COLLEGIUM
DA VINCI

Bazy danych

Spis treści

Info

Prowadzący: Krzysztof Hankiewicz

Mail: krzysztof.hankiewicz@cdv.pl

Literatura

- Beynon-Davies P., Systemy baz danych, WNT, Warszawa 2003
- Connolly T., Begg C., Systemy baz danych. Praktyczne metody projektowania implementacji i zarządzania, Wydawnictwo RM 2004 (głównie na tym opierane prezentacje)
- Stones R., Matthew N., Bazy danych i PostgreSQL. Od podstaw, Helion 2002

1. Informacje wstępne

W uproszczeniu możemy założyć, że **baza danych to zbiór powiązanych danych, a system zarządzania bazą danych (DBMS) to oprogramowanie, które zarządza i kontroluje dostęp do bazy danych**.

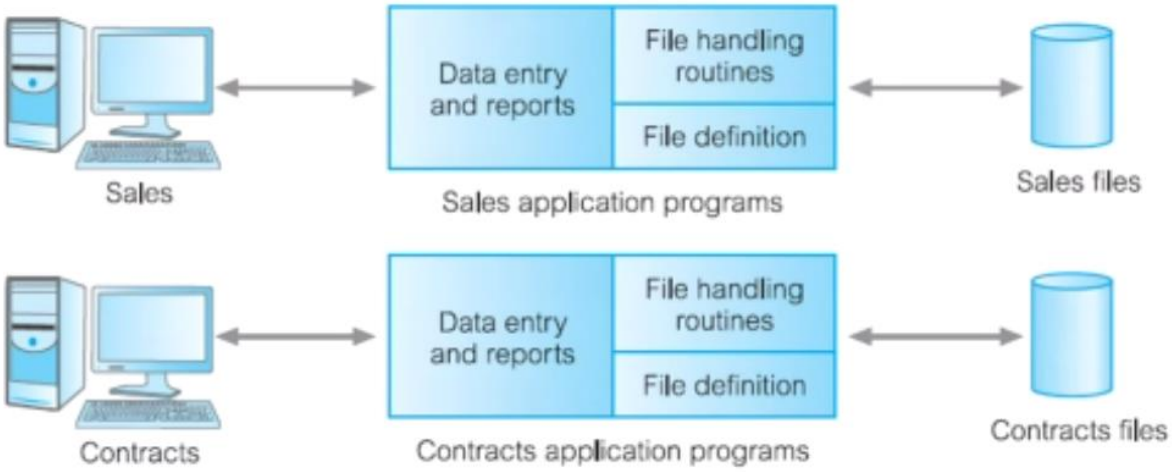
Natomiast **system bazy danych to zbiór programów użytkowych, które współdziałają z bazą danych wraz z DBMS i samą bazą danych**.

Powszechność stosowania baz danych

- Zakupy w supermarkecie oraz serwisach internetowych
- Płacenie w formie elektronicznej
- Rezerwacja biletów, wakacji, noclegów itp.
- Korzystanie z biblioteki, wypożyczanie pojazdów, wynajem lokali
- Ubezpieczenia, usługi medyczne
- Zatrudnienie, kształcenie (szkoła, studia, szkolenia)

System oparty na plikach

Zbiór programów użytkowych, które wykonują usługi dla użytkowników końcowych, takie jak tworzenie raportów. Każdy program definiuje i zarządza własnymi danymi.



Kiedy musimy coś znaleźć, przechodzimy do plików i przeszukujemy system, zaczynając od pierwszego wpisu, aż znajdziemy to, czego szukamy. Alternatywnie możemy mieć system indeksowania, który pomaga szybciej zlokalizować to, czego szukamy.

Ręczny system archiwizacji działa dobrze, o ile liczba przedmiotów do przechowywania jest niewielka. Działa to nawet w przypadku dużej liczby przedmiotów, a my musimy je tylko przechowywać i odzyskiwać. Jednak ręczny system archiwizacji załamuje się, gdy musimy odwoływać się lub przetwarzać jednocześnie informacje zawarte w różnych dokumentach.

Ograniczenia systemów plikowych:

- Separacja i izolacja danych
- Powielanie danych
- Zależność danych
- Niekompatybilne formaty plików
- Definiowanie zapytań
- Nadmierny wzrost ilości stosowanych programów użytkowych

Wprowadzenie baz danych i systemu zarządzania bazami danych (DBMS)

DBMS – Database Management System

Wszystkie wspomniane wcześniej ograniczenia podejścia opartego na plikach można przypisać dwóm czynnikom:

1. Definicja danych jest osadzona w programach użytkowych, a nie przechowywana oddzielnie i niezależnie
2. Nie ma kontroli nad dostępem i manipulacją danymi poza kontrolą narzuconą przez programy użytkowe

Aby stać się bardziej efektywnym, potrzebne było nowe podejście. Powstała baza danych i system zarządzania bazami danych (DBMS)

Definicja baz danych

Współdzielony zbiór logicznie powiązanych danych i ich opisu, zaprojektowany w celu zaspokojenia potrzeb informacyjnych organizacji.

Baza danych to jedno, możliwie duże repozytorium danych, z którego może korzystać jednocześnie wiele działów i użytkowników. Zamiast odłączonych plików z nadmiarowymi danymi, wszystkie elementy danych są zintegrowane z minimalną ilością duplikatów. Baza danych nie jest już własnością jednego działu, ale wspólnym zasobem korporacyjnym.

W bazie znajdują się nie tylko dane operacyjne organizacji, ale także opis tych danych. Z tego powodu baza danych jest również definiowana jako samoopisujący się zbiór zintegrowanych rekordów.

System zarządzania bazą danych

SZBD – System Zarządzania Bazą Danych

System oprogramowania, który umożliwia użytkownikom definiowanie, tworzenie, utrzymywanie i kontrolowanie dostępu do bazy danych.

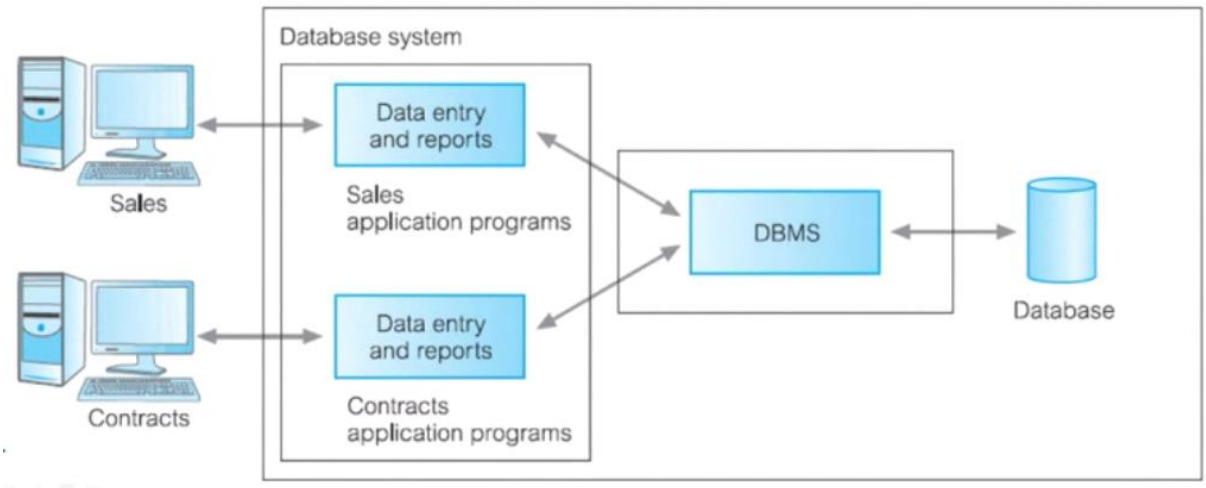
DBMS to oprogramowanie, które współdziała z programami użytkowymi użytkowników i bazą danych.

DBMS zapewnia następujące udogodnienia

- Umożliwia użytkownikom zdefiniowanie bazy danych
- Umożliwia użytkownikom wstawianie, aktualizowanie, usuwanie i pobieranie danych z bd
- Zapewnia kontrolowany dostęp do bazy danych przez:
    - System bezpieczeństwa
    - System integralności
    - System kontroli współbieżności
    - System kontroli odtwarzania
    - Katalog dostępny dla użytkownika

Programy aplikacji bazodanowych

Program komputerowy, który wchodzi w interakcję z bazą danych, wysyłając do DBMS odpowiednie żądanie (zazwyczaj wyrażenie SQL)



Widok (perspektywa)

W bazie danych widok jest zbiorem wyników przechowywanego zapytania dotyczącego danych, do którego użytkownicy bazy danych mogą wysyłać zapytania tak samo, jak w przypadku trwałego obiektu kolekcji bazy danych. To wstępnie ustalone polecenie zapytania jest przechowywane w słowniku bazy danych. W przeciwieństwie do zwykłych tabel bazodanowych w relacyjnej bazie danych widok nie stanowi części schematu fizycznego: jako zestaw wyników jest to wirtualna tabela obliczana lub porównywana dynamicznie na podstawie danych w bazie danych, gdy zażądano dostępu do tego widoku.

Oprócz zmniejszenia złożoności poprzez umożliwienie użytkownikom zobaczenia danych w taki sposób, w jaki chcą je widzieć, widoki mają kilka innych zalet:

- Widoki zapewniają poziom bezpieczeństwa
- Widoki zapewniają mechanizm dostosowywania wyglądu bazy danych
- Widok może przedstawiać spójny, niezmienny obraz struktury bazy danych, nawet jeśli podstawowa baza danych została zmieniona

Komponenty środowiska DBMS

Możemy zidentyfikować 5 głównych komponentów środowiska DBMS:

- Sprzęt
- Oprogramowanie
- Dane
- Procedury
- Ludzi

2. Relacyjny model baz danych

Model relacyjny

Model oparty jest na gałęziach matematyki zwanych teorią zbiorów i teorią predykatów.

Zasadą na której opiera się model relacyjny jest to, że typowa baza danych składa się z szeregu nieuporządkowanych tablic (lub relacji), którymi można manipulować używając nieproceduralnych operacji zwracających całe tablice.

Model relacyjny można zastosować zarówno do baz danych jak i do samych systemów zarządzania bazami danych.

Powstanie Modelu Relacyjnego

Za **twórcę** modelu relacyjnego uznaje się **Edgar**a **Frank**a **Codd**'a z laboratoriów **IBM**.

Kluczowe jego dzieło z tej dziedziny:

„*A Relational Model of Data for Large Shared Data Banks*" zostało wydane w **1970** roku. **Opisywało ono stosowane do dziś algorytmy zarządzania relacyjnymi bazami danych**.

Projektując bazę danych należy wybrać najlepszy sposób na odwzorowanie danego systemu, pochodzącego ze świata rzeczywistego, na abstrakcyjny model ujęty w bazie danych. Dotyczy to określenia, jakie tablice należy utworzyć, jakie kolumny mają one zawierać, a także, jakie związki zachodzą pomiędzy poszczególnymi tablicami.

Etapy tworzenia relacyjnej bazy danych:

- **Określenie zbiorów niezbędnych danych** – utworzenie listy danych z opisem do czego będą służyć (najlepiej w formie tabelarycznej)
- **Określenie zbiorów pól w tabelach** – zdecydowanie, jakie tabele będą przechowywały poszczególne dane (dekompozycja), ustalenie typu danych i rozmiaru pól
- **Określenie kluczy podstawowych** – jednoznaczna identyfikacja rekordów
- **Utworzenie diagramów i relacji**
- **Normalizacja danych**

Korzyści płynące z zaprojektowania bazy danych zgodnie z modelem relacyjnym

- Wydajne wprowadzanie, aktualizacja i usuwanie danych
- Sprawny odczyt danych, ich obróbka i generowanie raportów
- Przewidywalne zachowanie bazy danych, ponieważ podlega ona dobrze zdefiniowanemu modelowi
- Ponieważ większość informacji przechowywanych jest w bazie zamiast w samej aplikacji, taka baza danych jest w pewnym sensie samodokumentująca
- Łatwo dokonywać zmian w strukturze bazy

Tablice

Tablice w modelu relacyjnym stosowane są do reprezentacji elementów świata rzeczywistego. Każda tablica powinna reprezentować tylko jeden taki element. Elementami tymi mogą być rzeczywistymi obiektami bądź też zdarzeniami. Na przykład rzeczywistym obiektem może być klient, przedmiot z inwentarza lub faktura. Przykładami zdarzeń mogą być wizyty pacjentów, zlecenia, czy rozmowy telefoniczne.

Tablice zbudowane są z wierszy i kolumn.

Kolumna – najmniejsza część informacji o obiekcie, np. kolumna ‘Nazwa’ będzie zawierała nazwę klienta, a kolumna ‘Adres’ jego dane adresowe – np. nazwę ulicy.

Kolumna może mieć zdefiniowane pewne ograniczenia, np. kolumna nie może zostać pusta lub musi posiadać unikatową zawartość.

Wiersz / rekord / krotka - zbiór kolumn, pól z danymi o obiekcie, np. wiersz w tabeli 'Klienci' będzie zawierał dane określonego klienta (nazwę, adres, telefon, faks, e-mail itd.)

Inaczej mówiąc wiersz to ciąg wartości atrybutów dla jednego obiektu. Można dowolnie zmieniać kolejność wierszy.

Unikatowość

Model relacyjny nakazuje, żeby każdy wiersz w tablicy był unikatowy. Jeśli pozwoli się na powtarzające się wiersze, wtedy dla programu bazy danych nie będzie istniał żaden sposób na jednoznaczne określenie pozycji danego wiersza. To stwarza wszelkiego rodzaju niejednoznaczne sytuacje i problemy, których najlepiej unikać. **Unikatowość w tablicy zapewnia się wyznaczając klucz główny**.

Klucze

Klucze główne

Klucz główny – kolumna lub zbiór kolumn, które zawierają unikatowe wartości dla każdego wiersza w tablicy. Każda tablica może mieć tylko 1 klucz główny, nawet wtedy, gdy kilka kolumn lub ich kombinacje także zawierają unikatowe wartości.

Wśród kluczy głównych wyróżnia się:

- Klucze proste – opierają się na jednej kolumnie
- Klucze złożone – opierają się na wielu polach

Klucze potencjalne

Wszystkie kolumny (lub kombinacje kolumn) zawierające unikalne wartości nazywa się kluczami potencjalnymi. Spośród nich wybiera się **jeden klucz główny,** a **pozostałe** klucze nazywa się **kluczami alternatywnymi**. Klucze te mogą być proste lub złożone.

Kryteria wyboru klucza

Decyzja, który z kluczy potencjalnych ma być kluczem głównym, należy do projektanta. Nie ma ogólnej reguły na to, który klucz byłby najlepszy.

Mówi się, że decyzja powinna opierać się na zasadzie:

- **Minimalności** – klucz powinien zawierać najmniejszą niezbędną ilość kolumn
- **Stabilności** – klucz powinien rzadko ulegać zmianom
- **Prostoty/intuicyjności** – klucz powinien być zarówno prosty jak i łatwy do zrozumienia przez użytkowników

Zalecenia przy wyborze kluczy

Przeszukiwanie i sortowanie kolumn numerycznych jest bardziej wydajne niż kolumn tekstowych. Dlatego wielu projektantów woli numeryczne klucze główne.

Dobrymi kluczami głównymi są kolumny indeksowe, zwłaszcza kiedy mamy problemy z dobrymi kluczami potencjalnymi, a tablica nie zawiera jeszcze żadnej kolumny z nadrzędnym numerem identyfikacyjnym.

Kolumn indeksowych nie należy stosować, jeśli czasem zachodzić będzie potrzeba przenumerowania wartości lub jeśli niezbędny jest kod alfanumeryczny.

Dziedziny i klucze obce

Klucz obcy to taka kolumna, która zawiera odnośniki do klucza głównego z innej tablicy.

Ważne jest, żeby klucz główny i klucz obcy miały to samo znaczenie i posiadały tę samą dziedzinę. Dziedzina, to po prostu zbiór, z którego kolumna może czerpać swoje wartości. **Dziedziny** można traktować jako zdefiniowane przez użytkownika **typy kolumn**, nakładające na kolumnę reguły, do których muszą się stosować wartości zawarte w kolumnie, a także definiujące operacje, które można na kolumnach wykonywać.

(15)

(16)

Powiązania

Klucze obce definiuje się w bazie danych, aby odzwierciedlały powiązania w świecie rzeczywistym. Związki między jednostkami w świecie rzeczywistym mogą być bardzo złożone, zachodząc na różne sposoby pomiędzy licznymi obiektami. Na przykład rodzina posiada wielokrotne powiązania między wieloma osobami – wszystkie związki zachodzą jednocześnie. Jednakże w bazie relacyjnej, takiej jak Microsoft Access, rozpatruje się tylko związki w parach tablic.

Tablice mogą być powiązane na 1 z 3 sposobów:

- 1 do 1
- 1 do N
- N do N

Związki 1 do 1

Dwie tablice powiązane są w stosunku 1 do 1, jeśli dla każdego wiersza z pierwszej tablicy istnieje co najwyżej jeden związany z nim wiersz w drugiej tablicy. Prawdziwe związki 1-1 spotyka się rzadko w świecie rzeczywistym. Ten typ związków tworzony jest często sztucznie, aby obejść ograniczenia oprogramowania zarządzającego bazą danych. W bazie Microsoft Access związek 1-1 może być konieczny, kiedy potrzebne jest rozbicie tablicy na kilka innych tablic, np. ze względu na bezpieczeństwo lub wydajność. Tablice powiązane związkiem 1-1 powinny mieć zawsze ten sam klucz główny, służący jako kolumna łącząca.

Związki 1 do N

Dwie tablice są powiązane w stosunku 1 do N jeśli każdemu wierszowi z pierwszej tablicy odpowiada zero, jeden lub wiele wierszy z drugiej tablicy, ale każdemu wierszowi z drugiej tablicy odpowiada dokładnie jeden wiersz z tablicy pierwszej. Na przykład każde zamówienie dla firmy może wyszczególniać wiele różnych produktów. Związek 1-N często nazywa się związkiem rodzic-dziecko. Jest to najczęściej spotykany typ związków.

Związki N do N

Dwie tablice są powiązane w stosunku N do N jeśli każdemu wierszowi z pierwszej tablicy odpowiada wiele wierszy w drugiej tablicy, a każdemu wierszowi w drugiej tablicy odpowiada wiele wierszy w pierwszej tablicy. Związków N do N nie da się bezpośrednio modelować w programach zarządzających relacyjnymi bazami danych, także w Microsoft Access. Tego typu związki muszą być najpierw podzielone na wiele związków 1 do N. Na przykład, pacjent może podlegać wielu programom

ubezpieczeniowym, a dana firma ubezpieczeniowa (program ubezpieczeniowy) może ubezpieczać wielu pacjentów.

Ogólne zasady integralności

Model relacyjny podaje 2 ogólne reguły integralności danych. Nazywa się je ogólnymi, ponieważ stosuje się je do wszystkich baz danych.

Są to:

- **Integralność jednostek**
- **Integralność odniesień**

Integralność jednostek

Zasada integralności jednostek mówi, że **klucze główne nie mogą zawierać pustych pól**. Uzasadnienie tej zasady powinno być oczywiste: nie da się jednoznacznie zidentyfikować rzędu w tablicy, jeśli klucz główny może być pusty. Należy pamiętać, że ta zasada stosuje się zarówno do kluczy prostych jak i złożonych. Dla kluczy złożonych, żadna z indywidualnych kolumn należących do klucza nie może zawierać pustego pola. Na szczęście Access automatycznie wymusza integralność jednostek. Żaden składnik klucza głównego w Accessie nie może być pusty.

Integralność odniesień

Zasada integralności odniesień mówi, że **klucz obcy nie może odnosić się do nieistniejących rekordów**. Wynika z tego, że:

- Do tablicy zawierającej klucz obcy nie można dołączyć wiersza, który używa takiej wartości klucza, która nie ma odpowiednika w tablicy odniesień
- Jeśli wartość klucza głównego w tablicy, do której odnosi się klucz obcy (czyli w tablicy odniesień), ulegnie zmianie (lub cały wiersz zostanie usunięty), to w tablicy stosującej ten klucz nie mogą powstać wiersze „osierocone".

Na ogół, kiedy klucz główny, do którego się odwołujemy, ulega zmianie, mamy 3 możliwości podejścia do problemu:

- Zakazać dokonywania zmian w tablicy odniesień
- Propagować dokonane zmiany do wszystkich tablic zawierających referencje do zmienionych wierszy. W przypadku usunięcia wierszy, wszystkie zależne wiersze w innych tablicach również są usuwane.
- Wyzerować wszystkie wartości kluczy obcych odnoszących się do usuniętych wierszy.

Normalizacja bazy danych

Normalizacja bazy danych jest zasadniczą częścią analizy bazy danych.

Jest to proces mający na celu eliminację powtarzających się danych w relacyjnej bazie danych.

Działania takie zwiększają bezpieczeństwo danych i zmniejszają ryzyko powstania niespójności.

Sposoby ustalenia czy dany schemat bazy danych jest „znormalizowany", a jeżeli jest to jak bardzo - nazywane są postaciami normalnymi (ang. normal forms)

Wyróżnia się 5 poziomów normalizacji:

- Pierwsza postać normalna – podział na kolumny
- Druga postać normalna – jednoznaczny identyfikator
- Trzecie postać normalna – pozbycie się redundancji
- Czwarta postać normalna
- Piąta postać normalna

Pierwsza postać normalna

Zakłada eliminację powtarzających się grup i nie łączenie kilku informacji w pojedynczych polach.

Operacja ta wiąże się z rozdzieleniem informacji umieszczonych w jednym rekordzie na 2 lub kilka rekordów.

(np. adres -> ulica, numer budynku, numer mieszkania)

Pierwsza postać normalna to warunek, że wszystkie wartości kolumn muszą być elementarne, czyli niepodzielne.

Dla każdej pozycji wiersz-kolumn w tablicy powinna istnieć tylko jedna wartość, a nie tablica lub lista wartości. Korzyści płynące z tej reguły powinny być oczywiste. Jeśli w kolumnie przechowuje się całe listy wartości, wtedy trudno jest nimi operować. Odczytywanie informacji staje się bardziej pracochłonne i trudne w automatyzacji. Uzyskanie informacji z takiej tablicy byłoby niezmiernie kłopotliwe.

Pierwsza postać normalna zabrania także istnienia powtarzających się grup, nawet jeśli miałyby być one złożone z kolumn elementarnych.

Druga postać normalna

Spełnia warunki pierwszej postaci. Stosowanie identyfikatorów w miejsce tytułów i opisów.

Np. stosowanie identyfikatora książki zamiast jej tytułu w tabeli zawierającej zamówienia.

Tablica jest w drugiej postaci normalnej, jeśli jest w pierwszej postaci normalnej i każda kolumna nie należąca do żadnego klucza potencjalnego jest całkowicie zależna od (całego) klucza głównego. Innymi słowy, tablice powinny przechowywać dane dotyczące tylko jednej jednostki (obiektu, zdarzenia) oraz ta jednostka powinna być opisywalna przez jej klucz główny.

Trzecia postać normalna

Spełnia warunki pierwszej i drugiej postaci normalnej. Nie powinna przechowywać danych, które można uzyskać z innych tabel. Jeśli pewna wartość może być wyliczona, nie należy jej przechowywać.

Trzecia postać normalna jest zupełnie wystarczająca do tworzenia funkcjonalnej relacyjnej bazy danych.

Tablica jest w trzeciej postaci normalnej jeśli jest w drugiej postaci normalnej i wszystkie kolumny nie należące do żadnego klucza potencjalnego są wzajemnie niezależne.

Oczywistym przypadkiem zależności jest kolumna wyliczona.

Na przykład, jeśli tablica zawiera kolumny ilosc i koszt_elementu, można obliczyć sumę całkowitą jako (ilosc * koszt_elementu) i przechowywać ją w kolumnie koszt_calkowity należącej do tej samej tablicy. Tylko, że taka tablica nie byłaby w trzeciej postaci normalnej. Lepiej takiej kolumny nie wprowadzać, a obliczenia wykonywać w kwerendzie / na formularzu / na gotowym raporcie. Oszczędza to miejsce w bd i zapobiega konieczności przeliczania wartości koszt_calkowity za każdym razem, gdy ilosc lub koszt_elementu ulegną zmianie.

Czwarta postać normalna

Spełnia warunki poprzednich postaci normalnych. Zakłada izolację niezależnych, złożonych relacji. Żadna tabela nie może należeć do dwóch lub więcej relacji 1:N albo N:N, jeżeli relacje te nie są ze sobą powiązane.

Piąta postać normalna

Spełnia warunki poprzednich postaci normalnych. Zakłada izolację znaczeniowo powiązanych, złożonych relacji.

Izolacja ta polega zwykle na wprowadzeniu tabeli pośredniczącej łączącej informacje z dwóch tabel.

3. Podstawy SQL

SQL – Structured Query Language – to specjalny język programowania przeznaczony do zarządzania danymi przechowywanymi w relacyjnym systemie zarządzania bazami danych (RDBMS) lub do przetwarzania strumieniowego w relacyjnym systemie zarządzania strumieniem danych (RDSMS).

Dr E.F. Codd opublikował artykuł „A Relational Model of Data for Large Shared Data Banks" w czerwcu 1970 r. w czasopiśmie Association of Computer Machinery (ACM), Communications of the ACM. Model Codda jest obecnie akceptowany jako ostateczny model dla relacyjnych systemów zarządzania bazami danych.

Język, Structured English Query Language (SEQUEL), został opracowany przez IBM Corporation, w celu wykorzystania modelu Codda. SEQUEL później stał się SQL (wciąż wymawiany jako „sequel"). W 1979 roku firma Relational Software (obecnie Oracle) wprowadziła pierwszą komercyjnie dostępną implementację SQL. Obecnie SQL jest akceptowany jako standardowy język RDBMS.

Terminologia

Standard ISO SQL nie używa formalnych terminów: ~~relacji, atrybutów i krotek~~, zamiast tego używa: **tabel, kolumny i wierszy**. SQL nie trzyma się ściśle definicji modelu relacyjnego.

Komendy SQL

Większość składników instrukcji SQL **NIE uwzględnia wielkości liter**, co oznacza, że litery mogą być wpisywane zarówno dużymi, jak i małymi literami.

Jedynym ważnym **wyjątkiem** od tej reguły jest to, że dane znakowe muszą być wpisywane dokładnie tak, jak pojawiają się w bazie danych.

Chociaż komendy SQL można formatować swobodnie, instrukcje SQL są bardziej czytelne, jeśli używane są wcięcia i wiersze. Na przykład:

- Każda klauzula w poleceniu powinna zaczynać się w nowej linii;
- Początek każdej klauzuli powinien pokrywać się z początkiem innych klauzul;
- Jeśli klauzula składa się z kilku części, każda z nich powinna znajdować się w osobnym wierszu i być wcięta pod początkiem klauzuli, aby pokazać zależność.

Notacja Backusa-Naura (BNF)

- Wielkie litery są używane do reprezentowanie słów zastrzeżonych i muszą być zapisane dokładnie tak, jak pokazano;
- Małe litery są używane do reprezentowania słów zdefiniowanych przez użytkownika;
- Pionowa kreska (|) wskazuje wybór spośród alternatyw; np.: a | b | c;
- Nawiasy klamrowe wskazują wymagany element; np.: {a};
- Nawiasy kwadratowe wskazują element opcjonalny; np.: [a];
- Wielokropek (...) jest używany do wskazania opcjonalnego powtórzenia elementu zero lub więcej razy

Instrukcje DML i DDL

W praktyce instrukcje DDL są używane do tworzenia struktury bazy danych (czyli tabel) i mechanizmów dostępu (czyli tego, do czego każdy użytkownik ma legalny dostęp), a następnie instrukcje DML są używane do działań związanych z danymi.

DML – Data Manipulation Language

- SELECT – wyszukiwanie danych w bazie danych
- INSERT – wstawianie danych do tabeli
- UPDATE – aktualizowanie danych w tabeli
- DELETE – usuwanie danych w tabeli

DDL – Data Definition Language

W SQL DDL umożliwia tworzenie i usuwanie obiektów bazy danych, takich jak schematy, domeny, tabele, widoki i indeksy.

Główne instrukcje używane do tworzenia, zmiany usuwania struktur to:

- CREATE SCHEMA / DROP SCHEMA
- CREATE DOMAIN / ALTER DOMAIN / DROP DOMAIN
- CREATE TABLE / ALTER TABLE / DROP TABLE
- CREATE VIEW / DROP VIEW

I nie objęte standardem SQL:

- CREATE INDEX / DROP INDEX

Tworzenie bazy danych

Proces tworzenia bazy danych różni się znacznie w zależności od produktu.

Instrukcja definicji schematu ma następującą (uproszczoną) formę:

```
CREATE SCHEMA [Nazwa | AUTHORIZATION IdentyfikatorTworcy]
```

Dlatego, jeśli twórcą schematu jest Kowalski instrukcja SQL ma postać:

```
CREATE SCHEMA AUTHORIZATION Kowalski
```

Schemat można usunąć za pomocą instrukcji DROP SCHEMA, która ma następującą postać:

```
DROP SCHEMA Nazwa [RESTRICT | CASCADE]
```

Jeśli określono RESTRICT, co jest wartością domyślną, jeśli nie określono żadnego kwalifikatora, schemat musi być pusty lub operacja zakończy się niepowodzeniem.

Jeśli określono opcję CASCADE, operacja kaskadowo usuwa wszystkie obiekty powiązane ze schematem w kolejności zdefiniowanej powyżej.

```
CREATE TABLE nazwatabeli
     { (nazwakolumny typdanych [NOT NULL] [UNIQUE]
     [DEFAULT wartoscdomyslna] [CHECK (warunek)] [,…] }
     [PRIMARY KEY (listakolumn),]
     { [UNIQUE (listakolumn)] [,…]}
     { [FOREIGN KEY (listakluczyobcych)
     REFERENCES nazwatabelinadrzednej
          [(listakluczykandydujących)]
          [MATCH {PARTIAL | FULL}]
          [ON UPDATE dzialaniereferencyjne]
          [ON DELETE dzialaniereferencyjne] [,…]}
{ [CHECK (warunek)] [,…]}
```

Ograniczenie tabeli i opcjonalnie mogą być poprzedzone klauzulą:

```
CONSTRAINT nazwawarunku
```

Co umożliwia usunięcie ograniczenia w oparciu o podaną nazwę przy użyciu instrukcji ALTER TABLE.

Klauzula PRIMARY KEY określa kolumnę lub kolumny, które tworzą klucz podstawowy tabeli.

Klauzula FOREIGN KEY określa klucz obcy w tabeli (podrzędnej) oraz relację, jaką ma on z inną tabelą (nadrzędną).

Klauzula określa:

- listakluczyobcych – kolumna lub kolumny z tworzonej tabeli, które tworzą klucz obcy
- podpunkt REFERENCES, określający tabelę nadrzędną, czyli tabela zawierającą pasujący klucz kandydujący. Jeśli listakluczykandydujących jest pominięta, zakłada się, że klucz obcy jest zgodny z kluczem podstawowym tabeli nadrzędnej.
- Opcjonalna reguła aktualizacji (ON UPDATE) dla relacji, która określa działanie, które ma zostać podjęte, gdy klucz kandydujący zostanie zaktualizowany w tabeli nadrzędnej, która odpowiada kluczowi obcemu w tabeli podrzędnej.
  dzialaniereferencyjne może być CASCADE, SET NULL, SET DEFAULT albo NO ACTION

- Opcjonalna reguła usuwania (ON DELETE) dla relacji, która określa działanie, które ma zostać podjęte po usunięciu wiersza z tabeli nadrzędnej, która ma klucz kandydujący, który pasuje do klucza obcego w tabeli podrzędnej
- Domyślnie ograniczenie referencyjne jest spełnione, jeśli dowolny składnik klucza obcego ma wartość NULL lub w tabeli nadrzędnej znajduje się pasujący wiersz. Opcja MATCH zapewnia dodatkowe ograniczenia dotyczące wartości null w kluczu obcym.

Tworzenie tabeli PropertyForRent z wykorzystaniem wypowych elementów polecenia CREATE TABLE.

(przykład_SQL – slajdy: 18 i 19)

Zmiana definicji tabeli

Definicja instrukcji ALTER TABLE w standardzie ISO składa się z 6 opcji:

- `ADD` – Dodać nową kolumnę do tabeli
- `DROP` – Usunąć kolumnę z tabeli
- `ADD CONSTRAINT` – Dodać nowe ograniczenie tabeli
- `DROP CONSTRAINT` – Usunąć ograniczenie tabeli
- `ALTER [COLUMN] SET DEFAULT` – Ustawić wartość domyślną dla kolumny
- `ALTER [COLUMN] DROP DEFAULT` – Usunąć wartość domyślną dla kolumny

Składnia polecenia ALTER TABLE:

```
ALTER TABLE nazwatabeli
[ADD [COLUMN] nazwakolumny typdanych [NOT NULL] [UNIQUE]
[DEFAULT wartoscdomyslna] [CHECK (warunek)]]
[DROP [COLUMN] nazwakolumny [RESTRICT | CASCADE]]
[ADD [CONSTRAINT [nazwawarunku]] tabeladefinicjiwarunkow]
[DROP CONSTRAINT nazwawarunku [RESTRICT | CASCADE]]
[ALTER [COLUMN] SET DEFAULT opcjadomyslna]
[ALTER [COLUMN] DROP DEFAULT]
```

Zmiana tabeli Staff poprzez usunięcie domyślnej wartości „Asystent”

```
ALTER TABLE Staff
    ALTER position DROP DEFAULT;
```

W kolumnie position i ustawienie domyślnej płci na „F” (kobieta).

```
ALTER TABLE Staff
    ALTER sex SET DEFAULT ‘F’;
```

Zmiana tabeli PropertyForRent przez usunięcie ograniczenia, że personel nie może obsługiwać więcej niż 100 nieruchomości jednocześnie.

```
ALTER TABLE PropertyForRent
    DROP CONSTRAINT StaffNotHandlingTooMuch;
```

Zmiana tabeli Client przez dodanie nowej kolumny określającej preferowaną liczbę pokoi.

```
ALTER TABLE Client
    ADD prefNoRooms PropertyRooms;
```

Usuwanie tabeli

Możemy usunąć tabelę z bazy danych za pomocą polecenia DROP TABLE, które ma format:

```
DROP TABLE nazwatabeli [RESTRICT | CASCADE]
```

- **RESTRICT** – operacja **DROP** jest odrzucana, jeśli istnieją inne obiekty, których istnienie zależy od dalszego istnienia tabeli, która ma zostać usunięta.
- **CASCADE** – operacja **DROP** jest kontynuowana i SQL automatycznie usuwa wszystkie obiekty zależne (i obiekty zależne od tych obiektów)

Tworzenie indeksu

Indeks to struktura zapewniająca przyspieszony dostęp do wierszy tabeli na podstawie wartości jednej lub więcej kolumn. Tworzenie indeksów nie jest standardowym SQL. Jednak większość dialektów obsługuje co najmniej następujące możliwości:

```
CREATE [UNIQUE] INDEX nazwaindeksu ON nazwatabeli
        (nazwakolumny [ASC | DESC] [, …])
```

Indeksy można tworzyć tylko w tabelach podstawowych, a nie w widokach. W przypadku użycia klauzuli UNIQUE, unikatowość indeksowanej kolumny lub kombinacji kolumn będzie wymuszona przez DBMS.

Przykład

Dla tabeli Staff i PropertyForRent, możemy utworzyć przykładowo następujące indeksy:

```
CREATE UNIQUE INDEX StaffNoInd ON Staff (staffNo);

CREATE UNIQUE INDEX PropertyNoInd ON
    PropertyForRent (propertyNo);
```

Dla każdej kolumny możemy określić, że kolejność jest rosnąca (ASC) lub malejąca (DESC), przy czym ASC jest ustawieniem domyślnym.

Widoki

Widok jest dynamicznym wynikiem jednej lub więcej operacji relacyjnych działających na podstawowych relacjach w celu wytworzenia innej relacji.

Widok to wirtualna relacja, która niekoniecznie istnieje w bazie danych, ale może być tworzona na żądania konkretnego użytkownika w momencie żądania.

DBMS przechowuje definicję widoku w bazie danych. Kiedy DBMS odwołuje się do widoku, jednym z podejść jest wyszukanie tej definicji i przetłumaczenie żądania na równoważne żądania względem tabel źródłowych widoku, a następnie wykonanie równoważnego żądania.

Ten proces scalania nazywa się rozdzielczością widoku.

Alternatywne podejście, zwane materializacją widoku, przechowuje widok jako tabelę tymczasową w bazie danych.

Definiowanie widoku

Polecenie CREATE VIEW definiuje widok następująco:

```
CREATE VIEW Nazwawidoku [(newColumnName [, …])] AS subselect
    [WIDTH [CASCADED | LOCAL] CHECK OPTION]
```

Widok jest definiowany przez określenie instrukcji SQL SELECT. Opcjonalnie do każdej kolumny w widoku można przypisać nazwę. Jeśli określono listę nazw kolumn, musi ona zawierać taką samą liczbę elementów, jak liczba kolumn utworzonych przez podselekcję.

Podselekcja nazywana jest zapytaniem definiującym.

Jeśli określono WITH CHECK OPTION, SQL zapewnia, że jeśli wiersz nie spełni klauzuli WHERE zapytania definiującego widoku, nie zostanie on dodany do podstawowej tabeli widoku.

Widok poziomy – przykład

Utwórz widok, aby manager w oddziale B003 mógł zobaczyć tylko szczegóły dotyczące pracowników pracujących w jego oddziale.

Widok poziomy ogranicza dostęp użytkownika do wybranych wierszy jednej lub więcej tabel.

```
CREATE VIEW ManagerStaff
AS SELECT *
FROM Staff
WHERE branchNo = `B003`;
```

Jeżeli wykonamy polecenie wyświetlające całą zawartość widoku:

```
SELECT * FROM ManagerStaff;
```

Możemy uzyskać następujący wynik:

staffNo	fName	lName	position	sex	DOB	salary	branchNo
SG37	Ann	Beech	Assistant	F	10-Nov-60	12000.00	B003
SG14	David	Ford	Supervisor	M	24-Mar-58	18000.00	B003
SG5	Susan	Brand	Manager	F	3-Jun-40	24000.00	B003

Utwórz widok szczegółów personelu w oddziale B003, który wyklucza informacje o wynagrodzeniach, tak aby tylko menedżerowie mieli dostęp do szczegółów wynagrodzenia personelu, który pracuje w ich oddziale.

Widok pionowy ogranicza dostęp użytkownika do wybranych kolumn jednej lub kilku tabel.

```
CREATE VIEW Staff3
AS SELECT staffNo, fName, lName, position, sex
FROM Staff
WHERE branchNo = `B003`;
```

Możemy zmienić polecenie tak, aby używać widoku ManagerStaff zamiast tabeli Staff:

```
CREATE VIEW Staff3
AS SELECT staffNo, fName, lName, position, sex
FROM ManagerStaff;
```

staffNo	fName	lName	position	sex
SG37	Ann	Beech	Assistant	F
SG14	David	Ford	Supervisor	M
SG5	Susan	Brand	Manager	F

Grupowanie i łączenie w widokach

Utwórz widok pracowników zarządzających nieruchomościami do wynajęcia, w tym numer oddziału, w którym pracują, liczbę pracowników i liczbę zarządzanych nieruchomości.

```
CREATE VIEW StaffPropCnt (branchNo, staffNo, cnt)
AS SELECT s.branchNo, s.staffNo, COUNT(*)
FROM Staff s, PropertyForRent p
WHERE s.staffNo = p.staffNo
GROUP BY s.branchNo, s.staffNo;
```

Daje to dane przedstawione w tabeli:

branchNo	staffNo	cnt
B003	SG14	1
B003	SG37	2
B005	SL41	1
B007	SA9	1

Usuwanie widoków

Widok jest usuwany z bazy danych za pomocą instrukcji DROP VIEW:

```
DROP VIEW nazwawidoku [RESTRICT | CASCADE]
```

Na przykład możemy usunąć widok ManagerStaff za pomocą instrukcji:

```
DROP VIEW ManagerStaff;
```

Obsługiwanie widoków

Jak dokładnie obsługiwane jest zapytanie w widoku?

```
CREATE VIEW StaffPropCnt (branchNo, staffNo, cnt)
AS SELECT s.branchNo, s.staffNo, COUNT(*)
FROM Staff s, PropertyForRent p
WHERE s.staffNo = p.staffNo
GROUP BY s.branchNo, s.staffNo;

SELECT staffNo, cnt
FROM StaffPropCnt
WHERE branchNo = 'B003'
ORDER BY staffNo;
```

1. Nazwy kolumn widoku na liście SELECT są tłumaczone na odpowiadające im nazwy kolumn w zapytaniu definiującym. To daje:
   ```
   SELECT s.staffNo AS staffNo, COUNT(*) AS cnt;
   ```
2. Nazwy widoków w klauzuli FROM są zastępowane odpowiednimi nazwami tabel zapytania definiującego:
   ```
   FROM Staff s, PropertyForRent p
   ```
3. (41)
4. Klauzule GROUP BY i HAVING są kopiowane z zapytania definiującego. W tym przykładzie mamy tylko klauzulę GROP BY:
   ```
   GROUP BY s.branchNo, s.staffNo
   ```

(43, 44)

Aktualizowanie widoków

Wszystkie aktualizacje tabeli podstawowej są natychmiast odzwierciedlane we wszystkich widokach obejmujących tę tabelę podstawową. Podobnie możemy spodziewać się, że jeśli widok zostanie zaktualizowany, tabele bazowe będą odzwierciedlać tę zmianę.

Nie zawsze jest to jednak możliwe.

Definicja podana w normie ISO mówi, że widok można aktualizować wtedy i tylko wtedy, gdy:

- DISTINCT nie jest określony
- Każdy element na liście SELECT zapytania definiującego jest nazwą kolumny i żadna nazwa kolumny nie pojawia się więcej niż raz
- Klauzula FROM określa tylko jedną tabelę
- Klauzula WHERE nie zawiera żadnych zagnieżdżonych poleceń SELECT, które odwołują się do tabeli w klauzuli FROM
- W zapytaniu definiującym nie ma klauzuli GROUP BY ani HAVING

Zalety widoków

W przypadku DBMS działającego na samodzielnym komputerze, widoki są zwykle wygodną, zdefiniowaną w celu uproszczenia zapytań do bazy danych. Jednak w SZBD dla wielu użytkowników widoki odgrywają kluczową rolę w definiowaniu struktury bazy danych i egzekwowaniu bezpieczeństwa. Główne zalety widoków to:

- Niezależność danych
- Obieg
- Ulepszone bezpieczeństwo
- Zmniejszona złożoność
- Wygoda
- Dostosowywanie
- Integralność danych

Wady widoków

Chociaż widoki zapewniają wiele istotnych korzyści, istnieją również pewne wady widoków SQL. Główne wady widoków to:

- Ograniczona możliwość aktualizacji
- Ograniczenie do bieżącej struktury
- Wydajność

(?49)

5. Relacyjna struktura danych

Relacja – tabela z kolumnami i wierszami

RDMS wymaga jedynie, aby baza danych była postrzegana przez użytkownika jako tabele.

Atrybut – nazwana kolumna relacji

W modelu relacyjnym relacje służą do przechowywania informacji o obiektach, które mają być reprezentowane w bazie danych. Relacja jest reprezentowana jako dwuwymiarowa tabela, w której wiersze tabeli odpowiadają poszczególnym rekordom, a kolumny tabeli odpowiadają atrybutom.

('1-3', '1-4', '1-5')

Tabela przedstawia domeny dla niektórych atrybutów relacji Branch i Staff.

Attribute	Domain Name	Meaning	Domain Definition
branchNo	BranchNumbers	The set of all possible branch numbers	character: size 4, range B001–B999
street	StreetNames	The set of all street names in Britain	character: size 25
city	CityNames	The set of all city names in Britain	character: size 15
postcode	Postcodes	The set of all postcodes in Britain	character: size 8
sex	Sex	The sex of a person	character: size 1, value M or F
DOB	DatesOfBirth	Possible values of staff birth dates	date, range from 1-Jan-20, format dd-mmm-yy
salary	Salaries	Possible values of staff salaries	monetary: 7 digits, range 6000.00–40000.00

('1-7', '1-8', '1-9')

Relacyjna baza danych – zbiór znormalizowanych relacji z odrębnymi nazwami relacji

Relacyjna baza danych składa się z relacji o odpowiedniej strukturze – powstałej w wyniku normalizacji.

Alternatywna terminologia do modelu relacyjnego

Określenia formalne	Określenia standardu SQL	Inne
relacja	tabela	plik
krotka	wiersz	rekord
atrybut	kolumna	pole

(?'1-12')

6. SQL – Data Manipulation Language

SQL - Structured Query Language

DML - Data Manipulation Language

(2-4)

SELECT

Ogólna składnia:

```
SELECT [DISTINCT | ALL] {* | nazwa_kolumny lub wyrazenie}
[AS nowa_nazwa]] [,…]}
FROM nazwa_tabeli_i_lub_widoku [alias] [,…]
[WHERE warunek]
[GROUP BY lista_kolumn] [HAVING warunek]
[ORDER BY lista_kolumn]
```

Składniki przetwarzane w instrukcji SELECT to:

FROM – określa tabelę lub tabele, które mają być używane

WHERE – filtruje wiersze w oparciu o podane warunki

GROUP BY – tworzy grupy wierszy o tej samej wartości kolumny

HAVING – filtruje grupy w oparciu o podane warunki

SELECT – określa, które kolumny mają pojawić się na wyjściu

ORDER BY – określa kolejność wyjścia

Przykład:

Tabela Staff

staffNo	fName	lName	position	sex	DOB	salary	branchNo
SL21	John	White	Manager	M	1-Oct-45	30000.00	B005
SG37	Ann	Beech	Assistant	F	10-Nov-60	12000.00	B003
SG14	David	Ford	Supervisor	M	24-Mar-58	18000.00	B003
SA9	Mary	Howe	Assistant	F	19-Feb-70	9000.00	B007
SG5	Susan	Brand	Manager	F	3-Jun-40	24000.00	B003
SL41	Julie	Lee	Assistant	F	13-Jun-65	9000.00	B005

Zad: Pobierz wszystkie kolumny oraz wszystkie wiersze

Ponieważ w zapytaniu nie ma żadnych ograniczeń, klauzula WHERE jest zbędna i wymagane są wszystkie kolumny:

```
SELECT staffNo, fName, lName, position, sex, DOB, salary,
branchNo

FROM Staff;
```

Ponieważ wiele wyszukiwań SQL wymaga wszystkich kolumn tabeli, istnieje szybki sposób wyrażenia „wszystkich kolumn" w SQL za pomocą gwiazdki (*) zamiast nazw kolumn.

Poniższa instrukcja jest równoważnym i krótszym sposobem wyrażenia tego zapytania:

```
SELECT * FROM Staff;
```

Zad: Pobierz określone kolumny, wszystkie wiersze

Przygotuj listę wynagrodzeń dla wszystkich pracowników, zawierającą tylko numer pracownika, imię i nazwisko oraz pensję:

```
SELECT staffNo, fName, lName, salary FROM Staff;
```

Tabela wynikowa:

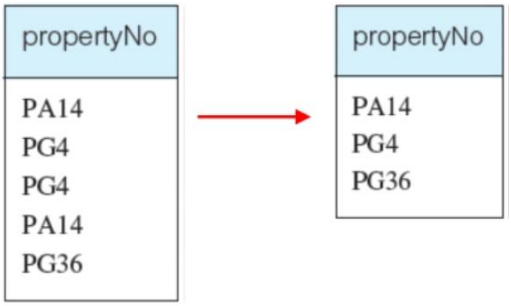
staffNo	fName	lName	salary
SL21	John	White	30000.00
SG37	Ann	Beech	12000.00
SG14	David	Ford	18000.00
SA9	Mary	Howe	9000.00
SG5	Susan	Brand	24000.00
SL41	Julie	Lee	9000.00

Użycie DISTINCT

SELECT nie eliminuje duplikatów, gdy pobiera jedną lub więcej kolumn. Aby wyeliminować duplikaty, używamy słowa kluczowego DISTINCT:

```
SELECT DISTINCT propertyNo FROM Viewing;
```

SELECT → SELECT DISTINCT



propertyNo
PA14
PG4
PG4
PA14
PG36

propertyNo
PA14
PG4
PG36

Pola obliczeniowe

Sporządź listę miesięcznych wynagrodzeń dla wszystkich pracowników, zawierającą numer pracownika, imię i nazwisko oraz wynagrodzenie:

```
SELECT staffNo, fName, lName, salary/12 FROM Staff;
```

staffNo	fName	lName	col4
SL21	John	White	2500.00
SG37	Ann	Beech	1000.00
SG14	David	Ford	1500.00
SA9	Mary	Howe	750.00
SG5	Susan	Brand	2000.00
SL41	Julie	Lee	750.00

Użycie AS

Czwarta kolumna tej tabeli wyników została wyprowadzona jako col4. Niektóre dialekty nadają kolumnie nazwę odpowiadającą jej pozycji w tabeli (np. col4); niektórzy mogą pozostawić pustą nazwę kolumny lub użyć wyrażenia wprowadzonego na liście SELECT.

Norma ISO pozwala na nazwanie kolumny za pomocą klauzuli AS:

```
SELECT staffNo, fName, lName, salary/12 AS monthlySalary
FROM Staff;
```

Wybór wiersza (klauzula WHERE)

Często musimy ograniczyć pobierane wiersze. Można to osiągnąć za pomocą klauzuli WHERE, która składa się ze słowa kluczowego WHERE, po którym następuje warunek wyszukiwania określające wiersze pobrania.

Oto 5 podstawowych warunków wyszukiwania:

- **Porównanie** (`=, <, >, <=, >=, !=, <>`) – porównaj wartość jednego wyrażenia z wartością innego wyrażenia
- **Zakres** (**`BETWEEN`**) – sprawdź, czy wartość wyrażenia mieści się w określonym zakresie wartości
- **Ustaw członkowstwo** (**`IN`**) – sprawdź, czy wartość wyrażenia jest równa jednej z zestawu wartości
- **Dopasowanie do wzorca** (**`LIKE, ILIKE`**) – sprawdź, czy ciąg pasuje do określonego wzorca
- **Null** (**`IS NULL`**)– sprawdź, czy kolumna ma wartość null (nieznaną)

Przykład:

- Warunek wyszukiwania porównawczego

Wymień wszystkich pracowników z pensją powyżej 10 000:

```
SELECT staffNo, fName, lName, position, salary
FROM Staff
WHERE salary > 10000;
```

Tabela wynikowa:

staffNo	fName	lName	position	salary
SL21	John	White	Manager	30000.00
SG37	Ann	Beech	Assistant	12000.00
SG14	David	Ford	Supervisor	18000.00
SG5	Susan	Brand	Manager	24000.00

Operatory porównania

W SQL dostępne są następujące proste operatory porównania:

<> nie jest równe (norma ISO)

!= nie jest równe (w niektórych dialektach)

= równa się

< jest mniejsze niż

<= jest mniejsze lub równe

> jest większe niż

>= jest większe lub równe

Bardziej złożone predykaty można generować za pomocą operatorów logicznych AND, OR i NOT z nawiasami, aby pokazać kolejność obliczeń.

Zasady obliczania wyrażenia warunkowego to:

- Wyrażenie jest obliczane od lewej do prawej
- Podwyrażenia w nawiasach są obliczane jako pierwsze
- NOT są obliczane przed AND i OR
- AND są obliczane przed OR

Przykłady:

- ?

(21)

Tabela wynikowa:

branchNo	street	city	postcode
B005	22 Deer Rd	London	SW1 4EH
B003	163 Main St	Glasgow	G11 9QX
B002	56 Clover Dr	London	NW10 6EU

- Warunek wyszukiwania zakresu (BETWEEN)

Wymień wszystkich pracowników z pensją od 20 000 do 30 000:

```
SELECT staffNo, fName, lName, position, salary
FROM Staff
WHERE salary BETWEEN 20000 AND 30000;
```

Tabela wynikowa:

staffNo	fName	lName	position	salary
SL21	John	White	Manager	30000.00
SG5	Susan	Brand	Manager	24000.00

- Warunek wyszukiwania zakresu (przy użyciu dwóch testów porównawczych)

Wymień wszystkich pracowników z pensją od 20 000 do 30 000:

```
SELECT staffNo, fName, lName, position, salary
FROM Staff
WHERE salary >= 20000 AND salary <= 30000;
```

- Ustaw warunek wyszukiwania członkowstwa (IN/NOT IN)

Wymień wszystkich menedżerów i przełożonych:

```
SELECT staffNo, fName, lName, position
FROM Staff
WHERE position IN('Manager', 'Supervisor');
```

(27)

- Ustaw warunek wyszukiwania członkowstwa (z OR)

Wymień wszystkich menedżerów i przełożonych:

```
SELECT staffNo, fName, lName, position
FROM Staff
WHERE position = 'Manager' OR position = 'Supervisor';
```

Dopasowanie do wzorca

SQL ma dwa specjalne symbole dopasowywania wzorców:

- Znak procent (%) reprezentuje dowolny ciąg zerowy lub więcej znaków (wieloznacznik)
- Znak podkreślenia (_) reprezentuje dowolny pojedynczy znak.

Wszystkie inne znaki we wzorze reprezentują same siebie. Na przykład:

- Adres **LIKE „H%”** oznacza, że pierwszym znakiem musi być H, ale reszta ciągu może być dowolna

- Adres **LIKE „H_ _ _”** oznacza, że ciąg musi zawierać dokładnie cztery znaki, z których pierwszy musi być H
- Adres **LIKE „%e”** oznacza dowolny ciąg znaków o długości co najmniej 1, z ostatnim znakiem e.
- Adres **LIKE „%Glasgow%”** oznacza ciąg znaków o dowolnej długości zawierający Glasgow
- Adres NOT **LIKE „H%”** oznacza, że pierwszym znakiem nie może być H.

Jeśli wyszukiwany ciąg może zawierać sam znak pasujący do wzorca, możemy użyć znaku zmiany znaczenia do reprezentowania znaku pasującego do wzorca.

Na przykład, aby sprawdzić ciąg „15%”, możemy użyć predykatu:

```
LIKE '15#%' ESCAPE '#';
```

Uwaga, niektóre RDBMS, takie jak Microsoft Office Access, używają symboli wieloznacznych * i ? zamiast % i _.

(31)

Tabela wynikowa:

ownerNo	fName	lName	address	telNo
CO87	Carol	Farrel	6 Achray St, Glasgow G32 9DX	0141-357-7419
CO40	Tina	Murphy	63 Well St, Glasgow G42	0141-943-1728
CO93	Tony	Shaw	12 Park Pl, Glasgow G4 0QR	0141-225-7025

NULL jako warunek wyszukiwania (IS NULL / IS NOT NULL)

Wymień szczegóły wszystkich wyświetleń dla obiektu PG4, w których nie podano komentarza:

```
SELECT clientNo, viewData
FROM Viewing
WHERE propertyNo = 'PG4' AND comment IS NULL;
```

Tabela wynikowa:

clientNo	propertyNo	viewDate	comment
CR56	PA14	24-May-04	too small
CR76	PG4	20-Apr-04	too remote
CR56	PG4	26-May-04	
CR62	PA14	14-May-04	no dining room
CR56	PG36	28-Apr-04	

clientNo	viewDate
CR56	26-May-04

Sortowanie wyników (ORDER BY)

Wiersze tabeli wynikowej zapytania SQL nie są ułożone w określonej kolejności. Możemy jednak zapewnić, że wyniki zapytania zostaną posortowane za pomocą klauzuli **ORDER BY** w instrukcji **SELECT**.

Klauzula **ORDER BY** składa się z rozdzielonych przecinkami listy identyfikatorów kolumn, według których ma być sortowany wynik. Identyfikatorem kolumny może być nazwa kolumny lub numer kolumny, który identyfikuje element listy **SELECT** poprzez jego pozycję na liście, 1 jest pierwszym (najbardziej od lewej) elementem na liście, 2 drugim elementem na liście, i tak dalej.

Klauzula **ORDER BY** umożliwia uporządkowanie pobranych w kolejności rosnącej (**ASC** – ustawienie domyślne) lub malejącej (**DESC**) w dowolnej kolumnie lub kombinacji kolumn, niezależnie od tego, czy kolumna ta pojawia się w wyniku. Jednak niektóre dialekty wymagają, aby elementy **ORDER BY** pojawiły się na liście **SELECT**. W obu przypadkach klauzula **ORDER BY** musi być zawsze ostatnią klauzulą instrukcji **SELECT**.

Przykład:

- Sortowanie jednokolumnowe:

Stwórz listę wynagrodzeń dla wszystkich pracowników, ułożoną w porządku malejącym wynagrodzeń:

```
SELECT staffNo, fName, lName, salary
FROM Staff
ORDER BY salary DESC;
```

Tabela wynikowa

staffNo	fName	lName	salary
SL21	John	White	30000.00
SG5	Susan	Brand	24000.00
SG14	David	Ford	18000.00
SG37	Ann	Beech	12000.00
SA9	Mary	Howe	9000.00
SL41	Julie	Lee	9000.00

- Sortowanie oparte na wielu kolumnach

Sporządź skróconą listę nieruchomości ułożonych w kolejności rodzaju nieruchomości oraz w kolejności malejącej kwoty wynajmu:

```
SELECT propertyNo, type, rooms, rent
```

```
FROM PropertyForRent
ORDER BY type, rent DESC;
```

Tabela wynikowa:

propertyNo	type	rooms	rent
PG16	Flat	4	450
PL94	Flat	4	400
PG36	Flat	3	375
PG4	Flat	3	350
PA14	House	6	650
PG21	House	5	600

Korzystanie z funkcji agregujących SQL

Norma ISO definiuje pięć funkcji agregujących:

a) **COUNT** – zwraca liczbę wartości w określonej kolumnie
b) **SUM** – zwraca sumę wartości w określonej kolumnie
c) **AVG** – zwraca średnią wartości w określonej kolumnie
d) **MIN** – zwraca najmniejszą wartość w określonej kolumnie
e) **MAX** – zwraca największą wartość w określonej kolumnie

Funkcje agregujące działają na pojedynczej kolumnie tabeli i zwracają pojedynczą wartość. **COUNT**, **MIN** i **MAX** dotyczą zarówno pól numerycznych, jak i nienumerycznych, ale **SUM** i **AVG** mogą być używane tylko w polach numerycznych.

Oprócz **COUNT(*)** każda funkcja najpierw eliminuje wartości null i działa tylko na pozostałych wartościach innych niż null. **COUNT(*)** to specjalne zastosowanie funkcji **COUNT**, która zlicza wszystkie wiersze tabeli, niezależnie od tego, czy występują wartości null, czy zduplikowane.

- użycie COUNT(*)

Ile nieruchomości kosztuje więcej niż 350 miesięcznie?

```
SELECT COUNT(*) AS myCount
FROM PropertyForRent
WHERE rent > 350;
```

myCount
2

- użycie COUNT i SUM

Znajdź całkowitą liczbę Menedżerów i sumę ich wynagrodzeń.

```
SELECT COUNT(staffNo) AS myCount,
       SUM(salary) AS mySum
FROM Staff
WHERE position = 'Manager';
```

myCount	mySum
2	54000.00

- użycie MIN, MAX, AVG

Znajdź minimalną, maksymalną I średnią pensję personelu.

```
SELECT MIN(salary) AS myMin,
       MAX(salary) AS myMax,
       AVG(salary) AS myAvg
FROM Staff;
```

myMin	myMax	myAvg
9000.00	30000.00	17000.00

Grupowanie wyników (GROUP BY)

Do uzyskania podsumowań służy klauzula **GROUP BY** instrukcji **SELECT**.

Zapytanie zawierające klauzulę **GROUP BY** nazywa się zapytaniem grupowym, ponieważ grupuje dane z tabel **SELECT** i tworzy jeden wiersz podsumowania dla każdej grupy. Kolumny wymienione w klauzuli **GROUP BY** nazywane są kolumnami grupującymi.

Norma ISO wymaga ścisłej integracji klauzuli **SELECT** i **GROUP BY**. Gdy używana jest funkcja **GROUP BY**, każdy element na liście **SELECT** musi mieć jedną wartość na grupę.

Polecenie SELECT może zawierać tylko:

- nazwy kolumn
- funkcje agregujące
- stałe
- wyrażenie obejmujące kombinacje powyższych

Przykład:

- Użycie GROUP BY

Znajdź liczbę pracowników pracujących w każdym oddziale i sumę ich wynagrodzeń.

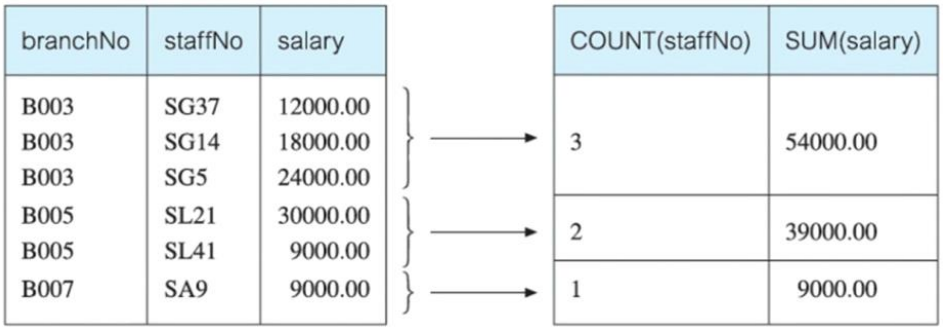
```
SELECT branchNo,
       COUNT(staffNo) AS myCount,
       SUM(salary) AS mySum
```

branchNo	myCount	mySum
B003	3	54000.00
B005	2	39000.00
B007	1	9000.00

```
FROM Staff
GROUP BY branchNo
ORDER BY branchNo;
```

Koncepcyjnie SQL wykonuje zapytanie w następujący sposób:

1. SQL dzieli personel na grupy zgodnie z ich numerami oddziałów. W każdej grupie wszyscy pracownicy mają ten sam numer oddziału. W tym przykładzie otrzymujemy trzy grupy:



branchNo	staffNo	salary
B003	SG37	12000.00
B003	SG14	18000.00
B003	SG5	24000.00
B005	SL21	30000.00
B005	SL41	9000.00
B007	SA9	9000.00

COUNT(staffNo)	SUM(salary)
3	54000.00
2	39000.00
1	9000.00

(50)

Ograniczenie grup (HAVING)

Klauzula **HAVING** jest przeznaczona do użytku z klauzulą **GROUP BY** w celu ograniczenia grup, które pojawiają się w końcowej tabeli wyników. Chociaż mają podobną składnię, **HAVING** i **WHERE** służą różnym celom.

Klauzula **WHERE** filtruje poszczególne wiersze przechodzące do końcowej tabeli wyników, natomiast **HAVING** filtruje grupy przechodzące do końcowej tabeli wyników.

Norma ISO wymaga, aby nazwy kolumn użyte w klauzuli **HAVING** pojawiały się również na liście **GROUP BY** lub były zawarte w funkcji agregującej.

W praktyce warunek wyszukiwania w klauzuli **HAVING** zawiera co najmniej jedną funkcję agregującą, w przeciwnym prazie warunek wyszukiwania można przenieść do klauzuli **WHERE** i zastosować do poszczególnych wierszy.

Pamiętaj, że w klauzuli **WHERE** nie można używać funkcji agregujących.

Przykład:

Dla każdego oddziału, w którym pracuje więcej niż jeden pracownik, znajdź liczbę pracowników pracujących w każdym oddziale i sumę ich wynagrodzeń.

```
SELECT branchNo,
```

```
    COUNT(staffNo) AS myCount,
    SUM(salary) AS mySum
FROM Staff
GROUP BY branchNo
HAVING COUNT(staffNo) > 1
ORDER BY branchNo;
```

branchNo	myCount	mySum
B003	3	54000.00
B005	2	39000.00

Podzapytania

Kompletną instrukcję `SELECT` można osadzić w innej instrukcji `SELECT`. Wyniki tej wewnętrznej instrukcji `SELECT` (lub podselekcji) są używane w zewnętrznej instrukcji, aby pomóc określić zawartość końcowego wyniku.

Podzapytania można użyć w klauzulach `WHERE` i `HAVING` zewnętrznej instrukcji `SELECT`, gdzie jest ona nazywana **podzapytaniem** lub **zapytaniem zagnieżdżonym**.

Podzapytania mogą również pojawiać się w instrukcjach INSERT, UPDATE i DELETE.

Istnieją trzy typy podzapytania:

- **Podzapytanie skalarne** – zwraca pojedynczą kolumnę i pojedynczy wiersz; czyli pojedynczą wartość. W zasadzie podzapytanie skalarne może być używane zawsze, gdy potrzebna jest pojedyncza wartość.
- **Podzapytanie wiersza** – zwraca wiele kolumn, ale tylko jeden wiersz. Podzapytanie wiersza może być używane zawsze, gdy potrzebny jest konstruktor wartości wiersza, zwykle w predykatach
- **Podzapytanie tabeli** – zwraca co najmniej jedną kolumnę i wiele wierszy. Podzapytanie tabeli może być używane zawsze, gdy potrzebna jest tabela, na przykład jako operand dla predykatu IN.

Przykłady:

- Używanie podzapytania z równością

Wymień pracowników, którzy pracują w oddziale na '163 Main St'.

```
SELECT staffNo, fName, lName, position
FROM Staff
WHERE branchNo = (
    SELECT branchNo
```

```
    FROM Branch
    WHERE street = '163 Main St'
);
```

Wewnętrzna instrukcja SELECT znajduje numer oddziału, który odpowiada oddziałowi o nazwie ulicy '163 Main St' (będzie tylko jeden taki numer oddziału, więc jest to przykład podzapytania skalarnego).

Zewnętrzny SELECT można zatem zapisać jako:

```
SELECT staffNo, fName, lName, position
FROM Staff
WHERE branchNo = 'B003';
```

Branch

branchNo	street	city	postcode
B005	22 Deer Rd	London	SW1 4EH
B007	16 Argyll St	Aberdeen	AB2 3SU
B003	163 Main St	Glasgow	G11 9QX
B004	32 Manse Rd	Bristol	BS99 1NZ
B002	56 Clover Dr	London	NW10 6EU

staffNo	fName	lName	position	sex	DOB	salary	branchNo
SL21	John	White	Manager	M	1-Oct-45	30000	B005
SG37	Ann	Beech	Assistant	F	10-Nov-60	12000	B003
SG14	David	Ford	Supervisor	M	24-Mar-58	18000	B003
SA9	Mary	Howe	Assistant	F	19-Feb-70	9000	B007
SG5	Susan	Brand	Manager	F	3-Jun-40	24000	B003
SL41	Julie	Lee	Assistant	F	13-Jun-65	9000	B005

staffNo	fName	lName	position
SG37	Ann	Beech	Assistant
SG14	David	Ford	Supervisor
SG5	Susan	Brand	Manager

- Używanie podzapytania z funkcją agregującą

Wymień wszystkich pracowników, których wynagrodzenie jest wyższe niż średnia pensja, i pokaż, o ile ich pensja jest wyższa od średniej.

```
SELECT staffNo, fName, lName, position
    salary – ( SELECT AVG(salary) FROM Staff ) AS salDiff
FROM Staff
WHERE salary > ( SELECT AVG(salary) FROM Staff );
```

Zauważ, że nie możemy napisać „`WHERE salary > AVG(salary)`", ponieważ w klauzuli WHERE nie można używać funkcji agregujących. Zamiast tego używamy zewnętrznego wyrażenia SELECT, aby znaleźć pracowników z pensją większą niż ta średnia. Zewnętrzne zapytanie zostaje zredukowane do:

```
SELECT staffNo, fName, lName, position
      salary - 17000 AS salDiff
FROM Staff
WHERE salary > 17000;
```

staffNo	fName	lName	position	salDiff
SL21	John	White	Manager	13000.00
SG14	David	Ford	Supervisor	1000.00
SG5	Susan	Brand	Manager	7000.00

W przypadku zapytań podrzędnych obowiązują następujące zasady:

1. Klauzula ORDER BY nie może być używana w podzapytaniu (chociaż może być używana w najbardziej zewnętrznej instrukcji SELECT)
2. Lista SELECT podzapytania musi składać się z pojedynczej nazwy kolumny lub wyrażenia, z wyjątkiem podzapytań używających słowa kluczowego EXISTS
3. Domyślnie nazwy kolumn w podzapytaniu odwołują się do nazwy tabeli w klauzuli FROM podzapytania. Możliwe jest odwołanie się do tabeli w klauzuli FROM zapytania zewnętrznego przez zakwalifikowanie nazwy kolumny
4. Gdy podzapytanie jest jednym z dwóch operandów biorących udział w porównaniu, podzapytanie musi pojawić się po prawej stronie porównania.

(62 / 63?)

Pracując od najbardziej wewnętrznego zapytania na zewnątrz pierwsze zapytanie wybiera numer oddziału przy '163 Main St'. Drugie zapytanie następnie wybiera tych pracowników, którzy pracują pod tym numerem oddziału. W takim przypadku może być znalezionych więcej niż jeden taki wiersz, więc nie możemy użyć warunku równości (=) w najbardziej zewnętrznym zapytaniu. Zamiast tego używamy słowa kluczowego IN. Zapytanie zewnętrzne pobiera następnie szczegóły nieruchomości zarządzanych przez każdego członka personelu w zapytaniu środkowym.

propertyNo	street	city	postcode	type	rooms	rent
PG16	5 Novar Dr	Glasgow	G12 9AX	Flat	4	450
PG36	2 Manor Rd	Glasgow	G32 4QX	Flat	3	375
PG21	18 Dale Rd	Glasgow	G12	House	5	600

ANY (SOME) i ALL

Słowa ANY I ALL mogą być używane z podzapytaniami, które tworzą pojedynczą kolumnę liczb. Jeśli podzapytanie jest poprzedzone słowem kluczowym ALL, warunek będzie spełniony tylko wtedy, gdy zostanie spełniony przez wszystkie wartości wytworzone przez podzapytanie. Jeśli

podzapytanie jest poprzedzone słowem kluczowym ANY, warunek będzie spełniony, jeśli zostanie spełniony przez dowolną (jedna lub więcej) wartości wytworzone przez podzapytanie. Jeśli podzapytanie jest puste, warunek ALL zwraca prawdę, a warunek ANY zwraca fałsz. Norma ISO pozwala również na użycie kwalifikatora SOME zamiast ANY.

Przykład:

Znajdź wszystkich pracowników, których wynagrodzenie jest wyższe niż wynagrodzenie co najmniej jednego pracownika w oddziale B003.

```
SELECT staffNo, fName, lName, position, salary
FROM Staff
WHERE salary > SOME (
     SELECT salary
     FROM Staff
     WHERE branchNo = 'B003'
);
```

staffNo	fName	lName	position	salary
SL21	John	White	Manager	30000.00
SG14	David	Ford	Supervisor	18000.00
SG5	Susan	Brand	Manager	24000.00

Chociaż to zapytanie można wyrazić za pomocą podzapytania, które znajduje minimalną pensję personelu w oddziale B003, a następnie zapytania zewnętrznego, które znajduje wszystkich pracowników, których wynagrodzenie jest wyższe niż ta liczba, w podejściu alternatywnym stosuje się słowo kluczowe SOME/ANY. Zapytanie wewnętrzne tworzy zbiór {12000, 18000, 24000}, a zapytanie zewnętrzne wybiera tych pracowników, których pensje są większe niż dowolna z wartości w tym zbiorze (tj. większe niż wartość minimalna 12000). Ta alternatywna metoda może wydawać się bardziej naturalna niż znalezienie minimalnej pensji w podzapytaniu.

Znajdź wszystkich pracowników, których wynagrodzenie jest wyższe niż wynagrodzenie każdego pracownika w oddziale B003.

```
SELECT staffNo, fName, lName, position, salary
FROM Staff
WHERE salary > ALL (
     SELECT salary
     FROM Staff
     WHERE branchNo = 'B003'
);
```

staffNo	fName	lName	position	salary
SL21	John	White	Manager	30000.00

Moglibyśmy użyć podzapytania, aby znaleźć maksymalną pensję pracowników w oddziale B003, a następnie użyć zapytania zewnętrznego, aby znaleźć wszystkich pracowników, których wynagrodzenie jest większe niż ta liczba. Jednak w tym przykładzie używamy słowa kluczowego ALL.

Zapytania dotyczące wielu tabel

Aby złączyć kolumny z kilku tabel w tabelę wynikową, musimy użyć operacji złączenia.

Operacja złączenia SQL łączy informacje z dwóch tabel, tworząc pary powiązanych wierszy z dwóch tabel.

Pary wierszy, które tworzą połączoną tabelę, to te, w których pasujące kolumny w każdej z dwóch tabel maja tę samą wartość.

Jeśli potrzebujemy uzyskać informacje z więcej niż jednej tabeli, wybór jest między użyciem podzapytania a użyciem złączenia. Jeśli ostateczna tabela wynikowa ma zawierać kolumny z różnych tabel, to musimy użyć złączenia.

Aby wykonać złączenie, po prostu dołączamy więcej niż jedną nazwę tabeli w klauzuli **FROM**, używając przecinka jako separatora i zazwyczaj dołączając klauzulę **WHERE** do określenia kolumny (kolumn) łączenia. Możliwe jest również użycie aliasu dla tabeli wymienionej w klauzuli **FROM**.

Proste złączenia

Wymień nazwiska wszystkich klientów, którzy oglądali nieruchomość wraz z dostarczonymi komentarzami.

```
SELECT c.clientNo, fNam, lName, propertyNo, comment
FROM Client c, Viewing v
WHERE c.clientNo = v.clientNo;
```

clientNo	fName	lName	propertyNo	comment
CR56	Aline	Stewart	PG36	
CR56	Aline	Stewart	PA14	too small
CR56	Aline	Stewart	PG4	
CR62	Mary	Tregear	PA14	no dining room
CR76	John	Kay	PG4	too remote

Standard SQL zapewnia następujące alternatywne sposoby określenia tego złączenia:

```
FROM Client c JOIN Viewing v ON c.clientNo = v.clientNo
FROM Client JOIN Viewing USING clientNo
```

```
FROM Client NATURAL JOIN Viewing
```

Sortowanie w złączeniach

Dla każdego oddziału podaj numery i nazwiska pracowników zarządzających nieruchomościami oraz nieruchomości, którymi zarządzają.

```
SELECT s.branchNo, s.staffNo, fName, lName, propertyNo
FROM Staff s, PropertyForRent p
WHERE s.staffNo = p.staffNo
ORDER BY s.branchNo, s.staffNo, propertyNo;
```

branchNo	staffNo	fName	lName	propertyNo
B003	SG14	David	Ford	PG16
B003	SG37	Ann	Beech	PG21
B003	SG37	Ann	Beech	PG36
B005	SL41	Julie	Lee	PL94
B007	SA9	Mary	Howe	PA14

Złączenie trzech tabel

Dla każdego oddziału podaj numery i nazwiska pracowników zarządzających nieruchomościami, w tym miasto, w którym znajduje się oddział oraz nieruchomości, którymi zarządzają pracownicy.

```
SELECT b.branchNo, b.city, s.staffNo, fName, lName, propertyNo
FROM Branch b, Staff s, PropertyForRent p
WHERE b.branchNo = s.branchNo AND s.staffNo = p.staffNo
ORDER BY b.branchNo, s.staffNo, propertyNo;
```

branchNo	city	staffNo	fName	lName	propertyNo
B003	Glasgow	SG14	David	Ford	PG16
B003	Glasgow	SG37	Ann	Beech	PG21
B003	Glasgow	SG37	Ann	Beech	PG36
B005	London	SL41	Julie	Lee	PL94
B007	Aberdeen	SA9	Mary	Howe	PA14

Wiele kolumn grupujących

Znajdź liczbę nieruchomości obsługiwanych przez każdego członka personelu.

```
SELECT s.branchNo, s.staffNo, COUNT(*) AS myCount
FROM Staff s, PropertyForRent p
WHERE s.staffNo = p.staffNo
```

branchNo	staffNo	myCount
B003	SG14	1
B003	SG37	2
B005	SL41	1
B007	SA9	1

```
GROUP BY s.branchNo, s.staffNo
ORDER BY s.branchNo, s.staffNo;
```

OUTER JOIN

Operacja złączenia łączy dane z dwóch tabel, tworząc pary powiązanych wierszy, w których pasujące kolumny w każdej tabeli mają tę samą wartość. Jeśli jeden wiersz tabeli jest niedopasowany, wiersz jest pomijany w tabeli wynikowej.

Norma ISO zapewnia kolejny zestaw operatorów złączenia, zwanych złączeniami zewnętrznymi. Złączenie zewnętrzne zachowuje wiersze, które nie spełniają warunku złączenia.

Istnieją trzy typy złączeń zewnętrznych:

- Lewe (LEFT)
- Prawe (RIGHT)
- Pełne zewnętrzne (FULL JOIN)

branchNo	bCity
B003	Glasgow
B004	Bristol
B002	London

propertyNo	pCity
PA14	Aberdeen
PL94	London
PG4	Glasgow

tabele źródłowe

branchNo	bCity	propertyNo	pCity
B003	Glasgow	PG4	Glasgow
B002	London	PL94	London

```
SELECT b.*, p.*
FROM Branch1 b, PropertyForRent1 p
WHERE b.bCity = p.pCity;
```

branchNo	bCity	propertyNo	pCity
B003	Glasgow	PG4	Glasgow
B004	Bristol	NULL	NULL
B002	London	PL94	London

```
SELECT b.*, p.*
FROM Branch1 b LEFT JOIN
PropertyForRent1 p ON b.bCity =
p.pCity;
```

branchNo	bCity	propertyNo	pCity
NULL	NULL	PA14	Aberdeen
B003	Glasgow	PG4	Glasgow
B002	London	PL94	London

```
SELECT b.*, p.*
FROM Branch1 b RIGHT JOIN
PropertyForRent1 p ON b.bCity =
p.pCity;
```

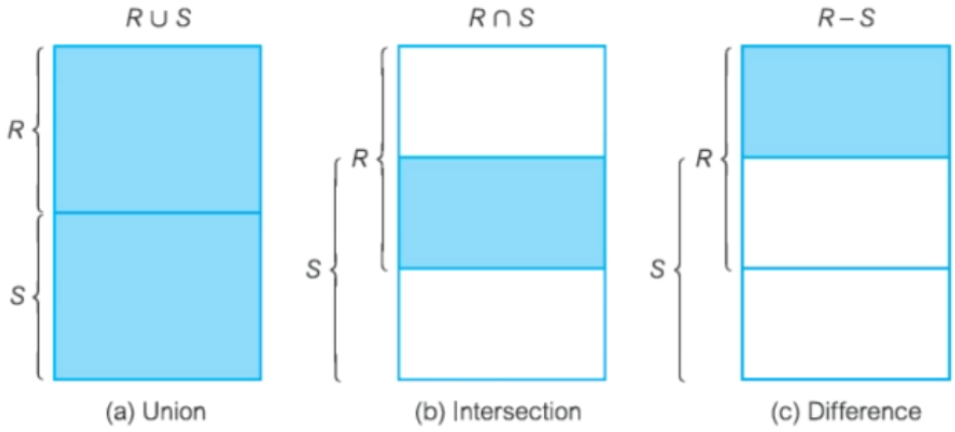
branchNo	bCity	propertyNo	pCity
NULL	NULL	PA14	Aberdeen
B003	Glasgow	PG4	Glasgow
B004	Bristol	NULL	NULL
B002	London	PL94	London

```
SELECT b.*, p.*
FROM Branch1 b FULL JOIN
PropertyForRent1 p ON b.bCity =
p.pCity;
```

Łączenie tabel wyników

W SQL możemy użyć normalnych operacji na zbiorach **Union**, **Intersection** i **Difference**, aby połączyć wyniki dwóch lub więcej zapytań w jedną tabelę wynikową:

- **Union** (A i B) to tabela zawierająca wszystkie wiersze znajdujące się w tabeli A lub tabeli B (lub w obu)
- **Intersection** (A i B) to tabela zawierająca wszystkie wiersze, które są wspólne dla obu tabel A i B
- **Difference** (A i B) to tabela zawierająca wszystkie wiersze, które znajdują się w tabeli A, ale nie znajdują się w tabeli B



Trzy operatory zbioru w standardzie ISO to:

- **UNION**
- **INTERSECT**
- **EXCEPT**

Forma klauzuli operatora zbioru to w każdym przypadku:

```
operator [ALL] [CORRESPONDING [BY {column1 [,…] } ] ]
```

Jeśli określono **CORRESPONDING BY**, operacja ustawiania jest wykonywana na nazwanych kolumnach; jeśli określono **CORRESPONDING**, ale nie podano klauzuli **BY**, operacja ustawiana jest wykonywana na kolumnach, które są

wspólne dla obu tabel. Jeśli określono **ALL**, wynik może zawierać zduplikowane wiersze.

Niektóre dialekty SQL nie obsługują **INTERSECT** i **EXCEPT**; inni używają **MINUS** zamiast **EXCEPT**.

Przykład:

Zbuduj listę wszystkich miast, w których znajduje się oddział lub nieruchomość.

```
(SELECT city                    or  (SELECT *
 FROM Branch                         FROM Branch
 WHERE city IS NOT NULL)             WHERE city IS NOT NULL)
 UNION                               UNION CORRESPONDING BY city
(SELECT city                        (SELECT *
 FROM PropertyForRent                FROM PropertyForRent
 WHERE city IS NOT NULL);            WHERE city IS NOT NULL);
```

city
London
Glasgow
Aberdeen
Bristol

Zbuduj listę wszystkich miast, w których znajduje się zarówno oddział, jak i nieruchomość.

```
(SELECT city                    or  (SELECT *
 FROM Branch)                        FROM Branch)
 INTERSECT                           INTERSECT CORRESPONDING BY city
(SELECT city                        (SELECT *
 FROM PropertyForRent);              FROM PropertyForRent);
```

city
Aberdeen
Glasgow
London

Zbuduj listę wszystkich miast, w których znajduje się oddział, ale nie ma nieruchomości.

```
(SELECT city                    or  (SELECT *
 FROM Branch)                        FROM Branch)
 EXCEPT                              EXCEPT CORRESPONDING BY city
(SELECT city                        (SELECT *
 FROM PropertyForRent);              FROM PropertyForRent);
```

city
Bristol

Aktualizacja bazy danych

SQL to kompletny język manipulacji danymi, który może być używany do modyfikowania danych w bazie danych, a także do wykonywania zapytań do bazy danych. Dostępne są trzy instrukcje SQL umożliwiające modyfikację zawartości tabel w bazie danych:

- **INSERT** – dodaje nowe wiersze danych do tabeli;
- **UPDATE** – modyfikuje istniejące dane w tabeli;
- **DELETE** – usuwa wiersze danych z tabeli.

Dodawanie danych do bazy danych (INSERT)

Istnieją dwie formy instrukcji INSERT. Pierwsza pozawala na wstawienie pojedynczego wiersza nazwanej tabeli i ma następujący format:

```
INSERT INTO NazwaTabeli [(ListaKolumn)] VALUES (ListaDanych)
```

Druga forma instrukcji INSERT umożliwia kopiowanie wielu wierszy z jednej lub kilku tabel do drugiej i ma następujący format:

```
INSERT INTO NazwaTabeli [(ListaKolumn)]
SELECT …
```

ListaDanych musi odpowiadać ListaKolumn następująco:

- Liczba pozycji na każdej liście musi być taka sama
- Pomiędzy pozycjami na dwóch listach musi istnieć bezpośrednia korespondencja, tak aby pierwsza pozycja w ListaDanych odnosiła się do pierwszej pozycji w ListaKolumn, druga pozycja w ListaDanych miała zastosowanie do drugiej pozycji w ListaKolumn itd.
- Typ danych każdego elementu w ListaDanych musi być zgodny z typem danych odpowiedniej kolumny

Przykłady:

Wstaw nowy wiersz do tabeli Staff wprowadzający dane dla wszystkich kolumn.

```
INSERT INTO Staff
VALUES ('SG16', 'Alan', 'Brown', 'Assistant', 'M', DATE '1957-05-25', 8300, 'B003');
```

Wstaw nowy wiersz do tabeli Staff zawierający dane dla wszystkich kolumn: staffNo, fName, lName, position, salary i branchNo.

```
INSERT INTO Staff(staffNo, fName, lName, position, salary, branchNo)
```

```
VALUES('SG44', 'Anne', 'Jones', 'Assistant', 8100, 'B003');
```

albo

```
INSERT INTO Staff
VALUES('SG44', 'Anne', 'Jones', 'Assistant', NULL, NULL, 8100,
'B003');
```

Załóżmy, że istnieje tabela StaffPropCount, która zawiera nazwiska pracowników i liczbę zarządzanych nieruchomości:

```
StaffPropCount(staffNo, fName, lName, propCount)
```

Wypełnij tabelę StaffPropCount, używając szczegółów z tabel Staff i PropertyForRent.

```
INSERT INTO StaffPropCount
(SELECT s.staffNo, fName, lName, COUNT(*)
FROM Staff s, PropertyForRent p
WHERE s.staffNo = p.staffNo
GROUP BY s.staffNo, fName, lName)
UNION
(SELECT staffNo, fName, lName, 0
FROM Staff s
WHERE NOT EXISTS (SELECT *
FROM PropertyForRent p
WHERE p.staffNo = s.staffNo));
```

Modyfikacja danych w bazie danych (UPDATE)

Instrukcja UPDATE umożliwia zmianę zawartości istniejących wierszy w nazwanej tabeli. Format polecenia następujący:

```
UPDATE NazwaTabeli
SET NazwaKolumny1 = Wartosc1 [, …]
[WHERE warunek]
```

Klauzula `SET` określa nazwy co najmniej jednej kolumny, która ma zostać zaktualizowana.

Klauzula `WHERE` jest opcjonalna; jeśli zostanie pominięta nazwane kolumny są aktualizowane dla wszystkich wierszy w tabeli. Jeśli określono klauzulę `WHERE`, aktualizowane są tylko te wiersze, które spełniają warunek wyszukiwania.

Nowe wartości Wartosc muszą być zgodne z typami danych dla odpowiednich kolumn.

Przykłady:

Daj wszystkim pracownikom podwyżkę płacy o 3%.

```
UPDATE Staff
SET salary = salary * 1.03;
```

Daj wszystkim menedżerom 5% podwyżkę wynagrodzenia.

```
UPDATE Staff
SET salary = salary * 1.05;
WHERE position = ‘Manager’;
```

Awansuj Davida Forda (staffNo = ‘SG14’) na menedżera i zmień jego pensję na 18 000.

```
UPDATE Staff
SET position = ‘Manager’, salary = 18000;
WHERE staffNo = ‘SG14’;
```

Usuwanie danych z bazy danych (DELETE)

Instrukcja DELETE umożliwia usuwanie wierszy z tabeli. Format polecenia jest następujący:

```
DELETE FROM NazwaTabeli
[WHERE Warunek]
```

Warunek jest opcjonalny; jeśli zostanie pominięty, wszystkie wiersze zostaną usunięte z tabeli. Nie powoduje to usunięcia samej tabeli – aby usunąć zawartość tabeli i definicję tabeli, należy zamiast tego użyć instrukcji DROP TABLE.

Jeśli określono Warunek, usuwane są tylko te wiersze, które spełniają warunek.

Przykłady:

Usuń wszystkie wyświetlenia związane z nieruchomością PG4.

```
DELETE FROM Viewing
WHERE propertyNo = ‘PG4’;
```

Usuń wszystkie wiersze z tabeli przeglądania

```
DELETE FROM Viewing;
```

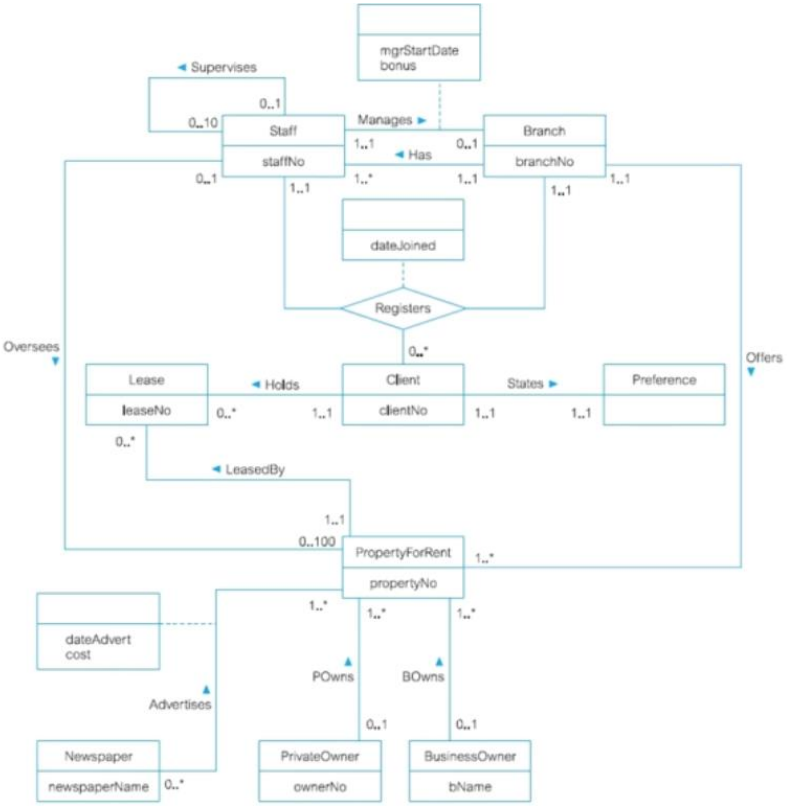
Encje

Zbiory encji

Encja – grupa obiektów o tych samych właściwościach, które identyfikuje się jako posiadające samodzielnie istnienie

Typy encji (https://afteracademy.com/blog/what-is-an-entity-entity-type-and-entity-set)

(4)

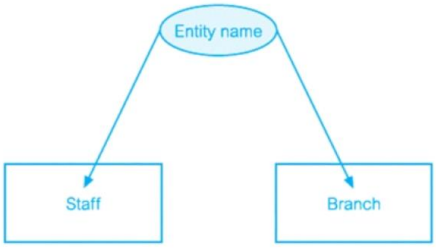


Podstawową koncepcją modelu ER jest zbiór encji, który reprezentuje grupę „obiektów” w „świecie rzeczywistym” o tych samych właściwościach.

Zbiory encji mają niezależne istnienie i mogą być obiektami z fizycznym („rzeczywistym") istnieniem lub obiektami z konceptualnym („abstrakcyjnym") istnienie.

Schematyczna reprezentacja zbiorów encji

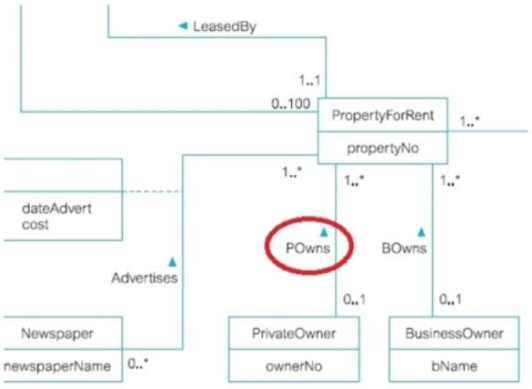
Każdy zbiór encji jest pokazany jako prostokąt oznaczony nazwą encji, która zwykle jest rzeczownikiem w liczbie pojedynczej. W **UML** (**Unified Modeling Language**) pierwsza litera każdego słowa w nazwie encji to wielka litera na przykład, Staff i PropertyForRent).



Związki

Związek - zestaw znaczących powiązań pomiędzy zbiorami encji.

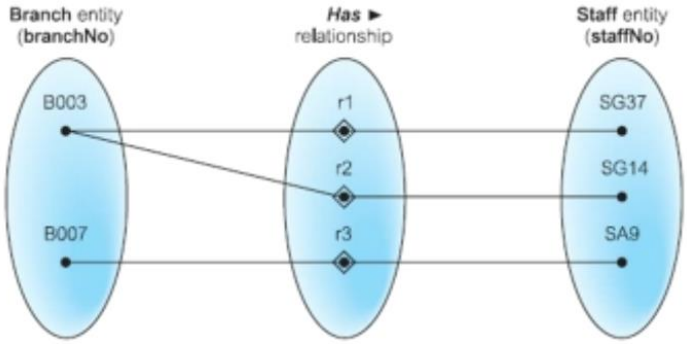
Związek to zestaw powiązań między co najmniej jednym uczestniczącym zbiorem encji. Każdy związek otrzymuje nazwę opisującą jego funkcję.



Przykład związku o nazwie Has

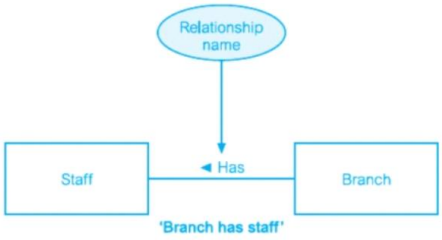
W tym przykładzie mamy powiązanie pomiędzy encjami Branch i Staff, czyli Branch Has Staff. Każde wystąpienie związku Has wiąże jedno wystąpienie encjami Branch z jednym wystąpieniem encji Staff. Możemy zbadać przykład pojedynczych wystąpień związku Has za pomocą sieci semantycznej. Sieć semantyczna to model na poziomie obiektu, który wykorzystuje symbol • do reprezentacji encji i symbol ◈ do reprezentacji związku.

Sieć semantyczna pokazująca pojedyncze wystąpienia związku Has



Schematyczne reprezentacja związków

Każdy związek jest wyświetlany jako linia łącząca skojarzone zbiory encji, oznaczona nazwą związku. Graficzna reprezentacja związku Branch Has Staff.



Stopień związku – Liczba zbiorów encji uczestniczących w związku

Encje zaangażowane w określony związek są jako uczestnikami tego związku. Liczba uczestników związku nazywana jest stopniem związku. Dlatego stopień związku wskazuje stopień związków zaangażowanych w związek. Związek drugiego stopnia nazywana się binarnym (najczęstszy stopień związku).

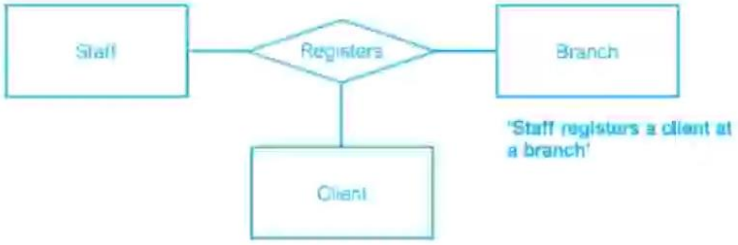
Wyróżnia się również związki trzeciego i czwartego stopnia.

Termin „związek złożony" jest używany do opisania związków o stopniach wyższych niż binarne.

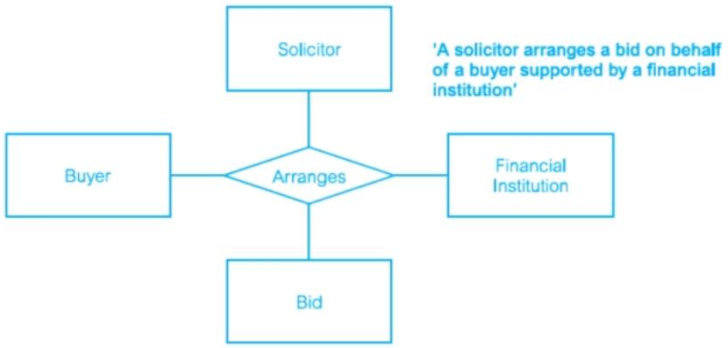
Przykład związku binarnego

Przykład związku trzeciego stopnia

Notacja UML używa rombu do reprezentowania związku o stopniach wyższych niż binarne. Nazwa związku jest wyświetlana wewnątrz rombu i w tym przypadku strzałka kierunkowa zwykle związana z nazwą jest pomijana...



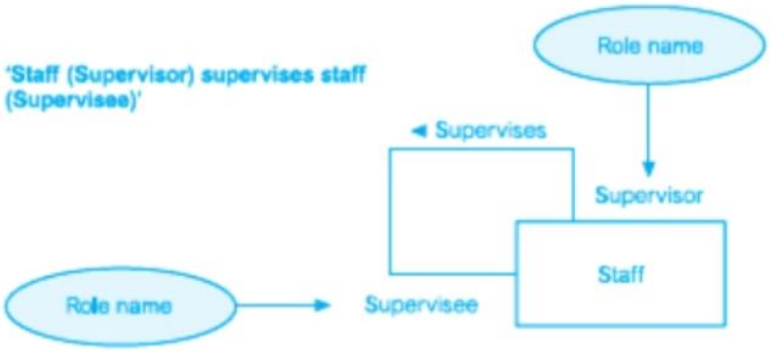
Przykład związku czwartego stopnia.

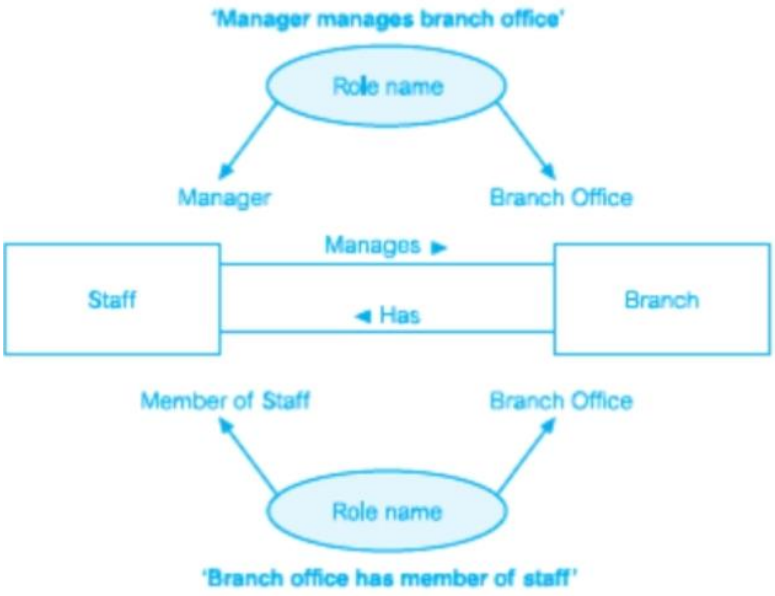


Związki rekurencyjne

Związki rekurencyjne – związki, w których ten sam związek encji występuje więcej niż raz w różnych rolach.

Przykład związku rekurencyjnego o nazwie Supervises z nazwami ról Supervisor i Supervise.





Atrybuty

Atrybuty – właściwość encji lub związku

Poszczególne właściwości zbiorów encji nazywane są atrybutami.

Atrybuty przechowują wartości opisujące każde wystąpienie encji i reprezentują główną część danych przechowywanych w bazie danych.

Domena atrybutu – zestaw dopuszczalnych wartości dla jednego lub większe liczby atrybutów.

Każdy atrybut jest powiązany z zestawem wartości zwanym domeną. Domena definiuje potencjalne wartości, które atrybut może posiadać i jest podobna do koncepcji domeny w modelu relacyjnym.

Atrybut prosty – atrybut złożony z jednego składnika o niezależnym istnieniu.

Proste atrybuty nie mogą być dalej dzielone na mniejsze elementy.

Atrybut złożony – atrybut złożony z wielu elementów, z których każdy ma niezależne istnienie.

Niektóre atrybuty można dalej podzielić, aby uzyskać mniejsze komponenty, które mają niezależne istnienie.

163 Main St, Glasgow, G11 9QX

Atrybut jednowartościowy - atrybut, który przechowuje pojedynczą wartość dla każdego wystąpienia zbioru encji.

Większość atrybutów ma jedną wartość.

branchNo B003

Atrybut wielowartościowy – atrybut, który przechowuje wiele wartości dla każdego wystąpienia zbioru encji.

Niektóre atrybuty mają wiele wartości dla każdego wystąpienia encji.

telNo 0141-339-2178 0141-339-4439

Atrybuty pochodne – atrybut reprezentujący wartość, którą można wyprowadzić z wartości powiązanego atrybutu lub zestawu atrybutów, niekoniecznie w tego samego zbioru encji.

Można wyprowadzić wartości posiadane przez niektóre atrybuty.

duration = rentFinish - rentStart

Klucze

Klucz potencjalny (kandydujący) – minimalny zestaw atrybutów, który jednoznacznie identyfikuje każde wystąpienie zbioru encji.

Klucz potencjalny nie może zawierać wartości null.

Klucz podstawowy – klucz potencjalny wybrany w celu jednoznacznej identyfikacji każdego wystąpienia zbioru encji.

Zbiór encji może mieć więcej niż jeden klucz potencjalny.

staff number (staffNo)
National Insurance Number (NIN)

Wybór klucza podstawowego dla encji opiera się na rozważaniach dotyczących:

- długości atrybutu
- minimalnej wymaganej liczby atrybutów
- pewności, że zachowa unikalność

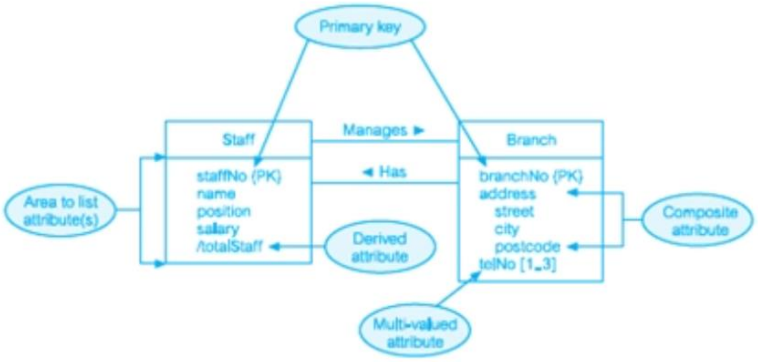
staffNo SG14
NIN (National Insurance Number) WL220658D

Pozostałe klucze potencjalne są określane jako klucze alternatywne.

Klucze złożone – klucz potencjalny, który składa się z co najmniej dwóch atrybutów.

W niektórych przypadkach klucz zbioru encji składa się z kilku atrybutów, których wartości razem są unikatowe dla każdego wystąpienia encji, ale nie oddzielnie.

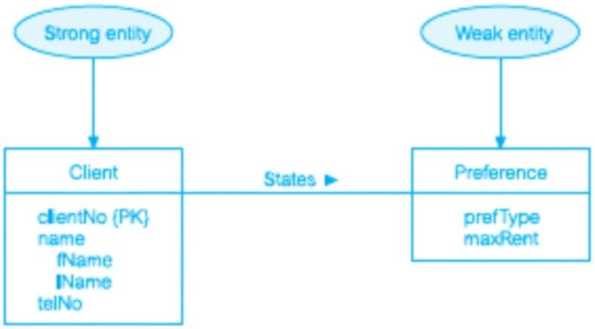
Schematyczne przedstawienie zbiorów encji Staff i Branch oraz ich atrybutów.



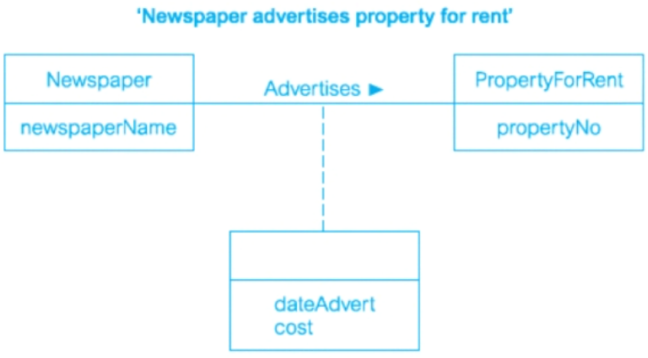
Silne i słabe zbiory encji

Silny zbiór encji – Zbiór encji, który nie jest zależny od istnienia innego zbioru encji.

Silny zbiór encji Client i słaby zbiór encji Preference.



Atrybuty można również przypisać do związków



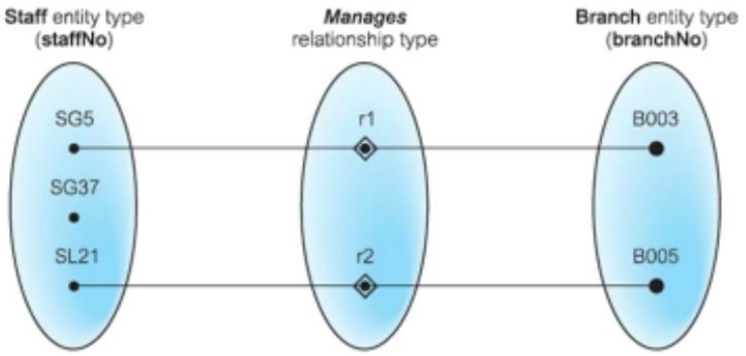
Więzy strukturalne

Krotność – liczba (lub zakres) możliwych wystąpień zbioru encji, które mogą odnosić się do pojedynczego wystąpienia powiązanego zbioru encji poprzez określony związek.

Najczęstszy stopień związku to binarny. Związki binarne są ogólnie określane jako:

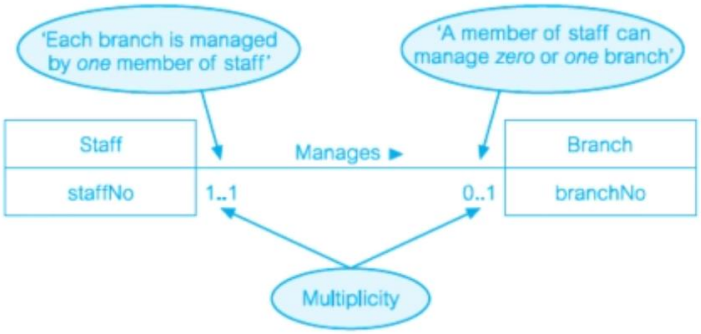
- jeden-do-jednego (1:1)
- jeden-do-wielu (1:*)
- wiele-do-wielu (*:*)

Związki wzajemnie jednoznaczne (1:1)



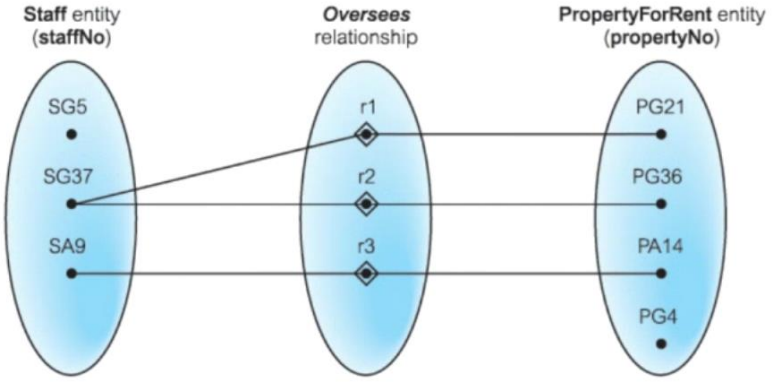
Sieć semantyczna pokazująca dwa wystąpienia związku Staff Manages Branch.

Schematyczne przedstawienie związku 1:1



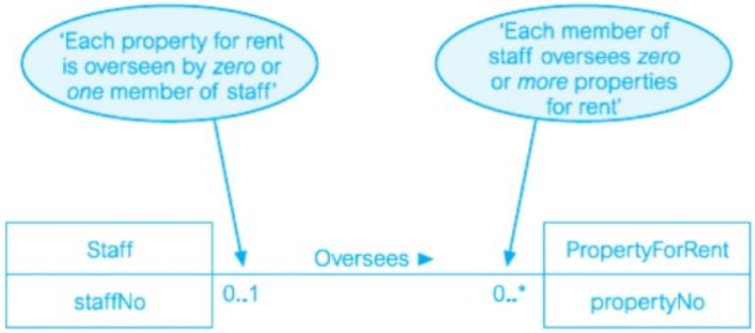
Krotność związku wzajemnie jednoznacznego Staff Manages Branch

Związki typu jeden do wielu



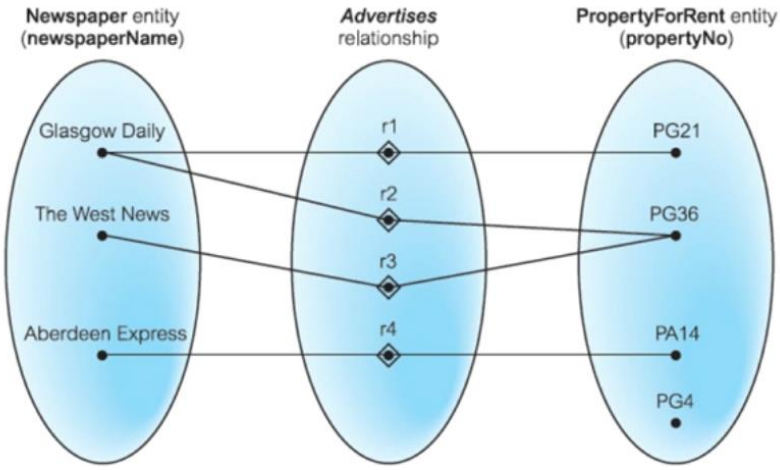
Sieć semantyczna pokazująca trzy wystąpienia związku Staff Oversees PropertyForRent

Schematyczne przedstawienie związku 1:*



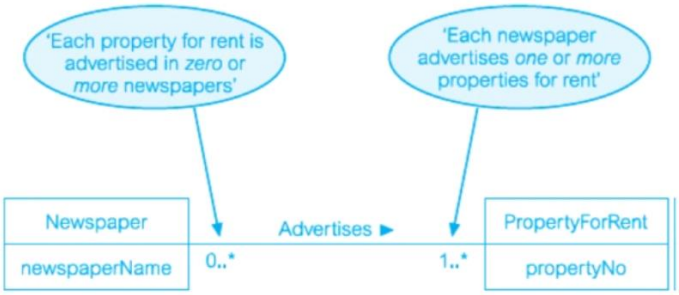
Krotność związku typu jeden do wielu Staff Oversees PropertyForRent

Związki typu wiele do wielu



Sieć semantyczna pokazująca cztery wystąpienia związku Newspaper Advertises PropertyForRent

Schematyczne przedstawienie związku *:*

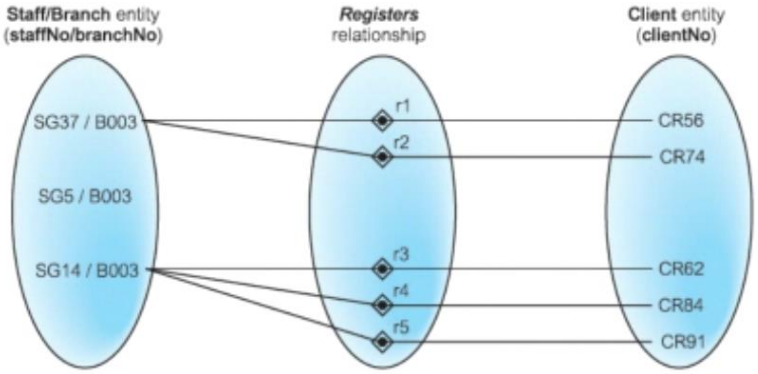


Krotność związku typu wiele do wielu
Newspaper Advertises PropertyForRent

Krotność związków złożonych

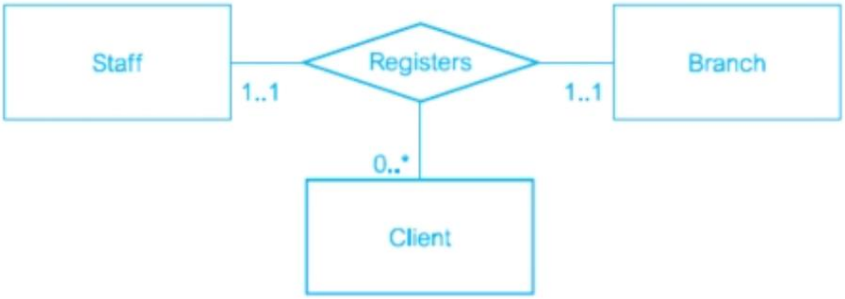
Krotność związków złożonych – liczba (lub zakres) możliwych wystąpień zbioru encji w n-argumentowym związku, gdy inne (n – 1) wartości są stałe.

Krotność związków n-argumentowych reprezentuje potencjalną liczbę wystąpień encji w związku, gdy (n-1) wartości ustalono dla innych uczestniczących zbiorów encji.



Sieć semantyczna pokazująca pięć wystąpień związku trójskładnikowego (potrójnego) Registers z ustalonymi wartościami dla zbiorów encji Staff i Branch.

Schematyczne przedstawienie krotności związku



Krotność w potrójnym związku Registers.

Sposoby przedstawienia krotności związku

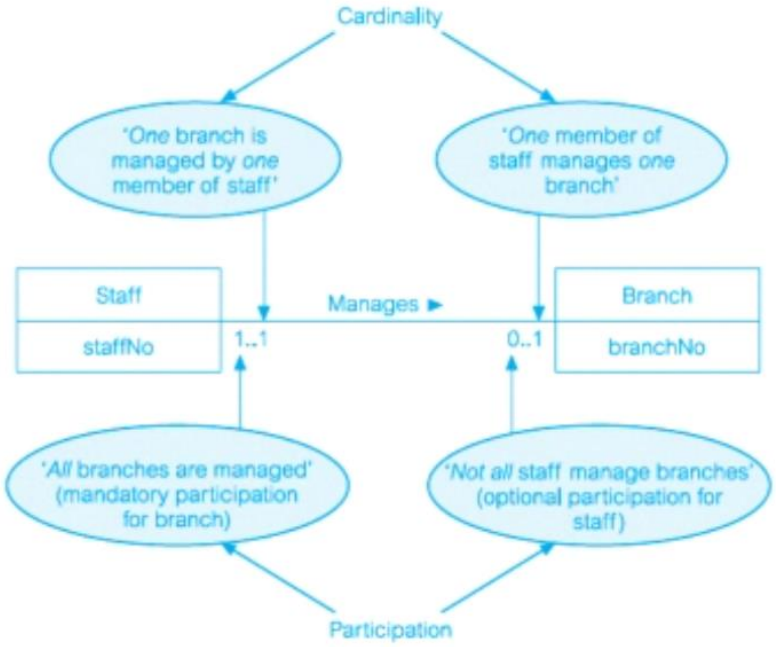
Oznaczenie krotności związku	Znaczenie (krotność wystąpienia związku)
0..1	zero lub jeden
1..1 (lub 1)	dokładnie jeden
0..* (lub *)	zero lub wiele
1..*	jeden lub wiele
6..12	minimum 6 i maximum 12
0, 4, 8–10	zero lub cztery lub osiem lub dziewięć lub dziesięć

Więzy liczności i uczestnictwa

Krotność składa się z dwóch oddzielnych ograniczeń znanych jako kardynalność i uczestnictwo.

Liczność – Cardinality – opisuje maksymalną liczbę możliwych wystąpień zależności dla encji uczestniczącej w danym związku.

Uczestnictwo – Participation – określa, czy w związku uczestniczą wszystkie, czy tylko niektóre wystąpienia encji.



Krotność opisywana jako więzy liczności i uczestnictwa dla związku Staff Manages Branch (1:1)

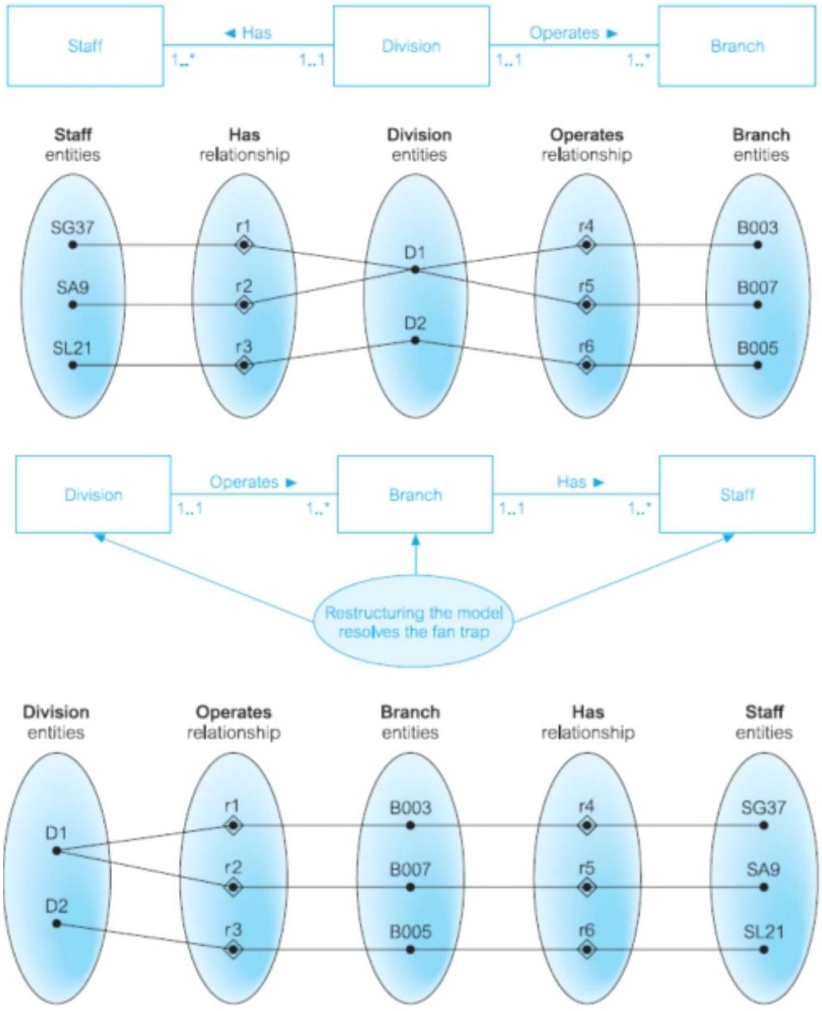
Problemy z modelami ER

Problemy z modelami ER są określane jako **pułapki połączeń** i zwykle występują z powodu błędnej interpretacji znaczenia niektórych związków.

Pułapki wachlarzowe

Pułapki wachlarzowe – gdy model reprezentuje związek między zbiorami encji, ale ścieżka między wystąpieniami niektórych encji jest niejednoznaczna.

Pułapka wachlarzowa może istnieć, gdy dwa lub więcej związków 1:* rozchodzi się od tej samej encji.

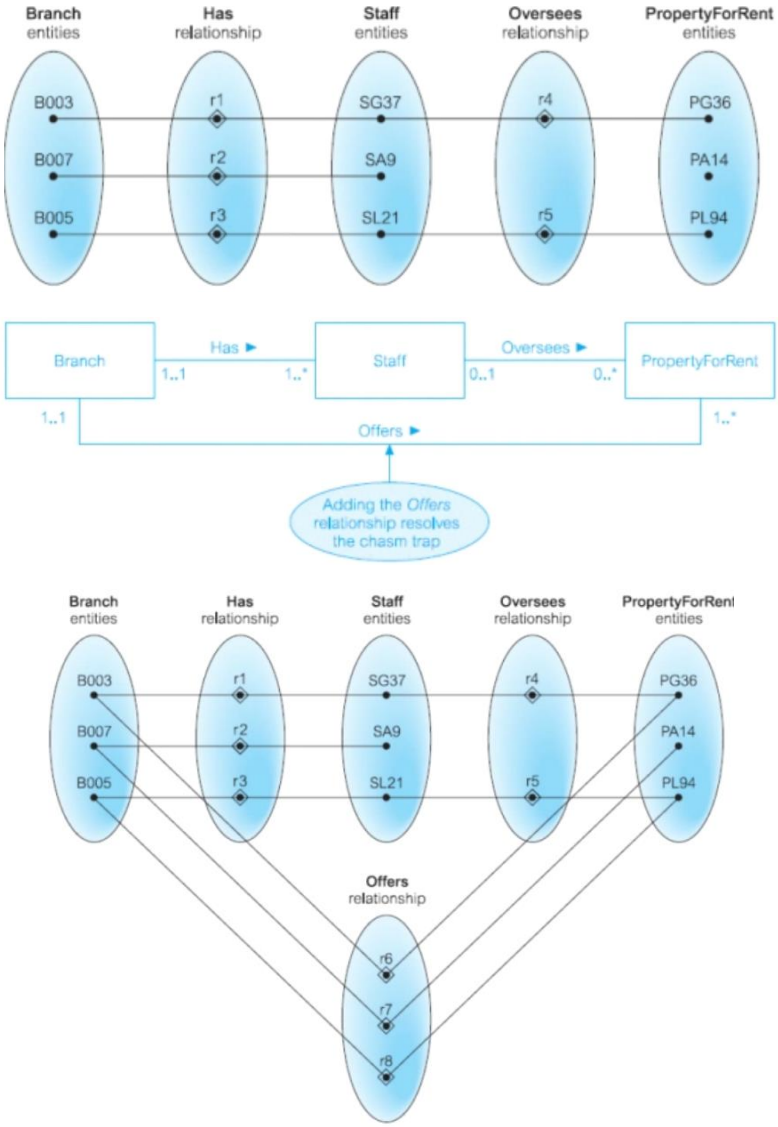


Pułapki szczelinowe

Pułapki szczelinowe – gdy model sugeruje istnienie związku między zbiorami encji, ale ścieżka powiązania nie istnieje między wystąpieniami niektórych encji.

Pułapki szczelinowe mogą wystąpić, gdy istnieje jeden lub więcej związków z minimalną krotnością zero (czyli opcjonalne uczestnictwo) tworzące część ścieżki między powiązanymi encjami.

Ten model przedstawia sytuację, gdy jeden oddział (branch) ma jednego lub więcej pracowników, którzy nadzorują zero lub więcej nieruchomości do wynajęcia. Jednak nie wszyscy pracownicy nadzorują nieruchomość i nie wszystkie nieruchomości są nadzorowane przez członków personelu.



Normalizacja

Normalizacja – technika służąca do wyznaczania zbioru relacji o pożądanych właściwościach, biorąca pod uwagę wymagania przedsiębiorstwa dotyczące danych.

Cel Normalizacji:

- minimalna liczba atrybutów niezbędnych do obsługi wymagań przedsiębiorstwa dotyczących danych;

- atrybuty o ścisłym związku logicznym (opisanym jako zależność funkcjonalna) znajdują się w tej samej relacji;

- minimalna nadmiarowość z każdym atrybutem reprezentowanym tylko raz, z ważnym wyjątkiem atrybutów, które tworzą całość lub część kluczy obcych, które są niezbędne do łączenia powiązanych relacji

Redundancja danych i anomalie aktualizacji

Korzyści ze zminimalizowania nadmiarowości danych:

- aktualizacje danych przechowywanych w bazie danych osiągane są przy minimalnej liczbie operacji, zmniejszając tym samym możliwości wystąpienia niespójności danych w bd;

- zmniejszenie przestrzeni przechowywania plików wymaganej przez relacje bazowe, minimalizując w ten sposób koszty.

(redundancja zajmuje więcej miejsca, ale dzieląc dane mamy powtórzone klucze co może spowodować problem w widoczności minimalizacji)

(tabelki z redundancją)

Relacje mają następujące...

(dotyczy tabelki)

Anomalie wstawiania

W tym przykładzie występują dwa główne typy anomalii wstawiania:

- aby wstawić dane nowych pracowników do relacji StaffBranch, musimy dołączyć...

Anomalie usuwania

Jeśli usuniemy rekord z relacji StaffBranch...

Anomalie modyfikacji

Jeśli chcemy

Cechy dekompozycji

Istnieją dwie ważne cechy związane z rozkładem większej relacji na mniejsze relacje:

- właściwość łączenia bezstratnego zapewnia, że każde wystąpienie oryginalnej relacji może zostać zidentyfikowane na podstawie odpowiednich wystąpień w mniejszych relacjach

- właściwość zachowywania zależności zapewnia...

Zależność funkcyjna

Opisuje związek pomiędzy atrybutami w relacji.

Np., jeśli A i B są atrybutami relacji R, to B jest funkcyjnie zależne od A (oznaczone jako A->B), jeśli każda wartość A jest powiązana z dokładnie jedną wartością B. (A i B mogą składać się z jednej lub więcej atrybutów)

Zależność między atrybutami A i B można przedstawić schematycznie:

(obrazek)

(obrazek)

Proces normalizacji

Początkowo zaproponowano trzy normalne postacie, nazwane:

- pierwszą postacią normalną (1NF)

- drugą postacią normalną (2NF)

- trzecią postacią normalną (3NF)

Następnie R.Boyce i EF Codd wprowadzili silniejszą definicję 3NF zwanej formą normalną Boyce-Codd (BCNF)

Z wyjątkiem 1NF, wszystkie te formy normalne są oparte na funkcjonalnych zależnościach między atrybutami relacji.

Wyższe formy normalne które wykraczają poza BCNF zostały wprowadzone później, takie jak 4NF i 5NF.

Jednak te późniejsze normalne formy radzą sobie z sytuacjami, które są bardzo rzadkie.

(forma graficzna normalizacji)

Schematyczna ilustracja relacji między postaciami normalnymi

Postać nieznormalizowana (Unnormalized Form UNF) to tabela zawierająca co najmniej jedną powtarzającą się grupę.

(nieznormalizowana tabela ClientRental)

Pierwsza postać normalna (1NF)

Relacja, w której przecięcie każdego wiersza i kolumny zawiera jedną i tylko jedną wartość

Istnieją dwa typowe podejścia do usuwania powtarzających się grup z nieznormalizowanych tabel:

1. wprowadzając odpowiednie dane w puste kolumny wierszy zawierających powtarzające się dane. Takie podejście powszechnie nazywane "spłaszczeniem" tabeli

2. umieszczając powtarzające się dane, wraz z kopią oryginalnych atrybutów kluczowych, w oddzielnej relacji

(pierwsza postać normalna relacji ClientRental)

(alternatywna pierwsza postać normalna relacji ClientRental)

Druga Postać Normalna (2NF)

Druga forma normalna dotyczy relacji z kluczami złożonymi, czyli relacji z kluczem podstawowym składającym się z 2 lub więcej atrybutów. Relacja z jednoatrybutowym kluczem podstawowym jest automatycznie w co najmniej 2NF.

Relacja, która jest w pierwszej postaci normalnej, a każdy atrybut niebędący kluczem podstawowym jest w pełni funkcjonalnie uzależniony od klucza podstawowego.

(relacje drugiej postaci normalnej pochodzące z relacji ClientRental)

Trzecia Postać Normalna (3NF)

Relacja, która jest w 1. i 2. postaci normalnej i w której żaden atrybut inny niż klucz podstawowy nie jest przechodnio zależny od klucza podstawowego

(relacje trzeciej postaci normalnej wprowadzone z PropertyOwner)

(rozkład relacji ClientRental w 1NF do relacji w 3NF)

Ogólne definicje 2NF i 3NF

Druga Postać Normalna (2NF) - relacja, która jest w 1. postaci normalnej i każdy atrybut klucza niekandydującego jest w pełni funkcjonalne zależny od dowolnego klucza kandydującego

Trzecie Postać Normalna (3NF) - relacja, która jest w 1. i 2. postaci normalnej i w której żaden atrybut klucza niekandydującego nie jest przejściowo zależny od żadnego klucza kandydującego

(kandydujący == potencjalny)

Postać Normalna Boyce'a-Codd'a (BCNF)

Postać Normalna BCNF - relacja jest w BCNF wtedy i tylko wtedy, gdy każdy wyznacznik jest kluczem kandydującym

Różnica pomiędzy 3NF i BCNF polega na tym, że 3NF dopuszcza zależność funkcyjną A->B, jeśli B jest atrybutem klucza głównego, a A nie jest kluczem kandydującym, podczas gdy BCNF zezwala na nią tylko wtedy, gdy atrybut A jest kluczem kandydującym.

(tabelka ClientInterview)

Relacja ClientInterview ma 3 klucze kandydujące:

(clientNo, interviewDate),

(staffNo, interviewDate, interviewTime),

(roomNo, interviewDate, interviewTime).

Te trzy złożone klucze kandydujące nakładają się na siebie, dzieląc wspólny atrybut interviewDate.

Wybieramy (clientNo, interviewDate) jako klucz podstawowy dla tej relacji.

(co jeżeli klient przyjdzie 2 razy tego samego dnia?)

(tabelka ClientInterview)

Relacja ClientInterview ma następujące zależności funkcyjne:

fd1 clientNo, interviewDate -> interviewTime, staffNo, roomNo (klucz podstawowy)

fd2 staffNo, interviewDate, interviewTime -> clientNo (klucz kandydujący)

fd3 roomNo, interviewDate, interviewTime -> staffNo, clientNo (klucz kandydujący)

fd4 staffNo, interviewDate -> roomNo (przypisany pokój do pracownika danego dnia)

Interview(clientNo, interviewDate, interviewTime, staffNo)

StaffRoom(staffNo, interviewDate, roomNo)

4NF

Możliwe istnienie wielowartościowych (multi-valued dependency MVD) w relacji wynika z 1NF, która uniemożliwia atrybutowi w rekordzie posiadanie zestawu wartości

Ten typ ograniczenia jest określany jako zależność wielowartościowa i skutkuje nadmiarowością danych.

(tabela BranchStaffOwner)

To ograniczenie reprezentuje wielowartościową zależność w relacji BranchStaffOwner.

Innymi słowy zależność wielowartościowa istnieje, ponieważ dwie niezależne 1:* są reprezentowane w relacji BranchStaffOwner.

Zależność wielowartościowa(mvd) - reprezentuje zależności między atrybutami (np A,B i C) w relacji, tak że dla każdej wartości A istnieje zestaw wartości dla B i zestaw wartości dla C. Jednak zestaw wartości...

Zależności wielowartościowe mogą być

- trywialne

MVD A-->>B w relacji R jeśli

a) B jest podzbiorem A

b) A u B = R

- nietrywialne, jeśli ani a) ani b) nie są spełnione

...

Reprezentujemy mvd między atrybutami A, B i C w relacji za pomocą następującej notacji

...

Wielowartościowa zależność w relacji BranchStaffOwner nie jest trywialna, ponieważ żaden warunek (a ani b) nie jest prawdziwy dla tej relacji.

Relacja BranchStaffOwner jest zatem ograniczona przez nietrywialny MVD do powtarzania rekordów, aby zapewnić spójność relacji pod względem relacji między atrybutami sName i oName.

Mimo że relacja BranchStaffOwner znajduje się w BCNF, relacja pozostaje słabo ustrukturyzowana ze względu na nadmiarowość danych spowodowaną obecnością nietrywialnej MVD.

Wyraźnie potrzebujemy silniejszej formy BCNF, która zapobiega strukturom relacyjnym, takim jak relacja BranchStaffOwner.

Definicja – relacja, która jest w postaci BCNF i NIE zawiera nietrywialnych zależności wielowartościowych.

5NF

Kiedy rozkładamy relację na 2 relacje, wynikające z nich relacje mają własność złączenia bezstratnego.

Ta własność odnosi się do faktu, że możemy połączyć powstałe relacje, aby wytworzyć pierwotną relację.

Zdarzają się jednak przypadki, w których istnieje wymóg złożenia relacji na więcej niż 2 relacje.

Chociaż...

Zależność łączenia bezstratnego – właściwość dekompozycji, która zapewnia, że żadne fałszywe rekordy nie są generowane, gdy relacje są ponownie łączone za pomocą operacji łączenia naturalnego.

W dzieleniu relacji przez rzutowanie bardzo wyraźnie określamy sposób dekompozycji.

W szczególności staramy się używać rzutów, które można odwrócić, łącząc powstałe relacje, tak aby zrekonstruować pierwotną relację. Taka dekompozycja nazywana jest dekompozycją bezstratną (zwaną także bezstratną lub nieaddytywną), ponieważ...

(tabelki)

Definicja 5NF – relacja, która nie ma zależności złączeniowych

(tabelki z wyposażeniem)

Ponieważ relacja PropertyItemSupplier zawiera zależność sprzężenia, dlatego nie znajduje się w 5NF. Aby usunąć zależność łączenia, rozkładamy relację PropertyItemSupplier na 3 relacje 5NF,

a mianowicie relacje PropertyItem (R1), ItemSupplier(R2) i PropertySupplier (R3)

Mówimy, że relacja...