia=Oceny.Iducznia AND Oceny.Ocena=2
      AND YEAR(DataWystawienia)=2009 AND MONTH(DataWystawienia)=2
```

Nazwisko	Imie	DataWystawienia	Ocena
Kotek	Katarzyna	2009-02-05	2.00
Kotek	Katarzyna	2009-02-05	2.00
Kotek	Katarzyna	2009-02-12	2.00
Kotek	Katarzyna	2009-02-12	2.00
Piesek	Jan	NULL	NULL
Lisek	Kasia	NULL	NULL
Kurka	Jola	2009-02-07	2.00
Kurka	Jola	2009-02-07	2.00

Rysunek 44.

Wynik zapytania z użyciem połączenia zewnętrznego

Przykładowy wynik tego zapytania jest pokazany na rys. 44. W porównaniu z przykładem wcześniejszym, istotne są trzy różnice:

- W operacji łączenia wykorzystano opcję **LEFT OUTER JOIN** (lewostronne łączenie zewnętrzne), która zapewnia, że do wyniku zapytania, oprócz wierszy spełniających warunek łączenia, zostaną dodane wiersze z tabeli po lewej stronie operatora **JOIN** (w naszym przypadku tabela Uczniowie), dla których warunek łączenia jest niespełniony.
- Warunki selekcji (**AND** Ocena=2 **AND** YEAR(DataWystawienia)=2009 **AND** MONTH(DataWystawienia)=2) zostały umieszczone w klauzuli **ON** a nie w klauzuli **WHERE**.
- W wyniku zapytania, dla tych wierszy, które nie spełniają warunku łączenia, w kolumnach DataWystawienia i Ocena, występuje wartość NULL

Istnieją jeszcze inne operatory połączeń, które można wykorzystywać zamiast operatora **JOIN** (np. **APPLY** lub **PIVOT**), ale ich omówienie wykracza poza ramy tych zajęć.

POLECENIE SELECT – WYKORZYSTANIE FUNKCJI AGREGUJĄCYCH

Zapytania SQL mogą być także wykorzystane do wykonywania obliczeń na podstawie danych zawartych w tabelach. Do tego celu służą funkcje agregujące. Język SQL udostępnia pięć podstawowych funkcji agregujących:

- **COUNT** – oblicza liczbę wierszy otrzymanych w wyniku zapytania,
- **SUM** – sumuje zawartość kolumny (lub wyrażenia obliczonego na podstawie danych) dla wszystkich wierszy w wyniku zapytania,
- **AVG** – oblicza średnią arytmetyczną zawartości kolumny (lub wyrażenia obliczonego na podstawie danych) dla wszystkich wierszy w wyniku zapytania,
- **MIN** – określa wartość minimalną dla kolumny w wyniku zapytania,
- **MAX** – określa wartość maksymalną dla kolumny w wyniku zapytania.

W różnych Systemach Zarządzania Bazami Danych mogą być dostępne jeszcze inne funkcje agregujące (głównie realizujące obliczanie wartości statystycznych), zaprezentowany zbiór pięciu funkcji jest powszechnie obowiązującym standardem.

Zapytania, które wykorzystują funkcje agregujące, zwracają jeden wiersz zawierający wynik obliczeń dla danej funkcji. W pierwszym przykładzie napiszemy zapytanie, w których chcemy obliczyć, ilu uczniów jest zapisanych w tabeli Uczniowie (wynik jest pokazany na rys. 45):

```
SELECT COUNT(*) AS IluUczniow
FROM Uczniowie
```

IluUczniow
24

Rysunek 45.

Wynik użycia funkcji agregującej **COUNT**

Zapytania używające funkcji agregujących mogą wykorzystywać łączenie tabel (klauzula **FROM**) oraz warunki selekcji (klauzula **WHERE**). W kolejnym przykładzie obliczamy, ilu uczniów jest w klasie IIa (wynik jest pokazany na rys. 46).

```
SELECT COUNT(*) AS IluUczniow
FROM Uczniowie JOIN Klasy ON Uczniowie.idklasy=Klasy.idklasy
WHERE Klasy.Nazwa='IIa'
```

IluUczniow
13

Rysunek 46.

Wyznaczanie liczby uczniów w klasie IIa

Jeśli chcemy w jednym zapytaniu wyznaczyć liczbę uczniów w poszczególnych klasach i wyniki umieścić w tabeli, to musimy zastosować dodatkową klauzulę **GROUP BY** (grupuj według), jak w następującym przykładzie (wynik jest pokazany na rys. 47).



```
SELECT Klasy.Nazwa,
       COUNT(*) AS IluUczniow
FROM Uczniowie JOIN Klasy ON Uczniowie.idklasy=Klasy.idklasy
GROUP BY Klasy.Nazwa
```

Nazwa	IluUczniow
la	5
lb	1
lla	13
llc	5

Rysunek 47.

Wynik zapytania z użyciem funkcji agregującej i grupowaniem wyników

Działanie klauzuli **GROUP BY** polega na zastosowaniu funkcji agregującej do każdej grupy wierszy w wyniku zapytania, które mają tę samą wartość w kolumnie (lub w kolumnach, bo można podać w tej klauzuli listę kolumn) podanej jako parametr grupowania. W naszym przykładzie kolumną, według której są grupowane dane, jest Nazwa z tabeli Klasy, czyli funkcja **COUNT** zlicza wiersze dla każdej klasy oddzielnie. W kolejnym chcemy otrzymać listę uczniów z klasy lla oraz ich średnią ocen otrzymanych w roku 2009 (wynik jest pokazany na rys. 48).

```
SELECT Uczniowie.Nazwisko, Uczniowie.Imie, AVG(Oceny.Ocena) as Średnia
FROM Uczniowie JOIN Oceny ON Uczniowie.Iducznia=Oceny.Iducznia
      JOIN Uczniowie.Idklasy=Klasy.Idklasy
WHERE YEAR(Oceny.DataWystawienia)=2009 AND Klasy.Nazwa='lla'
GROUP BY Uczniowie.Nazwisko, Uczniowie.Imie
ORDER BY Średnia DESC
```

Nazwisko	Imie	Średnia
Łoś	Jola	4.714285
Konik	Kasia	4.000000
Kura	Kasia	3.111111
Różyczka	Basia	3.000000
Kurka	Jola	3.000000
Kotek	Katarzyna	2.958333
Lisek	Kasia	2.866666
Okoń	Rysio	2.857142
Gazela	Basia	2.800000
Miś	Wacek	2.777777
Płotka	Wojtek	2.777777
Ryba	Jan	2.500000

Rysunek 48.

Wynik zapytania ze średnimi ocenami uczniów

Język SQL udostępnia jeszcze jedną klauzulę wykorzystywaną przy grupowaniu z zastosowaniem funkcji agregujących. Przypuśćmy, że w poprzednim zapytaniu chcielibyśmy otrzymać listę tylko tych uczniów, dla których obliczona średnia ocena jest większa od 3.00. Takiego warunku selekcji nie możemy jednak zapisać w klauzuli **WHERE**, ponieważ jest ona wykonywana w momencie, gdy nie jest znany jeszcze wynik obliczeń funkcji agregującej. Potrzebujemy więc sprawdzenia warunku Średnia>3.00 dopiero wtedy, gdy znane będą wartości średnie. Klauzula **HAVING**, dzięki której rozwiążemy ten problem, nazywana jest **opóźnionym warunkiem selekcji** i jest wykorzystywana do selekcji według wartości obliczonych przez funkcje agregujące. Zmodyfikujemy nasz przykład i zastosujemy klauzulę **HAVING** – wynik na rys. 49.

```
SELECT Uczniowie.Nazwisko, Uczniowie.Imie, AVG(Oceny.Ocena) as Średnia
FROM Uczniowie JOIN Oceny ON Uczniowie.Iducznia=Oceny.Iducznia
      JOIN Uczniowie.Idklasy=Klasy.Idklasy
WHERE YEAR(Oceny.DataWystawienia)=2009 AND Klasy.Nazwa='lla'
GROUP BY Uczniowie.Nazwisko, Uczniowie.Imie
HAVING AVG(Oceny.Ocena) > 3.00
ORDER BY Średnia
```



Nazwisko	Imie	Średnia
Łoś	Jola	4.714285
Konik	Kasia	4.000000
Kura	Kasia	3.111111

Rysunek 49.

Wynik zapytania z selekcją niektórych

POLECENIE SELECT – ZAPYTANIA ZŁOŻONE

Polecenie **SELECT** języka SQL umożliwia **zagnieżdżanie zapytań**, czyli wykorzystanie zapytania wewnątrz innego zapytania. Dzięki tej właściwości można za pomocą jednego polecenia wykonywać bardzo złożone operacje na danych. Omówimy to, chcąc przygotować listę uczniów (zawierającą nazwisko i imię ucznia oraz nazwę klasy), którzy w roku 2009 nie otrzymali oceny niedostatecznej z fizyki. Należy zwrócić uwagę na fakt, że chcemy pobrać z bazy dane, które nie są bezpośrednio w niej zapisane, bo jeżeli uczeń nie otrzymał oceny to w bazie danych nie ma żadnego zapisu tego faktu. Rozwiązując ten problem skorzystamy z pewnych zależności logicznych. Pomyślmy o tym problemie jako o działaniu na następujących zbiorach:

- A – zbiór wszystkich uczniów,
- B – zbiór uczniów, którzy otrzymali w roku 2009 ocenę niedostateczną z fizyki,
- C – poszukiwany zbiór uczniów, którzy w roku 2009 nie otrzymali oceny niedostatecznej z fizyki.

Wyrażenie: $C = A - B$ opisuje rozwiązanie naszego problemu, czyli poszukiwany zbiór możemy otrzymać jako różnicę dwóch innych zbiorów.

Zbiór **A**, czyli zbiór wszystkich uczniów możemy otrzymać za pomocą prostego zapytania, które przygotuje zbiór uczniów zawierający nazwisko i imię ucznia oraz nazwę klasy:

```
SELECT Uczniowie.Nazwisko, Uczniowie.Imie, Klasy.Nazwa,
FROM Uczniowie JOIN Klasy ON Uczniowie.idklasy=Klasy.idklasy
```

Zbiór **B**, czyli zbiór uczniów, którzy otrzymali ocenę niedostateczną z fizyki otrzymamy za pomocą innego zapytania:

```
SELECT DISTINCT Iducznia
FROM Oceny JOIN Przedmioty ON Oceny.Idprzedmiotu=Przedmioty.Idprzedmiotu
WHERE Przedmioty.Nazwa='Fizyka' AND YEAR(Oceny.DataWystawienia)=2009
AND Oceny.Ocena=2
```

Działanie nowej opcji **DISTINCT** polega na eliminowaniu w wyniku zapytania powtarzających się wierszy (jeżeli uczeń otrzymał kilka ocen niedostatecznych z fizyki w roku 2009, to w zbiorze wynikowym jego identyfikator (Iducznia) pojawiłby się wiele razy).

Pozostaje nam teraz zrealizować operację różnicy zbiorów:

```
SELECT Uczniowie.Nazwisko, Uczniowie.Imie, Klasy.Nazwa,
FROM Uczniowie JOIN Klasy
ON Uczniowie.idklasy=Klasy.idklasy
WHERE Iducznia NOT IN
(SELECT DISTINCT Iducznia
FROM Oceny JOIN Przedmioty
ON Oceny.Idprzedmiotu=Przedmioty.Idprzedmiotu
WHERE Przedmioty.Nazwa='Fizyka' AND
YEAR(Oceny.DataWystawienia)=2009 AND Oceny.Ocena=2)
```

Pokazaliśmy jeden przykład zapytania złożonego, ILUSTRUJĄCY dodatkowe możliwości, jakimi dysponujemy przy pisaniu zapytań do baz danych z wykorzystaniem języka SQL. Trudno wymienić wszystkie sytuacje, w których można wykorzystywać podzapytania ale jest jedna zasada ogólna:

Podzapytanie może być wykorzystane wszędzie tam, gdzie ma sens wynik tego podzapytania



Ponieważ podstawową formą wyników zapytań jest tabela, to możemy podzapytania zwracające tabele wykorzystywać zamiast fizycznych tabel będących częścią bazy danych, na przykład w klauzuli **FROM**.

Zapytania mogą zwracać także jedną wartość, gdy zapytanie zwraca jedną kolumnę i w wyniku powstaje jeden wiersz, to w takim przypadku możemy umieścić podzapytanie wszędzie tam, gdzie ma sens taka wartość.

```
SELECT AVG(YEAR(GETDATE()) – YEAR(Uczniowie.DataUrodzenia)) as ŚredniWiek
FROM Uczniowie
```

Takie zapytanie powinno zwrócić średni wiek uczniów zapisanych w tabeli Uczniowie (funkcja **AVG** oblicza średnią arytmetyczną wyrażenia, w którym od roku pobranego z daty systemowej (**GETDATE()**) odejmujemy rok pobrany z daty urodzenia), czyli zapytanie zwróci jedną wartość i moglibyśmy taką postać zapytania wykorzystać w innym, które ma zwrócić dane uczniów, których wiek jest poniżej średniego wieku wszystkich uczniów.

Nie byłoby problemu gdybyśmy chcieli wybrać uczniów, których wiek jest mniejszy od pewnej danej wartości np. 17.5, wtedy zapytanie miaoby postać:

```
SELECT Nazwisko,Imie,Pesel
FROM Uczniowie
WHERE (YEAR(GETDATE()) – YEAR(Uczniowie.DataUrodzenia) < 17.5
```

Ponieważ chcemy, żeby wiek ucznia był mniejszy od średniego wieku wszystkich uczniów, to musimy konkretną wartość (17.5) zastąpić zapytaniem obliczającym tę średnią, czyli tak jak poniżej:

```
SELECT Nazwisko,Imie,Pesel
FROM Uczniowie
WHERE (YEAR(GETDATE()) – YEAR(Uczniowie.DataUrodzenia)
< (SELECT AVG(YEAR(GETDATE()) – YEAR(Uczniowie.DataUrodzenia)) as ŚredniWiek
FROM Uczniowie)
```

Przeczytaliśmy kilka przykładów zapytań złożonych, żeby pokazać dodatkowe możliwości języka SQL, budowania takich zapytań.

Zapytanie **SELECT** umożliwia także przekształcanie wyniku do postaci dokumentu XML, tworzyć tabele przestawne i wiele innych operacji ale my robimy tylko pierwszy krok w zagadnienia zapytań pisanych w języku SQL.

Ćwiczenie 12. Wykonywanie zapytań SQL.

Z pomocą prowadzącego kurs próbujemy napisać zapytania odwołujące się do bazy danych ElektronicznyDziennikOcen:

- Napisać zapytanie, które przygotuje tabelę, zawierającą następujące dane: nazwisko i imię ucznia, jego datę urodzenia i numer Pesel dla tych uczniów, którzy urodzili się później niż 31-12-1992. Wynik zapytania uporządkować malejąco według daty urodzenia.
- Napisać zapytanie, które przygotuje tabelę zawierającą następujące dane: nazwisko i imię ucznia, nazwę przedmiotu, datę wystawienia oceny oraz wartość tej oceny dla ocen z fizyki wystawionych uczniom klasy IIa w roku 2009. Wynik zapytania uporządkować malejąco według daty wystawienia oceny.
- Napisać zapytanie, które przygotuje tabelę zawierającą następujące dane: nazwę przedmiotu oraz średnią ocen wystawionych uczniom klasy IIa w roku 2009 z poszczególnych przedmiotów. W wyniku zapytania powinny znaleźć się tylko te przedmioty, dla których wartość średniej jest niższa od 3.5. Wynik zapytania uporządkować malejąco względem wartości średniej oceny.
- Napisać zapytanie, które przygotuje tabelę zawierającą następujące dane: nazwisko i imię nauczyciela dla tych nauczycieli, którzy w marcu 2009 nie wystawili oceny niedostatecznej uczniom klasy IIa. Wynik zapytania uporządkować rosnąco według nazwiska nauczyciela "
- Napisać zapytanie, które przygotuje listę uczniów, którzy otrzymali ocenę 5 z fizyki w marcu 2009, z wyjątkiem tych uczniów, które otrzymały w tym samym miesiącu ocenę 2 z matematyki

8. INNE OBIEKTY BAZY DANYCH

8.1. WIDOKI

Widokami w relacyjnych bazach danych nazywamy wirtualne tabele tworzone w standardzie SQL na podstawie określonego zapytania **SELECT**. Widok tworzymy poleceniem **CREATE VIEW** dołączając zapytanie **SELECT**. Widok nie przechowuje danych – jest on przeznaczony do pokazywania danych zawartych w innych tabelach. Zdefiniowany widok możemy wykorzystywać, podobnie jak tabele, w zapytaniach a także można modyfikować dane w tabeli wykonując operacje modyfikacji w odniesieniu do widoku, gdy widok jest utworzony na podstawie zapytania korzystającego z jednej tabeli.

Stosowanie widoków przynosi wiele korzyści:

- Wygoda – uproszczenie zapytań kierowanych do systemu.
- Utworzenie dodatkowego poziomu zabezpieczenia tabeli.
- Ograniczenie dostępu do określonych kolumn lub wierszy tabeli bazowej.
- Pokazywanie danych z innej perspektywy, na przykład widok może zostać użyty do zmiany nazwy kolumny bez zmiany rzeczywistych danych zapisanych w tabeli.
- Tworzenie warstwy abstrakcji.

Przykładowe polecenie definiujące widok w bazie danych Elektroniczny DziennikOcen:

```
CREATE VIEW [test]
AS
SELECT Uczniowie.Nazwisko,
       Uczniowie.Imie,
       Oceny.DataWystawienia,
       Oceny.Ocena,
       Przedmioty.Nazwa AS Przedmiot
FROM Uczniowie INNER JOIN
     Oceny ON Uczniowie.iducznia =
     Oceny.iducznia INNER JOIN
     Przedmioty ON Oceny.idprzedmiotu = Przedmioty.idprzedmiotu
```

Widoki można definiować z dodatkowymi opcjami:

- Schemabinding – powoduje, że SZBD zabroni modyfikacji tabel, które mogą spowodować błąd realizacji widoku (np. usunięcie kolumny z tabeli a kolumnę tę udostępnia widok).
- Encryption – powoduje blokadę dostępu do definicji widoku.
- With Check Option – w sytuacjach, gdy widok umożliwia modyfikację danych w tabeli macierzystej i w definicji widoku występuje klauzula **WHERE** – opcja powoduje, że zmiany dokonywane poprzez widok muszą być zgodne z warunkiem zawartym w klauzuli **WHERE**.

Ćwiczenie 13. Definiowanie widoków.

Zdefiniować widok zawierający dane z tabeli uczniowie dla uczniów z klasy o wartość idklasy=1. Sprawdzić z wykorzystaniem Help składnie polecenia **CREATE VIEW** i sposoby definiowania opcji **Schemabinding**, **Encryption** oraz **WITH CHECK OPTION**. Sprawdzić, jak zachowuje się widok w zależności od zastosowanej przy jego definicji opcji.

W celu realizacji powyższych poleceń wykonujemy następujące czynności:

1. W bazie danych ElektronicznyDziennikOcen w tabeli Uczniowie dodajemy nową kolumnę o nazwie tmp typu **varchar(50)** (dopuszczamy możliwość zapisywania wartości **null**)
2. Otwieramy okno edycji i wpisujemy następujące polecenie:

```
CREATE VIEW Widok_Uczniow_Klasy
AS
SELECT Nazwisko,
       Imie,
       DataUrodzenia,
```




```

    CzyChłopak,
    Pesel,
    idklasy,
    test
FROM Uczniowie
WHERE idklasy=1

```

- Wykonujemy polecenie naciskając klawisz F5.
- W oknie Object Explorer wybieramy zdefiniowany widok i z menu uzyskanego poprzez kliknięcie prawym klawiszem myszy wybieramy opcje Design. Pojawi się okno projektanta widoków. Zamykamy to okno bez dokonywania zmian. Sprawdziliśmy jedynie, że zdefiniowany widok możemy uruchomić do edycji.
- Wykonujemy polecenie:
 Select * from Widok_Uczniow_Klasy

Powinniśmy otrzymać wynik taki, jak pokazano na rys. 50. Widać, że widok udostępnia jedynie dane uczniów, którzy są związani z klasą o wartości idklasy=1 (zgodnie z definicją widoku).

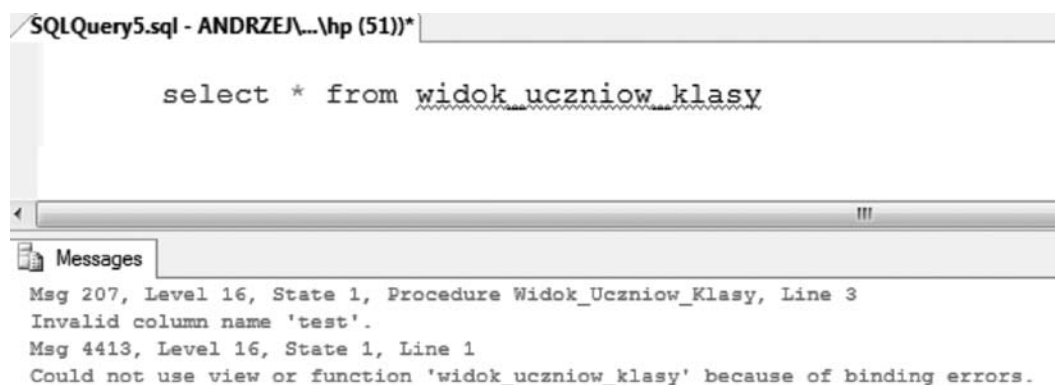
	Nazwisko	Imie	DataUrodzenia	CzyChłopak	Pesel	idklasy	test
1	Piesek	Jan	1992-05-15	1	92051587746	1	NULL
2	Gąska	Wacek	1991-03-11	1	91031199123	1	NULL
3	Krówka	Rysio	1992-05-15	1	92051577646	1	NULL
4	Zebra	Wojtek	1993-03-13	1	93030399846	1	NULL
5	Sarenka	Rysio	1992-12-12	0	92121278766	1	NULL

Rysunek 50.
Wynik zapytania

- Wykonujemy polecenie wstawienia do widoku nowego wiersza:
 Insert into Widok_Uczniow_Klasy(nazwisko,imie,DataUrodzenia,CzyChłopak,Pesel,idklasy,test)
 values('Nowak', 'Piotr', '1992-09-12',1,'92091298795',2,'cokolwiek')

Polecenie wykonuje się poprawnie (do widoku pokazującego uczniów o wartości idklasy=1 wstawiono wiersz z inną wartością tej kolumny, ale w wyniku zapytania Select* from Widok_Uczniow_Klasy, nowo wprowadzone dane nie będą widoczne).

- Z tabeli Uczniowie usuwamy kolumnę o nazwie test i wykonujemy ponownie zapytanie Select* from Widok_Uczniow_Klasy. Tym razem zapytanie spowoduje błąd, jak pokazano na rys. 51. Komunikat informuje, że nie można wykonać zapytania, ponieważ w definicji widoku jest odwołanie do kolumny o nazwie test, którą usunęliśmy z tabeli.



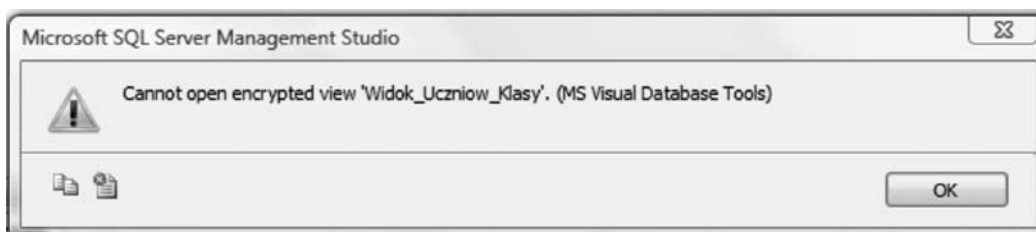
Rysunek 51.
Błąd wygenerowany przy uruchomieniu zapytania

8. Dodajemy ponownie do tabeli Uczniowie kolumnę o nazwie test typu varchar(50).
9. Przystępujemy teraz do zademonstrowania działania opcji wykorzystywanych przy definiowaniu widoku, piszemy polecenie modyfikujące definicję naszego przykładowego widoku:

```
ALTER VIEW dbo.Widok_Uczniow_Klasy
WITH SCHEMABINDING, ENCRYPTION
AS
SELECT  Nazwisko, Imie, DataUrodzenia, CzyChlopak, Pesel, idklasy, test
FROM    dbo.Uczniowie
WHERE   idklasy=1
WITH CHECK OPTION
```

W poleceniu wykorzystujemy wszystkie możliwe opcje definiowania widoku. Proszę zwrócić uwagę, że poprzedziliśmy nazwę tabeli członem dbo. – jest to nazwa domyślnego schematu, w którym są definiowane tabele. Jest to wymagane, jeżeli chcemy skorzystać z opcji przy definiowaniu widoku.

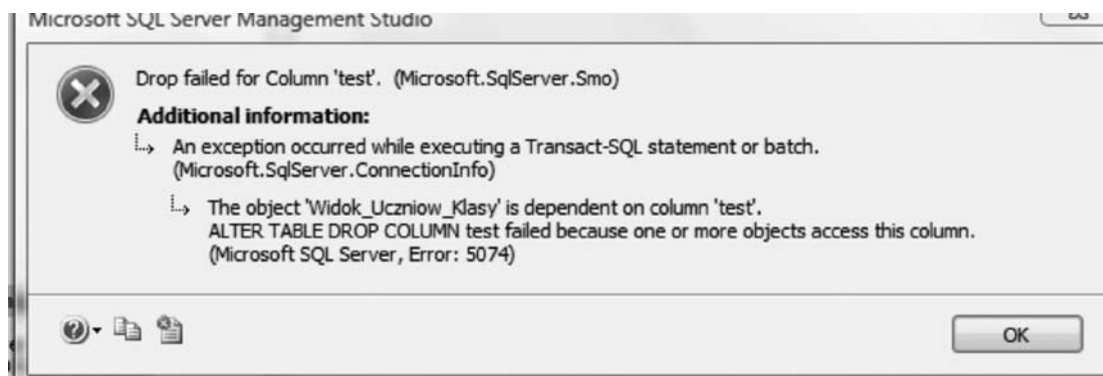
10. W oknie Obiekt Explorer wybieramy zdefiniowany widok i z menu uzyskanego poprzez kliknięcie prawym klawiszem myszy wybieramy opcję Design. Pojawi się okno z komunikatem pokazanym na rys. 52. Powodem tego komunikatu jest wykorzystanie opcji ENCRYPTION w definicji widoku, która uniemożliwia podglądanie definicji widoku.



Rysunek 52.

Okno z komunikatem

11. Usuwamy z tabeli Uczniowie kolumnę test. Operacja zostanie anulowana i pojawi się komunikat widoczny na rys. 53. System uniemożliwił usunięcie kolumny, ponieważ widok zdefiniowany z opcją SCHEMABINDING wykorzystuje tę kolumnę.



Rysunek 53.

Komunikat przy usuwaniu kolumny

12. Wykonujemy polecenie wstawienia do widoku nowego wiersza:

```
Insert into Widok_Uczniow_Klasy(nazwisko,imie,DataUrodzenia,CzyChlopak,Pesel,idklasy,test)
values('Nowak', 'Jan', '1992-07-12',1,'9207298795',2,'cokolwiek')
```

Polecenie wygeneruje błąd pokazany na rys. 54, ponieważ widok zdefiniowaliśmy z opcją WITH CHECK OPTION, czyli nie można wprowadzać do widoku danych, które są niezgodne z wyrażeniem WHERE.

```

Insert into Widok_Uczniow_Klasy(nazwisko, imie, DataUrodzenia, CzyChlopak, Pesel, idklasy, test)
values ('Nowak', 'Jan', '1992-07-12', 1, '9207298795', 2, 'cokolwiek')
    
```

Messages

Msg 550, Level 16, State 1, Line 3
 The attempted insert or update failed because the target view either specifies
 WITH CHECK OPTION or spans a view that specifies
 WITH CHECK OPTION and one or more rows resulting from the operation did not qualify under the CHECK OPTION constraint.
 The statement has been terminated.

Rysunek 54.
 Okno błędu przy zapisie nowego wiersza do widoku

13. Proszę zdefiniować widok, w przykładowej bazie danych ElektronicznyDziennikOcen, który wyświetla dane z tabeli o nazwie Oceny, zastępując klucze obce wybranymi kolumnami powiązanych tabel.

8.2. PROCEDURY SKŁADOWANE

Procedura składowana jest obiektem bazy danych zawierającym zestaw instrukcji T-SQL, przechowywanym na serwerze, zapewniającym szybszą i wydajniejszą realizację. Procedury składowane są podobne do procedur w innych językach programowania i mogą: akceptować parametry, zawierać programowe instrukcje do realizacji zadań w bazie (np. uruchamianie innych procedur), zwracać wartość stanu do procedury wywołującej, aby sygnalizować udane lub nieudane wykonanie (i przyczynę błędu) oraz zwracać wiele wartości w postaci parametrów do procedury wywołującej. Procedury składowane mają wiele zalet. Mogą one:

- Realizować logikę aplikacji dla aplikacji klienckich, zapewniając spójność danych i manipulacji nimi.
- Dostarczać logikę przetwarzania. Logika przetwarzania i założenia zawarte w procedurach składowanych mogą być zmieniane w jednym miejscu. Wszystkie aplikacje mogą używać tych samych procedur składowanych, aby zapewnić spójność modyfikacji danych.
- Dostarczać mechanizmy bezpieczeństwa. Użytkownik może mieć prawo do uruchamiania procedury składowanej nawet jeśli nie ma praw do używanych przez procedurę tabel i widoków.
- Zwiększać wydajność, gdyż są prekompilowane, a po pierwszym uruchomieniu pozostają w pamięci dla dalszych wywołań.
- Redukować obciążenie sieci. Operacje mogą być wykonywane przez przesłanie pojedynczych instrukcji, zamiast wysyłania setek wierszy kodu języka Transact-SQL.
- Procedury składowane redukują liczbę żądań wysyłanych przez klienta do serwera.

Tworzenie procedur składowanych jest podobne do tworzenia widoków. Najpierw należy zapisać i przetestować instrukcje języka Transact-SQL, które mają być umieszczone w procedurze. Jeśli uzyskane wyniki są pozytywne, to należy utworzyć procedurę. Procedury składowane tworzy się instrukcją **CREATE PROCEDURE**.

Poniższy kod tworzy procedurę składowaną, która zwraca listę ocen wystawionych uczniowi, którego identyfikator przekazano jako parametr.

```

CREATE PROCEDURE ListaOcenUcznia
    @iducznia int
AS
SELECT  przedmioty.nazwa,
        DataWystawienia,
        Ocena
FROM    Oceny JOIN Przedmioty
        ON Oceny.idprzedmiotu = Przedmioty.idprzedmiotu
WHERE   iducznia = @iducznia
    
```

Po wykonaniu powyższego polecenia w bazie danych zostanie zapisana procedura o nazwie ListaOcenUcznia. Procedury składowane uruchamia się przy użyciu instrukcji EXECUTE razem z nazwą procedury i parametrami:

```
EXECUTE ListaOcenUcznia @iducznia=1
```



Wykonanie polecenia zwróci wynik zapytania jak pokazano na rys.55.

	nazwa	DataWystawienia	Ocena
1	Infomatyka	2009-02-05	2.00
2	Geografia	2009-01-29	2.00
3	Fizyka	2009-01-24	1.00
4	Fizyka	2009-01-04	2.00
5	Matematyka	2008-11-14	5.00
6	Fizyka	2008-11-09	4.00
7	Matematyka	2008-10-20	5.00
8	Geografia	2008-10-03	2.00
9	Fizyka	2008-10-02	3.00
10	Fizyka	2008-08-29	1.00
11	Geografia	2008-08-01	4.00
12	Geografia	2009-02-07	5.00
13	Matematyka	2009-02-06	3.00
14	Fizyka	2009-02-05	1.00

Rysunek 55.

Wynik wykonania procedury ListaOcenUcznia

Ćwiczenie 14. Definiowanie procedur składowanych.

Wspólnie z prowadzącym kurs piszemy procedurę składowaną realizującą zapisywanie nowego wiersza do tabeli Uczniowie.

8.3. FUNKCJE SKŁADOWANE

Funkcje składowane, podobnie jak procedury składowane, są obiektami bazy danych zawierającymi kod w języku T-SQL. Podstawowa różnica polega na tym, że funkcje mają określony typ zwracanej wartości i mogą być wykorzystywane w zapytaniach **SELECT** (procedury nie można uruchomić w poleceniu **SELECT**). W MS SQL Server 2008 możemy definiować funkcje skalarne (zwracające jedną wartość) oraz funkcje tabelaryczne zwracające tabelę. W rozdziale 8.2 jest zawarty przykład procedury składowanej o nazwie ListaOcenUcznia, zwracającej tabelę zawierającą dane o ocenach wystawionych danemu uczniowi. Podobny przykład zdefiniujemy jako funkcję. Poniższy kod tworzy funkcję składowaną, która zwraca listę ocen wystawionych uczniowi, którego identyfikator przekazano jako parametr.

```
CREATE FUNCTION F_ListaOcenUcznia( @iducznia int )
RETURNS TABLE
AS
RETURN
(
    SELECT przedmioty.nazwa,
           DataWystawienia,
           Ocena
    FROM Oceny JOIN Przedmioty
    ON Oceny.idprzedmiotu = Przedmioty.idprzedmiotu
    WHERE iducznia = @iducznia
)
```

Po wykonaniu powyższego polecenia w bazie danych zostanie zapisana funkcja tabelaryczna o nazwie F_ListaOcenUcznia. Tabelaryczne funkcje składowane możemy wykorzystywać w zapytaniach tak jak każdą inną tabelę lub widok. Polecenie:

```
SELECT* FROM dbo.F_ListaOcenUcznia(1)
```

zwróci tabelę pokazaną na rysunku 55.



Ćwiczenie 15. Definiowanie funkcji składowanych.

Wspólnie z prowadzącym kurs piszemy funkcję składowaną zwracającą listę uczniów z klasy podanej jako parametr funkcji.

8.4. WYZWALACZE

Wyzwalacz jest specjalnym typem procedury składowanej, związanej z jedną tabelą w bazie. Jest on uruchamiany automatycznie w momencie zajścia pewnego zdarzenia na tej tabeli. Zdarzenia wyzwalające to polecenia manipulacji danych, czyli: **INSERT, UPDATE, DELETE**.

Dla pojedynczej tabeli można zdefiniować wiele wyzwalaczy, dowiązanych do różnych kombinacji operacji modyfikacji danych. Microsoft SQL Server 2008 udostępnia dwa typy wyzwalaczy dla zdarzeń języka manipulacji danymi oraz wyzwalacze (wprowadzone w wersji SQL Server 2005) reagujące na zdarzenia języka definiowania danych, czyli polecenia **CREATE, ALTER** oraz **DROP**.

Typy wyzwalaczy:

- **INSTEAD OF** – wyzwalacz jest uruchamiany zamiast zdarzenia wyzwalającego;
- **AFTER** – wyzwalacz jest uruchamiany po zajściu zdarzenia wyzwalającego.

Składnia polecenia definiującego wyzwalacz jest następująca:

```
CREATE TRIGGER nazwa_wyzwalacza
ON nazwa_tabeli
{AFTER | INSTEAD OF}
{DELETE | INSERT | UPDATE}
AS
wyrażenia SQL
```

W trakcie swego działania wyzwalacz ma dostęp do starych i nowych wartości atrybutów tabeli, do której jest przypisany (tzn. wartości sprzed wykonania i po wykonaniu zdarzenia wyzwalającego). Dostęp ten jest realizowany za pomocą tabel **DELETED** i **INSERTED**; mają one taki sam schemat jak tabela, do której przypisany jest wyzwalacz:

- tabela **DELETED** przechowuje kopie wierszy modyfikowanych podczas zdarzeń **DELETE** i **UPDATE**;
- tabela **INSERTED** przechowuje kopie wierszy modyfikowanych podczas zdarzeń **INSERT** i **UPDATE**.

Ćwiczenie 16. Definiowanie wyzwalaczy.

W ramach ćwiczenia utworzymy mechanizm automatycznego rejestrowania zmian dokonywanych w tabeli **Uczniowie**. W tym celu realizujemy następujące czynności:

1. Zdefiniować nową tabelę o nazwie **LOG_zmian** jak pokazano na rys. 56.

	Column Name	Data Type	Allow Nulls
PK	id	int	<input type="checkbox"/>
	Kto	varchar(128)	<input type="checkbox"/>
	Kiedy	datetime	<input type="checkbox"/>
	NaJakimKomputerze	varchar(128)	<input type="checkbox"/>
	WykonanaOperacja	char(1)	<input type="checkbox"/>
	iducznia	int	<input type="checkbox"/>

Rysunek 56.

Schemat tabeli **LOG_zmian**

W kolumnie o nazwie **KTO** będzie zapisywana nazwa użytkownika (nazwę tę można uzyskać za pomocą funkcji **SYSTEM_USER**), w kolumnie **Kiedy** – dokładna data i czas wykonanej operacji (czas systemowy

możemy uzyskać za pomocą funkcji GETDATE()), w kolumnie NaJakimKomputerze zapiszemy nazwę komputera, z którego wykonano daną operację, w kolumnie WykonanaOperacja zapiszemy literę określającą operację na danych (I – dla Insert, U- dla Update i D – dla Delete) oraz w kolumnie iducznia zapiszemy wartość klucza zmienianego wiersza. W tej tabeli będziemy rejestrować informacje o wszystkich zmianach dokonywanych w tabeli Uczniowie.

2. Z pomocą prowadzącego kurs, zdefiniować wyzwalacz, który będzie zapisywał do tabeli LOG_zmian niezbędne dane przy wykonywaniu zmian w tabeli Uczniowie.
3. Przetestować działanie wyzwalacza.

LITERATURA

1. Ben-Gan I., Kollar L., Sarka D., *MS SQL Server 2005 od środka: Zapytania w języku T-SQL*, APN PROMISE, Warszawa 2006
2. Coburn R., *SQL dla każdego*, Helion, Gliwice 2001
3. Rizzo T., Machanic A., Dewson R., Walters R., Sack J., Skin J., *SQL Server 2005*, WNT, Warszawa 2008
4. Szeliga M., *ABC języka SQL*, Helion, Gliwice 2002
5. Vieira R., *SQL Server 2005. Programowanie. Od Podstaw*, Helion, Gliwice 2007









W projekcie **Informatyka +**, poza wykładami i warsztatami,
przewidziano następujące działania:

- 24-godzinne kursy dla uczniów w ramach modułów tematycznych
- 24-godzinne kursy metodyczne dla nauczycieli, przygotowujące do pracy z uczniem zdolnym
 - nagrania 60 wykładów informatycznych, prowadzonych przez wybitnych specjalistów i nauczycieli akademickich
 - konkursy dla uczniów, trzy w ciągu roku
 - udział uczniów w pracach kół naukowych
 - udział uczniów w konferencjach naukowych
 - obozy wypoczynkowo-naukowe.

Szczegółowe informacje znajdują się na stronie projektu

www.informatykaplus.edu.pl