

Literatura

1. Baker R., CASE*Method. Modelowanie związków encji, WNT, Warszawa 1996.
2. Baker R., Longman C., CASE*Method. Modelowanie funkcji i procesów, WNT, Warszawa 1996.
3. Beynon-Davies P., *Systemy baz danych*, WNT, Warszawa 2003.
4. Connolly T., Begg C., *Systemy baz danych t.1,2*, Wydawnictwo RM, Warszawa 2004.
5. Delobel C., Adiba M., *Relacyjne bazy danych*, WNT, Warszawa 1989.
6. Flasiński M., *Wstęp do analitycznych metod projektowania systemów informatycznych*, WNT, Warszawa 1997.
7. Poźniak-Koszałka I., *Relacyjne bazy danych w środowisku Sybase*, Oficyna Wyd. PWr, Wrocław 2004.
8. Ullman J. D., Widom J., *Podstawowy wykład z systemów baz danych*, WNT, Warszawa 2002.
9. Yourdon E., *Nowoczesna analiza strukturalna*, WNT, Warszawa 1996.

Historia zarządzania danymi

do ok. 1900 r – ręczne zarządzanie zapisami na różnego rodzaju „nośnikach”

1890 – 1955 wykorzystanie kart perforowanych i maszyn do zbierania danych (Herman Hollericht → IBM)

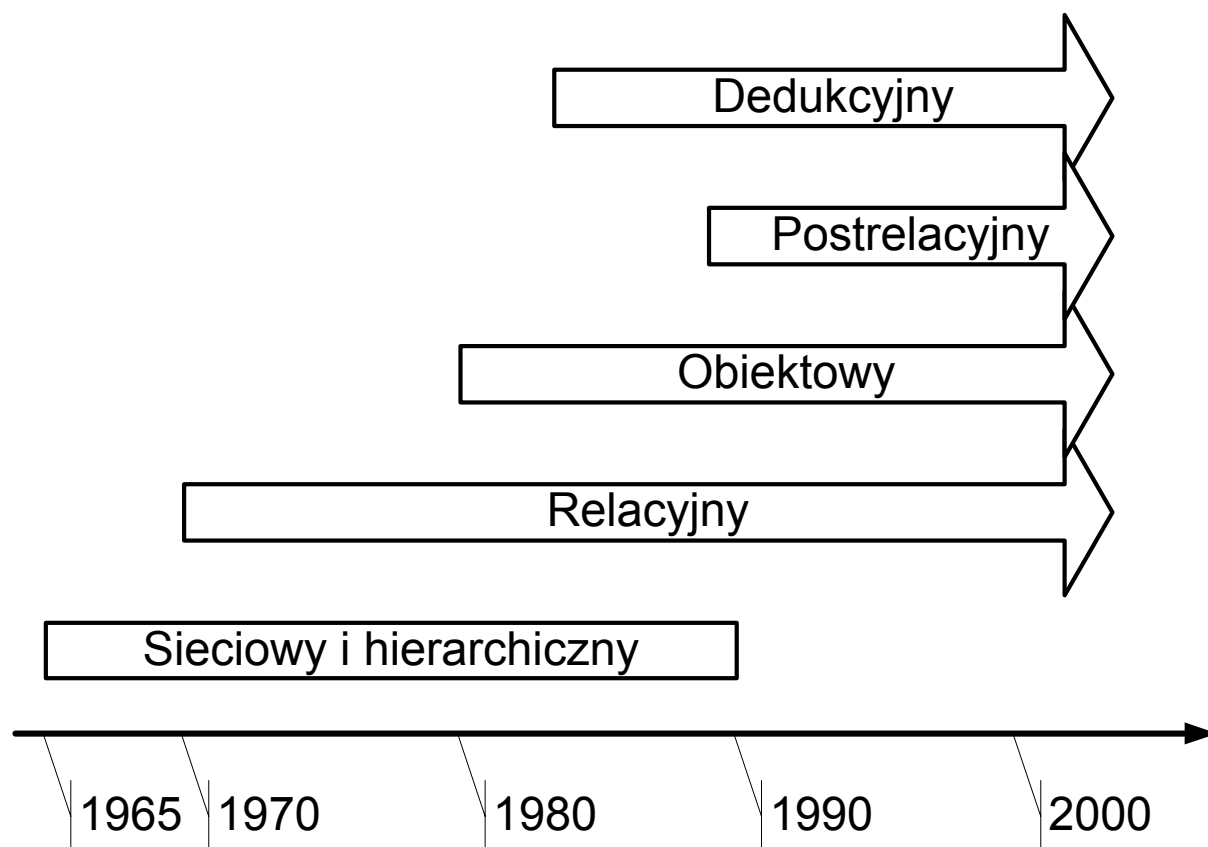
1955 – 1970 wykorzystanie taśmy magnetycznej do przechowywania danych → przetwarzanie wsadowe plików sekwencyjnych

1965 – 1980 wykorzystanie nośników danych o dostępie swobodnym (dyski), co umożliwia przetwarzanie danych on-line → powstaje model hierarchiczny i sieciowy

1970 E.F.Codd - podstawy relacyjnego modelu danych, praca „A relational model for large shared data banks”

1970-te IBM opracowuje prototyp relacyjnego SZBD o nazwie System/R

od 1980 szybki rozwój modelu relacyjnego dzięki upowszechnieniu się komputerów osobistych



Rys. 1 Chronologia modeli danych

Baza danych – pewna pula powiązanych ze sobą danych.

„Bazą danych nazywamy zbiór danych o określonej strukturze, zapisany na zewnętrznym nośniku pamięciowym komputera, mogący zaspokoić potrzeby wielu użytkowników korzystających z niego w sposób selektywny w dogodnym dla siebie czasie”. (C.Delobel, M.Adiba)

Baza danych – dostępny dla wielu użytkowników zbiór powiązanych logicznie danych wraz z definicją ich struktury, zaprojektowany dla zaspokojenia potrzeb przetwarzania danych przez instytucję.

Dla abstrakcyjnej bazy danych prawdziwe są następujące stwierdzenia:

- baza danych jest modelem pewnego wycinka rzeczywistości. Tę rzeczywistość nazywamy obszarem analizy,
- baza danych ma własność integralności, jest dokładnym odzwierciedleniem swojego obszaru analizy,
- baza danych nie powinna zawierać powtarzających się danych, należy unikać redundancji danych,
- w bazie danych zapisywane są dane na temat pewnego wycinka rzeczywistości aby móc te dane odtworzyć i analizować w dowolnej chwili. Dane same w sobie nie mają znaczenia, aby były użyteczne muszą być zinterpretowane w pewnym kontekście. Zinterpretowane dane to informacja,

- baza danych zawsze znajduje się w pewnym stanie, który odzwierciedla obszar analizy w danej chwili. Zawartość bazy danych zmienia się w czasie, przechodząc od stanu do stanu w sposób dyskretny,
- zmianę stanu bazy danych powodują zdarzenia nazywane transakcjami,
- dane w bazie danych są traktowane jako trwałe,
- baza danych zawiera oprócz danych katalog systemowy (słownik danych, metadane), który jest repozytorium opisującym dane przechowywane w bazie.

Bazy danych są stosowane w szerokim kontekście systemów informacyjnych, których celem jest dostarczanie informacji dla przedsiębiorstwa lub jego części.

Jedna z definicji mówi, że informacja to „przyrost wiedzy, który może być uzyskany na podstawie danych”.

dane → informacje → wiedza

Technologia informatyczna zapewnia środki służące do konstruowania nowoczesnych systemów informacyjnych.

System zarządzania bazą danych (SZBD) jest to system oprogramowania, który pozwala użytkownikom definiować, tworzyć i utrzymywać bazę danych oraz kontrolować do niej dostęp.

SZBD spełnia swoje funkcje dla jednej lub wielu baz danych budowanych dla konkretnego obszaru analizy.

SZBD często mylnie utożsamiany z bazą danych, jest to zbiór programów umożliwiających tworzenie i eksploatację bazy danych.

Elementy środowiska SZDB:

- sprzęt,
- oprogramowanie,
- dane,
- procedury (instrukcje i polecenia do projektowania i wykorzystania bazy danych),
- ludzie,
 - administratorzy danych i bazy danych,
 - projektanci bazy danych,
 - twórcy aplikacji,
 - użytkownicy.

Funkcje SZBD:

- (1) zapis, odczyt i aktualizacja danych,
- (2) udostępnienie katalogu systemowego (słownika danych, metadanych) użytkownikom.

Typowy katalog systemowy pamięta:

- nazwy, typy i rozmiary elementów danych,
- nazwy związków,
- więzy integralności,
- nazwy użytkowników wraz z prawami dostępu,
- schematy
- statystyki.

(3) obsługa transakcji,

Użytkownicy i programy użytkowe kontaktują się z systemem za pomocą tzw. transakcji. Transakcja stanowi elementarną jednostkę pracy, taką że baza danych jest w stanie spójnym przed i po zakończeniu transakcji. Jeżeli dana transakcja została wykonana poprawnie, to zmiany, które wprowadziła do bazy zostaną w niej zapamiętane. Transakcja stanowi ciąg akcji elementarnych.

(4) sterowanie współbieżnością,

(5) odporność na awarie (niezawodność bazy danych) - możliwość odtworzenia poprawnego stanu bazy danych sprzed awarii,

- (6) obsługa autoryzacji i ochrona danych (uniemożliwienie dostępu nieuprawnionych użytkowników do poufnych danych innych użytkowników),
- (7) zapewnienie integralności danych - uniemożliwienie przejścia bazy do stanu, który nie istnieje w modelowanej rzeczywistości,
- (8) obsługa transmisji danych,
- (9) optymalizacja zapytań - takie przekształcanie zapytań kierowanych do bazy przez jej użytkowników aby czas oczekiwania na odpowiedź był możliwie najkrótszy.

SZBD + baza danych = System bazy danych (SBD) – baza danych wraz ze środkami programowymi umożliwiającymi operowanie na niej.

Trójwarstwowa architektura ANSI-SPARC

- Warstwa zewnętrzna opisuje, jak użytkownicy widzą bazę danych i w jaki sposób uzyskują do niej dostęp. W tej warstwie zawarte są wszystkie istotne dla jej użytkowników informacje o bazie danych.
- Warstwa konceptualna zawiera logiczną strukturę całej bazy: taką, jaką widzi administrator. Warstwa ta opisuje, jakie dane są przechowywane w bazie i jakie są ich wzajemne związki. Jest zatem pełnym opisem wszystkich danych, ale niezależnym od sposobu przechowywania.
- Warstwa wewnętrzna opisuje sposób przechowywania danych w bazie. Opisuje między innymi organizację plików czy też budowę indeksów.
- Poniżej warstwy wewnętrznej znajduje się warstwa fizyczna, która jest realizowana przez SO pod kierunkiem SZBD.

Głównym zadaniem trójwarstwowej architektury jest oddzielenie fizycznej reprezentacji bazy danych od różnych sposobów widzenia danych w bazie przez różnych użytkowników. Każdy użytkownik powinien mieć dostęp do tych samych danych, ale sposób ich widzenia powinien być dostosowany do indywidualnych wymagań.

Użytkownicy nie powinni mieć bezpośredniego do czynienia ze szczegółami fizycznego sposobu pamiętania danych.

Administrator bazy danych powinien mieć możliwość zmiany struktury służących do przechowywania danych i te zmiany powinny być niezauważalne dla użytkowników.

Wewnętrzna struktura bazy danych powinna być odporna na zmiany fizycznych parametrów nośnika danych.

Głównym zadaniem trójwarstwowej architektury jest zapewnienie niezależności danych, co oznacza, że zmiany dokonane w niższych warstwach nie powinny mieć wpływu na warstwy wyższe.

Architektura klient-serwer

- strona klienta - na stacji roboczej użytkownika,
- strona serwera – na komputerze zawierającym serwer bazy danych czyli bazę danych wraz z jej systemem zarządzania (SZBD).

Funkcje aplikacji po stronie serwera bazy danych

- przechowywanie i organizacja dostępu do danych,
- wykonywanie instrukcji języka baz danych (SQL),
- sprawowanie kontroli nad spójnością danych,
- zarządzanie zasobami bazy danych w tym kontami użytkowników.

Funkcje aplikacji po stronie klienta

- kontakt z użytkownikiem (interfejs użytkownika).
- wyjaśnianie użytkownikowi stanu obliczeń w tym błędów i sytuacji wyjątkowych.
- przyjmowanie od niego zleceń na operacje, wykonywanie tych zleceń lub przesyłanie ich w postaci instrukcji języka SQL do serwera bazy danych.

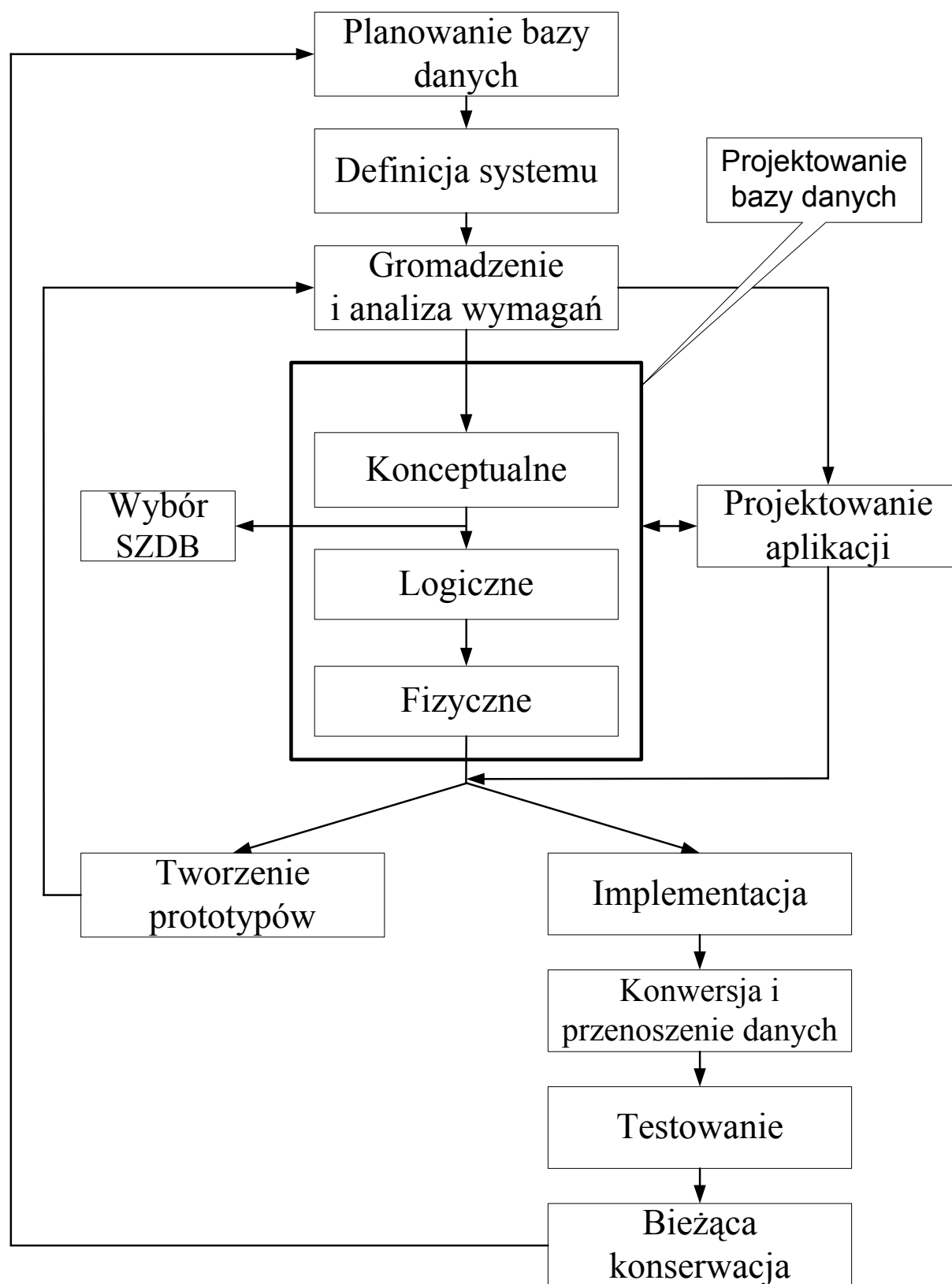
System informacyjny to zestaw środków i zasobów, które pozwalają na uzyskiwanie informacji, zarządzanie i sterowanie nimi oraz ich rozpowszechnianie w obrębie instytucji.

Komputerowy system informacyjny (czyli system informatyczny) obejmuje: bazę danych i jej oprogramowanie, aplikacje, sprzęt komputerowy oraz użytkowników korzystających z niego wraz z personelem rozwijającym ten system.

Baza danych jest podstawowym składnikiem systemu informacyjnego, zatem jej powstanie i późniejsze wykorzystanie powinno być widziane z szerokiej perspektywy danej instytucji.

Fazy cyklu życia systemu informacyjnego:

- planowanie, gromadzenie i analiza wymagań,
- projektowanie,
- implementacja (+ prototyp),
- testowanie,
- konserwacja eksploatacyjna.



Rys. 2 Fazy cyklu życia aplikacji baz danych

Planowanie bazy danych to czynności związane z wyborem skutecznych i wydajnych metod realizacji faz cyklu życia oraz odpowiedź na pytania związane z następującymi zagadnieniami:

- identyfikacja celów i planów przedsiębiorstwa i powiązanie ich z wymaganiami wobec systemu informacyjnego,
- ocena aktualnie używanego systemu informacyjnego (mocne i słabe strony),
- oszacowanie możliwości technologii informacyjnych (informatycznych).

Definicja systemu określa zakres i granice stosowania danej aplikacji bazy danych oraz główne perspektywy użytkowników. Formalizacja tych wymagań może być realizowana za pomocą diagramów kontekstowych lub diagramu przepływu danych.

Perspektywa użytkownika definiuje, jakie są oczekiwania wobec aplikacji z punktu widzenia konkretnego użytkownika (kierownik, magazynier itp.) lub pewnego obszaru zastosowań (kadry, marketing).

Gromadzenie i analiza wymagań – formalizacją są np. diagram hierarchii funkcji, diagram przepływu danych, a techniki to: analiza dokumentacji, wywiady, obserwacja działalności, prowadzenie badań oraz ankietowanie.

Etapy projektowania baz danych

Konceptualne – proces konstrukcji modelu dla danych, który jest niezależny od wszelkich aspektów fizycznych.

Poziom abstrakcji: analitycy i projektanci bazy danych

Formalizacja opisu: projektowanie strukturalne lub notacja obiektowa

Logiczne (implementacyjne) – proces konstrukcji modelu dla danych, który jest oparty o specyficzny model danych, ale niezależny od konkretnego SZBD i innych aspektów fizycznych.

Poziom abstrakcji: programiści i administratorzy bazy danych.

Formalizacja opisu: uszczegółowienie modelu konceptualnego, normalizacja, SQL.

Fizyczne – proces tworzenia opisu bazy danych w pamięci zewnętrznej.

Poziom abstrakcji: struktura pamięci gdzie będzie przechowywana baza danych.

Formalizacja opisu: tworzony automatycznie przez SZBD.

Metody projektowania schematu relacyjnego

Z góry na dół (*Top-Down*)

- utworzyć model ER,
- zastosować reguły transformacji modelu ER na schemat relacyjny.

Z dołu do góry (*Down-Top*)

- zebrać jak najwięcej danych, które będą tworzyć zawartość bazy danych,
- zidentyfikować tematy oraz ich właściwości: zdefiniować tabele relacyjne,
- przeprowadzić proces normalizacji do 3 lub 4 postaci normalnej.

Mieszana

- utworzyć model ER,
- zastosować reguły transformacji modelu ER na schemat relacyjny,
- przeprowadzić proces normalizacji do 3 lub 4 postaci normalnej.

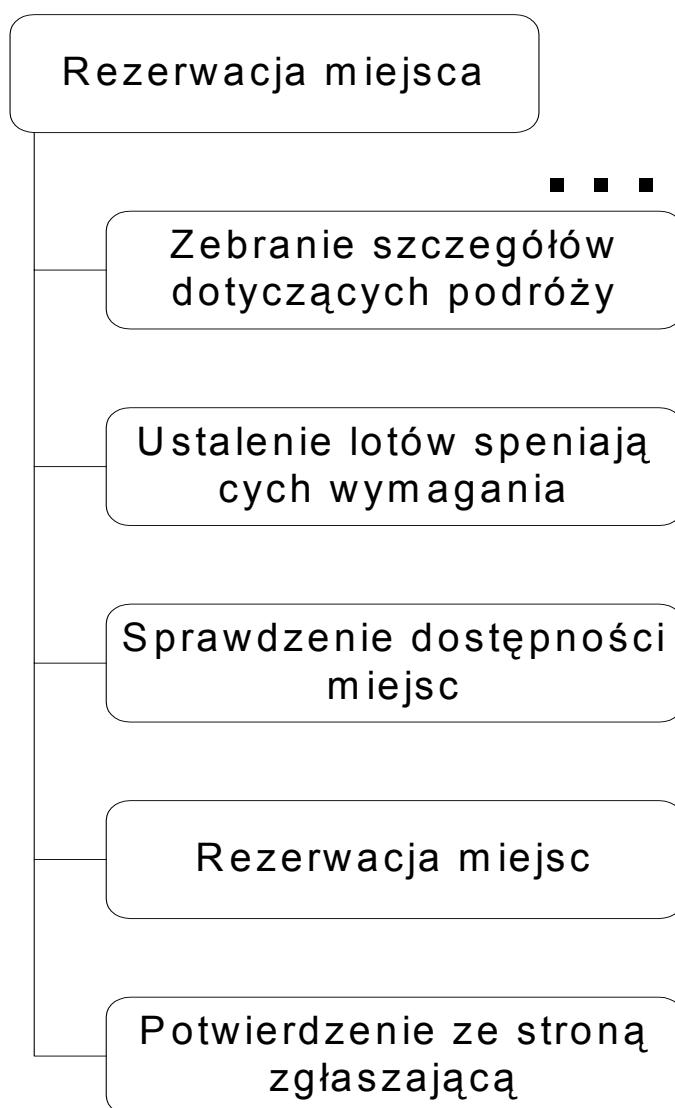
Cele modelowania funkcji

- Dostarczenie dokładnego modelu potrzeb funkcjonalnych przedsiębiorstwa, który zostanie następnie wykorzystany do opracowywania nowych lub ulepszania istniejących systemów.
- Dostarczenie modelu, który jest niezależny od wszelkich mechanizmów lub metod przetwarzania, pozwalając na podejmowanie obiektywnych decyzji oraz umożliwiając współistnienie z istniejącymi systemami.

Zasady tworzenia hierarchii funkcji

- Każda funkcja opisuje to, co przedsiębiorstwo robi obecnie lub powinno robić w przyszłości
- Brak odpowiedzi na pytania „kto”, „kiedy”, „gdzie” i „jak” tzn.
- Brak mechanizmów np. faks, telefon
- Brak struktur organizacyjnych np. zespoły, działy

- Brak nazw stanowisk np. sprzedawca, badacz, mechanik
- Dekompozycja funkcji zawiera wszystko, co jest potrzebne do jej wykonania
- Każda funkcja w dekompozycji jest konieczna do realizacji funkcji nadrzędnej



Rys. 3 Przykładowy diagram hierarchii funkcji

Modelowanie hierarchii funkcji tworzy diagramy pokazujące dekompozycję funkcji na różnych poziomach działalności przedsiębiorstwa.

Funkcja:

- elementarna – po wywołaniu musi wykonać swoje działanie z powodzeniem albo anulować wszystkie operacje w przypadku niepowodzenia wykonania. Alternatywna definicja to przejście z jednego stanu zgodności przedsiębiorstwa do drugiego albo brak zmiany stanów w przypadku niepowodzenia,
- atomowa – które nie podlegają dalszej dekompozycji,
- wspólna – występuje w kilku miejscach w hierarchii i reprezentując tą samą operację.

Cele modelowania związków encji

- Dostarczenie dokładnego modelu potrzeb informacyjnych przedsiębiorstwa, który stanowiłby podstawę do konstruowania nowych lub ulepszonych systemów.
- Dostarczenie modelu niezależnego od sposobu przechowywania danych i od metod dostępu do nich, umożliwiającego podejmowanie celowych decyzji, jeśli chodzi o metody implementacyjne i współdziałanie z istniejącymi systemami.

Modelowanie związków encji w najprostszej postaci obejmuje:

- identyfikowanie rzeczy ważnych w analizowanym przedsiębiorstwie (**encji**)
- własności tych rzeczy (**atrybutów**)
- sposoby, jakimi te encje są ze sobą powiązane (**związki**)

Encja jest rzeczą lub obiektem mającym dla nas znaczenie, rzeczywistym bądź wyobrażonym, o którym informacje muszą być znane lub przechowywane.

Synonim słowa klasa stosowanego w obiektowo zorientowanych językach programowania. Każde unikalne i rozpoznawalne wystąpienie encji to *wystąpienie* (*przykład, instancja*) encji lub jako synonim obiekt.

Atrybut jest dowolnym szczegółem służącym do identyfikowania, klasyfikowania, określania ilości lub wyrażania stanu encji.

Wartości jakie mogą być przyjmowane przez atrybuty są ograniczane przez typ, wielkość i zbiór wartości dopuszczalnych.

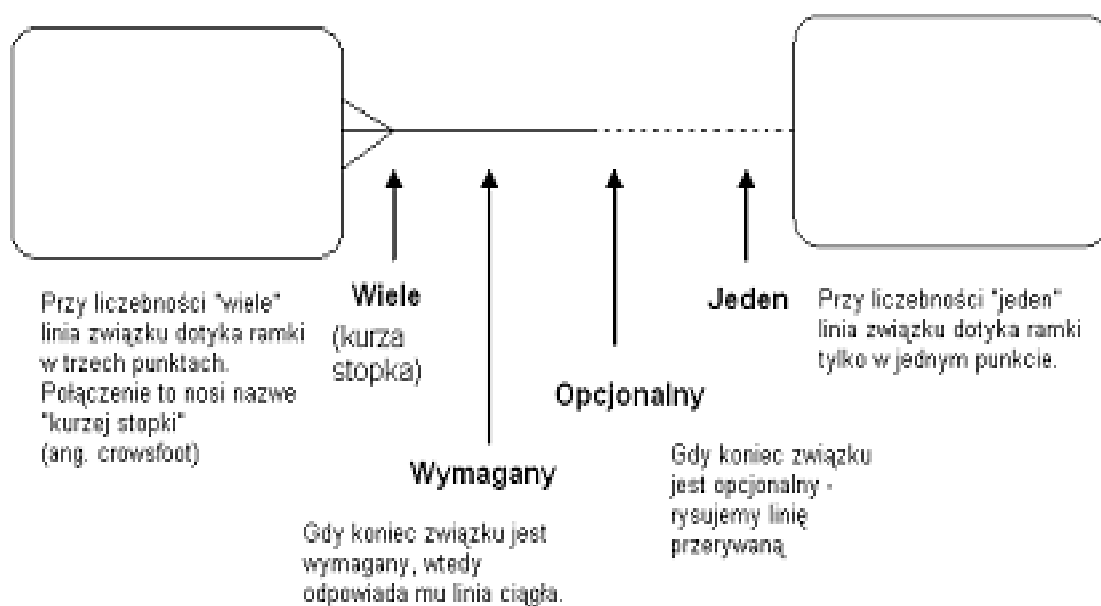
Każda encja musi być jednoznacznie identyfikowalna, to znaczy, że każde wystąpienie encji musi być wyraźnie odróżnialne od wszystkich innych wystąpień tego typu encji. Uzyskuje się to poprzez definicję jednoznacznego identyfikatora.

Jednoznacznym identyfikatorem może być atrybut, kombinacja atrybutów, kombinacja związków lub kombinacja atrybutów i związków.

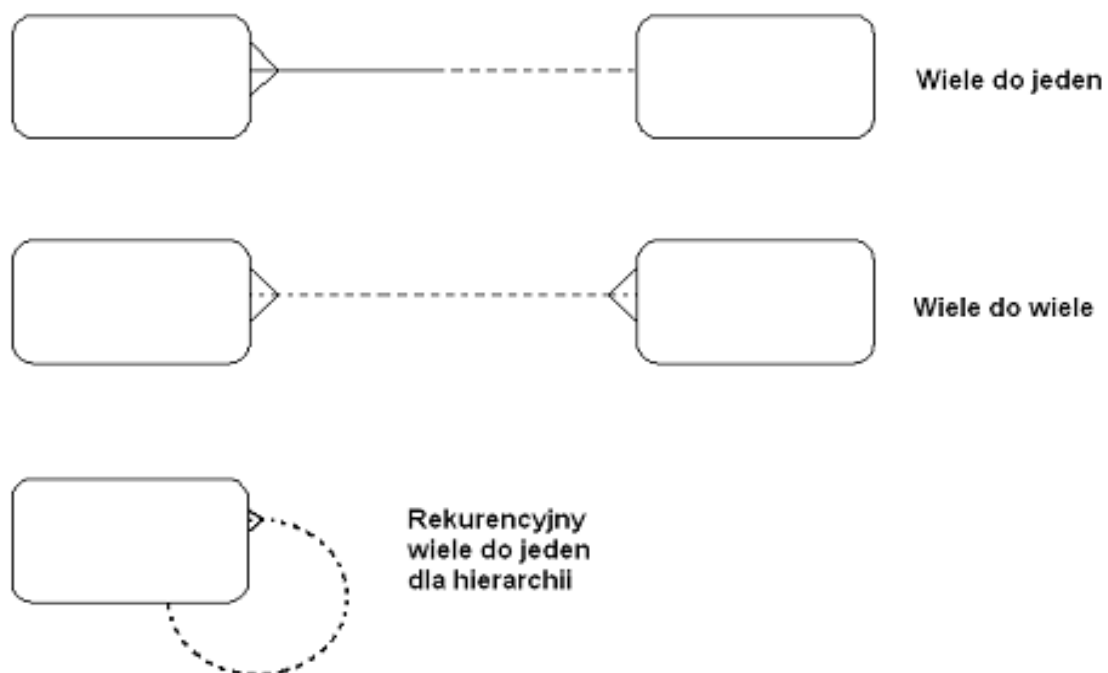
Związek – nazwane, istotne powiązanie pomiędzy encjami.

Każdy związek ma dwa końce, z których każdy ma przypisaną:

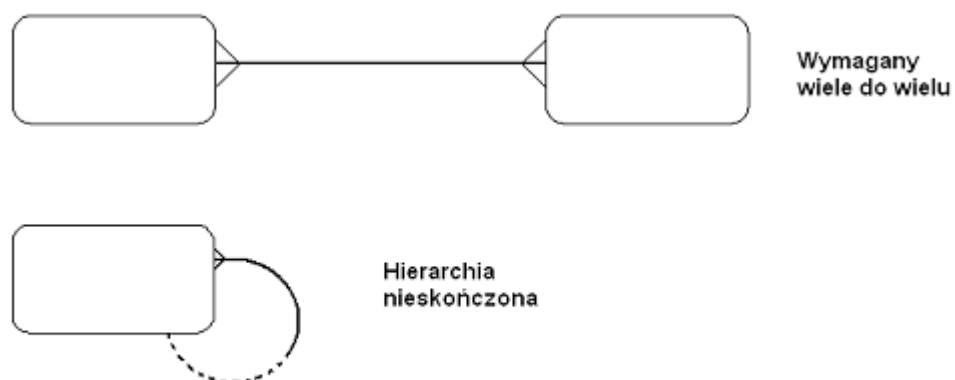
- nazwę,
- stopień/liczebność,
- opcjonalność (opcjonalny/wymagany).



Rys. 4 Konwencja w związkach

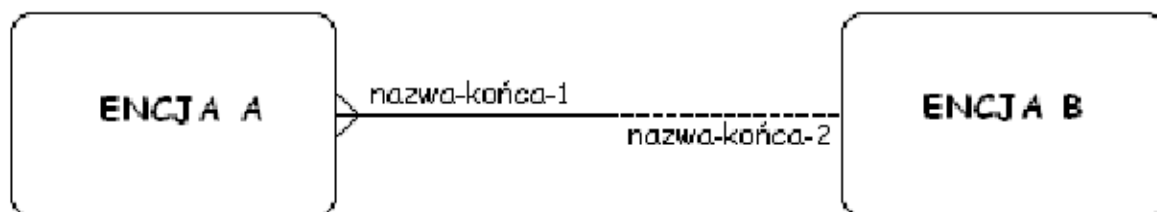


Rys. 5 Przykłady poprawnych związków



Rys. 6 Przykłady niepoprawnych związków

Nazewnictwo związków:



Rys. 4 Model nazewnictwa związków

- każda *ENCJA-A* **musi być** *nazwa-końca-1* jedna i tylko jedna *ENCJA-B* – (czytane od lewej do prawej),
- każda *ENCJA-B* **może być** *nazwa-końca-2* jedna lub więcej *ENCJA-A* – (czytane z prawej do lewej).

Nazwy atrybutów:

- powinna być prosta,
- musi być podana w liczbie pojedynczej,
- nie powinna zawierać nazwy encji.

Umieszczenie atrybutu na diagramach związków encji nie jest konieczne.

Konstrukcje specjalne:

(1) Związki wykluczające

- występują w postaci łuku łączącego dwa (lub więcej) końce związków dochodzących do tej samej encji,
- opisują sytuacje kiedy dla pojedynczego wystąpienia encji może wystąpić tylko jeden ze związków wykluczających.

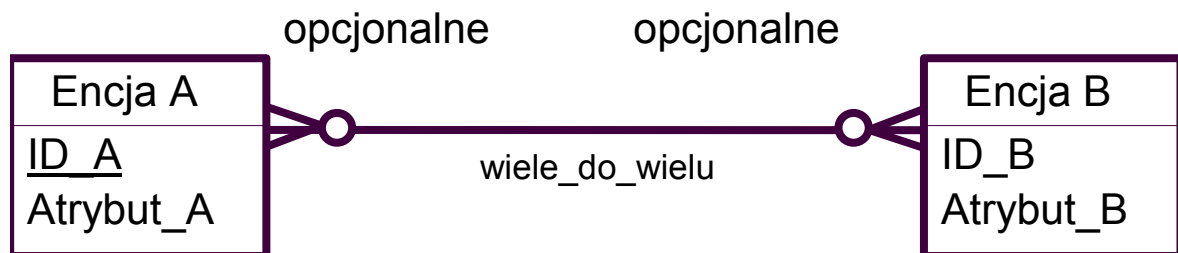
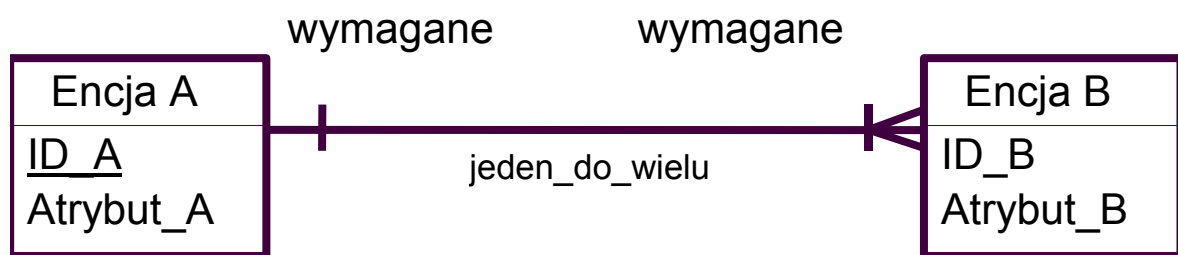
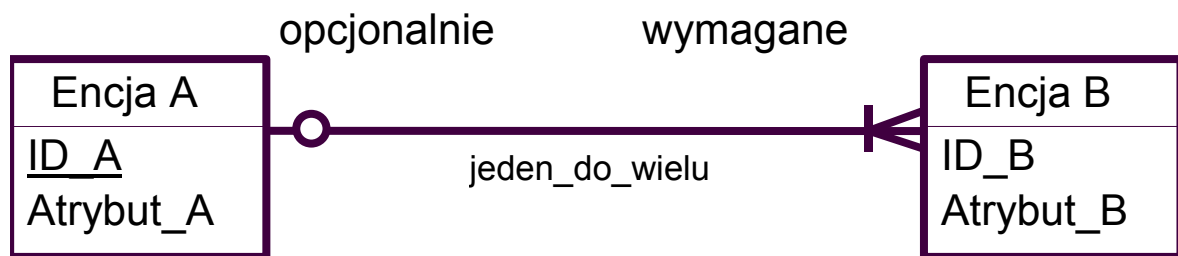
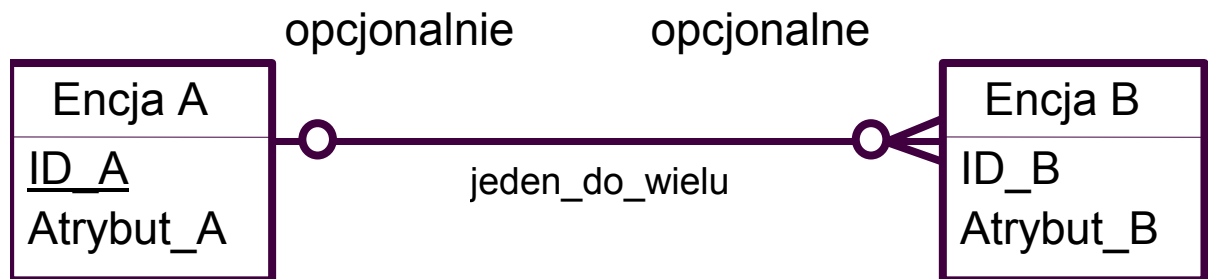
(2) Hierarchie encji

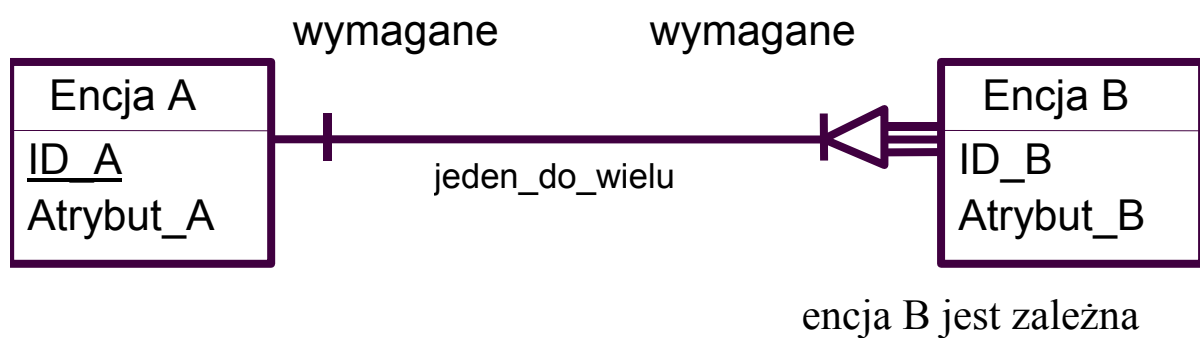
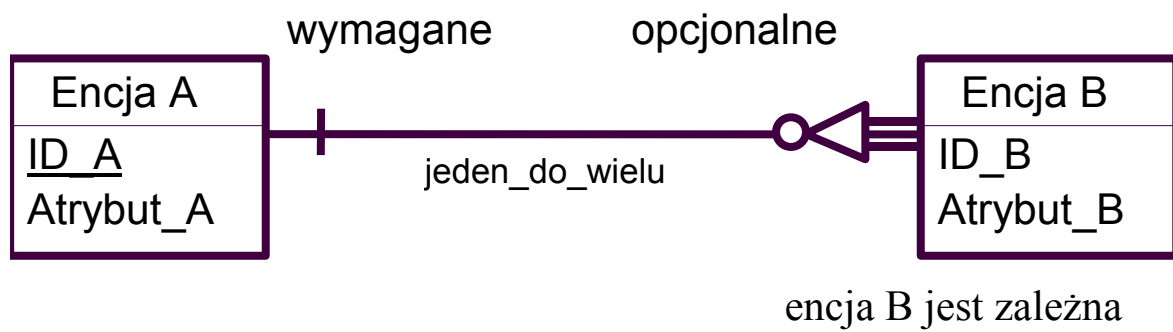
- hierarchie encji składają się z encji-nadtypu i zawartych w niej encji-podtypów,
- podtyp oprócz swoich własnych atrybutów i związków, posiada wszystkie atrybuty, związki i funkcje dziedziczone z encji-nadtypu.

Związki nieprzechodnie

- oznaczane są za pomocą rombu przy jednym z końców związku,
- wystąpienie encji, przy której istnieje związek nieprzechodni nie może zmieniać przypisania do innego wystąpienia encji wynikającego z tego związku.

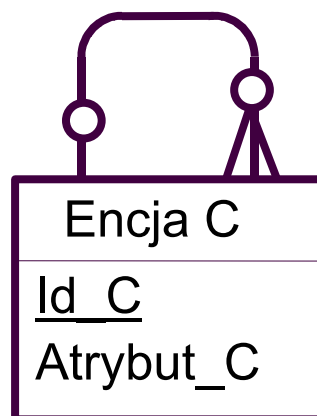
DataArchitect – modelowanie związków encji



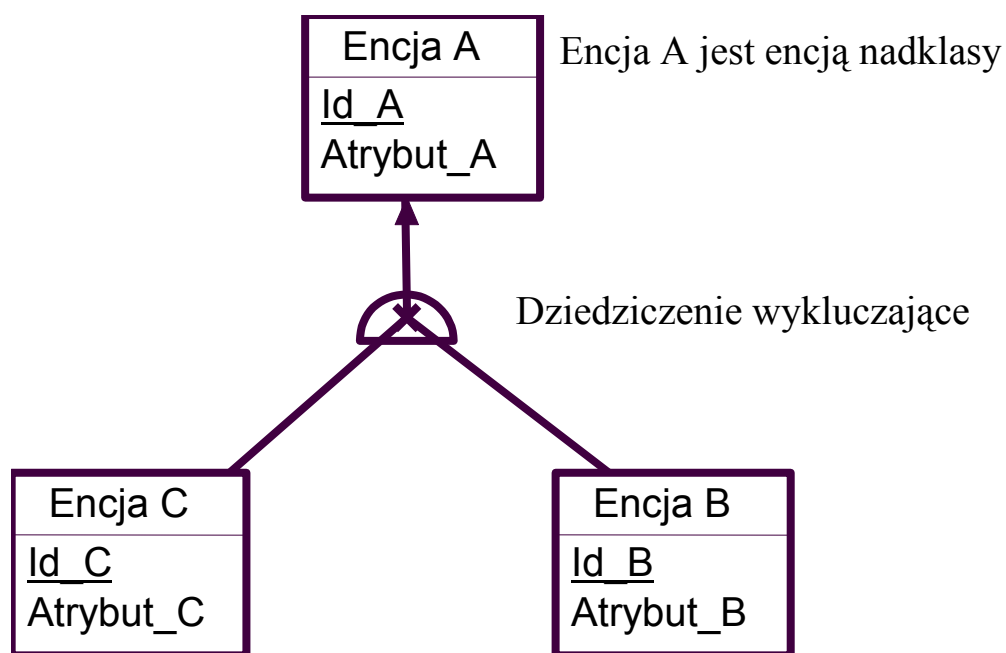


Rys. 8 Przykłady związków encji

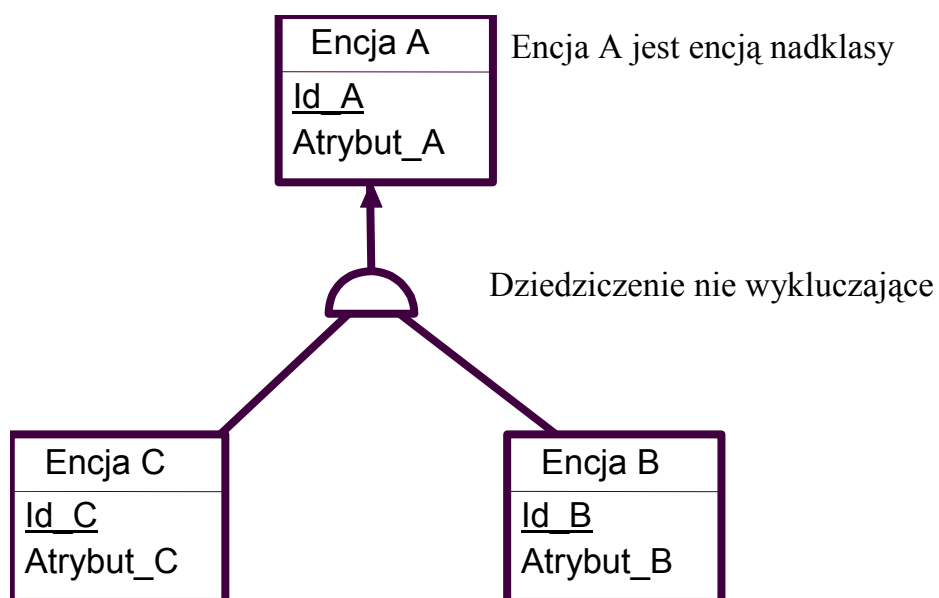
hierarchia encji



Rys. 9 Przykład hierarchii encji

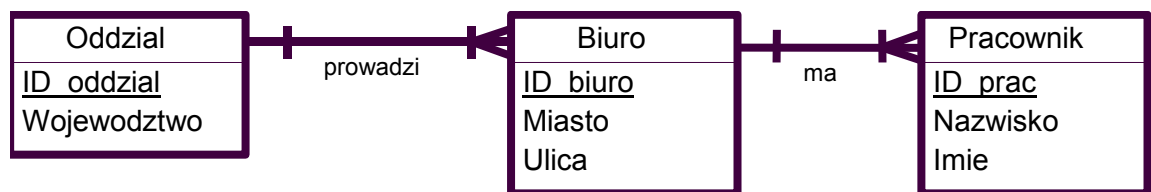
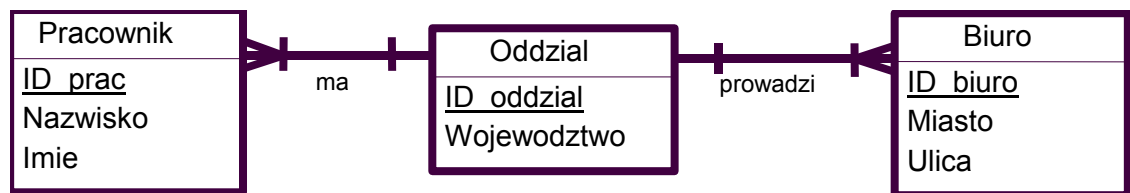


Encje C i B są encjami podklas

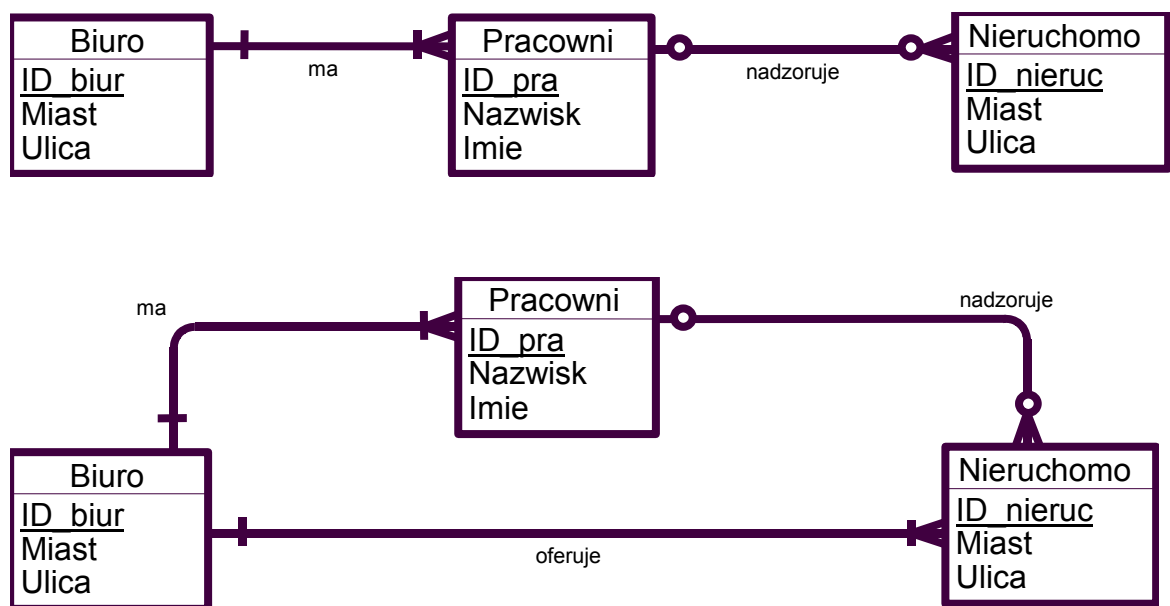


Encje C i B są encjami podklas

Rys. 10 Przykłady dziedziczenia encji



Rys. 11 Przykład pułapki „wachlarzowej”



Rys. 12 Przykład pułapki „szczelinowej”

Relacje matematyczne

Dla dowolnej ilości zbiorów (D_1, D_2, \dots, D_n) można zdefiniować ich iloczyn kartezjański jako:

$$D_1 \times D_2 \times \dots \times D_n = \{(d_1, d_2, \dots, d_n) \mid d_1 \in D_1, \dots, d_n \in D_n\}$$

Dowolny podzbiór takiego iloczynu jest relacją.

Np.

$$D_1 = \{2, 4\}, D_2 = \{3, 5, 7\}.$$

$$D_1 \times D_2 = \{(2, 3), (2, 5), (2, 7), (4, 3), (4, 5), (4, 7)\}.$$

$$R_1 = \{(2, 3), (2, 5), (4, 5), (4, 7)\},$$

$$R_2 = \{(2, 7), (4, 5)\},$$

$$R_3 = \{(2, 3), (2, 5), (2, 7), (4, 3), (4, 5), (4, 7)\}.$$

Terminologia modelu relacyjnego

Model ten oparty jest na matematycznym pojęciu relacji, która fizycznie reprezentowana jest jako tabela.

Relacja – tabela posiadająca kolumny i wiersze.

Atrybut – kolumna relacji opatrzona nazwą.

Dziedzina – zbiór dopuszczalnych wartości dla jednego lub większej liczby atrybutów.

Krotka – wiersz relacji.

Stopień (krotność) relacji – liczba atrybutów relacji.

Moc relacji – liczba krotek, które znajdują się w relacji.

Struktura relacji wraz z definicjami dziedzin jest nazywana schematem bazy danych (**intencjonalną bazą danych**). Konkretny stan bazy nazywany jest **bazą ekstensjonalną**.

Tab 1. Odpowiedniki terminologiczne dla modelu relacyjnego

Nazwa formalna	Pierwszy odpowiednik	Drugi odpowiednik
Relacja	Tabela	Plik
Krotka	Wiersz	Rekord
Atrybut	Kolumna	Pole

Własności relacji:

- relacja ma nazwę, która jest inna niż nazwy wszystkich pozostałych relacji w schemacie relacyjnym,
- każda komórka relacji zawiera jedną wartość atomową,
- każdy atrybut ma inną nazwę (w ramach jednej relacji),
- wszystkie wartości atrybutu pochodzą z tej samej dziedziny,
- każda krotka jest inna (nie ma dwóch krotek o tych samych wartościach dla wszystkich atrybutów),

- kolejność atrybutów nie ma znaczenia,
- kolejność krotek nie ma znaczenia (teoretycznie) – w praktyce może mieć wpływ na efektywność wyszukiwania.

Klucze relacji

Nadklucz – atrybut lub zbiór atrybutów, które pozwalają jednoznacznie zidentyfikować krotkę w relacji.

Klucz kandydujący – minimalny nadklucz.

Klucz złożony – klucz składający się z więcej niż jednego atrybutu.

Klucz główny – klucz kandydujący wybrany, by jednoznacznie identyfikować krotki relacji.

Klucz obcy – atrybut bądź zbiór atrybutów z jednej relacji, który odpowiada kluczowi kandydującemu pewnej relacji.

Klucz główny \Rightarrow relacja nadrzędna

Klucz obcy \Rightarrow relacja podrzędna

Integralność relacji

(formalna poprawność)

Każdy atrybut jest związany z pewną dziedziną, co oznacza, że istnieją więzy dziedzinowe nakładające ograniczenia na zbiór dopuszczalnych wartości atrybutów relacji.

Wartość pusta (NULL) – reprezentuje wartość atrybutu, która w danej chwili nie jest znana lub nie może być ustalona.

Integralność encji – w relacji bazowej żaden atrybut klucza głównego nie może być pusty.

Relacja bazowa – relacja o ustalonej nazwie, odpowiadająca encji (zbiorowi encji) ze schematu conceptualnego. Krotki tej relacji są fizycznie obecne w bazie danych.

Integralność referencyjna – jeśli w relacji istnieje klucz obcy, to jego wartość albo musi być równa wartości klucza kandydującego pewnej krotki w relacji nadrzędnej, albo klucz obcy musi mieć wartości puste dla wszystkich atrybutów.

Perspektywy

Perspektywa – dynamicznie obliczany wynik jednej lub wielu operacji relacyjnych tworzący nową relację z relacji bazowych lub innych perspektyw. Perspektywa jest relacją wirtualną, która nie musi fizycznie istnieć w bazie danych, ale może być wyliczana w każdej chwili na żądanie użytkownika.

Zadania perspektyw:

- ograniczenie dostępu do danych – zabezpieczenie przed nieautoryzowanym dostępem,
- zapewnienie użytkownikom dostęp do danych w sposób dostosowany do ich potrzeb,
- uproszczenie schematu bazy danych,
- uniezależnienie aplikacji od zmian w strukturze bazy danych,
- prezentowanie danych w inny sposób niż dane w relacjach i perspektywach bazowych (m.in. zmiana nazwy atrybutów, formaty danych).

Normalizacja – technika służąca do wyznaczania zbioru relacji o pożądanych cechach.

Wyznacza relacje bazowe opierając się o pojęcia takie jak: klucz główny, klucz kandydujący oraz zależność funkcyjna.

Poprzez proces normalizacji unikamy redundancji danych oraz anomalii aktualizacji:

- wstawiania,
- usuwania,
- modyfikacji.

Np. dla relacji

PersonelBiuro (ID_prac, imie, nazwisko, stanowisko, pensja, ID_biuro, miasto, ulica, kod).

Można wyszczególnić wszystkie wyżej wymienione anomalie:

- nowy pracownik → dane o biurze (powtórzenie),
- nowe biuro (bez pracownika) → integralność encji,
- usunięcie ostatniego pracownika z biura,
- zmiana adresu biura → aktualizacja wielu krotek.

Zależność funkcyjna – opisuje związek pomiędzy atrybutami w relacji. Jeśli A i B są atrybutami relacji R, to B jest funkcjonalnie zależny od A ($A \rightarrow B$), jeśli każda wartość A jest powiązana z dokładnie jedną wartością B. Atrybut (grupa atrybutów) oznaczony jako A nazywany jest wyznacznikiem zależności funkcyjnej.

W czasie wyznaczania zależności funkcyjnych ważne jest wyraźne rozróżnienie między wartościami atrybutu w danym momencie czasu a zbiorem wszystkich możliwych wartości.

Zależność funkcyjna jest własnością schematu relacyjnego (intencjonalnej bazy danych), a nie konkretnego stanu schematu (ekstensjonalnej bazy danych).

Pierwsza postać normalna (1NF) – relacja jest w 1NF jeśli każde przecięcie wiersza i kolumny zawiera jedną i tylko jedną wartość.

Druga postać normalna (2NF) – oznacza relację w pierwszej postaci normalnej, w której każdy atrybut spoza klucza głównego jest od niego w pełni funkcyjnie zależny.

Pełna zależność funkcyjna: atrybut B jest w pełni zależnie funkcyjnie od atrybutu A, jeśli B jest funkcyjnie zależny od A, ale nie jest funkcyjnie zależny od żadnego właściwego podzbioru A.

Zależność funkcyjna $A \rightarrow B$ jest pełną zależnością funkcyjną, jeśli skutkiem usunięcia jakiegokolwiek atrybutu z A jest utrata tej zależności.

Przykład procesu normalizacji (2NF)

KlientWynajęcie (ID_klienta, ID_nieruch, imie_k, nazwisko_k, miasto, kod, wynajecie_od, wynajecie_do, czynsz, ID_wlasciciela, imie_wl, nazwisko_wl)

Zależności częściowe:

$ID_klienta \rightarrow imie_k, nazwisko_k$

$ID_nieruch \rightarrow miasto, kod, czynsz, ID_wlasciciela, imie_wl, nazwisko_wl,$

Relacja KlientWynajęcie nie spełnia wymogów 2PN, należy stworzyć nowe relacje, tak by usunięte zostały z niej atrybuty spoza klucza głównego wraz z kopią tej części klucza głównego, od której są w pełni funkcyjnie zależne.

Trzecia postać normalna (3NF) – oznacza relację w pierwszej i w drugiej postaci normalnej, w której żaden atrybut spoza klucza głównego nie jest od niego przechodnio zależny.

Zależności przechodnia (tranzytywna): występuje w sytuacji, gdy A, B i C są atrybutami relacji, takimi że $A \rightarrow B$ i $B \rightarrow C$. Wtedy C jest tranzytywnie zależne od A poprzez B (pod warunkiem, że A nie jest zależnie funkcyjnie od B lub C).

Postać normalna Boyce'a–Codd (BNCF) – oznacza relację, w której każdy wyznacznik zależności jest kluczem kandydującym.

Opis nie formalny: jeśli wyszczególnimy wszystkie zależności funkcyjne, to nie może być zależności, której wyznacznik nie należy do klucza kandydującego (należy pamiętać o wyznacznikach wielowartościowych).

Przykład procesu normalizacji (3NF)

WlascicielNieruchomosc (ID_nieruch, miasto, kod, czynsz,
ID_wlasciciela, imie_wl, nazwisko_wl)

Zależność przechodnia: $ID_nieruch \rightarrow ID_wlasciciela$

$ID_wlasciciela \rightarrow imie_wl, nazwisko_wl.$

Atrybuty imie_wl, nazwisko_wl są funkcyjnie zależne do klucza głównego oraz od atrybutu spoza klucza głównego ID_wlasciciela.

Należy relację WlascicielNieruchomosc podzielić na dwie relacje, pamiętając o pozostawieniu w jednej z relacji atrybutu, który stanie się kluczem obcym.

Przykład procesu normalizacji (BCNF)

WizytaPacjent (ID_pac, data, godzina, ID_prac, nr_pokoju)

Klucze kandydujące: ID_pac, data

data, godzina, ID_prac

data, godzina, nr_pokoju

WizytaPacjent (ID_pac, data, godzina, ID_prac, nr_pokoju)

Zależności funkcyjne: ID_pac, data \rightarrow godzina, ID_prac, nr_pokoju

data, godzina, ID_prac \rightarrow ID_pac

data, godzina, nr_pokoju \rightarrow ID_prac, ID_pac

ID_prac, data \rightarrow nr_pokoju

Wyznacznik ostatniej zależności funkcyjnej nie należy do klucza kandydującego, w konsekwencji relacja ta nie spełnia wymogów BCNF i może podlegać anomalią aktualizacji.

Należy relację WizytaPacjent podzielić na dwie relacje, pamiętając o pozostawieniu w jednej z relacji atrybutu, który stanie się kluczem obcym.

Wizyta (ID_pac, data, godzina, ID_prac)

Pracownik (ID_prac, data, nr_pokoju)

Wykorzystanie zależności funkcyjnych pozwala do przeprowadzenia normalizacji, dzięki której uzyskujemy BCNF.

Mogą jednak wystąpić relacje, które zawierają redundancje danych, chociaż są w BCNF.

ID_pracownika	tel_prywatny	jezyk_obcy
1230	3403012	francuski
1230	3403012	angielski
1230	603992312	francuski
1230	603992312	angielski
1240	3219090	rosyjski
1240	609909090	rosyjski

W powyższej relacji:

- informacje o telefonach i językach są wzajemnie niezależne,
- brak jest zależności funkcyjnych,
- wszystkie atrybuty tworzą klucz główny.

Przyczyną redundancji danych są zależności wielowartościowe.

Zależność wielowartościowa – reprezentuje w relacji taki związek pomiędzy atrybutami (np. A, B, C), że dla każdej wartości atrybutu A istnieje zbiór wartości atrybutu B i zbiór wartości atrybutu C. Zbiory te są jednak od siebie całkowicie niezależne (notacja $A \twoheadrightarrow B$, $A \twoheadrightarrow C$).

Zależność wielowartościową $A \twoheadrightarrow B$ w relacji R nazywamy trywialną jeśli:

B jest podzbiorem A

lub $A \cup B = R$.

Czwarta postać normalna (4NF) – oznacza relację w postaci normalnej Boyce’a–Codda, która nie zawiera żadnych nietrywialnych zależności wielowartościowych.

Wyżej przedstawiona relacja ulegnie projekcji na dwie relacje o następujących schematach:

telefony (ID_pracownika, tel_prywatny),

znaj_jez (ID_pracownika, jezyk_obcy).

Utworzone relacje zawierają jedynie trywialne zależności wielowartościowe ($A \cup B = R$).

Piąta postać normalna (5NF) – nie występują zależności wielowartościowe zależne od siebie. (opis nie formalny).

Dla relacji o schemacie (IDbiuro, IDdostawca, wyposażenie),

Występują następujące zależności wielowartościowe:

$IDbiuro \twoheadrightarrow IDdostawca$,

$IDbiuro \twoheadrightarrow wyposażenie$,

$IDdostawca \twoheadrightarrow wyposażenie$.

Powyższą relację należy podzielić na 3, które są związane z wyszczególnionymi zależnościami wielowartościowymi.

Przegląd normalizacji od 2NF do BCNF

PersonelNieruchomoscWizyta

(ID_nieruch, data_w, godzina_w, ulica_n, nr_domu_n, ID_pracownika, imie, nazwisko, nr_samoch)

Zależności funkcyjne:

ID_nieruch, data_w \rightarrow godzina_w, ID_pracownika, imie, nazwisko, nr_samoch,

ID_nieruch \rightarrow ulica_n, nr_domu_n,

ID_pracownika \rightarrow imie, nazwisko,

ID_pracownika, data_w \rightarrow nr_samoch,

data_w, godzina_w, nr_samoch \rightarrow ID_nieruch, ulica_n, nr_domu_n, ID_pracownika, imie, nazwisko,

2NF – należy usunąć częściowe zależności od klucza głównego.

Nieruchomosc (ID_nieruch, ulica_n, nr_domu_n)

PersonelWizyta (ID_nieruch, data_w, godzina_w, ID_pracownika,
imie, nazwisko, nr_samoch)

Zależności funkcyjne dla relacji PersonelWizyta:

ID_nieruch, data_w → godzina_w, ID_pracownika, imie, nazwisko,
nr_samoch,

ID_pracownika → imie, nazwisko,

ID_pracownika, data_w → nr_samoch,

data_w, godzina_w, nr_samoch ID_nieruch, ID_pracownika,
imie, nazwisko,

ID_pracownika, data_w, godzina_w \rightarrow ID_nieruch,

3NF – należy usunąć zależności przechodnie.

Personel (ID_pracownika, imie, nazwisko)

Wizyta_w_nieruch (ID_nieruch, data_w, godzina_w, ID_pracownika,
nr_samoch)

Zależności funkcyjne dla relacji Wizyta_w_nieruch:

$ID_nieruch, data_w \rightarrow godzina_w, ID_pracownika, nr_samoch,$

$ID_pracownika, data_w \rightarrow nr_samoch,$

$data_w, godzina_w, nr_samoch \rightarrow ID_nieruch, ID_pracownika,$

$ID_pracownika, data_w, godzina_w \rightarrow ID_nieruch,$

BCNF – każdy wyznacznik zależności funkcyjnej jest kluczem kandydującym.

PersonelSamochod (data_w, ID_pracownika, nr_samoch)

Wizyta (ID_nieruch, data_w, godzina_w, ID_pracownika)

Podsumowanie:

- 1NF każdy atrybut niekluczowy jest funkcyjnie zależny od klucza głównego,
- 2NF eliminuje niepełne zależności funkcyjne, dla których wyznacznikiem jest klucz główny,
- 3NF usuwa wszystkie przechodnie zależności funkcyjne,
- BCNF eliminuje zależności funkcyjne, których wyznaczniki nie należą do klucza kandydującego,
- 4NF usuwa wszystkie wielokrotne zależności wielowartościowe, których wyznaczniki nie należą do klucza kandydującego,
- 5NF usuwa zależności wielowartościowe zależne od siebie, których wyznaczniki nie należą do klucza kandydującego.

Reguły transformacji modelu ER na schemat relacyjny

Encja \Rightarrow relacja

Instancja encji \Rightarrow krotka

Atrybut encji \Rightarrow kolumna relacji

Unikalny identyfikator encji
 \Rightarrow klucz główny relacji

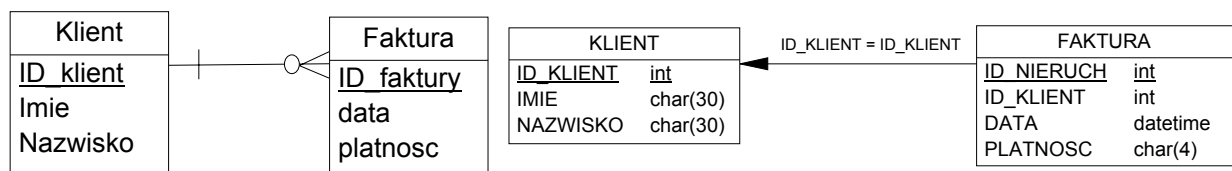
Związek \Rightarrow klucz obcy

Związek jeden–do–wielu:

encja po stronie „jeden” – nadrzędna

encja po stronie „wiele” – podrzędna

umieszczenie atrybutów klucza głównego encji nadrzędnej w encji podrzędnej, które stają się kluczem obcym.



W tym przypadku zostanie nałożone ograniczenia *not null* na kolumnę będącą kluczem obcym.

Związki nadencja – podencja

(1) transformacja do pojedynczej relacji

Zasady:

jedna relacja,
schemat relacji: atrybuty wszystkich encji z hierarchii + dodatkowa kolumna, określająca typ specjalizacji.

Kiedy stosować:

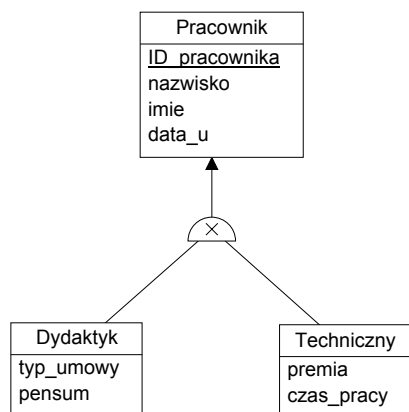
większość atrybutów w nadtypie,
większość związków do nadtypu.

Zalety:

uproszczenie schematu bazy danych.

Wady:

atrybuty obowiązkowe podtypu stają się kolumnami opcjonalnymi.



PRACOWNIK	
ID_PRACOWNIKA	integer
NAZWISKO	char(30)
IMIE	char(30)
DATA_U	date
TYP_PRACOWNIKA	numeric(1)
TYP_UMOWY	char(12)
PENSUM	decimal(6)
PREMIA	decimal(5)
CZAS_PRACY	char(2)

(2) transformacja do oddzielnych relacji dla podtypów

Zasady:

jedna relacja dla każdego podtypu,
schemat relacji: atrybuty nadtypu + atrybuty podtypu.

Kiedy stosować:

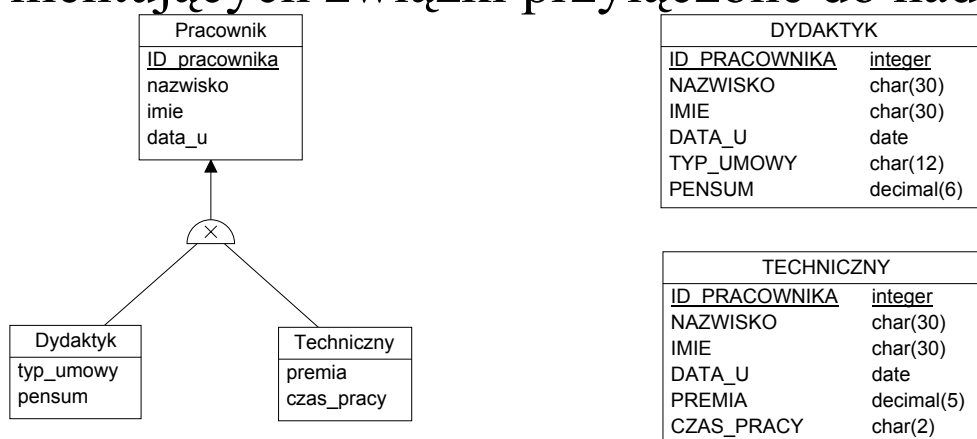
większość atrybutów w podtypach
większość związków do podtypów

Zalety:

zachowanie obowiązkowości atrybutów w podtypach.

Wady:

komplikacja schematu
konieczność powielenia kluczy obcych implementujących związki przyłączone do nadtypu



(3) transformacja do oddzielnych relacji dla podtypów i nadtypu

Zasady:

jedna relacja dla każdego podtypu oraz nadtypu.

Kiedy stosować:

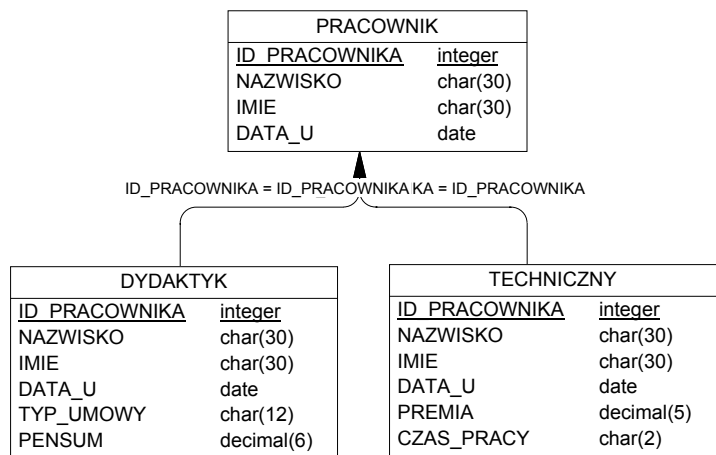
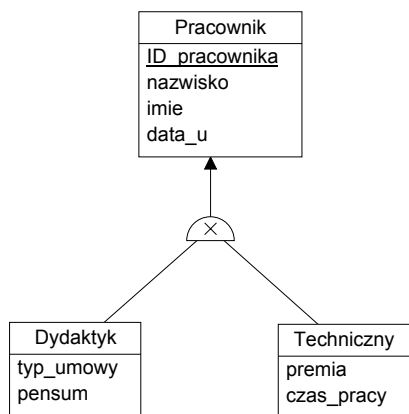
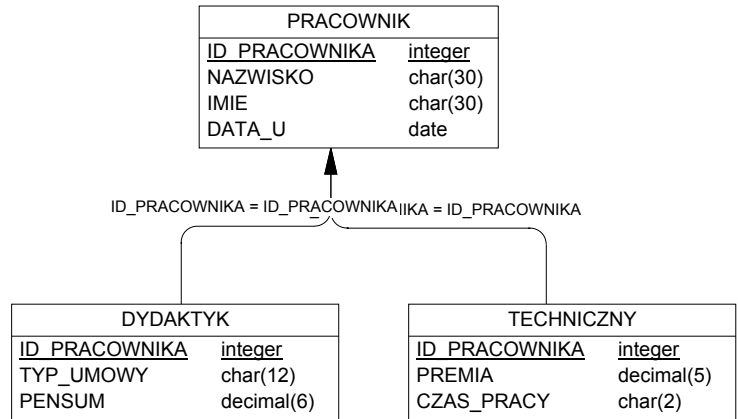
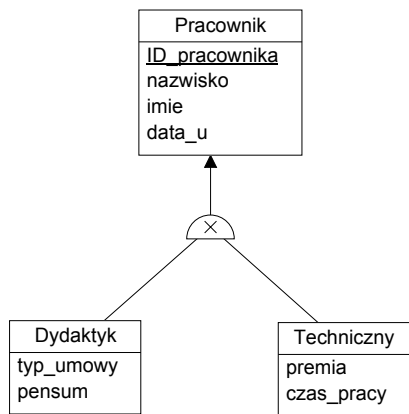
większość atrybutów w podtypach
większość związków do podtypów

Zalety:

zachowanie obowiązkowości atrybutów w podtypach.

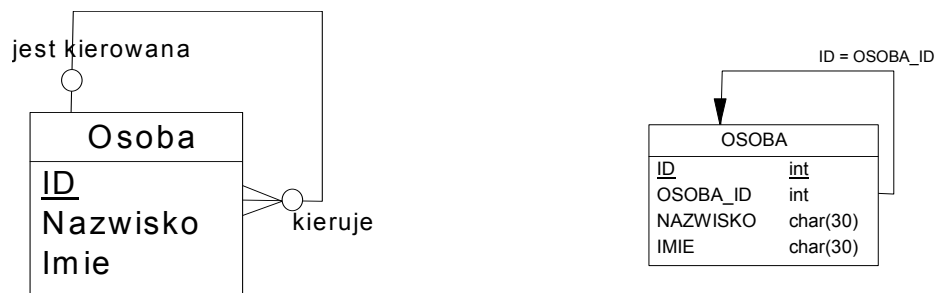
Wady:

komplikacja schematu

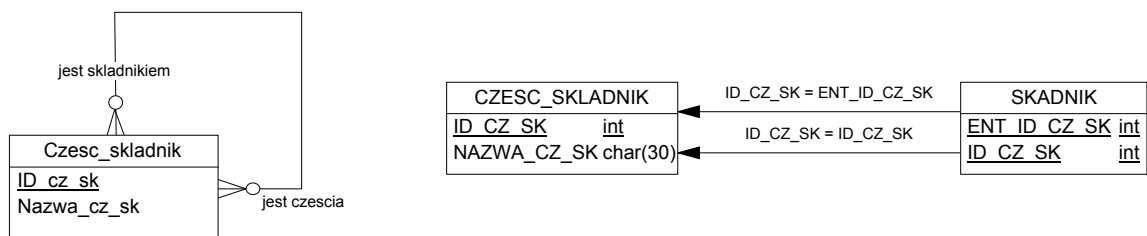
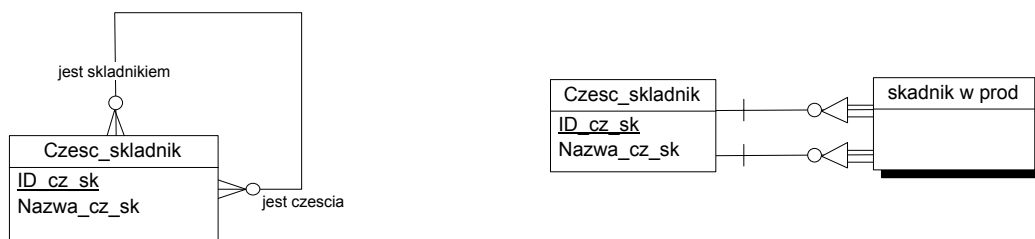


Związek rekurencyjny

(1) związek jeden–do–wiele



(2) związek wiele–do–wielu

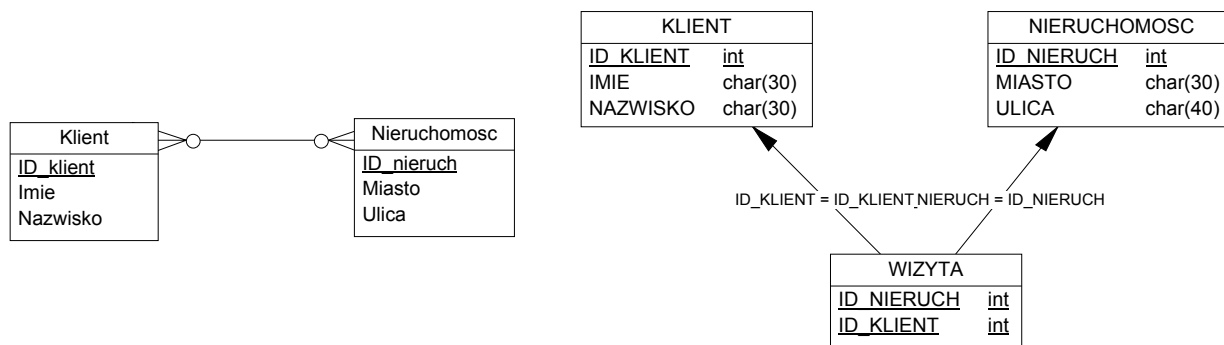
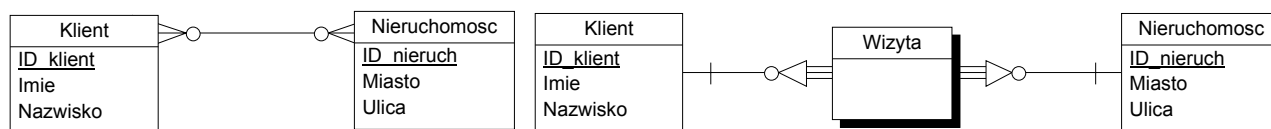


Związek wiele-do-wielu:

zostaje utworzona nowa relacja (encja),

w nowej relacji zostają utworzone klucze obce, wskazujące na klucze główne relacji w związku,

kolumny obu kluczy obcych tworzą klucz główny relacji.

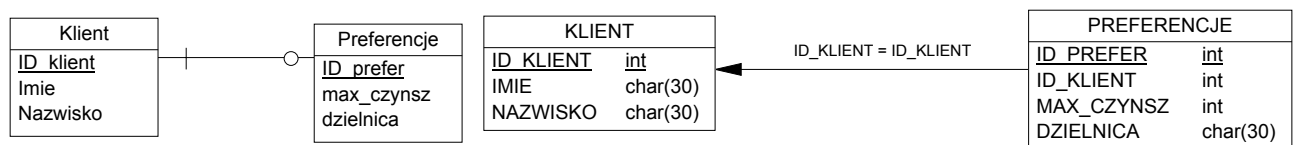
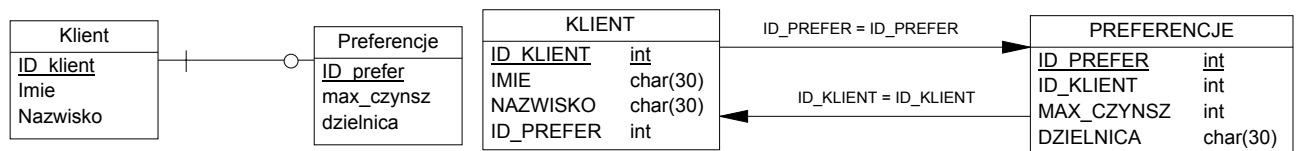


Związek jednoznaczny jeden–do–jednego

Przypadek występowania takiego związku jest bardzo rzadko spotykany.

Uczestnictwo obowiązkowe po obu stronach – należy encje połączyć w jedną relację, wybierając klucz główny jednej z nich jako klucz główny nowej relacji.

Uczestnictwo obowiązkowe po jednej stronie – do encji występującej opcjonalnie zostanie dodany klucz obcy wskazujący na encję nadrzędną.



Uczestnictwo opcjonalne po obu stronach – wybór encji nadrzędnej dokonuje projektant.

Integralność relacji

Dziedzina atrybutu \Rightarrow ograniczenie *check*

Dozwolona wartość pusta \Rightarrow *null*

Zabroniona wartość pusta \Rightarrow *not null*
(*mandatory*)

Integralność referencyjna – klucz obcy wiąże krotkę relacji podrzędnej z krotką relacji nadrzędnej o pasującej wartości odpowiedniego klucza kandydującego (najczęściej klucza głównego).

Zasadnicza reguła dla wartości pustych dla klucza obcego:

- uczestnictwo obowiązkowe relacji podrzędnej – wartości puste nie są dozwolone,
- uczestnictwo opcjonalne relacji podrzędnej – wartości puste są dozwolone.

Więzy występowania definiują warunki, które muszą być spełnione przy wstawianiu, modyfikacji, usuwaniu klucza kandydującego lub obcego:

- wstawienie krotki do relacji podrzędnej
- usunięcie krotki z relacji podrzędnej
- wstawienie krotki do relacji nadrzędnej
- usunięcie (aktualizacja) krotki z relacji nadrzędnej:
 - *no action (restricts)* – uniemożliwienie usunięcia jeżeli są krotki zależne w relacji podrzędnej,
 - *cascade* – usunięcie krotki nadrzędnej pociąga za sobą usunięcia wszystkich związanych z nią krotek podrzędnych,
 - *set null* – usunięcie krotki nadrzędnej powoduje ustawienie wartości *null* w odpowiadających kluczach obcych
 - *set default* – j.w. tylko wartość domyślna
 - *no check (none)* – usunięcie krotki nadrzędnej nie powoduje zmian służącym zachowaniu integralności (czy jest właściwe?)

Język **SQL** (*Structured Query Language*) służy do definiowania baz danych oraz operowania na niej. Jest językiem **uniwersalnym**, dzięki czemu praca na różnych systemach baz danych sprowadza się do wydawania tych samych komend (istnieją jednak różne dialekty).

Składnię języka SQL można podzielić na trzy części:

- **język definiowania struktur danych** - DDL (*Data Definition Language*) - operacje na tabelach, takie jak: tworzenie, modyfikacja oraz usuwanie,
- **język do wybierania i manipulowania danymi** - DML (*Data Manipulation Language*) - manipulowanie danymi umieszczonymi w tabelach: wstawienie, modyfikowanie, usuwanie oraz wyszukiwanie,
- **język do zapewnienia bezpieczeństwa dostępu do danych** - DCL (*Data Control Language*) - jest używany głównie przez administratorów systemu baz danych do nadawania odpowiednich uprawnień do korzystania z bazy danych.

SELECT – wyszukiwanie danych z jednej lub wielu relacji.

```
SELECT  [DISTINCT|ALL]  { * | wyrażenie
[ [AS] alias ] } [, ... ]
FROM      nazwa_tabeli [ alias ] [, ... ]
[WHERE      warunek selekcji wierszy]
[GROUP BY lista_kol] [HAVING sel_grup]
[ORDER BY wyrażenie [ASC|DESC] ] ;
```

DISTINCT – eliminuje powtarzające się wiersze,

wyrażenie – nazwa kolumny lub wyrażenie zawierające nazwy kolumn,

***** – wszystkie kolumny,

alias – nazwa nadana wyrażeniu na liście **SELECT**,

ASC – uporządkowanie rosnące (domyślnie),

DESC – uporządkowanie malejące,

Kolejność przetwarzania klauzul polecenia SELECT:

FROM	tabela (tabele), z których korzystamy
WHERE	wiersze spełniające warunek selekcji
GROUP BY	grupuje wiersze o tej samej wartości wskazanej kolumny
HAVING	wybór grup ze względu na podany warunek selekcji grup
SELECT	wskazuje, które kolumny mają pojawić się w wyniku
ORDER BY	uporządkowanie wyniku

Przykłady:

Wyszukanie wszystkich kolumn i wierszy

```
SELECT    id_klient, imie, nazwisko,  
           nr_tel, max_czynsz
```

```
FROM      Klient;
```

równoważnie

```
SELECT    *
```

```
FROM      Klient;
```

Wyszukanie wszystkich imion właścicieli

```
SELECT    imie  
FROM      Wlasciciel;
```

eliminacja powtórzeń w powyższym pytaniu

```
SELECT      DISTINCT    imie  
FROM        Wlasciciel;
```

Podaj imię i nazwisko pracownika wraz z pensją miesięczną (pola wyliczane)

```
SELECT    imie, nazwisko, pensja/12 AS  
           pensjaMiesiecz  
FROM      Personel;
```

WHERE – wybieranie wierszy, możliwe jest:

porównanie

= równe

<> różne != różne

< mniejsze <= mniejsze niż lub równe

> większe >= większe niż lub równe

można stosować nawiasy oraz operatory logiczne

NOT przed AND przed OR

Podaj dane pracowników, których pensja jest wyższa niż 1000zł

```
SELECT    imie, nazwisko, stanowisko,  
            pensja  
FROM      Personel  
WHERE     pensja>1000;
```

sprawdzenie zakresu

BETWEEN **NOT BETWEEN**

Podaj dane pracowników, których pensja jest pomiędzy 1000zł a 2000zł

```
SELECT    imie, nazwisko, stanowisko,  
            pensja  
FROM      Personel  
WHERE     pensja BETWEEN 1000 AND 2000;
```

przynależność do zbioru

IN NOT IN

Podaj dane pracowników, którzy są kierownikami bądź dyrektorami

```
SELECT    imie, nazwisko, stanowisko
FROM      Personel
WHERE     stanowisko
           IN ('kierownik', 'dyrektor');
```

równoważnie

```
SELECT    imie, nazwisko, stanowisko
FROM      Personel
WHERE     stanowisko='kierownik' OR
           stanowisko='dyrektor';
```

dopasowanie do wzorca

LIKE NOT LIKE

Symbole zastępcze w SQL:

% – zastępuje ciąg znaków dowolnej długości,

_ – zastępuje dowolny (jeden) znak.

np.: LIKE 'W%' pierwszym znakiem musi być *W*

 LIKE 'a__' dokładnie trzy znaki, pierwszy *a*

 NOT LIKE 'H%' pierwszym znakiem słowa nie
 może być *H*

LIKE '15#%' ESCAPE '#' – zdejmujemy znaczenie
 specjalne znaku %

Znajdź te uwagi w których wystąpiło słowo „hałas”

```
SELECT    id_klienta,  
            id_nieruchomosci, uwagi  
  
FROM      Wizyta  
  
WHERE     uwagi LIKE '%hałas%';
```

wartości puste

IS NULL IS NOT NULL

Znajdź identyfikatory tych nieruchomości do których nie ma uwag

```
SELECT    DISTINCT id_nieruchomosci
FROM      Wizyta
WHERE      uwagi IS NULL;
```

ORDER BY – porządkowanie wyniku.

Podaj dane pracowników, których pensja jest wyższa niż 1000zł, uporządkuj według malejącej pensji

```
SELECT     imie, nazwisko, stanowisko,
             pensja
FROM       Personel
WHERE      pensja>1000
ORDER BY   pensja DESC;
```

równoważnie

```
ORDER BY 4 DESC
```

4 oznacza sortowanie według czwartej kolumny

Podaj dane pracowników, których pensja jest wyższa niż 1000zł, uporządkuj według malejącej pensji i rosnąc według nazwiska

```
SELECT    imie, nazwisko, stanowisko,  
            pensja  
FROM      Personel  
WHERE      pensja>1000  
ORDER BY pensja DESC, nazwisko;
```

Funkcje sumaryczne (agregujące):

COUNT — liczba rekordów,

AVG — średnia arytmetyczna, (tylko pola liczbowe)

SUM — suma wartości, (tylko pola liczbowe)

MAX — wartość maksymalna,

MIN — wartość minimalna w kolumnie.

Powyższe funkcje pomijają wartości puste.

COUNT(*) — nie pomija wartości pustych.

DISTINCS — wyeliminuje wartości powtarzające się

W ilu nieruchomościach czynsz jest wyższy niż 500zł

```
SELECT    COUNT(*) AS liczba
FROM      Nieruchomosc
WHERE      czynsz>500;
```

Ilość nieruchomości odwiedzonych w maju 2004r

```
SELECT    COUNT(DISTINCTS ID_nieruch)
           AS liczba
FROM      Wizyta
WHERE      dataWizyty BETWEEN
           '1.05.2004' AND '31.05.2005' ;
```

Ilu jest stażystów i jaka jest ich sumaryczna pensja

```
SELECT    COUNT(ID_prac) AS liczba,
           SUM(pensja) AS suma
FROM      Personel
WHERE      stanowisko='stażysta' ;
```


Podaj najmniejszą, największą i średnią pensję

```
SELECT    MIN(pensja) AS minimum,  
           MAX(pensja) AS maksimum,  
           AVG(pensja) AS średnia  
  
FROM      Personel;
```

GROUP BY – grupowanie wyniku.

Dzieli krotki na grupy i umożliwia wykonanie funkcji sumarycznych na wartościach należących do poszczególnych grup.

Wszystkie nazwy kolumn na liście **SELECT** muszą występować w klauzuli **GROUP BY**, chyba że nazwa kolumny jest używana tylko jako argument funkcji grupującej.

Wartości puste uważane są za równe.

Oblicz dla każdego biura liczbę zatrudnionych pracowników oraz ich sumaryczną pensję

```
SELECT    id_biura, COUNT(id_prac) AS
            liczba, SUM(pensja) AS suma
FROM      Personel
GROUP BY  id_biura
ORDER BY  id_biura;
```

HAVING – wybór grup.

Klauzulę **HAVING** stosuje się z klauzulą **GROUP BY** w celu wybrania grup.

Oblicz dla każdego biura zatrudniającego więcej niż 1 pracownika liczbę zatrudnionych pracowników oraz ich sumaryczną pensję

```
SELECT    id_biura, COUNT(id_prac) AS
            liczba, SUM(pensja) AS suma
FROM      Personel
GROUP BY  id_biura
HAVING COUNT(id_prac) > 1
ORDER BY  id_biura;
```

Podzapytania

W podzapytaniu nie można używać ORDER BY

Lista SELECT podzapytania musi składać się z pojedynczej nazwy kolumny lub wyrażenia

Podaj imię i nazwisko pracowników pracujących we Wrocławiu

```
SELECT    imie, nazwisko
FROM      Personel
WHERE      id_biura =
              ( SELECT    id_biura
                FROM      Biuro
                WHERE      miasto='Wrocław' );
```

Podaj imię, nazwisko i stanowisko pracowników, których pensja jest wyższa od średniej wraz z różnicą pensji i średniej

```
SELECT    imie, nazwisko, stanowisko,
            pensja - ( SELECT AVG(pensja)
                      FROM Personel ) AS roznica
FROM      Personel
WHERE      pensja > ( SELECT AVG(pensja)
                      FROM Personel );
```

Błędnie !!!

- **WHERE** pensja > **AVG**(pensja)

Funkcje agregujące tylko w **SELECT** i **HAVING**

- **WHERE** (**SELECT** **AVG**(pensja)
FROM Personel) < pensja;

Podzapytanie musi być po prawej stronie porównania

Podaj adresy nieruchomości nadzorowanych przez pracowników z Wrocławia

```
SELECT    miasto, kodPocz, ulica,  
           nr_dom, nr_miesz  
  
FROM      Nieruchomosc  
  
WHERE     id_prac IN  
           (SELECT id_prac  
           FROM     Personel  
           WHERE    id_biuro =  
                   (SELECT id_biuro  
                   FROM     Biuro  
                   WHERE    miasto=  
                           'Wrocław'));
```

Zapytania dotyczące wielu tabel

Podaj identyfikatory nieruchomości z uwagami oraz imiona, nazwiska i identyfikatory odwiedzających je klientów

```
SELECT    id_nieruch, uwagi,  
            k.id_klient, imie, nazwisko  
  
FROM      Klient AS k, Wizyta AS w  
  
WHERE      k.id_klient=w.id_klient  
  
ORDER BY id_nieruch, nazwisko;
```

Znajdź liczbę nieruchomości nadzorowanych przez poszczególnych pracowników – podaj identyfikator biura identyfikator pracownika oraz odpowiednią liczbę

```
SELECT      p.id_biura, p.id_prac,  
              COUNT(*) AS liczba  
  
FROM        Personel AS p,  
              Nieruchomosc AS n  
  
WHERE        p.id_prac=n.id_prac  
  
GROUP BY    p.id_biura, p.id_prac  
  
ORDER BY    p.id_biura, p.id_prac;
```

Powyżej przedstawione złączenia są tzw. złączeniami wewnętrznymi – operacja złączenia kojarzy dane z dwóch tabel, tworząc pary wierszy, w których kolumny złączenia mają taką samą wartość. Jeżeli wiersz tabeli nie zostanie połączony w parę, to nie trafi do tabeli wynikowej.

Np. dla zapytania „*Podaj identyfikatory...*”, które dotyczy stanu bazy jak poniżej:

Klient

id_klient	imie	nazwisko
1	Ania	Kowalska
2	Piotr	Nowak
3	Tomasz	Nowak
4	Zygmunt	Zarzeczny

Wizyta

id_wizyty	id_klient	id_nieruch	data	uwagi
1	1	2	2005-10-11	
2	1	3	2005-10-12	hałas
3	2	1	2005-10-16	daleko
4	2	3	2005-11-16	
5	3	2	2005-10-14	wysoko, glosno

otrzymujemy:

id_nieruch	uwagi	id_klient	imie	nazwisko
1	daleko	2	Piotr	Nowak
2		1	Ania	Kowalska
2	wysoko, glosno	3	Tomasz	Nowak
3	hałas	1	Ania	Kowalska
3		2	Piotr	Nowak

Powyższej własności nie ma złączenie zewnętrzne – zachowane są wiersze niespełniające warunków złączenia.

Lewostronne złączenie zewnętrzne – wynikowej tabeli wiersze, które spełniają podany warunek oraz wiersze z pierwszej (lewej) tabeli, dla których nie istnieje pasujący wiersz z drugiej (prawej) tabeli.

Podaj identyfikatory nieruchomości z uwagami oraz imiona, nazwiska i identyfikatory odwiedzających je klientów (podaj również odpowiednie dane dla osób nie oglądających nieruchomości)

```
SELECT    id_nieruch, uwagi,
            k.id_klient, imie, nazwisko

FROM      Klient AS k LEFT JOIN
            Wizyta AS w ON
            k.id_klient=w.id_klient

ORDER BY id_nieruch, nazwisko;
```

id_nieruch	uwagi	id_klient	imie	nazwisko
		4	Zygmunt	Zarzeczny
1	daleko	2	Piotr	Nowak
2		1	Ania	Kowalska
2	wysoko, glosno	3	Tomasz	Nowak
3	hałas	1	Ania	Kowalska
3		2	Piotr	Nowak

W podobny sposób można określić:

prawostronne złączenie zewnętrzne

RIGHT JOIN

pełne złączenie zewnętrzne

FULL JOIN

Oblicz dla każdego biura podając identyfikator oraz miasto liczbę zatrudnionych pracowników oraz ich sumaryczną pensję i średnią w biurze

```
SELECT    b.id_biura, miasto,  
            COUNT(id_prac) AS liczba,  
            SUM(pensja) AS suma,  
            AVG(pensja) AS srednia  
  
FROM      Biuro AS b RIGHT JOIN  
            Personel AS p ON  
            p.id_biura=b.id_biura  
  
GROUP BY b.id_biura, miasto;
```

Dla powyższego pytania i stanu bazy:

Personel

id_prac	imie	nazwisko	stanowisko	pensja	id_biura
1	Piotr	Nowak	dyrektor	10 000,00 zł	1
2	Ania	Nowak	stażysta	5 000,00 zł	1
3	Zenon	Glinka	portier	3 000,00 zł	1
4	Adam	Szulik	dyrektor	9 000,00 zł	2
5	Andrzej	Lamik	stażysta	5 000,00 zł	2
6	Paweł	Kowalski	kierownik	7 000,00 zł	3

Biuro

id_biura	miasto	ulica	kod
1	Wrocław	Górna	50-135
2	Opole	Długa	44-158
3	Poznań	Kręta	75-125
4	Wrocław	Wąska	52-333

tabela przejściowa:

Biuro **AS** b **RIGHT JOIN** Personel **AS** p **ON**
p.id_biura=b.id_biura

b.id_biura	miasto	ulica	kod	id_prac	imie	nazwisko	stanowisko	pensja	p.id_biura
1	Wrocław	Górna	50-135	1	Piotr	Nowak	dyrektor	10 000,00 zł	1
1	Wrocław	Górna	50-135	2	Ania	Nowak	stażysta	5 000,00 zł	1
1	Wrocław	Górna	50-135	3	Zenon	Glinka	portier	3 000,00 zł	1
2	Opole	Długa	44-158	4	Adam	Szulik	dyrektor	9 000,00 zł	2
2	Opole	Długa	44-158	5	Andrzej	Lamik	stażysta	5 000,00 zł	2
3	Poznań	Kręta	75-125	6	Paweł	Kowalski	kierownik	7 000,00 zł	3

tabela wynikowa:

id_biura	miasto	liczba	suma	srednia
1	Wrocław	3	18 000,00 zł	6 000,00 zł
2	Opole	2	14 000,00 zł	7 000,00 zł
3	Poznań	1	7 000,00 zł	7 000,00 zł

Łączenie tabel wynikowych

W standardzie zdefiniowane są trzy operacje, które dotyczą operacji na zbiorach:

- suma **UNION**
- przekrój **INTERSECT**
- różnica **EXCEPT**

Tabele dla których wykonujemy powyższe operacje muszą spełniać pewne warunki:

- taka sama liczba kolumn,
- kolumny tego samego typu i rozmiaru.

Duplikaty wierszy są pomijane.

Podaj listę wszystkich miast, w których znajduje się biuro lub nieruchomość

```
(SELECT miasto FROM Biuro)
```

UNION

```
(SELECT miasto FROM Nieruchomosc) ;
```

Podaj listę wszystkich miast, w których znajduje się biuro i nieruchomość

```
(SELECT miasto FROM Biuro)
```

INTERSECT

```
(SELECT miasto FROM Nieruchomosc);
```

Uwaga

W dialekcie Sybase nie jest zaimplementowany operator tej operacji, więc należy zapisać ją równoważnie:

```
SELECT    DISTINCTS b.miasto  
FROM      Biuro AS b, Nieruchomosc AS n  
WHERE     b.miasto=n.miasto;
```

Podaj listę wszystkich miast, w których znajduje się biuro, ale nie ma żadnej nieruchomości

```
(SELECT miasto FROM Biuro)
```

EXCEPT

```
(SELECT miasto FROM Nieruchomosc);
```

Uwaga

W dialekcie Sybase nie jest zaimplementowany operator tej operacji, więc należy zapisać ją równoważnie:

```
SELECT    DISTINCTS miasto
FROM      Biuro
WHERE      miasto    NOT IN
                (SELECT  miasto
                FROM      Nieruchomosc) ;
```

INSERT – dodanie wierszy do tabeli.

```
INSERT INTO nazwaTabeli[(lista_kol)]
VALUES (lista_wartosci);
```

```
INSERT    INTO Biuro
VALUES    ( '5', 'Gdynia', 'Morska', '30-
                230' );
```

W przypadku umieszczania wartości znakowych lub daty należy je podać w apostrofach.

Wypełnij tabelę „PracLiczNieruch”, wykorzystując tabele Personel i Nieruchomosc.

PracLiczNieruch	(<u>id_prac</u> ,	imie,	nazwisko,
	licz_nieruchomosci)		

```
INSERT INTO PracLiczNieruch
(SELECT p.id_prac, imie, nazwisko
COUNT(*))
FROM Personel AS p
Nieruchomosc AS n
WHERE p.id_prac= n.id_prac
GROUP BY p.id_prac, imie, nazwisko)
UNION
(SELECT p.id_prac, imie, nazwisko, 0
FROM Personel
WHERE p.id_prac NOT IN
(SELECT DISTINCT p.id_prac
FROM Nieruchomosc));
```

UPDATE – modyfikacja danych.

```
UPDATE    nazwaTabeli
SET       naz_kol1=wart1
            [,naz_kol1=wart1 ...]
[WHERE     warunek selekcji];
```

Podnieś wszystkim pracownikom pensję o 3 %

```
UPDATE    Personel
SET       pensja=pensja*1,03;
```

Podnieś wszystkim stażystom pensję o 3 %

```
UPDATE    Personel

SET       pensja=pensja*1,03

WHERE     stanowisko='stażysta';
```

DELETE – usuwanie danych.

```
DELETE    FROM    nazwaTabeli  
[WHERE    warunek selekcji];
```

Usuń wszystkich stażystów

```
DELETE    FROM    Personel  
  
WHERE      stanowisko='stażysta';
```

Instrukcje z grupy DDL

[CREATE, DROP]	DATABASE
[CREATE, ALTER, DROP]	DOMAIN
[CREATE, ALTER, DROP]	TABLE
[CREATE, DROP]	VIEW
[CREATE, DROP]	INDEKS

Właściwości dotyczące więzów integralności:

- dane wymagane

miasto **VARCHAR(30) NOT NULL**

- więzy dziedzinowe

- zastosowanie klauzuli CHECK dla konkretnej kolumny

płeć **CHAR NOT NULL CHECK (płeć IN ('K', 'M'))**

- definiowanie dziedziny

```
CREATE DOMAIN naz_dzie[AS] typ_danych  
[DEFAULT      wartosc_domyslna]  
[CHECK (warunek selekcji)];
```



```
CREATE DOMAIN RodzajPlci AS CHAR  
DEFAULT 'M'  
CHECK (VALUE IN ('M', 'K')) ;
```

płeć RodzajPlci **NOT NULL**

W parametrze *warunek selekcji* można odwołać się do innej tabeli.

```
CREATE DOMAIN NumerBiura AS CHAR(5)  
CHECK (VALUE IN  
(SELECT id_biura FROM Biuro)) ;
```

- integralność encji

```
PRIMARY KEY (id_klienta)
```

```
PRIMARY KEY (id_klienta, id_nieruch)
```

- integralność referencyjna

```
FOREGIN KEY (id_biura) REFERENCES Biuro
```

W podklauzulach ON DELETE i ON UPDATE zdefiniowane są akcje referencyjne związane z kasowaniem i aktualizacją klucza kandydującego (głównego) tabeli nadrzędnej (NO ACTION, SET DEFAULT, SET NULL, CASCADE)

```
FOREGIN KEY (id_biura)
```

```
REFERENCES Biuro ON DELETE SET NULL
```

```
FOREGIN KEY (id_biura)
```

```
REFERENCES Biuro ON UPDATE CASCADE
```

- więzy ogólne (asercje)

Stanowią dodatkowe ograniczenia nałożone na bazę danych.

```
CREATE ASSEARATION naz_aseracji  
CHECK (warunek selekcji);
```

Osoba nie może nadzorować więcej niż 100 nieruchomości.

```
CREATE    ASSERTION Pacownik_do_100  
CHECK (NO EXIST ( SELECT id_prac  
                        FROM Nieruchomosc  
                        GROUP BY id_prac  
                        HAVING COUNT (*) > 100 ) )
```

Perspektywa – dynamicznie obliczany wynik jednej lub wielu operacji relacyjnych tworzących nową relację z relacji bazowych. Perspektywa jest relacją wirtualną, która nie musi fizycznie istnieć w bazie danych, ale może być wyliczona w każdej chwili. Nie przechowuje danych, a jedynie udostępnia dane zawarte w tabelach lub innych perspektywach. Oparta jest o klauzulę SELECT.

```
CREATE VIEW naz_persp [(n_kol[,...])]
AS zap_SELECT [WITH[CASCADED|LOCAL]
                CHECK OPTION] ;
```

Utwórz perspektywę, taka w której widziane są tylko dane osób pracujących w biurze o identyfikatorze 33.

```
CREATE      VIEW Personel_B_33
AS          SELECT *
              FROM Personel
              WHERE id_biura='33' ;
```

Utwórz perspektywę, taka w której widziane są tylko dane osób (bez pensji) pracujących w biurze o identyfikatorze 33.

```
CREATE      VIEW Personel_B_33_bp
AS          SELECT imie, nazwisko, stanowisko, plec
              FROM Personel
              WHERE id_biura='33' ;
```

```
CREATE    VIEW Personel_B_33_bp
AS        SELECT imie, nazwisko, sta-
            nowisko, plec
            FROM Personel_B_33;
```

Utwórz perspektywę zawierającą liczbę nieruchomości nadzorowanych przez poszczególnych pracowników – podaj identyfikator biura identyfikator pracownika oraz odpowiednią liczbę

```
CREATE    VIEW Licz_N_Prac
            (id_biura, p.id_prac, liczba)
SELECT    p.id_biura, p.id_prac,
            COUNT(*)
FROM      Personel AS p,
            Nieruchomosc AS n
WHERE     p.id_prac=n.id_prac
GROUP BY p.id_biura, p.id_prac;
```

Podaj odpowiednie dane dla biura o identyfikatorze 33

```
SELECT    id_prac, liczba
FROM      Licz_N_Prac
WHERE     id_biura='33';
```

Ograniczenia dla perspektyw:

- Jeśli kolumna perspektywy jest wyliczana to nazwa tej kolumny może pojawiać się tylko w klauzulach **SELECT** i **ORDER BY** zapytań wykorzystujących perspektywę (nie może być również argumentem funkcji agregujących).

Np. dla perspektywy *Licz_N_Prac* błędne są następujące zapytania:

```
SELECT    COUNT(liczba)
FROM      Licz_N_Prac;
```

```
SELECT    *
FROM      Licz_N_Prac
WHERE      liczba>5;
```

- Nie wolno wykonać złączenia perspektywy grupującej z tabelą bazową lub inną perspektywą.

W przypadku modyfikowania danych poprzez perspektywy istnieje szereg ograniczeń dotyczących tych perspektyw:

- perspektywa dotyczy tylko jednej tabeli (klauszula FROM),
- nie występują kolumny, które są: stałą, wyrażeniem, funkcją agregującą oraz kolumny mogą wystąpić tylko raz,
- klauszula WHERE nie zawiera żadnego zagnieżdżonego podzapytania SELECT,
- w definicji perspektywy nie występuje: GROUP BY, HAVING, DISTINCTS.

Klauszula WITH CHECK OPTION

Zastosowanie tej klauszuli zapobiega zmianie ilości wierszy (zapobiega istnieniu wierszy migrujących). Oznacza to, że po wykonaniu polecenia aktualizacji dla perspektywy ilość wierszy musi być taka sama jak przed aktualizacją (pozostaje zgodność kryterium zapytania).

Opcja domyślna to **CASCADDED**.

```

CREATE    VIEW Personel_B_33
AS        SELECT *
            FROM Personel
            WHERE id_biura='33'

WITH CHECK OPTION;

```

Chcemy zmienić numer biura w jednym z wierszy perspektywy:

```

UPDATE    Personel_B_33
SET       id_biura='44'
WHERE      id_prcow='2';

```

Omawiana klauzula uniemożliwi zmianę tego wiersza.

Tabela *Biuro*

id_nrac	imie	nazwisko	stanowisko	nensia	id_biura
1	Piotr	Nowak	dvrektor	10 000.00	33
2	Ania	Wilk	stażvsta	5 000.00 zł	33
3	Zenon	Glinka	nortier	3 000.00 zł	33
4	Adam	Szulik	dvrektor	9 000.00 zł	22
5	Andrzej	Lamik	stażvsta	5 000.00 zł	22
6	Paweł	Kowalski	kierownik	7 000.00 zł	5

Widok *Personel_B_33*

id_nrac	imie	nazwisko	stanowisko	nensia	id_biura
1	Piotr	Nowak	dvrektor	10 000.00	33
2	Ania	Wilk	stażvsta	5 000.00 zł	33
3	Zenon	Glinka	nortier	3 000.00 zł	33

Najważniejsze zalety i wady perspektyw w SQL:

- niezależność danych,
- uproszczenie zapytań,
- dostosowanie do użytkownika,
 - ograniczona możliwość modyfikacji,
 - ograniczenie struktury,
 - wydajność.

Kontrola dostępu:

Każdy użytkownik bazy danych ma przypisany identyfikator, który jest nazwą w SQL. Jest to możliwe dzięki mechanizmowi zabezpieczeń istniejącemu w każdym SZBD.

Prawa dostępu (dla tabeli lub perspektywy) to: SELECT, INSERT, UPDATE, DELETE, REFERENCES, ALTER.

Nadawanie uprawnień – GRANT

```
GRANT {lista_praw|ALL PRIVILEGES}  
ON nazwa_obiektu  
TO {lista_id_uzytkownikow|PUBLIC}  
[WITH GRANT OPTION]
```

Nadaj prawa prawo UPDATE do kolumny „pensja” dla użytkownika o identyfikatorze Kierownik

```
GRANT      UPDATE (pensja)  
  
ON         Personel  
  
TO         Kierownik;
```

Nadaj wszystkie prawa dla użytkownika o identyfikatorze Kierownik do tabeli Personel

```
GRANT      ALL PRIVILEGES  
  
ON         Personel  
  
TO         Kierownik  
  
WITH GRANT OPTION;
```

Odbieranie uprawnień – REVOKE

Uwaga!! – polecenie REVOKE odbiera prawa nadane tylko przez użytkownika, który je nadał i nie odbiera praw nadanych przez innych użytkowników.

Transakcja – operacja lub ciąg operacji odwołujących się do bazy danych, wykonywanych przez jednego użytkownika lub jeden program aplikacji. Jest ona logiczną jednostką pracy, która przeprowadza bazę danych z jednego stanu spójnego w drugi stan spójny.

Podstawowe własności transakcji:

- atomowość (*atomicity*) – transakcja może być wykonana w całości albo wcale,
- Spójność (*consistency*) – transakcja przekształca bazę danych z jednego stanu spójnego w drugi (zapewnić musi SZBD oraz twórca aplikacji),
- niezależność (*isolation*) – transakcje są wykonywane niezależnie od siebie (są od siebie izolowane),
- trwałość (*durability*) – rezultaty zakończonej z powodzeniem transakcji są na trwałe zapisane w bazie danych.

Sytuacje konfliktowe dla dwóch transakcji:

- *odczyt – odczyt*

Obie transakcje T_1 i T_2 żądają odczytu wartości A . Każda z nich uzyskuje taką samą wartość A – konflikt nie występuje.

- *zapis – zapis (utrata aktualizacji)*

Utrata aktualizacji nastąpi wtedy, gdy transakcja T_2 dokona zmiany wartości pewnej danej przed zakończeniem transakcji T_1 , nie uwzględniając aktualizacji dokonanej przez T_1 .

- *zapis – odczyt (brudny odczyt)*

Sytuacja taka zdarza się wtedy, gdy transakcja T_2 odczytuje wartość zmienioną przez T_1 , po czym transakcja T_1 zostaje anulowana z dowolnego powodu. Transakcja T_2 odczytała zatem wartość niewłaściwą - niepotwierdzoną. Wszystkie modyfikacje dokonane przez T_1 zostały anulowane.

- *odczyt – zapis (niepowtarzalny odczyt)*

Jeżeli transakcja T_2 wykonuje kilkakrotnie odczyt wartości pewnego obiektu A , przy każdym z kolejnych odczytów powinna uzyskiwać tą samą wartość. Jeśli jednak pomiędzy kolejnymi odczytami wartości A inna transakcja dokona zmiany tej wartości, to w wyniku następujących po sobie odczytów transakcja T_2 uzyska dwie różne wartości A .

- *wiersze widmowe*

Transakcja T_1 czyta zbiór wierszy spełniających określone kryterium wyboru. Transakcja T_2 wykonuje operacje INSERT lub UPDATE i zatwierdza zmiany. Jeśli transakcja T_1 powtórzy operację czytania, uzyska inny zbiór wierszy.

Transakcje mogą kończyć się:

- zatwierdzeniem zmian – COMMIT,
- odwołaniem zmian – ROLLBACK.

W SQL zdefiniowane są 4 klauzule definiujące poziom niezależności odnoszący się do problemu sytuacji konfliktowych.

Poziom niezależności	Brudny odczyt	Niepowtarzalny odczyt	Wiersze widmowe
READ UNCOMMITTED	T	T	T
READ COMMITTED	N	T	T
REPEATABLE READ	N	N	T
SERIALIZABLE	N	N	N

W środowisku Sybase poziom izolacji podany jest jako wartość numeryczna i ma następujące znaczenie:

Poziom niezależności	Brudny odczyt	Niepowtarzalny odczyt	Wiersze widmowe
0	T	T	T
1	N	T	T
2	N	N	T
3	N	N	N

W wielu środowiskach bazodanowych są dostępne procedury, które składają się ze zdań języka SQL i są zapisane w bazie danych. Procedury mogą być wykorzystywane przez aplikacje i użytkowników, którzy posiadają odpowiednie uprawnienia.

Procedury można podzielić na:

- procedury pamiętane,
- procedury zdarzeń (*triggery*).