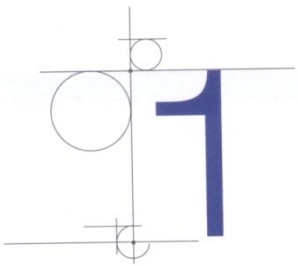


Zasady projektowania relacyjnych baz danych

1.1. Wprowadzenie do baz danych

Tradycyjne dane przechowywane są w postaci dokumentów papierowych. Dokumenty takie zawierają opisy przechowywanych obiektów. Obiektami mogą być na przykład samochody, książki lub osoby. Opisywane są także związki zachodzące między obiektami — kto jest właścicielem samochodu, kto wypożyczył książkę, gdzie pracuje dana osoba. Dokumenty opisujące obiekty i związki zachodzące między nimi są gromadzone w postaci kartotek, katalogów lub archiwów i są przechowywane w kopertach, teczkach albo segregatorach. Dostęp do tak opracowanych dokumentów jest trudny i zajmuje dużo czasu. Aby przyspieszyć wyszukiwanie danych, sporządza się różne spisy, wyciągi i katalogi.

Dobrym rozwiązaniem tych problemów jest przeniesienie takich dokumentów do komputera. Po zapisaniu danych w pamięci komputera można obsługiwać tak utworzoną bazę danych, korzystając z dostępnych narzędzi.

Zalety korzystania z komputerowych baz danych to:

- szybkie wyszukiwanie informacji,
- łatwe wykonywanie obliczeń,
- możliwość przechowywania dużej ilości danych na małej powierzchni,
- szybkie porządkowanie danych.

WSKAZÓWKA

Baza danych to uporządkowany zbiór danych z określonej dziedziny tematycznej zorganizowany w sposób ułatwiający do nich dostęp.

WSKAZÓWKA

System zarządzania bazą danych to program zarządzający danymi w bazie i umożliwiający ich przetwarzanie.

WSKAZÓWKA

System bazy danych to baza danych i system zarządzania bazą danych.

U podstaw konstruowania bazy danych leży założenie, że użytkownik, dla którego ta baza jest przeznaczona, nie musi być specjalistą z dziedziny baz danych, może w ogóle ich nie znać. Mimo to powinien bez problemów radzić sobie z obsługą zaprojektowanej bazy danych. Aby to było możliwe, podstawą tworzonej bazy musi być solidny projekt określający potrzeby użytkownika dotyczące gromadzenia, przechowywania i przetwarzania danych oraz definiujący czynności składające się na obsługę bazy danych.

1.2. Modele baz danych

Aby przechowywać dane na komputerze, konieczne jest określenie formy ich przechowywania. Na potrzeby baz danych zostały zdefiniowane klasyczne techniki organizowania informacji, zwane modelami baz danych.

Model danych to abstrakcyjny opis sposobu przedstawiania i wykorzystania danych.

Na model danych składają się:

- *struktura* — opis sposobu przedstawiania obiektów (encji) modelowanego wycinka świata oraz ich związków,
- *ograniczenia* — reguły kontrolujące spójność i poprawność danych,
- *operacje* — zbiór działań, które umożliwiają dostęp do struktur.

Głównymi modelami baz danych są:

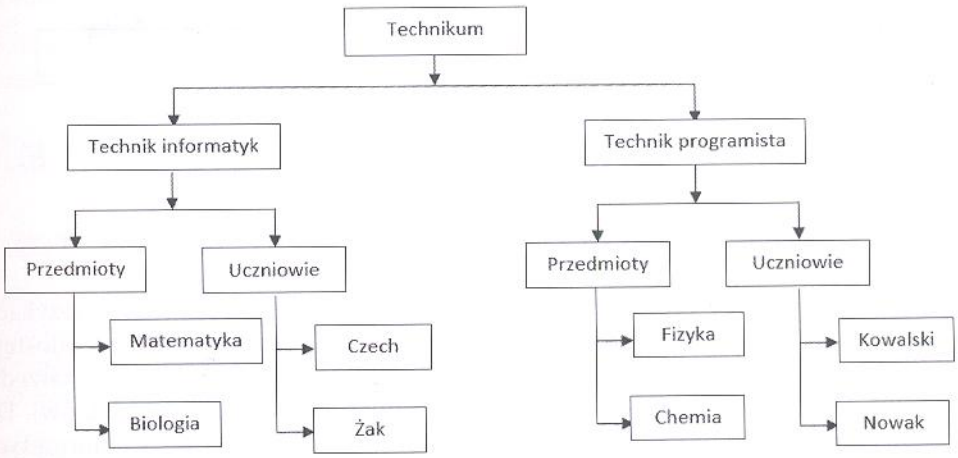
- model hierarchiczny,
- model sieciowy,
- model obiektowy,
- model relacyjny.

1.2.1. Model hierarchiczny

W modelu hierarchicznym przechowywane dane są zorganizowane w postaci odwróconego drzewa (rysunek 1.1). Każdy obiekt (z wyjątkiem obiektu podstawowego) jest połączony z jednym obiektem nadrzędnym.

Tak zbudowana baza danych umożliwia proste wyszukiwanie danych. Rozpoczyna się ono od obiektu podstawowego i poprzez rozgałęzienia oraz kolejne obiekty dochodzi się do obiektu szukanego. Informacja jest zawarta w kolejnych dokumentach oraz w strukturze drzewa (podobnej do drzewa folderów na dysku komputera).

Hierarchiczna baza danych zakłada podstawowe warunki integralności danych.

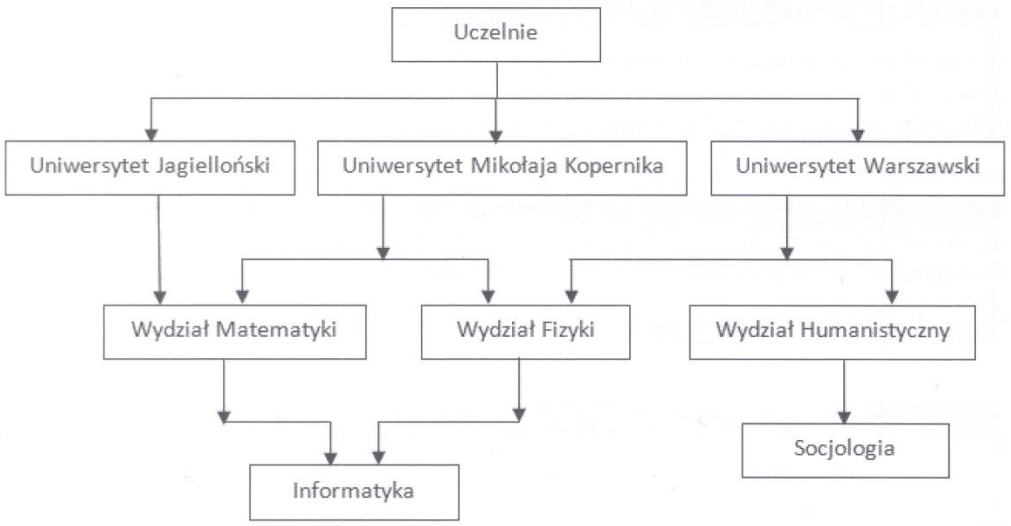


Rysunek 1.1. Model hierarchiczny

Każdy dokument (z wyjątkiem głównego — na górze) jest połączony z dokumentem nadrzędnym. Dokument podrzędny może zostać utworzony dopiero, gdy zostanie powiązany z dokumentem nadrzędnym. Kiedy zostanie usunięty dokument nadrzędny, automatycznie zostaną usunięte wszystkie dokumenty podrzędne.

1.2.2. Model sieciowy

W modelu sieciowym połączenia między dokumentami tworzą sieć (rysunek 1.2). Jest to zmodyfikowana wersja modelu hierarchicznego, w której dozwolone są połączenia na tym samym poziomie drzewa danych. Informacja jest zawarta w dokumentach oraz w przebiegu połączeń sieci.



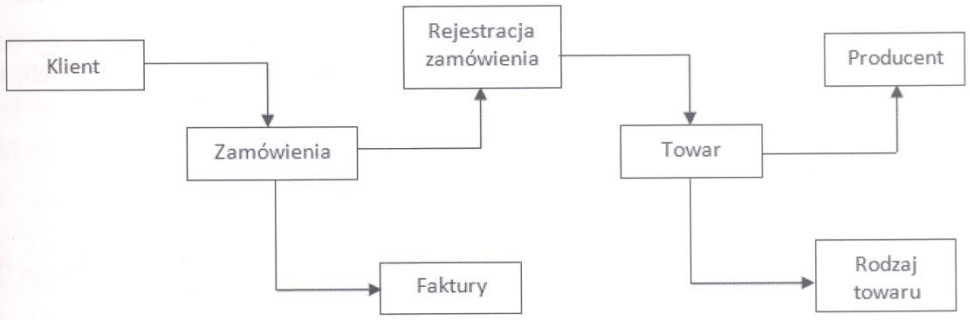
Rysunek 1.2. Model sieciowy

1.2.3. Model obiektowy

Model obiektowy łączy cechy programów komputerowych tworzonych w językach programowania obiektowego z cechami aplikacji bazodanowych. Dane są udostępniane w postaci obiektów. Ich stan i zachowanie są opisywane za pomocą narzędzi dostępnych w programowaniu obiektowym (własności, metody, klasy obiektów). Do przechowywania danych stosowane są obiekty. Obiekty obsługiwane za pomocą tych samych metod i własności są instancjami tej samej klasy.

1.2.4. Model relacyjny

Model relacyjny baz danych został oparty na matematycznym modelu organizacji danych i pojęciu relacji. W tym modelu dane są przedstawiane w postaci relacji reprezentowanych przez tabele. Relacje (tabele) składają się z rekordów (wierszy) o takiej samej strukturze. Tworzą one między sobą powiązania zwane relacjami (rysunek 1.3). Relacje (tabele) grupuje się w schematy bazy danych.



Rysunek 1.3. Model relacyjny

Ze względu na funkcjonalność model relacyjny jest najczęściej wykorzystywany przy projektowaniu baz danych.

1.3. Relacyjny model danych

1.3.1. Model relacyjny według E.F. Codda

Twórcą teorii relacyjnych baz danych jest Edgar Frank Codd. W 1970 roku opublikował on pracę pt. „Relacyjny model danych dla dużych banków danych", która wprowadzała główne założenia dotyczące modelu relacyjnego baz danych. Później zostały uszczegółowione terminy *algebra relacji* i *rachunek relacyjny*.

Według podanych założeń określenie *relacyjna baza danych* oznacza bazę zbudowaną z relacji. Podstawowy obiekt takiej bazy danych, tabela, jest reprezentacją relacji w ujęciu pojęć matematycznych. Ale terminy *relacja* i *tabela* nie są jednoznaczne, ponieważ jedna relacja może być odwzorowana za pomocą wielu różnych tabel. Nie używa się również pojęć związanych z tabelą: *kolumna* i *wiersz*, lecz *atrybut* i *krotka*.

W modelu relacyjnym przyjmuje się, że:

- kolejność wierszy i kolumn w tabelach jest nieistotna,
- wiersze zawierające takie same dane są identyczne.

Każda tabela składa się z pewnej liczby wierszy i kolumn. Na przecięciu wiersza z kolumną znajduje się pole. Pole zawiera najmniejszą niepodzielną wartość, czyli taką część informacji, która nie może być dalej dzielona ze względu na spójność logiczną.

Definicja relacji według E.F. Codda

Relacja *r* to dowolny podzbiór iloczynu kartezjańskiego jednego lub więcej zbiorów:

$r \subset D_1 \times D_2 \times \ldots \times D_k$

$D_1 \times D_2 \times \ldots \times D_k = \{(a_1, a_2, \ldots, a_k) : a_i \in D_i, i \in \{1, 2, \ldots, k\}\}$

Schematem R relacji nazywamy zbiór atrybutów $\{A_1, \ldots, A_n\}$.

Relacją r o schemacie $R = \{A_1, \ldots, A_n\}$ nazywamy skończony zbiór $r = \{t_1, \ldots, t_m\}$ odwzorowań t_i: $R \rightarrow D$, gdzie D jest równe sumie dziedzin atrybutów $A_1, \ldots, A_n$ takich, że $t_i(A_j) \in D_j$ dla $i = 1, \ldots, m, j = 1, \ldots, n$.

Każde takie odwzorowanie nazywamy krotką.

Przykład 1.1

Schematem relacji nazywamy zbiór atrybutów $R = \{A_1, A_2, \ldots, A_n\}$, gdzie $A_1, A_2, \ldots, A_n$ są atrybutami reprezentowanymi w tabeli poprzez nazwy kolumn.

Przykładowym schematem relacji *towar* będzie zbiór:

```
towar{Id_towaru, nazwa, cena, ilość}.
```

Każdemu atrybutowi $(A_1, A_2, \ldots, A_n)$ przyporządkowana jest dziedzina (zakres dopuszczalnych wartości atrybutu) reprezentowana przez typ danych.

W tym przypadku dla poszczególnych kolumn zdefiniowano następujące typy danych:

```
Id_towaru int,
nazwa varchar(64),
cena numeric(7,2),
ilość numeric(6).
```

Dziedziną relacji o schemacie $R = \{A_1, A_2, \ldots, A_n\}$ nazywamy sumę dziedzin wszystkich jej atrybutów:

$D(R)=D(A_1) \cup D(A_2) \cup \ldots \cup D(A_n)$.

Przykładową dziedziną relacji *towar* będzie dziedzina:

```
D(towar) = int ∪ varchar (64) ∪ numeric(7,2) ∪ numeric(6).
```

Relacją o schemacie $R = \{A_1, A_2, \ldots, A_n\}$ nazywamy skończony zbiór $r = \{t_1, t_2, \ldots, t_m\}$ odwzorowań t_i: $R \rightarrow D(R)$ takich, że dla każdego j z zakresu $1 <= j <= n$ zachodzi zależność:

$t_i(A_j) \in D(A_j)$.

Tak zdefiniowane pojedyncze odwzorowanie nosi nazwę krotki i odpowiada mu pojedynczy wiersz tabeli. Wartością krotki jest suma wartości poszczególnych atrybutów.

Przykładową relacją *towar* będzie:

```
komputer, 2300, 6
drukarka, 350, 13
monitor, 400, 7
klawiatura, 90, 16
```

Klucz

Kluczem schematu *R* relacji nazywamy taki zbiór atrybutów *K* tego schematu, że na podstawie wartości atrybutów z tego zbioru można jednoznacznie zidentyfikować każdą krotkę.

Właściwości klucza:

- Wartość klucza pozwala jednoznacznie identyfikować krotki.
- Dany schemat może posiadać kilka kluczy.
- Każdy nadzbiór klucza jest kluczem.
- Klucz, którego żaden podzbiór właściwy nie jest kluczem, nazywamy kluczem właściwym lub kandydującym.

Wśród kluczy wybiera się jeden, który staje się kluczem głównym (podstawowym).

Integralność danych

Integralność danych (spójność danych) oznacza poprawność struktury bazy danych (czyli zgodność ze schematem bazy danych) oraz poprawność przechowywanych w niej danych. W relacyjnym modelu danych występują następujące rodzaje więzów integralności:

- *Integralność encji* — każda relacja posiada klucz główny i żaden element klucza głównego nie może posiadać wartości pustej (*NULL*).
- *Integralność referencyjna* — każda wartość klucza obcego jest równa wartości klucza głównego określonej krotki lub wynosi *NULL*.
- *Więzy ogólne* — dodatkowe warunki dotyczące poprawności danych określane przez użytkowników lub administratorów baz danych.

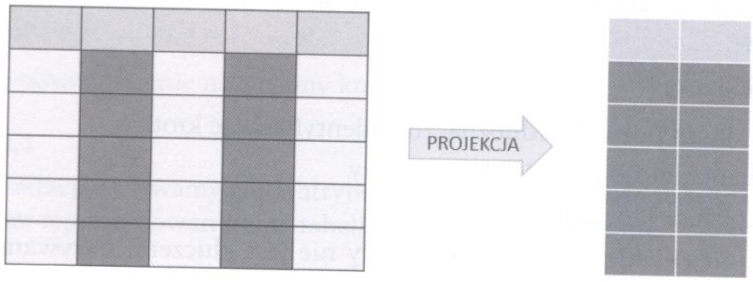
Algebra relacji

Algebra relacji zawiera zbiór operacji, które pozwalają na tworzenie potrzebnych relacji z relacji dostępnych w bazie danych.

Selekcja to wybór tych krotek relacji, które spełniają określone warunki (rysunek 1.4).

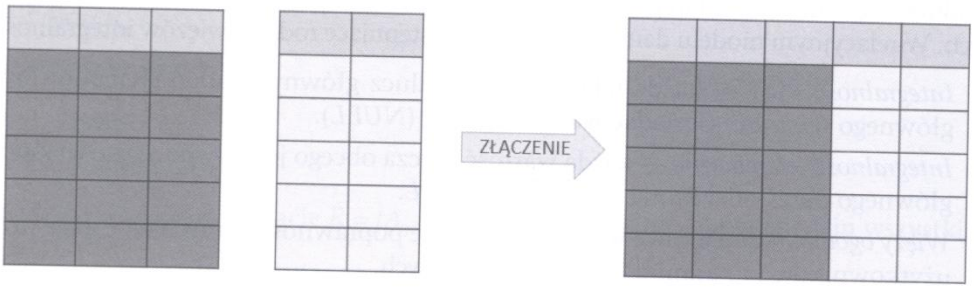
Rysunek 1.4. Wybór krotek z relacji (selekcja)

Projekcja to wybór z relacji określonych atrybutów (rysunek 1.5). Projekcja nazywana jest czasami selekcją pionową lub rzutem.



Rysunek 1.5. Wybór atrybutów z relacji (projekcja)

Złączenie to utworzenie z danych zawartych w dwóch relacjach jednej relacji (rysunek 1.6). Najczęściej łączone są relacje, które mają taki sam atrybut.



Rysunek 1.6. Złączenie dwóch relacji

Iloczynem kartezjańskim dwóch relacji — relacji r o schemacie $R = \{A_1, \ldots, A_k\}$ oraz relacji s o schemacie $S = \{B_1, \ldots, B_m\}$ takich, że $R \cap S = \emptyset$ — nazywamy relację $q = r \times s$ o schemacie $Q = R \cup S$ taką, że

$$t \in q \Leftrightarrow (\exists u \in r)(t \mid R = u) \wedge (\exists v \in s)(t \mid S = v)$$

$$t \in q \Leftrightarrow t \mid R \in r \wedge t \mid S \in s.$$

Iloczyn kartezjański dwóch relacji jest zbiorem wszystkich możliwych połączeń krotek obu relacji.

Wartość NULL

Jednym z kluczowych problemów relacyjnego modelu danych było podejście do brakującej informacji (na przykład nieznany numer telefonu lub numer domu). Ostatecznie E.F. Codd wprowadził do modelu relacyjnego dodatkową wartość: *NULL*. Wskutek tego nastąpiło rozszerzenie logiki dwuwartościowej operatorów porównania (*Tak*, *Nie*) do logiki trójwartościowej (na każde pytanie można odpowiedzieć: *Tak*, *Nie*, *Nieznane*).

Wartość *NULL* określa wartość atrybutu, która w danej chwili nie jest znana lub nie może zostać ustalona.

1.3.2. Relacyjny model baz danych

Istnieje wiele różnych podejść do relacyjnego modelu baz danych. Dwa główne to podejście formalne (opis relacyjnego modelu baz danych za pomocą reguł matematycznych) oraz podejście intuicyjne (podejście do relacyjnego modelu baz danych w sposób czysto użytkowy).

Cechy relacyjnego modelu baz danych:

- Podstawową formą przechowywania danych jest tabela (relacja).
- Użycie kombinacji „wartość klucza podstawowego, nazwa tabeli i nazwa kolumny" zapewnia dostęp do dowolnych danych.
- Manipulacje na danych są realizowane za pomocą selekcji, projekcji i złączenia.
- Musi być obsługiwana wartość *NULL* (wartość *NULL* przedstawia brakujące lub bezużyteczne informacje, na przykład nieznany numer telefonu).
- Integralność danych powinna być naturalną cechą projektu bazy danych.

Korzyści płynące z używania modelu relacyjnego:

- efektywność przechowywania danych,
- pewność integralności danych,
- możliwość rozbudowy bazy danych,
- możliwość łatwej zmiany w strukturze bazy danych,
- zwiększenie szybkości dostępu do danych.

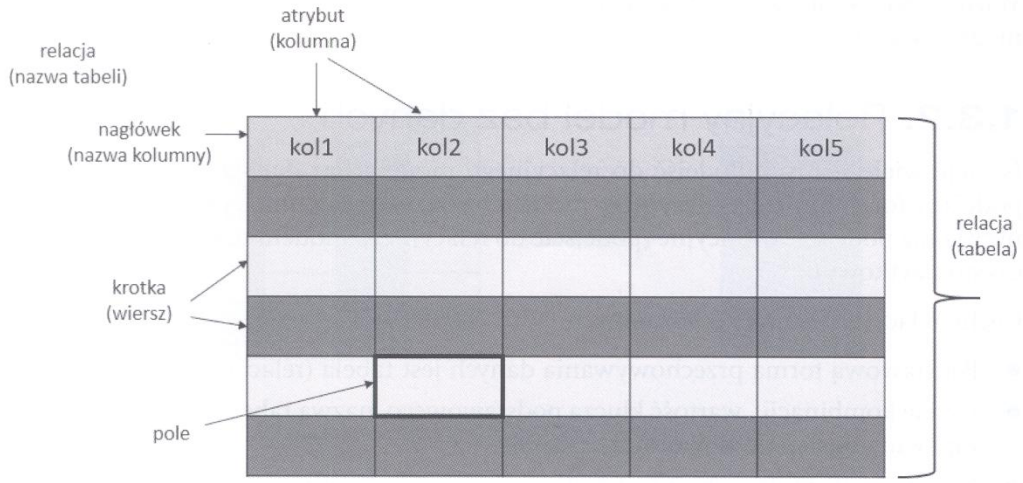
Opisując relacyjny model danych, musimy określić:

- struktury danych — czyli w jaki sposób i według jakich zasad zostanie zorganizowane przechowywanie danych oraz według jakich zasad będziemy je projektować,
- język manipulowania danymi — czyli w jaki sposób będą pobierane, zapisywane, modyfikowane i usuwane dane znajdujące się w bazie danych,
- integralność danych — czyli w jaki sposób zostanie zapewniona poprawność przechowywanych danych.

Tabele

W modelu relacyjnym baz danych wszystkie dane są przechowywane w dwuwymiarowych tabelach (relacjach). W skład bazy danych może wchodzić wiele relacji.

Tabela to zbiór powiązanych ze sobą danych. Jest to układ poziomych wierszy, nazywanych rekordami lub krotkami, i pionowych kolumn, nazywanych polami rekordu lub atrybutami. Tabela jest identyfikowana poprzez nazwę (rysunek 1.7).



Rysunek 1.7. Struktura tabeli

Przecięcie kolumny i wiersza tworzy pole. Nazwa kolumny jest jednocześnie nazwą pola. Pojęcia *pole* używa się również do określenia kolumny.

W modelu relacyjnym:

- każda tabela musi mieć jednoznaczną nazwę,
- każda kolumna w tabeli musi mieć jednoznaczną nazwę w obrębie tej tabeli,
- wszystkie wartości w kolumnie są tego samego typu,
- w tabeli nie mogą istnieć dwa identyczne wiersze,
- tabela nie może istnieć bez wierszy,
- przechowywane w tabeli dane oparte są na typach prostych (dane elementarne),
- kolejność wierszy i kolejność kolumn w tabeli nie ma żadnego znaczenia.

Klucz podstawowy

Tabela musi mieć zestaw pól, które będą jednoznacznie identyfikować dany rekord. W najlepszym przypadku w tabeli może istnieć jedno pole spełniające to założenie. To klucz podstawowy, nazywany również kluczem głównym lub pierwotnym.

WSKAZÓWKA

Klucz podstawowy to minimalna kombinacja pól identyfikująca każdy rekord w tabeli w sposób jednoznaczny.

Klucz podstawowy pozwala w sposób efektywny przeszukiwać i odczytywać dane w bazie oraz łączyć dane zapisane w różnych tabelach.

Klucz podstawowy nie może zawierać powtarzających się danych oraz nie może być pusty. Oznacza to, że w tabeli musi się znaleźć jedno lub kilka pól, które pozwolą odróżniać dane zapisane w jednym rekordzie od danych zapisanych w innym rekordzie.

Jeżeli tabela zawiera dane osobowe, takim kluczem może być kolumna z nazwiskiem. Każdą osobę zapisaną w tabeli rozpoznamy po nazwisku i odczytamy jej dane z właściwego rekordu. Co jednak zrobić, jeżeli w tabeli zapisaliśmy informacje o kilku osobach, które mają takie samo nazwisko?

W takim wypadku kluczem podstawowym może być kombinacja pól, czyli do pola *Nazwisko* możemy dodać pole *Imię*. Teraz każdą osobę w tabeli znajdziemy, podając jej nazwisko i imię. A co zrobić, jeśli w tabeli są umieszczone informacje o dwóch osobach, które mają identyczne nazwisko i imię? Można do klucza dodać kolejne pole, na przykład *Data urodzenia*.

W celu uniknięcia dodawania kolejnych pól do klucza bardzo często funkcję klucza podstawowego pełni klucz sztuczny. Najprostszym sposobem utworzenia klucza sztucznego jest dodanie do tabeli dodatkowego pola i umieszczenie w nim kolejnych numerów.

WSKAZÓWKA

Klucz sztuczny to pole zawierające unikatowy numer identyfikacyjny nadany w sposób sztuczny każdemu obiektowi umieszczonemu w tabeli.

Jeżeli w tabeli zostało utworzone takie pole, staje się ono automatycznie kluczem podstawowym, ponieważ spełnia wszystkie wymagania dotyczące tego klucza.

Klucz obcy to jedno pole lub więcej pól tabeli (kolumn), które odwołują się do pola lub pól klucza podstawowego w innej tabeli. Klucz obcy pokazuje, w jaki sposób tabele są powiązane. Jest niezbędny do zdefiniowania połączenia między tabelami.

Integralność danych

W modelu relacyjnym baz danych integralność danych zachowana jest poprzez:

- klucz podstawowy,
- klucz obcy,
- zawężenie dziedziny,
- unikatowość wartości,
- możliwość nadawania wartości pustych.

Połączenia

Projektując bazę danych, dzielimy dane na wiele tabel tematycznych, tak aby każda informacja została zapisana tylko raz. Aby zestawić razem dane zapisane w różnych tabelach, tworzy się między nimi połączenia. W wyniku powstaje nowa tabela (relacja) zawierająca połączone wiersze tabel, dla których wartości klucza obcego i klucza

podstawowego są takie same. Definiowanie logicznego połączenia między tabelami bazy danych nazywane jest również relacją.

Typy połączeń (związków)

W rzeczywistości (w życiu codziennym) występują trzy typy połączeń (związków).

Związek „jeden do jednego" (1:1)

W związku „jeden do jednego" każdemu rekordowi z pierwszej tabeli może odpowiadać tylko jeden rekord z drugiej tabeli i każdemu rekordowi z drugiej tabeli może odpowiadać tylko jeden rekord z pierwszej tabeli. Jest to nietypowy rodzaj związku, ponieważ najczęściej informacje powiązane w ten sposób są przechowywane w jednej tabeli. Opisywanego związku używa się w celu odizolowania części tabeli ze względu na bezpieczeństwo danych lub w celu podzielenia danych na podzbiory.

Związek „jeden do wielu" (1:n lub 1:∞)

W związku „jeden do wielu" każdemu rekordowi z pierwszej tabeli może odpowiadać wiele rekordów z drugiej tabeli, a każdemu rekordowi z drugiej tabeli może odpowiadać najwyżej jeden rekord z pierwszej tabeli. Jest to typ związku najczęściej występujący w relacyjnych bazach danych.

Związek „wiele do wielu" (m:n)

W związku „wiele do wielu" każdemu rekordowi z pierwszej tabeli może odpowiadać wiele rekordów z drugiej tabeli i każdemu rekordowi z drugiej tabeli może odpowiadać wiele rekordów z pierwszej tabeli.

Związek „wiele do wielu" zawsze jest realizowany jako dwa związki „jeden do wielu".

1.4. Projektowanie bazy danych

W bazie danych przechowujemy tylko niektóre informacje o świecie rzeczywistym. Wybór właściwych wycinków rzeczywistości i dotyczących ich danych jest bardzo istotny — od niego zależy prawidłowe działanie bazy. Aby ten wybór był właściwy, należy wskazać informacje, które powinny być przechowywane w bazie danych, oraz określić ich strukturę.

1.4.1. Zasady projektowania bazy danych

Cały proces projektowania bazy danych możemy podzielić na kilka etapów:

- planowanie bazy danych,
- tworzenie modelu konceptualnego (diagramy ERD),
- transformacja modelu konceptualnego na model relacyjny,
- proces normalizacji bazy danych,
- wybór struktur i określenie zasad dostępu do bazy danych.

1.4.2. Podstawowe pojęcia

Z punktu widzenia relacyjnej bazy danych świat rzeczywisty widzimy i analizujemy jako zestaw **encji** i związków zachodzących między nimi.

Encja

WSKAZÓWKA

Encją jest każdy przedmiot, zjawisko, stan lub pojęcie, czyli każdy obiekt, który potrafimy odróżnić od innych obiektów (na przykład: osoba, samochód, książka, stan pogody).

Encje podobne do siebie (opisywane za pomocą podobnych parametrów) grupujemy w zbiory encji. Projektując bazę danych, należy precyzyjnie zdefiniować encje i określić parametry, przy użyciu których będą opisywane.

Atrybut

Encje mają określone cechy wynikające z ich natury. Cechy te nazywamy atrybutami. Zestaw atrybutów, które określamy dla encji, zależy od potrzeb bazy danych.

Dziedzina

Atrybuty encji mogą przyjmować różne wartości. Projektując bazę danych, możemy określić, jakie wartości może przyjmować dany atrybut. Zbiór wartości atrybutu nazywamy dziedziną (domeną).

1.4.3. Planowanie bazy danych

Prostym przykładem bazy danych jest lista klientów sklepu internetowego zajmującego się sprzedażą książek. Zawiera ona zbiór informacji o wycinku otaczającej nas rzeczywistości, czyli o klientach dostępnego w internecie sklepu oraz o oferowanych książkach. Encją jest klient. Zbiór klientów tworzy zbiór encji.

Klient jest encją i jako encja jest określany przez kilka atrybutów (nazwisko, imię, adres zamieszkania, PESEL). Dla każdego atrybutu można ustalić zbiór wartości, na przykład atrybut *Nazwisko* to zbiór wszystkich kombinacji liter alfabetu łacińskiego. W rzeczywistości zbiór nazwisk jest podzbiorem tego zbioru, ponieważ niektóre kombinacje liter nigdy nie wystąpią w nazwisku.

Podobnie *Książka* jest encją i jest określana przez atrybuty: tytuł, autor, cena, rok wydania, wydawnictwo, miejsce wydania, język.

Ocena, które z atrybutów encji są istotne, wynika z przeznaczenia bazy danych. Należy zwrócić uwagę, czy w bazie nie zostały umieszczone atrybuty, które tylko pozornie

należą do danej encji, a naprawdę są atrybutami innej encji. Załóżmy, że w bazie danych trzeba określić narodowość autora książki. Jeżeli atrybut *Narodowość* zostanie dodany do encji *Książka*, nastąpi pomieszanie atrybutów, ponieważ narodowość jest atrybutem autora, a nie książki.

Można wymienić następujące niekorzystne skutki wspomnianego pomieszania informacji:

- Narodowość autora jest powtórzona tyle razy, ile książek tego autora znajdzie się w bazie.
- Jeżeli w bazie nie została umieszczona jeszcze żadna książka danego autora, nie ma o nim informacji w tej bazie.
- Znalezienie informacji o dowolnym autorze staje się wyjątkowo żmudne.
- Występują problemy z poprawianiem błędów popełnionych w czasie wprowadzania danych.

Ponieważ trudno uniknąć błędów podczas wprowadzania danych, należy odpowiednio zabezpieczyć się przed nimi na etapie projektowania bazy danych. Jeśli na przykład pojawią się różne narodowości dla tego samego autora, jak rozstrzygnąć, która jest właściwa? Jak usunąć ten błąd?

Właściwym postępowaniem jest zaprojektowanie oddzielnej encji dla autora. Każdy autor w tym zbiorze encji występuje raz, a jego atrybutami, oprócz nazwiska i imienia, mogą być: narodowość, okres tworzenia, rodzaj twórczości. Oto zalety tego rozwiązania:

- Wyszukiwanie autora i jego danych jest proste.
- Każdy autor występuje na liście tylko raz i jeżeli nawet jego książek na razie nie ma w bazie, to autor pojawia się na liście.
- Ewentualne błędy występują tylko jednokrotnie i łatwo je poprawić.

Należy pamiętać, że prawidłowa klasyfikacja danych jest podstawą dobrego projektu bazy danych. Najgorsze efekty otrzymujemy, próbując zaprojektować bazę danych, która będzie gromadziła wszystkie możliwe dane i przetwarzała je na wszystkie możliwe sposoby.

Każda encja powinna mieć przynajmniej jeden atrybut lub kombinację kilku atrybutów, które identyfikują ją jednoznacznie. Ten atrybut to **klucz podstawowy** encji.

Projektując bazę danych na przykład dla klientów sklepu, widzimy, że tym atrybutem będzie nazwisko klienta. Aby uniknąć niejasności, gdy w bazie pojawi się kilku klientów o tym samym nazwisku, a czasami również imieniu, możemy przyjąć, że kluczem podstawowym będzie kombinacja atrybutów *Nazwisko*, *Imię* oraz *PESEL*. Jeśli klucz podstawowy składa się z kilku atrybutów, dobrym rozwiązaniem jest zastąpienie go **kluczem sztucznym**. Najczęściej zawiera on unikatowe liczby przypisane kolejnym encjom.

Projektowanie modelu bazy danych powinno składać się z następujących działań:

- określenie występujących zbiorów encji,
- określenie atrybutów przypisanych do poszczególnych encji,
- określenie dziedziny poszczególnych atrybutów,
- ustalenie kluczy podstawowych,
- określenie typów występujących związków,
- zweryfikowanie utworzonego modelu.

1.4.4. Tworzenie modelu konceptualnego (diagramy ERD)

Konceptualne projektowanie bazy danych to konstruowanie schematu danych niezależnego od wybranego modelu danych, docelowego systemu zarządzania bazą danych, programów użytkowych czy języka programowania.

Do tworzenia modelu graficznego schematu bazy danych wykorzystywane są diagramy związków encji, z których najpopularniejsze są diagramy ERD (ang. *Entity Relationship Diagram*). Pozwalają one na modelowanie struktur danych oraz związków zachodzących między tymi strukturami. Nadają się szczególnie do modelowania relacyjnych baz danych, ponieważ umożliwiają prawie bezpośrednie przekształcenie diagramu w schemat relacyjny. Pozwalają na analizę struktury bazy danych, mogą też stanowić część dokumentacji tworzonego systemu baz danych.

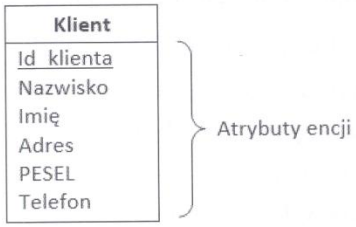
Na diagramy ERD składają się trzy rodzaje elementów:

- zbiory encji,
- atrybuty encji,
- związki zachodzące między encjami.

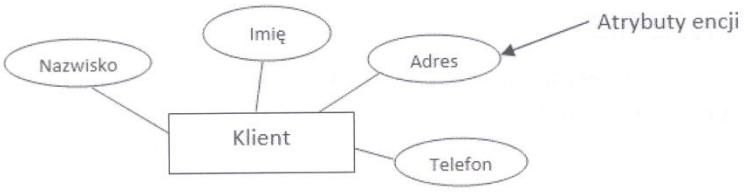
Encja to reprezentacja obiektu przechowywanego w bazie danych. Graficzną reprezentacją encji jest najczęściej prostokąt (rysunek 1.8).

Rysunek 1.8. Graficzna reprezentacja encji

Atrybut opisuje encję. Może on być liczbą, tekstem lub wartością logiczną. W relacyjnym modelu baz danych atrybut jest reprezentowany przez kolumnę tabeli. Graficzna reprezentacja atrybutów dla encji *Klient* została pokazana na rysunkach 1.9 i 1.10.



Rysunek 1.9. Atrybuty encji Klient

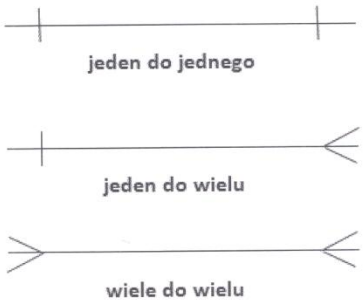


Rysunek 1.10. Atrybuty encji Klient

Związek to powiązanie między dwoma zbiorami encji. Każdy związek ma dwa końce, do których są przypisane następujące atrybuty:

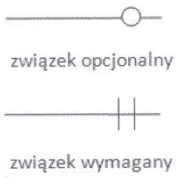
- nazwa,
- stopień związku,
- uczestnictwo lub opcjonalność związku.

Stopień związku określa, jakiego typu związek zachodzi między encjami. „Jeden do jednego" oznacza, że encji odpowiada dokładnie jedna encja. „Jeden do wielu" oznacza, że encji odpowiada jedna lub kilka encji. „Wiele do wielu" oznacza, że encji lub kilku encjom odpowiada jedna lub kilka encji. Opis reprezentacji graficznej stopnia związku został pokazany na rysunku 1.11.



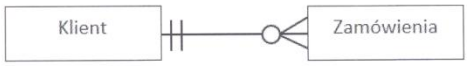
Rysunek 1.11. Graficzna reprezentacja związków zachodzących między encjami

Opcjonalność związku określa, czy związek jest opcjonalny, czy wymagany. Opis reprezentacji graficznej opcjonalności związku został pokazany na rysunku 1.12.



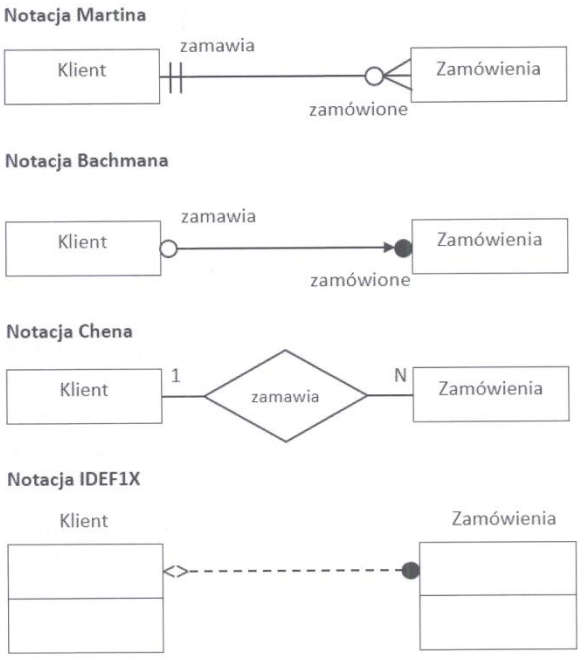
Rysunek 1.12. Graficzna reprezentacja opcjonalności związku

Przykład prostego diagramu związków encji został pokazany na rysunku 1.13. *Zamówienie* musi mieć przypisanego *Klienta*, natomiast *Klient* może złożyć *Zamówienie*. *Klient* może złożyć wiele *Zamówień*, ale złożone *Zamówienie* dotyczy tylko jednego *Klienta*.



Rysunek 1.13. Diagram związków encji

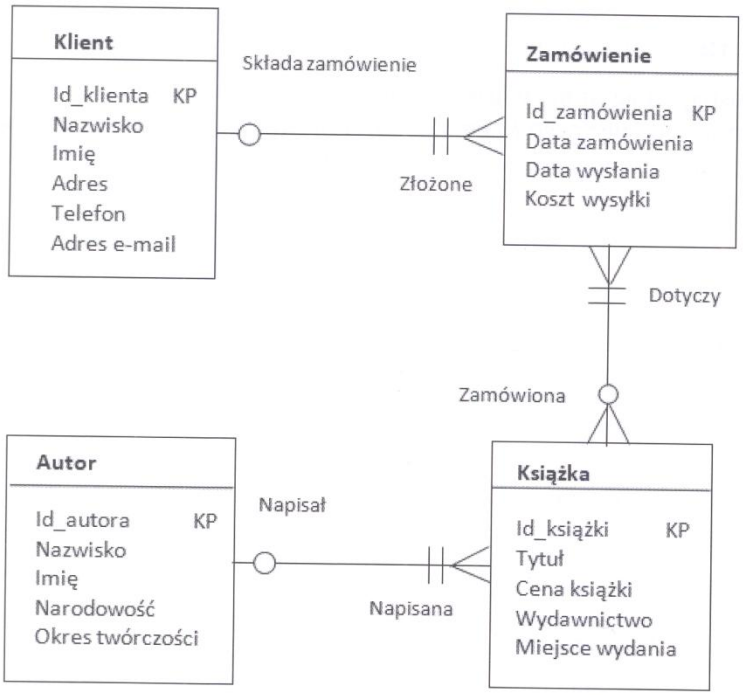
Diagramy ERD spotyka się w wielu różnych notacjach, na przykład: Martina, Bachmana, Chena, IDEF1X (rysunek 1.14).



Rysunek 1.14. Diagram związków encji zapisany w różnych notacjach

Istnieje wiele narzędzi wspomagających rysowanie diagramów ERD, ale jedynie w przypadku narzędzi klasy CASE (ang. *Computer Aided Software Engineering*) można mówić o określonej notacji. Narzędzia CASE są wykorzystywane do projektowania oprogramowania. Pozwalają tworzyć model graficzny oraz poprzez generowanie gotowych skryptów wspomagają wytwarzanie oprogramowania.

Prosty przykład diagramu ERD w notacji Martina dla księgarni został przedstawiony na rysunku 1.15.



Rysunek 1.15. Diagram ERD w notacji Martina dla księgarni

W pokazanym na rysunku schemacie encje zostały przedstawione za pomocą prostokątów zawierających listę atrybutów. Klucze główne zostały oznaczone symbolem KP. Stopień związku i uczestnictwo zostały oznaczone liniami łączącymi z odpowiednimi symbolami opisującymi stopień oraz opcjonalność związku. Należy zwrócić uwagę na to, że w encjach nie umieszcza się kluczy obcych. Zostaną one dodane na etapie przekształcania encji w tabele.

Tak przygotowany diagram ERD pozwala na późniejszą weryfikację i optymalizację bazy danych, a także stanowi podstawową dokumentację projektowanej bazy danych. Można go również wykorzystać w jednym z narzędzi CASE do wygenerowania fizycznej struktury bazy danych.

1.4.5. Projektowanie bazy danych za pomocą narzędzi CASE

Narzędzia CASE (ang. *Computer Aided Software Engineering*) są wykorzystywane podczas projektowania różnego rodzaju oprogramowania, najczęściej wspomagają proces jego wytwarzania.

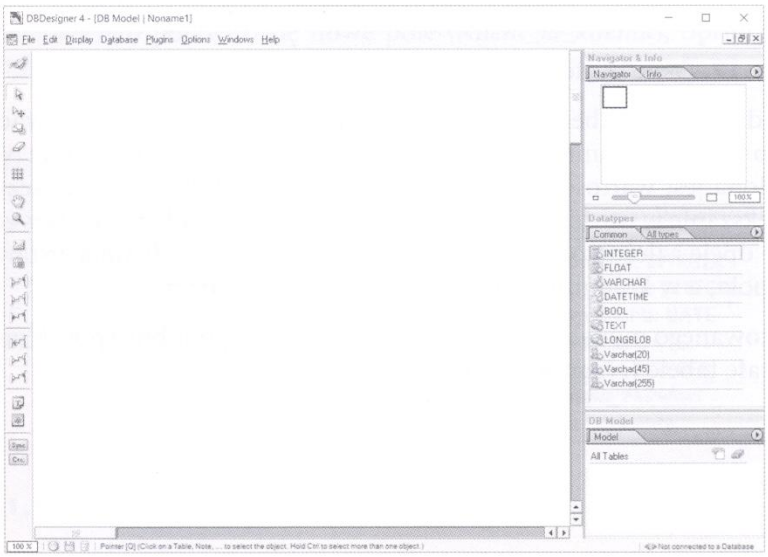
Narzędzia te pozwalają tworzyć modele graficzne odpowiadające konstrukcjom programistycznym. Wykorzystywane są tutaj edytory notacji graficznych, które dają możliwości tworzenia diagramów i powiązań między poszczególnymi elementami. Bardziej zaawansowane edytory umożliwiają przetwarzanie informacji i udostępnianie danych do aplikacji zewnętrznych, na przykład kodów w językach Visual Basic, SQL, ODBC.

Narzędzia CASE mogą być stosowane do generowania kodu na podstawie zaprojektowanego modelu danych, można również za ich pomocą, na podstawie analizy kodu źródłowego, odtworzyć projekt i specyfikację bazy danych.

Przykładem narzędzia typu CASE jest program DBDesigner4. Jest to narzędzie do wizualnego projektowania, modelowania i tworzenia baz danych. Program został stworzony z myślą o bazie MySQL, ale obsługuje również bazy danych Oracle, SQLite, MS SQL. Jest rozpowszechniany jako open source i jest dostępny na stronie *http://fabforce.net/index.php*.

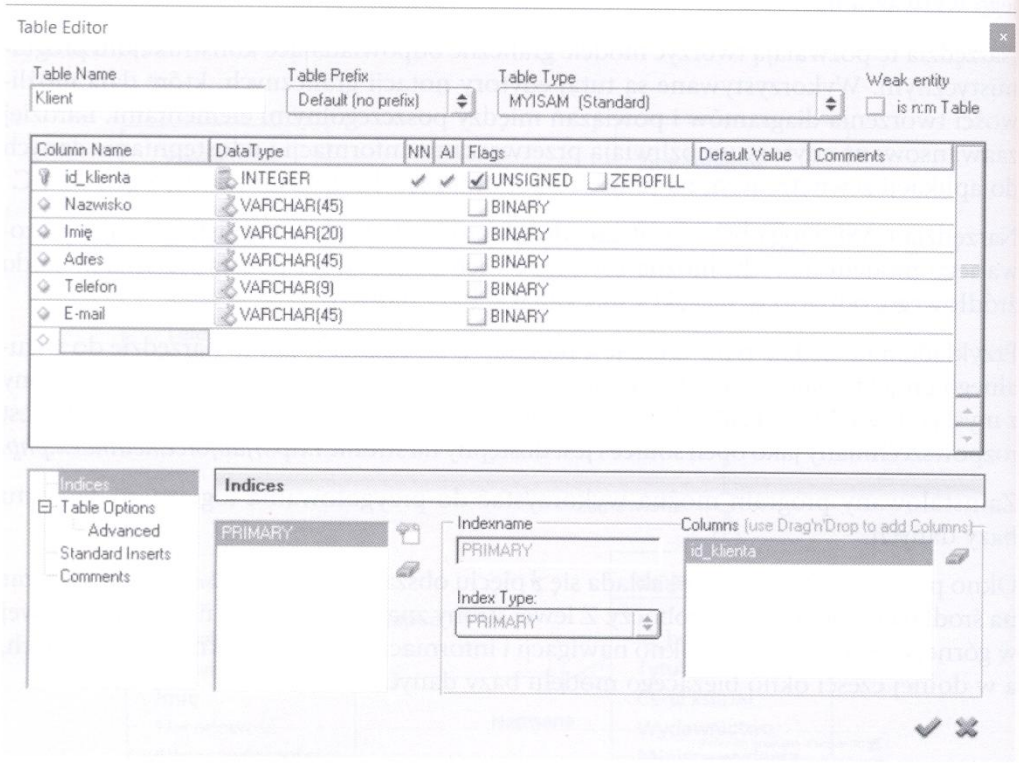
Zainstalowany program można wykorzystać do przygotowania logicznego projektu bazy danych.

Okno programu DBDesigner4 składa się z pięciu obszarów (rysunek 1.16). Pusty obszar na środku ekranu to obszar roboczy. Z lewej strony znajduje się pasek narzędzi. Z prawej w górnej części znajduje się okno nawigacji i informacji, na środku okno typów danych, a w dolnej części okno bieżącego modelu bazy danych.



Rysunek 1.16. Okno programu DBDesigner4

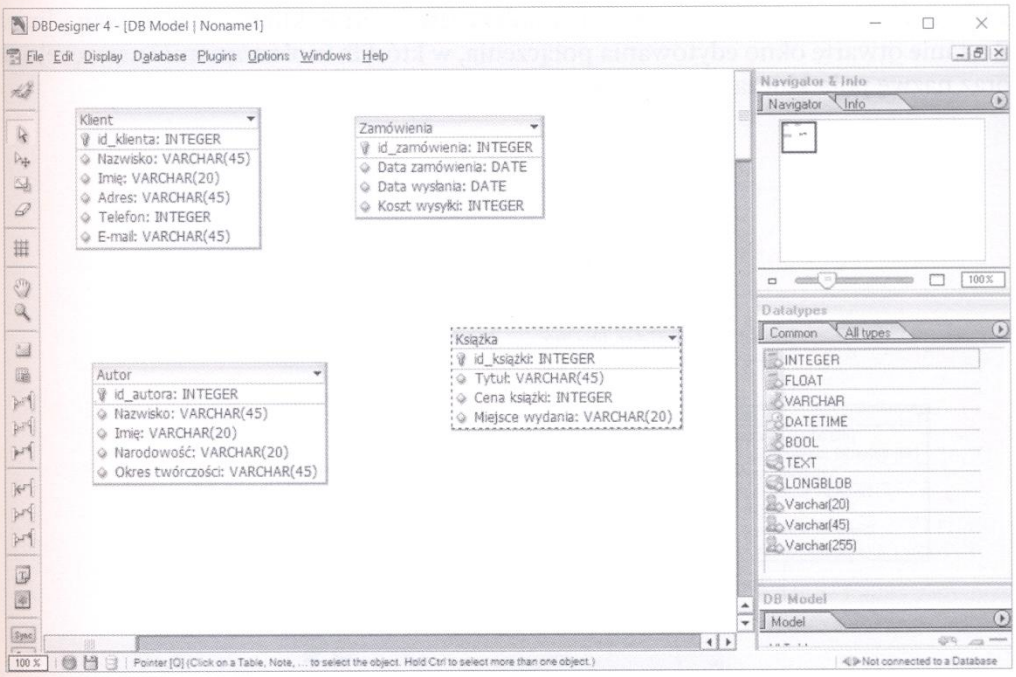
Pracę w programie rozpoczynamy od utworzenia nowego projektu (menu *File/New*). Kolejnym etapem jest utworzenie pierwszej tabeli. Należy wybrać na pasku narzędzi ikonę tworzenia tabeli (*New Table*) i kliknąć obszar roboczy. W efekcie zostanie utworzona tabela. Po dwukrotnym kliknięciu tabeli można otworzyć okno jej edytowania, w którym należy wprowadzić nazwę tabeli i nazwy pól oraz wybrać ich typy (rysunek 1.17).



Rysunek 1.17. Okno edytowania tabeli

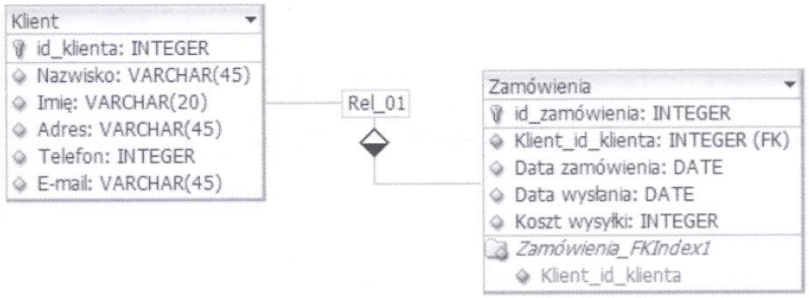
W oknie edytowania tabeli w kolumnie *Column Name* należy wprowadzić nazwę tworzonego pola, w kolumnie *DataType* określić typ danych, w kolumnie *NN* określić, czy dozwolona jest wartość *NULL* (*NOT NULL*), w kolumnie *AI* zaznaczyć automatyczne zwiększanie wartości o 1 (*AUTO INCREMENT*), w kolumnie *Flags* zdefiniować dodatkowe opcje zależne od typu danych, w kolumnie *Default Value* ustawić wartość domyślną pola, a w kolumnie *Comments* wstawić komentarz.

Dla projektowanego modelu graficznego bazy danych w podobny sposób należy utworzyć pozostałe tabele (rysunek 1.18).



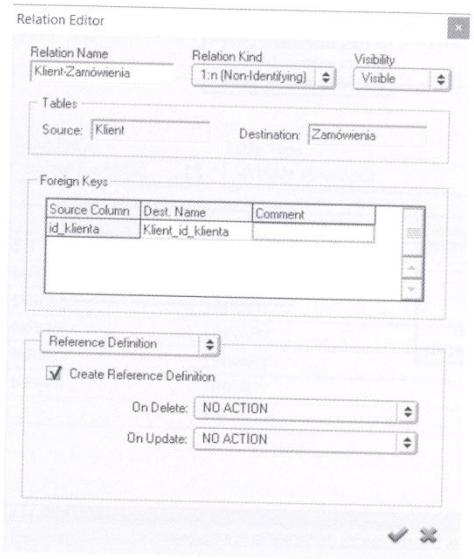
Rysunek 1.18. Tabele zaprojektowane w programie DBDesigner4

Po utworzeniu wszystkich tabel należy zdefiniować połączenia między nimi. Program DBDesigner4 obsługuje wszystkie rodzaje połączeń występujących w bazie danych. Ikony odpowiednich połączeń są dostępne na pasku narzędzi. Aby dodać połączenie typu „jeden do wielu", należy wybrać ikonę *New 1:n*. Po wybraniu ikony rodzaju połączenia klikamy najpierw tabelę ze strony „jeden" (*Klient*), a następnie tabelę ze strony „wiele" (*Zamówienia*). W wyniku zdefiniowania połączenia w tabeli ze strony „wiele" (*Zamówienia*) pojawiło się nowe pole (*Klient_id_klienta*), opisujące związek między tabelami, które stanie się kluczem obcym (rysunek 1.19).



Rysunek 1.19. Definiowanie połączenia jeden do wielu

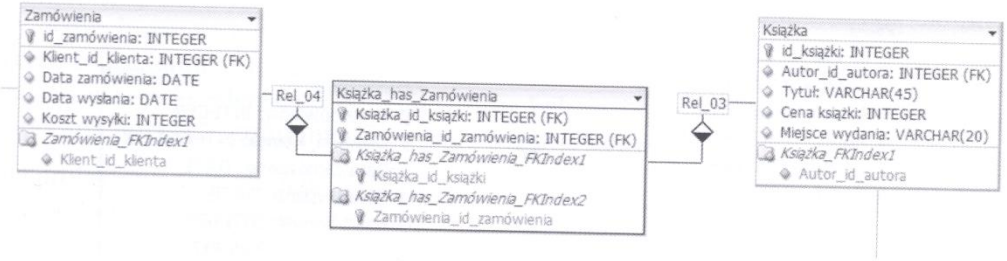
Aby edytować utworzone połączenie, należy dwukrotnie kliknąć narysowaną linię. Zostanie otwarte okno edytowania połączenia, w którym można zmienić nazwę relacji oraz nazwę pola klucza obcego (rysunek 1.20).



Rysunek 1.20. Okno edytowania połączenia

W podobny sposób należy zdefiniować połączenie między tabelami *Autor* i *Książka*.

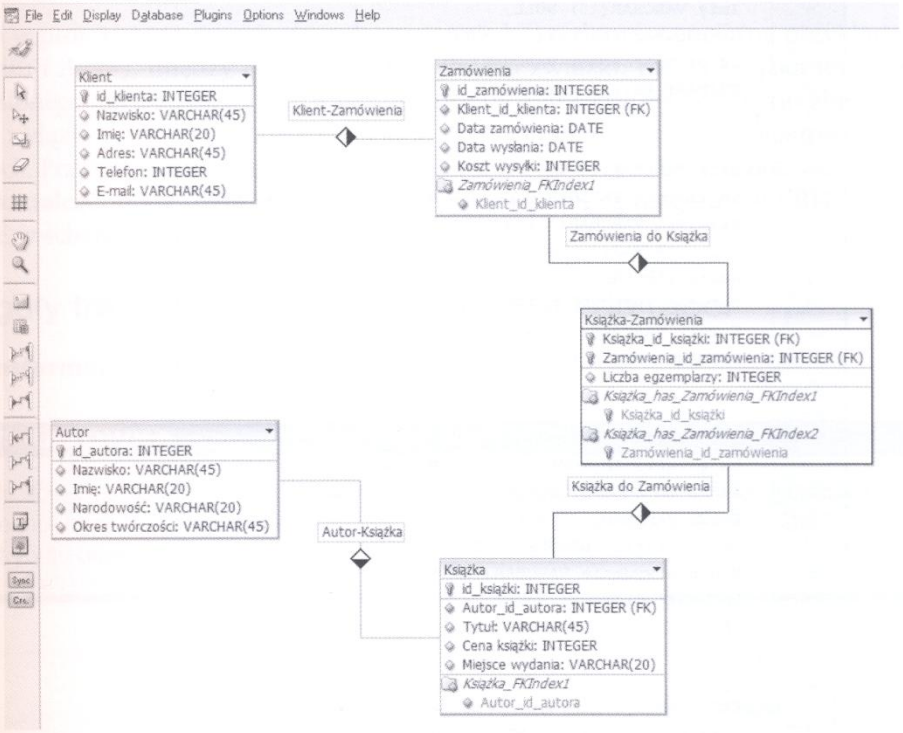
Ponieważ w jednym zamówieniu może znaleźć się kilka książek, a jedna książka może zostać wskazana w wielu zamówieniach, między tabelami *Zamówienia* i *Książki* występuje połączenie „wiele do wielu". Aby dodać taki typ połączenia, należy wybrać ikonę *New n:m*. Po wybraniu tej ikony klikamy na przykład tabelę *Książka*, a następnie tabelę *Zamówienia* (lub w odwrotnej kolejności). W wyniku zdefiniowania połączenia w projekcie zostanie umieszczona dodatkowa tabela opisująca zdefiniowane połączenie. Będzie ona zawierała klucze podstawowe łączonych tabel (rysunek 1.21).



Rysunek 1.21. Definiowanie połączenia „wiele do wielu"

Do utworzonej tabeli można dodać pola będące atrybutami połączenia, np. gdy chcemy przechowywać informację, w ilu egzemplarzach została zamówiona każda książka.

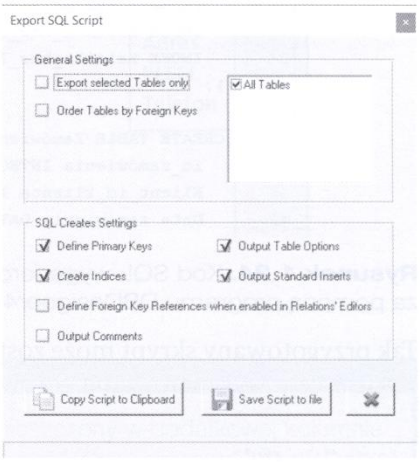
Po utworzeniu połączeń i dodaniu potrzebnych atrybutów uzyskamy efekt podobny do pokazanego na rysunku 1.22.



Rysunek 1.22. Schemat bazy danych uzyskany w programie DBDesigner4

Utworzony projekt należy zapisać w pliku, wybierając z menu *File/Save*. Można również zaprojektowaną bazę danych wyeksportować do pliku *.sql*. W tym celu należy wybrać z menu *File/Export/SQL Create Script* i w otwartym oknie zaznaczyć opcje, tak jak pokazano na rysunku 1.23, a następnie kliknąć przycisk *Save Script to file*.

Po wykonaniu tych czynności zostanie wygenerowany skrypt, którego zawartość można zobaczyć, otwierając plik na przykład w edytorze tekstowym Notepad++ (rysunek 1.24).



Rysunek 1.23. Opcje eksportowania projektu bazy do kodu SQL

```
CREATE TABLE Autor (
  id_autora INTEGER UNSIGNED NOT NULL AUTO_INCREMENT,
  Nazwisko VARCHAR(45) NULL,
  Imię VARCHAR(20) NULL,
  Narodowość VARCHAR(20) NULL,
  Okres twórczości VARCHAR(45) NULL,
  PRIMARY KEY(id_autora)
);

CREATE TABLE Klient (
  id_klienta INTEGER UNSIGNED NOT NULL AUTO_INCREMENT,
  Nazwisko VARCHAR(45) NULL,
  Imię VARCHAR(20) NULL,
  Adres VARCHAR(45) NULL,
  Telefon INTEGER UNSIGNED NULL,
  E-mail VARCHAR(45) NULL,
  PRIMARY KEY(id_klienta)
);

CREATE TABLE Książka (
  id_książki INTEGER UNSIGNED NOT NULL AUTO_INCREMENT,
  Autor_id_autora INTEGER UNSIGNED NOT NULL,
  Tytuł VARCHAR(45) NULL,
  Cena książki INTEGER UNSIGNED NULL,
  Miejsce wydania VARCHAR(20) NULL,
  PRIMARY KEY(id_książki),
  INDEX Książka_FKIndex1(Autor_id_autora)
);

CREATE TABLE Książka-Zamówienia (
  Książka_id_książki INTEGER UNSIGNED NOT NULL,
  Zamówienia_id_zamówienia INTEGER UNSIGNED NOT NULL,
  Liczba egzemplarzy INTEGER UNSIGNED NULL,
  PRIMARY KEY(Książka_id_książki, Zamówienia_id_zamówienia),
  INDEX Książka_has_Zamówienia_FKIndex1(Książka_id_książki),
  INDEX Książka_has_Zamówienia_FKIndex2(Zamówienia_id_zamówienia)
);

CREATE TABLE Zamówienia (
  id_zamówienia INTEGER UNSIGNED NOT NULL AUTO_INCREMENT,
  Klient_id_klienta INTEGER UNSIGNED NOT NULL,
  Data zamówienia DATE NULL,
```

Rysunek 1.24. Kod SQL wygenerowany automatycznie za pomocą programu DBDesigner4

Tak przygotowany skrypt może zostać uruchomiony na jednym z serwerów bazodanowych w celu wygenerowania gotowej bazy danych.

1.4.6. Transformacja modelu konceptualnego do modelu relacyjnego

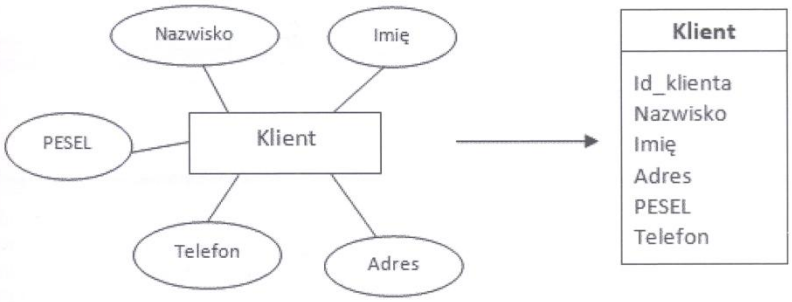
Jeżeli w projektowanej bazie danych występuje kilka zbiorów encji (na przykład encje *Książka* i *Autor*), między tymi encjami zachodzą związki. Każda książka jest dziełem określonego autora, każdy autor ma przypisany zbiór książek. Zarówno zbiory encji, jak i związki zachodzące między encjami zapisujemy w tabelach projektowanej bazy danych. Przy projektowaniu tabel stosujemy reguły, które określają, w jaki sposób należy przekształcać model bazy danych utworzony za pomocą diagramów ERD i narzędzi CASE w schemat bazy danych.

Reguły transformacji diagramów ERD do tabel

Transformacja encji do tabeli

WSKAZÓWKA

Do opisu każdego zbioru podobnych encji stosuje się oddzielną tabelę (rysunek 1.25). Jednej encji odpowiada jeden wiersz. Atrybutowi odpowiada kolumna. Dla każdego atrybutu określa się typ informacji.

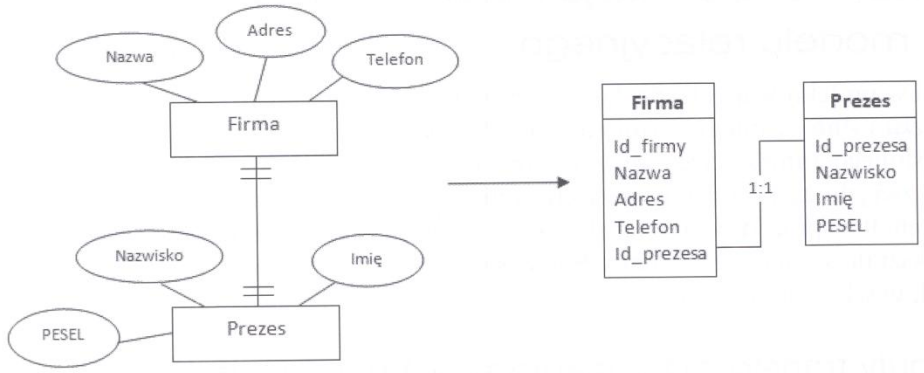


Rysunek 1.25. Transformacja encji do tabeli

Transformacja związku „jeden do jednego"

WSKAZÓWKA

Zapis związku „jeden do jednego" może zostać umieszczony w dodatkowej kolumnie w tabeli należącej do związku. Kolumna ta może znaleźć się w dowolnej tabeli. Dołączona kolumna zawiera klucz tabeli, z którą zachodzi związek (rysunek 1.26).

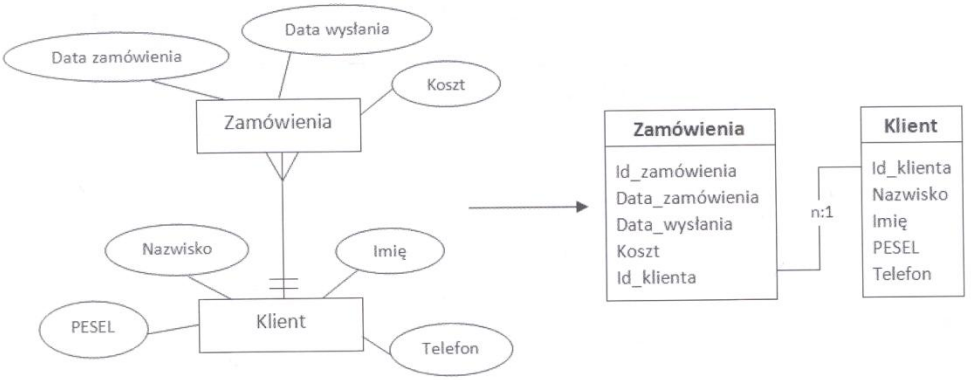


Rysunek 1.26. Transformacja związku „jeden do jednego" do tabeli

Transformacja związku „wiele do jednego"

WSKAZÓWKA

Zapis związku „wiele do jednego" może być umieszczony w dodatkowej kolumnie w tabeli należącej do związku. Kolumna ta musi znaleźć się w tabeli ze strony „wiele". Dołączona kolumna zawiera klucz tabeli, z którą zachodzi związek (rysunek 1.27).

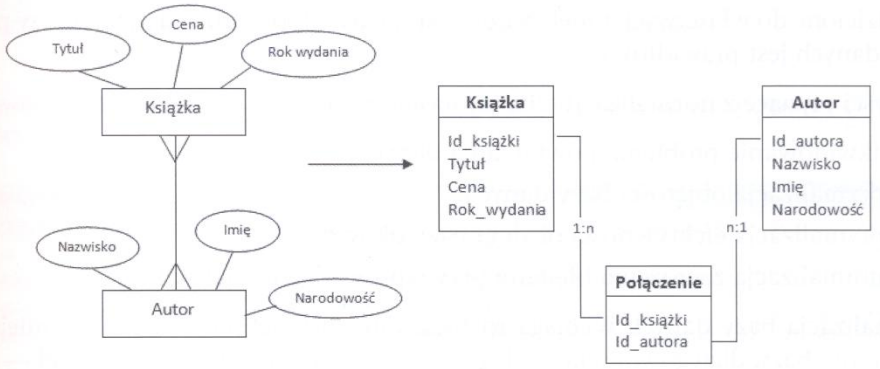


Rysunek 1.27. Transformacja związku „wiele do jednego" do tabeli

Transformacja związku „wiele do wielu”

WSKAZÓWKA

Związek „wiele do wielu” opisuje się w oddzielnej tabeli, której kolumny tworzone są z kluczy encji należących do związku (rysunek 1.28).

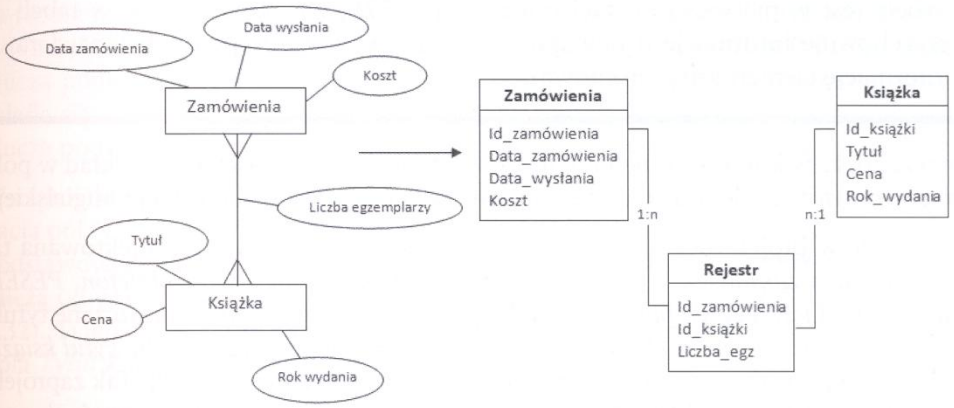


Rysunek 1.28. Transformacja związku „wiele do wielu” do tabeli

Transformacja związku „wiele do wielu” z atrybutami

WSKAZÓWKA

Jeżeli związek „wiele do wielu” posiada atrybuty, należy je umieścić w tabeli opisującej związek (rysunek 1.29).



Rysunek 1.29. Transformacja związku „wiele do wielu” z atrybutami

Jeśli klucze w tabeli opisującej związek składają się z wielu atrybutów lub są długie, należy zastąpić je kluczami sztucznymi.

1.4.7. Normalizacja tabel

Normalizację stosuje się, aby sprawdzić, czy zaprojektowane tabele mają prawidłową strukturę. Proces normalizacji rozpoczynamy, gdy zostanie utworzony wstępny projekt tabel. Pozwala ona określić, czy informacje przewidziane w projekcie bazy zostały przydzielone do właściwych tabel. Natomiast nie da odpowiedzi na pytanie, czy projekt bazy danych jest prawidłowy.

Korzyści płynące z normalizacji tabel są następujące:

- zlikwidowanie problemu powtarzania danych,
- optymalizacja objętości bazy danych,
- optymalizacja efektywności obsługi bazy danych,
- minimalizacja zagrożenia błędami przy wprowadzaniu danych.

Normalizacja bazy danych wymaga rozbicia dużych tabel na mniejsze. Zmniejsza to wydajność bazy, dlatego w niektórych przypadkach nie normalizuje się tabel — szczególnie w systemach niekorzystających z modelu relacyjnego.

Stosowane są cztery reguły normalizacji, ale w większości projektów baz danych wystarczy sprawdzić trzy pierwsze. Zostaną one omówione poniżej.

Dla każdej z nich stosowane są określenia: pierwsza postać normalna (**I PN**), druga postać normalna (**II PN**) i trzecia postać normalna (**III PN**).

Pierwsza postać normalna

DEFINICJA

Tabela jest w pierwszej postaci normalnej (**I PN**), gdy każdy wiersz w tabeli przechowuje informacje o pojedynczym obiekcie, a każde pole tabeli zawiera informację elementarną (atomową).

Oznacza to, że w komórce tabeli nie może wystąpić lista wartości, na przykład w polu *Narodowość autora* nie można umieścić dwóch narodowości — polskiej i angielskiej.

Załóżmy, że w projektowanej bazie danych dla księgarni została zaprojektowana tabela *Realizacja zamówień* z polami: *Nazwisko klienta*, *Imię*, *Adres*, *Telefon*, *PESEL*, *Tytuł książki*, *Liczba egzemplarzy*, *Cena*. W polu *Tytuł książki* będą umieszczane tytuły książek zakupionych przez klienta. Gdy klient kupi dwie książki, w polu *Tytuł książki* należałoby wpisać dwa tytuły. Powstałaby lista wartości (rysunek 1.30). Tak zaprojektowana tabela nie jest w **I PN**. Nie będzie możliwe prawidłowe przetwarzanie danych zapisanych w tabeli. Rozwiązaniem jest zapisanie informacji o zakupionych książkach

w dwóch wierszach. W pierwszym wierszu w polu *Tytuł książki* należy wpisać tytuł pierwszej książki, w drugim — tytuł drugiej książki, natomiast nazwisko klienta zostanie powtórzone w liczbie wierszy równej liczbie zakupionych książek (rysunek 1.31). Teraz tabela jest w **I PN**.

Realizacja zamówień

Nazwisko kli	Imię	Adres	Telefon	PESEL	Tytuł książki	Liczł	Cena
Nowak	Marek	Toruń	(56) 6589234	79120307431	Dziady	2	20,00 zł
Kowalski	Adam	Warszawa	(22) 3451234	80122401871	Balladyna, Tango	1	15,00 zł
Górecki	Grzegorz	Poznań	(45) 2367897	82061203983	Pan Tadeusz	1	21,00 zł
Zan	Marcin	Gdańsk	(33) 8373635	82020201875			

Rysunek 1.30. Tabela nie jest w I PN, ponieważ w polu Tytuł książki pojawiła się lista wartości

Realizacja zamówień

Nazwisko klie	Imię	Adres	Telefon	PESEL	Tytuł książki	Liczł	Cena
Nowak	Marek	Toruń	(56) 6589234	79120307431	Dziady	2	20,00 zł
Kowalski	Adam	Warszawa	(22) 3451234	80122401871	Balladyna	1	15,00 zł
Kowalski	Adam	Warszawa	(22) 3451234	80122401871	Tango	2	18,00 zł
Górecki	Grzegorz	Poznań	(45) 2367897	82061203983	Pan Tadeusz	1	21,00 zł

Rysunek 1.31. Tabela jest w I PN, ponieważ w polu Tytuł książki występują pojedyncze wartości

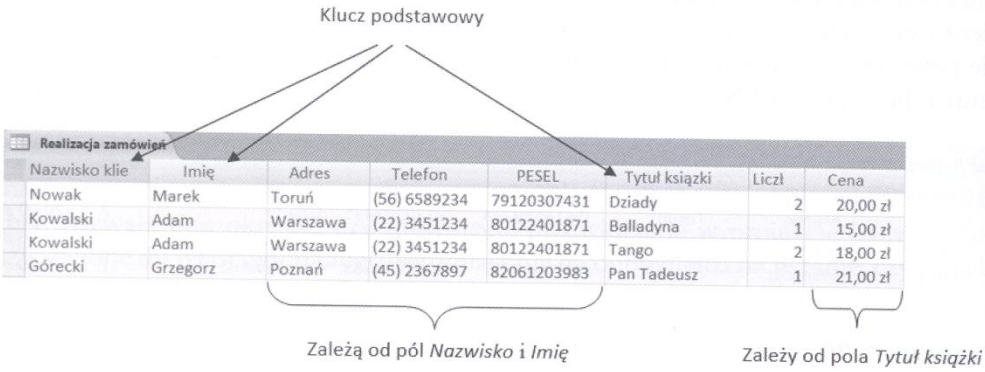
Druga postać normalna

DEFINICJA

Tabela jest w drugiej postaci normalnej (**II PN**), jeżeli jest w pierwszej postaci normalnej (**I PN**) oraz każde z pól niewchodzących w skład klucza podstawowego zależy od całego klucza, a nie od jego części.

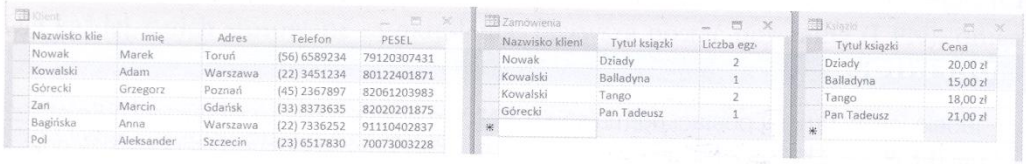
Ta reguła i następna służą do sprawdzenia, czy w tabeli i bazie danych nie doszło do redundancji, czyli niepotrzebnego powtarzania danych. Ponieważ **II PN** odnosi się do klucza podstawowego, należy określić ten klucz dla tabeli. Jeżeli klucz podstawowy składa się z jednego pola, tabela jest w **II PN**, ponieważ wszystkie pola, poza polem klucza podstawowego, muszą odnosić się do pola klucza podstawowego.

W zaprojektowanej tabeli *Realizacja zamówień* kluczem podstawowym będzie kombinacja pól *Nazwisko klienta*, *Imię* oraz *Tytuł książki*. Pola niewchodzące w skład klucza podstawowego (*Adres*, *Telefon*, *PESEL*) zależą od pól *Nazwisko* i *Imię*, natomiast nie zależą od pola *Tytuł książki*. Pole *Cena* zależy jedynie od pola *Tytuł książki* (rysunek 1.32). Tylko pole *Liczba egzemplarzy* zależy zarówno od pól *Nazwisko* i *Imię*, jak i od pola *Tytuł książki*. Tabela nie jest w **II PN**.



Rysunek 1.32. Tabela nie jest w II PN, ponieważ tylko pole Liczba egzemplarzy zależy od całego klucza

Normalizacja polega na podzieleniu tabeli na takie tabele, które spełnią warunek **II PN**. Tabelę *Realizacja zamówień* należy podzielić na trzy tabele: *Klient*, z polami *Nazwisko*, *Imię*, *Adres*, *Telefon*, *PESEL*; *Zamówienia*, z polami *Nazwisko klienta*, *Tytuł książki*, *Liczba egzemplarzy*; oraz *Książki*, z polami *Tytuł książki* i *Cena* (rysunek 1.33). Kluczem podstawowym w tabeli *Klient* są pola *Nazwisko* i *Imię*, w tabeli *Zamówienia* pola *Nazwisko klienta* i *Tytuł książki*, a w tabeli *Książki* pole *Tytuł książki*. Zostało zlikwidowane powtarzanie danych w tabeli *Realizacja zamówień* i wszystkie tabele są w II PN.



Rysunek 1.33. Podział tabeli Realizacja zamówień na tabele spełniające warunek II PN

Aby wyświetlić zestawienie dotyczące klienta i kupionych przez niego książek, należy zdefiniować połączenie między tabelami.

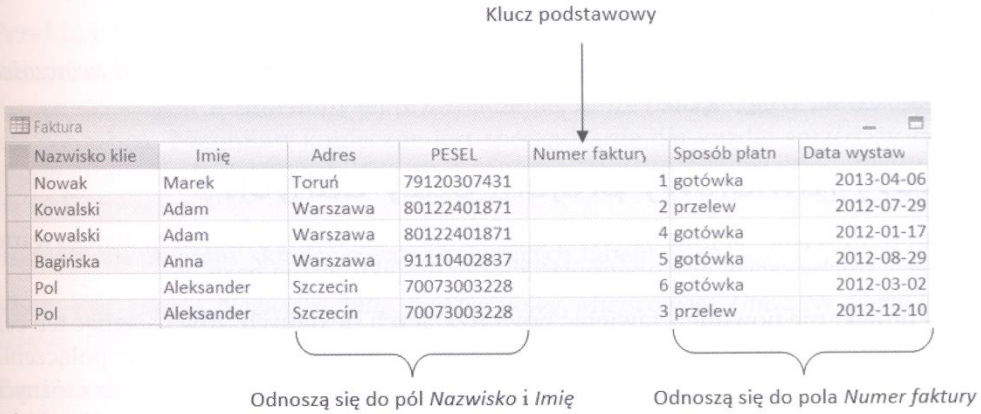
Trzecia postać normalna

DEFINICJA

Tabela jest w trzeciej postaci normalnej (**III PN**), jeżeli jest w pierwszej i w drugiej postaci normalnej oraz każde z pól niewchodzących w skład klucza podstawowego niesie informację bezpośrednio o kluczu i nie odnosi się do żadnego innego pola.

Załóżmy, że w projektowanej bazie danych dla księgarni została zaprojektowana tabela *Faktura* z polami: *Nazwisko klienta*, *Imię*, *Adres*, *PESEL*, *Numer faktury*, *Sposób płatności* i *Data wystawienia faktury*. Załóżmy również, że w pola tabeli będą wpisywane tylko

wartości elementarne, czyli tabela jest w **I PN**. Klucz podstawowy to pole *Numer faktury*. Wszystkie pola niewchodzące w skład klucza zależą od całego klucza, czyli tabela jest w **II PN**. Sprawdźmy, czy tabela jest w **III PN**. Pola *Sposób płatności* i *Data wystawienia faktury* odnoszą się do faktury, czyli zawierają informacje o kluczu. Natomiast pola *Adres* i *PESEL* zawierają informacje na temat klienta, a nie faktury (rysunek 1.34), czyli nie niosą informacji bezpośrednio o kluczu. Tabela nie jest w **III PN**.

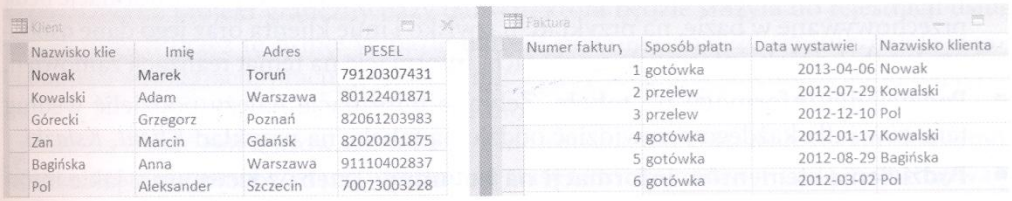


Nazwisko klie	Imię	Adres	PESEL	Numer faktury	Sposób płatn	Data wystaw
Nowak	Marek	Toruń	79120307431	1	gotówka	2013-04-06
Kowalski	Adam	Warszawa	80122401871	2	przelew	2012-07-29
Kowalski	Adam	Warszawa	80122401871	4	gotówka	2012-01-17
Bagińska	Anna	Warszawa	91110402837	5	gotówka	2012-08-29
Pol	Aleksander	Szczecin	70073003228	6	gotówka	2012-03-02
Pol	Aleksander	Szczecin	70073003228	3	przelew	2012-12-10

Rysunek 1.34. Tabela nie jest w III PN, ponieważ pola Adres i PESEL nie niosą informacji o kluczu

Normalizacja, podobnie jak w przypadku **II PN**, polega na podzieleniu tabeli na takie tabele, które spełnią warunek **III PN**.

Tabelę *Faktura* należy podzielić na dwie tabele: *Faktura* (z polami *Numer faktury*, *Sposób płatności* i *Data wystawienia faktury*) oraz *Klient* (z polami *Nazwisko klienta*, *Imię*, *Adres*, *PESEL*) (rysunek 1.35). Zostało zlikwidowane powtarzanie danych o kliencie w tabeli *Faktura*. Dane o kliencie będą zapisane tylko raz, w tabeli *Klient*.



Klient

Nazwisko klie	Imię	Adres	PESEL
Nowak	Marek	Toruń	79120307431
Kowalski	Adam	Warszawa	80122401871
Górecki	Grzegorz	Poznań	82061203983
Zan	Marcin	Gdańsk	82020201875
Bagińska	Anna	Warszawa	91110402837
Pol	Aleksander	Szczecin	70073003228

Faktura

Numer faktury	Sposób płatn	Data wystawie	Nazwisko klienta
1	gotówka	2013-04-06	Nowak
2	przelew	2012-07-29	Kowalski
3	przelew	2012-12-10	Pol
4	gotówka	2012-01-17	Kowalski
5	gotówka	2012-08-29	Bagińska
6	gotówka	2012-03-02	Pol

Rysunek 1.35. Podział tabeli Faktura na tabele spełniające warunek III PN

Przykład 1.2

Przestrzegając reguł tworzenia tabel, po sprawdzeniu za pomocą normalizacji, czy tabele mają prawidłową strukturę, baza danych dla księgarni mogłaby składać się z następujących tabel:

Klient → *Id_klienta*, *Nazwisko*, *Imię*, *Kod pocztowy*, *Miejscowość*, *Ulica*, *Nr domu*, *PESEL*, *Telefon*, *Adres e-mail*.

Zamówienia → *Id_zamówienia*, *Id_klienta*, *Data złożenia zamówienia*, *Data wysłania*, *Koszt wysyłki*.

Rejestr → *Id_zamówienia*, *Id_książki*, *Liczba egzemplarzy*.

Książka → *Id_książki*, *Tytuł*, *Id_autora*, *Cena książki*, *Rok wydania*, *Wydawnictwo*, *Rodzaj literatury*, *Miejsce wydania*, *Język książki*, *Opis*, *Zdjęcie okładki*.

Autor → *Id_autora*, *Nazwisko*, *Imię*, *Narodowość*, *Okres tworzenia*, *Rodzaj twórczości*, *Język tworzenia*, *Osiągnięcia*.

1.4.8. Prawidłowy projekt bazy danych

Prawidłowy projekt bazy danych jest bardzo istotny dla efektywnej pracy, dlatego warto poświęcić trochę czasu, aby opanować zasady projektowania bazy.

Dobry projekt nie powinien zawierać powtarzających się danych. Aby osiągnąć ten cel, musimy podzielić dane na wiele tabel. Następnie powinniśmy zdefiniować połączenia między tabelami, aby można było tworzyć zestawienia danych pochodzących z różnych tabel. Na przykład w bazie danych *Księgarnia* podzieliliśmy dane na oddzielne zbiory (*Klient*, *Zamówienia*, *Rejestr*, *Książka*, *Autor*, *Faktura*), następnie zdefiniowaliśmy połączenia między tabelami, aby utworzyć zestawienie dotyczące realizacji zamówienia (*Nazwisko*, *Imię*, *Tytuł książki*, *Nazwisko autora*, *Cena książki*, *Liczba egzemplarzy*).

Proces projektowania bazy danych składa się z następujących kroków:

- **Określenie celu, jakiemu ma służyć baza danych**. Baza danych może na przykład służyć do gromadzenia informacji na temat sprzedaży książek, do wystawiania faktur dotyczących sprzedaży, do modyfikowania na bieżąco tych danych, do przetwarzania zgromadzonych danych.
- **Określenie zakresu potrzebnych informacji**. Należy określić, jakie informacje będą przechowywane w bazie, na przykład: nazwisko i imię klienta oraz jego dane osobowe, tytuły książek, informacje o autorach, informacje na temat realizacji zamówień.
- **Podzielenie informacji na tabele**. Zebrane informacje należy podzielić według tematów i dla każdego przewidzieć oddzielną tabelę, na przykład *Klient*, *Książki*.
- **Podzielenie elementów informacji na kolumny**. Trzeba zdecydować, jakie informacje mają być przechowywane w poszczególnych tabelach. Każdy element informacji zostanie przypisany do kolumny, na przykład tabela *Klient* będzie zawierała kolumny *Nazwisko klienta* i *Adres*.
- **Wybranie kluczy podstawowych**. Należy wybrać klucz podstawowy dla każdej tabeli, na przykład w tabeli *Klient* może to być identyfikator przypisany do każdego klienta lub PESEL.
- **Zastosowanie reguł normalizacji**. Za pomocą reguł normalizacji można sprawdzić, czy tabele mają prawidłową strukturę.

- **Poprawienie projektu.** Po sprawdzeniu, jeżeli to konieczne, trzeba skorygować projekt bazy.
- **Utworzenie relacji pomiędzy tabelami.** Należy przejrzeć projekt i zdecydować, jakie relacje powinny znaleźć się w bazie.

Po zaprojektowaniu bazy danych zgodnie z podanymi regułami można przystąpić do jej tworzenia, korzystając z aplikacji przeznaczonych do obsługi relacyjnych baz danych.

Przykład 1.3

Podczas analizowania przeznaczenia bazy danych tworzymy jej strukturę. Jeżeli w bazie danych dla księgarni zmienimy jej przeznaczenie, może okazać się, że tabela *Autor* nie jest potrzebna. Natomiast konieczne jest sporządzanie dla każdej sprzedaży faktury. Wtedy niezbędna będzie tabela do przechowywania informacji, które powinny znaleźć się na fakturze.

Baza danych mogłaby składać się z następujących tabel:

Klient → *Id_klienta*, *Nazwisko*, *Imię*, *Kod pocztowy*, *Miejscowość*, *Ulica*, *Nr domu*, *PESEL*, *Telefon*, *Adres e-mail*.

Zamówienia → *Id_zamówienia*, *Id_klienta*, *Data złożenia zamówienia*, *Data wysłania*, *Koszt wysyłki*, *Numer faktury*.

Rejestr → *Id_zamówienia*, *Id_książki*, *Liczba_egzemplarzy*.

Książka → *Id_książki*, *Tytuł*, *Nazwisko i imię autora*, *Cena książki*, *Wydawnictwo*, *Rodzaj literatury*, *Miejsce wydania*, *Język książki*, *Opis*.

Faktura → *Numer faktury*, *Sposób płatności*, *Data wystawienia faktury*.

Ćwiczenie 1.1

Firma Globtroter wynajmuje klientom autokary do przewozu osób. Posiada flotę dobrze wyposażonych autokarów oraz zatrudnia grupę kierowców z najwyższymi umiejętnościami. Utwórz projekt graficzny bazy danych, która będzie służyła do rejestracji usług świadczonych przez tę firmę. W bazie danych powinna znaleźć się informacja o posiadanych autokarach, pracujących w firmie kierowcach oraz klientach korzystających z usług firmy. Każda usługa zamówienia może dotyczyć jednego lub kilku autokarów i powinna zostać zarejestrowana w bazie z informacją, dla kogo realizowane jest zamówienie, które autokary zostaną wysłane na trasę i którzy kierowcy będą je obsługiwali.

Rozwiązanie

Określenie potrzebnych informacji:

Przechowywane informacje o autokarach: model, liczba miejsc, rok produkcji, pojemność silnika, spalanie, stawka za 1 km.

Przechowywane informacje o kierowcach: nazwisko, imię, PESEL, adres, telefon, kategoria prawa jazdy, data zatrudnienia, doświadczenie.

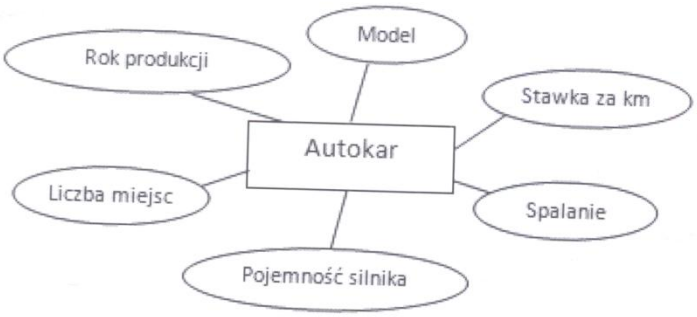
Przechowywane informacje o klientach: nazwisko, imię, PESEL, adres, telefon.

Przechowywane informacje o realizowanych usługach: cel, liczba km, data realizacji, opłata za wynajęcie.

Przechowywane informacje o fakturach: nr faktury, data wystawienia, sposób płatności.

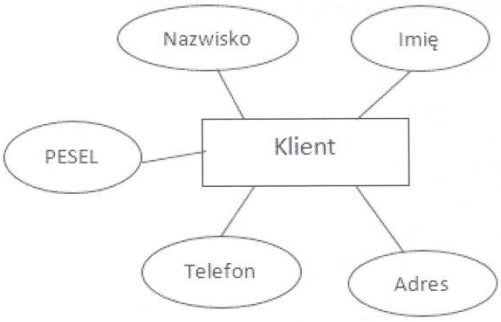
Tworzenie diagramu ERD:

Tworzone zbiory encji zostały pokazane na rysunkach 1.36–1.40.

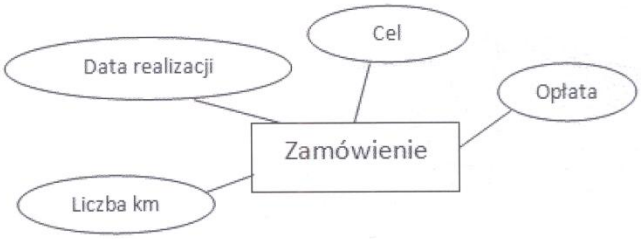


Rysunek 1.36. Zbiór encji Autokar

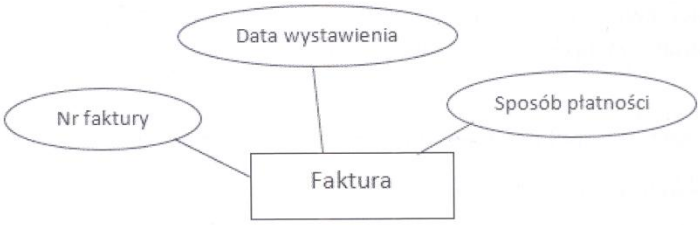
Rysunek 1.37. Zbiór encji Kierowca



Rysunek 1.38. Zbiór encji Klient



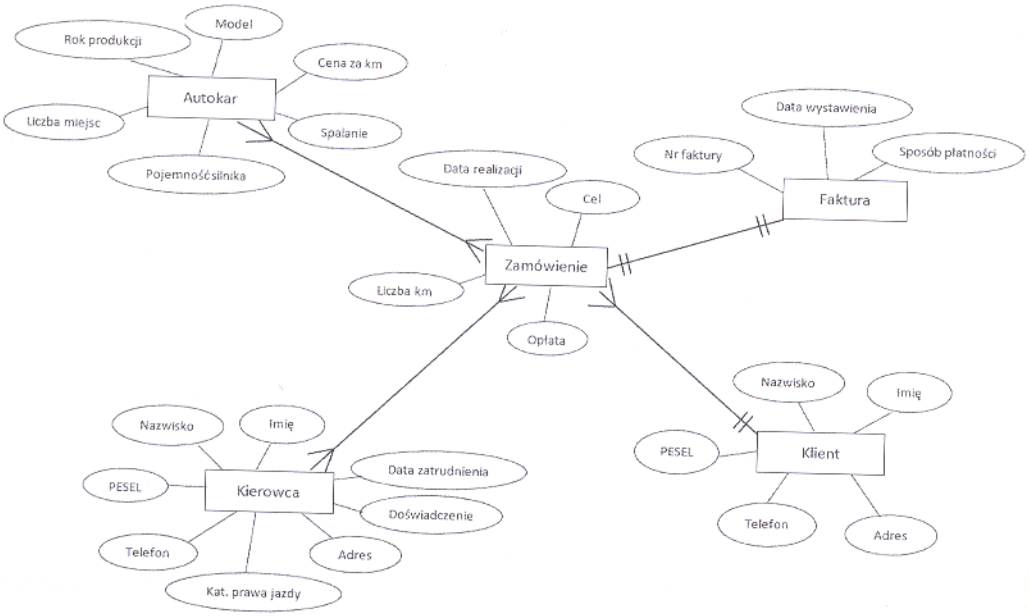
Rysunek 1.39. Zbiór encji Zamówienie



Rysunek 1.40. Zbiór encji Faktura

Definiowanie związków między zbiorami encji:

Tworzone związki zostały pokazane na rysunku 1.41.



Rysunek 1.41. Związki między zbiorami encji bazy danych dla firmy Globtroter

Ćwiczenie 1.2

Na podstawie utworzonego w ćwiczeniu 1.1 projektu graficznego bazy danych dla firmy Globtroter zaprojektuj w modelu relacyjnym tabele do przechowywania danych oraz tabele opisujące związki. Umieść w nich wszystkie atrybuty zdefiniowanych encji oraz atrybuty opisujące związki zachodzące między encjami.

Rozwiązanie

Przykładowy zestaw tabel dla bazy danych firmy Globtroter:

Klient → *Id_klienta*, *Nazwisko*, *Imię*, *Kod pocztowy*, *Miejscowość*, *Ulica*, *Nr domu*, *PESEL*, *Telefon*.

Zamówienia → *Id_zamówienia*, *Id_klienta*, *Data realizacji*, *Cel*, *Opłata*, *Liczba km*, *Nr faktury*.

Kierowca → *Id_kierowcy*, *Nazwisko*, *Imię*, *Kod pocztowy*, *Miejscowość*, *Ulica*, *Nr domu*, *PESEL*, *Telefon*, *Data zatrudnienia*, *Doświadczenie*, *Kat. prawa jazdy*.

Autokar → *Id_autokaru*, *Model*, *Rok produkcji*, *Liczba miejsc*, *Pojemność silnika*, *Spalanie*, *Stawka za km*.

Faktura → *Nr faktury*, *Sposób płatności*, *Data wystawienia*.

Zamówienie-Kierowca → *Id_zamówienia*, *Id_kierowcy*.

Zamówienie-Autokar → *Id_zamówienia*, *Id_autokaru*.

1.4.9. Typy danych

Projektując bazę danych, zwracamy uwagę na to, aby wartości przechowywane w polach były wartościami elementarnymi (atomowymi). W praktyce wymóg ten jest realizowany przez określenie typu pola.

Teoria relacyjnych baz danych mówi, że wszystkie wartości danych oparte są na prostych typach danych. Oznacza to, że dla każdego pola powinien zostać wybrany typ danych najmniejszy z możliwych, który spełni związane z nim wymagania projektowe. Typ danych określi, jaki rodzaj informacji może zostać zapisany w wybranym polu. Poza tym definiowanie typów danych znacząco wpływa na optymalizację działania bazy danych.

Jeżeli w tabeli występuje pole *Imię*, dla którego wybierzemy typ `char(100)` — typ tekstowy o długości 100 znaków — to na każde imię zostanie zarezerwowane 100 bajtów. Przyjmując, że najdłuższe imię nie zawiera więcej niż 20 znaków, dodatkowo rezerwujemy 80 bajtów, które nigdy nie zostaną wykorzystane. Gdy w tabeli wystąpi 200 tysięcy rekordów, które zajmą kilkanaście GB nigdy nie wykorzystanej pamięci.

To samo dotyczy pola *Data_urodzenia*. Jeżeli zostanie wybrany typ `date`, to zajmie on 3 bajty, jeżeli zostanie zastosowany typ `datetime`, to dla każdego rekordu zostanie zarezerwowane 8 bajtów na datę urodzenia.

Typy danych stosowane w językach komunikowania się z bazą danych można podzielić na kilka grup. Wybrane typy zostały pokazane w tabeli 1.1.

Tabela 1.1. Typy danych stosowane w MS SQL i MySQL

Grupa	*Typy w MS SQL*	*Typy w MySQL*
Tekstowe	char, varchar, nchar, ntext, nvarchar	char, varchar
Liczbowe	int, smallint, bigint, tinyint, float, real, decimal, numeric	int, smallint, bigint, tinyint, float, real, decimal, numeric
Daty i czasu	date, datetime	date, datetime, timestamp, time
Binarne	binary, varbinary	binary, varbinary
Waluty	money, smallmoney	
Specjalne	text, image, bit	text, bit, enum, set

Typy tekstowe

Typy tekstowe to pola, które mogą zawierać litery, liczby i symbole. Po zadeklarowaniu długości można przechowywać teksty o określonej liczbie znaków.

Typ char(długość) — można przechowywać do 255 znaków. Jest to pole o stałej liczbie znaków, np. `char(10)`. Jeżeli tekst jest krótszy, zostanie uzupełniony do zadeklarowanej

długości spacjami. Jeśli tekst jest dłuższy niż zadeklarowana liczba znaków, zostanie obcięty.

Typ varchar(długość) — można przechowywać do 255 znaków. Jest to pole o zmiennej liczbie znaków, np. `varchar(10)`. Jeżeli tekst jest krótszy niż zadeklarowana długość, nie jest uzupełniany spacjami, co powoduje zmniejszenie zasobów pamięci zajmowanych przez bazę danych. Przy deklarowaniu typów pól zaleca się stosowanie typów o zmiennej liczbie znaków.

Typ nvarchar(długość) — można przechowywać do 255 znaków. Ten typ działa w standardzie Unicode, który zapisuje pojedynczy znak na 2 bajtach. Jest to związane z kodowaniem znaków narodowych. Powoduje to zwiększenie objętości bazy danych, ale jeżeli będzie ona funkcjonowała w środowisku wielojęzycznym, należy zastosować typ, który będzie poprawnie interpretował znaki narodowe, np. `nvarchar(10)`.

Typ ntext(długość) — pozwala na wprowadzanie tekstu o wielkości do 2 GB. Typ stosowany, jeśli w polu będzie przechowywany tekst o dużej liczbie znaków.

Typy liczbowe

Typy liczb całkowitych

Jeżeli w polu będą wprowadzane tylko liczby całkowite, powinien zawsze być wybierany typ liczb całkowitych. Typy całkowite różnią się tylko liczbą bajtów przeznaczonych na zapisanie liczby, a więc również zakresem liczb, które mogą zostać zapisane.

Typ tinyint — 1 bajt, zapis liczb z zakresu od –128 do 127 lub bez znaku z zakresu od 0 do 255.

Typ smallint — 2 bajty, zapis liczb z zakresu od –32 768 do 32 767 lub bez znaku z zakresu od 0 do 65 535.

Typ int — 4 bajty, zapis liczb z zakresu od –2 147 483 648 do 2 147 483 647 lub bez znaku z zakresu od 0 do 4 294 967 295.

Typ bigint — 8 bajtów, zapis liczb z zakresu od –2^63 do 2^63–1.

Typy liczb rzeczywistych

Jeżeli w polu będą przechowywane liczby, dla których należy dokładnie określić liczbę miejsc po przecinku, powinien zostać zastosowany typ `decimal` lub `numeric`.

Typ numeric(precyzja, skala) — dla tego typu należy podać dwa parametry: precyzję i skalę. Pierwszy parametr określa całkowitą liczbę cyfr, drugi mówi, ile cyfr znajduje się po przecinku, np. `numeric(4,2)`. W podanym przykładzie możliwe będzie przechowywanie liczb nie większych od 9 999 z dokładnością do dwóch miejsc po przecinku. Jeśli nie podamy tych parametrów, zostaną one ustawione automatycznie na wartości 10 jako precyzja i 0 jako skala.

Typ decimal(precyzja, skala) — podobnie jak dla typu `numeric` należy podać dwa parametry: precyzję i skalę.

Jeżeli dokładność przechowywanych w bazie liczb jest mniej istotna, natomiast najważniejsza jest ich rozpiętość (np. dane statystyczne), należy stosować typ `float` lub `real`.

Typ float(n) — 4 bajty lub 8 bajtów, liczba zmiennoprzecinkowa. Parametr *n* określa precyzję w bitach. Jeśli *n* jest w zakresie od 1 do 24, to mamy do czynienia z pojedynczą precyzją (dokładność do około 7 cyfr po przecinku), jeżeli wynosi od 25 do 53, to mamy do czynienia z podwójną precyzją (dokładność do około 15 cyfr po przecinku). Używanie typu `float` może powodować problemy przy wykonywaniu obliczeń.

Typ float(m,d) — liczba zmiennoprzecinkowa o pojedynczej precyzji. Typ ten jest niestandardowym rozszerzeniem MySQL.

Typ real — 4 bajty, liczba zmiennoprzecinkowa. Definiowana podobnie jak typ `float`.

Jeżeli nie jest to konieczne, należy unikać używania typów przybliżonych. Jeżeli jest to możliwe, należy stosować typy `decimal` lub `numeric`.

Typy daty i czasu

Typ datetime — 8 bajtów, zapis daty i czasu z dokładnością do sekundy w postaci RRRR-MM-DD HH:MM:SS.

Typ date — 3 bajty, zapis daty w postaci RRRR-MM-DD.

Typ time — 3 bajty, zapis czasu w postaci HH:MM:SS.

Typ daty powinien być dostosowany do rodzaju przechowywanej informacji. Na ogół nie ma potrzeby przechowywania zarówno daty, jak i czasu. Jeśli chcemy przechowywać datę urodzin, w zupełności wystarczy pole typu `date`. Jeśli wymagane jest przechowywanie tylko roku urodzin, wystarczy zastosować pole typu `int`. Dzięki temu zostanie zaoszczędzona znaczna ilość miejsca w pamięci i uzyskamy większą efektywność działań na danych.

Typy binarne

Typy binarne służą do przechowywania danych binarnych. Dane reprezentowane są za pomocą liczb heksadecymalnych.

Typ binary — do przechowywania danych binarnych o stałej długości, długość maksymalna 8000 bajtów.

Typ varbinary — do przechowywania danych binarnych o zmiennej długości, długość maksymalna do 8000 bajtów.

Typy walutowe

Typy walutowe służą do przechowywania wartości walutowych. Mogą również służyć do przechowywania wartości innych niż wartości walutowe.

Typ money — 8 bajtów, do przechowywania wartości walutowych z zakresu od –2^31 do 2^31–1 z dokładnością do jednej dziesięciotysięcznej (cztery cyfry po przecinku).

Typ smallmoney — 4 bajty, do przechowywania wartości walutowych z zakresu od –2^63 do 2^63–1 z dokładnością do jednej dziesięciotysięcznej (cztery cyfry po przecinku).

Typy specjalne

Typ text — do przechowywania łańcuchów tekstowych o zmiennej długości. Może przechowywać do 2 GB danych.

Typ image — do przechowywania danych binarnych o zmiennej długości. Może przechowywać do 2 GB danych.

Mimo że typy `text` i `image` pozwalają na przechowywanie dużej ilości informacji, powinniśmy zastanowić się, czy lepszym rozwiązaniem nie byłoby umieszczenie pliku z tą informacją poza bazą, a w bazie danych przechowywanie jedynie ścieżki do pliku.

Typ bit — 1 bit, do przechowywania wartości logicznych w postaci *0* lub *1*, *włączony/wyłączony*, *tak/nie*.

1.5. Cechy relacyjnej bazy danych

Baza danych powinna charakteryzować się następującymi cechami:

- trwałość danych,
- integralność danych,
- bezpieczeństwo danych,
- współdzielenie danych,
- abstrakcja danych,
- niezależność danych,
- integracja danych.

1.5.1. Trwałość danych

Trwałość danych zapisanych w bazie jest podstawową cechą baz danych. Trwałość danych oznacza, że dane zostały zapisane w bazie w sposób nieulotny. Wszystkie współczesne systemy baz danych muszą spełniać ten wymóg. Trwałość danych jest niezależna od działania aplikacji oraz od platformy sprzętowej i programowej. Dane gromadzone w bazie danych są przechowywane w pamięci zewnętrznej (dyskowej, optycznej, taśmowej). Powinny one być przechowywane w pamięci dopóty, dopóki wymagają tego użytkownicy systemu baz danych.

1.5.2. Integralność, czyli poprawność danych

Integralność (spójność) danych oznacza, że dane muszą:

- wiernie odzwierciedlać dane rzeczywiste (dane są prawdziwe oraz są aktualizowane, gdy ulega zmianie rzeczywistość),

- spełniać ograniczenia nałożone przez użytkowników (istnieje system kontroli danych wejściowych),
- wykazywać brak anomalii wynikających ze współbieżnego dostępu do danych (istnieją mechanizmy, które zabezpieczają dane przed błędami pojawiającymi się podczas współbieżnego dostępu do nich).

Integralność bazy danych to także:

- odporność na błędy i awarie wynikające z zawodności sprzętu i oprogramowania (istnieją mechanizmy, które zabezpieczają dane przed skutkami awarii sprzętu i oprogramowania),
- odporność na błędy użytkowników (istnieją mechanizmy, które zabezpieczają dane przed następstwami błędów logicznych).

Na poziomie danych wyróżniamy dwa rodzaje integralności danych:

- *semantyczna* (spójność logiczna) — oznacza zgodność danych z rzeczywistością, czyli poprawność odwzorowania rzeczywistości (baza danych odpowiada zaprojektowanemu schematowi i zadeklarowanym ograniczeniom),
- *bazowa* (spójność fizyczna) — oznacza poprawność procesów zachodzących w bazie (operacje wykonywane w bazie danych kończą się sukcesem).

Na poziomie struktur bazy danych można wyróżnić następujące rodzaje integralności:

- *referencyjna* — odnosi się do powiązań między tabelami i oznacza, że każdej wartości klucza obcego odpowiada dokładnie jedna wartość klucza podstawowego;
- *encji* — odnosi się do schematu bazy danych i oznacza, że każda encja musi posiadać klucz podstawowy, czyli pole lub zbiór pól o wartościach unikatowych i niebędących wartością NULL. Z tego wynika, że w danej tabeli nie mogą istnieć dwa identyczne wiersze;
- *atrybutu* — oznacza, że wartość każdego atrybutu należy do jego dziedziny.

1.5.3. Bezpieczeństwo danych

Bezpieczeństwo danych oznacza, że dostęp do bazy danych mają tylko jej użytkownicy identyfikowani unikatową nazwą (loginem) i hasłem. Ponadto każdy użytkownik posiada określone uprawnienia w bazie danych.

Podstawowym sposobem zapewnienia bezpieczeństwa danych jest stosowanie procedur uwierzytelniania oraz ograniczania uprawnień dostępu do danych. Sposobem pośrednim może być szyfrowanie danych oraz ograniczenie fizycznego dostępu do systemu komputerowego.

Bezpieczeństwo danych oznacza również, że baza danych jest zabezpieczona przed awarią sprzętu lub oprogramowania.

1.5.4. Współdzielenie danych

Współdzielenie danych oznacza, że istnieje możliwość równoczesnej pracy wielu użytkowników z tą samą bazą danych. Użytkownicy mogą też jednocześnie pracować z tym samym zbiorem danych. Skutkiem takiej pracy mogą być konflikty w dostępie do danych, gdy jeden użytkownik modyfikuje zbiór danych, a drugi próbuje ten sam zbiór odczytać lub zmodyfikować w inny sposób. W bazie danych muszą istnieć mechanizmy zapewniające poprawne rozwiązanie takich konfliktów (np. stosowanie mechanizmów transakcji i współbieżności).

1.5.5. Abstrakcja danych

Abstrakcja to uogólnienie (uproszczenie) rozpatrywanego problemu, które polega na wyodrębnieniu wspólnych jego cech.

Baza danych zawiera pewien model rzeczywistości. Ale przechowywane są w niej tylko niektóre dane o obiektach. Powinny one opisywać wyłącznie istotne cechy obiektów świata rzeczywistego. Baza danych to abstrakcyjny model pewnego wycinka rzeczywistości, a jej struktura powinna poprawnie odzwierciedlać obiekty świata rzeczywistego i powiązania pomiędzy tymi obiektami.

Wyodrębniamy trzy poziomy abstrakcji:

- *poziom wewnętrzny* — określa sposób pamiętania danych;
- *poziom pojęciowy* (*konceptualny*) — wyższy poziom abstrakcji danych. Definiuje dane oraz związki zachodzące między nimi (przedstawia on logiczną strukturę bazy danych);
- *poziom zewnętrzny* — najwyższy poziom abstrakcji danych. Opisuje sposób, w jaki dane są widziane przez użytkownika. Dla każdego użytkownika obraz danych definiuje się za pomocą schematu zewnętrznego (podschematu).

Dla tej samej bazy danych istnieje jeden model pojęciowy i wiele schematów zewnętrznych.

1.5.6. Niezależność danych

Niezależność danych polega na oddzieleniu danych od aplikacji, które używają tych danych. Niezależność danych może być rozpatrywana w dwóch aspektach:

- *Logiczna niezależność danych* — niezależność danych widzianych przez użytkownika (przez jego aplikacje) od logicznej struktury bazy danych (schematu pojęciowego) oraz zmian tej struktury w czasie.
- *Fizyczna niezależność danych* — niezależność logicznej struktury bazy danych (schematu pojęciowego) od sposobu przechowywania danych w pamięci zewnętrznej (na nośnikach zewnętrznych).

We współczesnych bazach danych niezależność danych jest osiągana tylko częściowo.

1.5.7. Integracja danych

Integracja danych polega na zapewnieniu współpracy danych przechowywanych w różnych miejscach i obsługiwanych przez różne systemy. Często bazy danych fizycznie są umieszczane na różnych komputerach połączonych ze sobą w taki sposób, że użytkownik nie wie, iż dane, z którymi pracuje, pochodzą z różnych baz i komputerów. Zmiany zawartości bazy na jednym komputerze powinny być uwzględniane również na innych komputerach. Tak funkcjonujące bazy nazywamy *rozproszonymi bazami danych*. Bazy danych powinny być zaprojektowane tak, aby dane były zawsze dostępne, niezależnie od tego, gdzie i w jakiej postaci zostały zapisane i jak są przechowywane. Serwery bazodanowe powinny zapewniać integrację rozproszonych danych znajdujących się na wielu komputerach. Mechanizmy integracyjne dostępne na nich powinny działać w czasie rzeczywistym i współpracować z różnorodnymi bazami danych.

1.6. Pytania i zadania

1.6.1. Pytania

1. Podaj zalety korzystania z komputerowych baz danych.
2. Podaj definicję bazy danych.
3. Omów poznane modele baz danych.
4. Omów występujące w modelu relacyjnym rodzaje więzów integralności.
5. Podaj podstawowe cechy relacyjnego modelu baz danych.
6. Podaj definicję klucza podstawowego.
7. Jakiego typu relacje mogą wystąpić w bazie danych?
8. Co oznacza występujące w bazach danych pojęcie *encja*?
9. Do czego służą diagramy *ERD*?
10. Jakie zastosowanie w projektowaniu bazy danych mają narzędzia *CASE*?
11. W jaki sposób w tabelach opisywany jest związek „wiele do wielu"?
12. Na czym polega normalizacja tabel?
13. Wymień cechy, którymi powinna charakteryzować się baza danych.
14. Na czym polega integralność referencyjna bazy danych?

1.6.2. Zadania

Zadanie 1.

Zaprojektuj zgodnie z poznanymi zasadami bazę danych dotyczącą Twojej szkoły. Nazwij ją na przykład *Moja_szkoła*. Umieść w niej informacje na temat uczniów, klas, przedmiotów i nauczycieli, sal lekcyjnych i pracowni. Określ funkcje tworzonej bazy danych oraz zdefiniuj zbiory przechowywanych informacji. Wykorzystując diagramy ERD, opracuj model graficzny schematu bazy danych. Zaprojektuj tabele i sprawdź za pomocą reguł normalizacji, czy mają one prawidłową strukturę.