Bazy danych i systemy zarządzania

Wykład III

Relacyjny model danych

Pojęcie atrybutu

Dane reprezentowane w jakiejkolwiek formie, np. w pewnym pliku, arkuszu kalkulacyjnym, bazie danych, bazie wiedzy, itp. odnoszą się zawsze do pewnego *obiektu* lub *zbioru obiektów* i charakteryzują wybrane ich *własności* będące przedmiotem zainteresowania.

Dla danego zagadnienia, zazwyczaj na etapie projektu systemu informatycznego, dokonuje się wyboru tych własności oraz stopnia szczegółowości ich reprezentacji. Do opisu własności obiektów reprezentowanych w bazie danych wykorzystuje się wybrany zestaw *atrybutów*, a dla wyrażenia ich wartości z zadaną dokładnością określa się *dziedziny atrybutów*. W praktyce często dziedzinę atrybutu definiuje się poprzez podanie *typu* danych, *rozmiaru pola*, *więzów spójności* (lokalnych, np. reguły poprawności).

Przez **atrybut obiektu** rozumie się pewną jego własność, cechę lub charakterystykę, pozwalającą częściowo opisać ten obiekt. Przykładami atrybutów mogą być kolor, kształt, waga, cena, wiek, marka, typ, numer seryjny, itp. Atrybuty przyjmują wartości z określonego zbioru (dziedziny), przy czym normalnie (choć milcząco) zakłada się, że w danym momencie dla danego obiektu wybrany atrybut przyjmuje *pojedynczą wartość*; co więcej, przyjmuje się najczęściej, że jest to *wartość atomiczna*, tzn. taka, która traktowana jest jako pojedynczy symbol i nie jest dalej analizowana po kątem jej struktury wewnętrznej, zawartości, elementów składowych, itd.

Wymaganie co do atomicznego charakteru wartości atrybutów stanowi jeden z podstawowych kanonów relacyjnych baz danych; w bardziej zaawansowanych modelach może ono być jednak osłabione, poprzez dopuszczenie wartości złożonych, takich jak zbiory, przedziały, czy nawet struktury. Wymaganie atomicznego charakter wartości atrybutów uwarunkowane jest dążeniem do maksymalnej przejrzystości modelu relacyjnych baz danych i jego zgodności z teorią, a przede wszystkim dążeniem do uzyskania dużej efektywnoći operacji algebraicznych.

Wartości i dziedziny atrybutów

Każdy atrybut przyjmuje wartości z określonego zbioru, zadanego jawnie lub domyślnie, stanowiącego jego dziedzinę. Przyjmijmy następujące standardowe oznaczenia:

- $\mathbf{A} = \{A_1, A_2, \ldots, A_n\}$ – ustalony zbiór atrybutów,
- $D_1, D_2, \ldots, D_n$ – ustalony zbiór dziedzin tych atrybutów.

Z każdym atrybutem $A_i \in \mathbf{A}$ związana jest dziedzina D_i, $i = 1, 2, \ldots, n$.

Dziedzina atrybutu może być zadana jawnie, np. poprzez wyliczenie elementów w postaci skończonego zbioru (ekstensjonalnie), lub może być zadana niejawnie w przypadku trudności wyliczenia wszystkich potencjalnych elementów tej dziedziny. Przykładem pierwszej dziedziny może być zbiór wartości atrybutu *dzień_tygodnia* w postaci {*poniedziałek, wtorek, środa, czwartek, piątek, sobota, niedziela*}. Przykładem drugiej dziedziny może być zbiór wartości atrybutu *nazwisko* – trudno tutaj wyliczyć wszystkie możliwe napisy, które mogłyby być uznane za nazwiska; z drugiej strony, przyjęcie za dziedzinę tego atrybutu zbioru napisów o ograniczonej długości maksymalnej dopuszcza napisy typu „Aaaaa" czy „xyz", które trudno uznać za realne nazwiska. W takiej sytuacji dziedzinę zadaje się zazwyczaj częściowo, poprzez określenie typu i rozmiaru pola, oraz ewentualnie regułę poprawności.

Dany atrubut dla danego obiektu może przyjmować w danej chwili czasu pojedynczą wartość – mówimy tutaj o *funkcyjnym* charakterze zależności wartości atrybutu od wybranych parametrów (obiektu, chwili czasu, ewentualnie dodatkowych czynników). Dobrymi przykładami wartości atrybutów są np. wartości zmiennych opisujących stan i zachowanie się układów fizycznych, precyzyjne dane pomiarowe, wartości wielkości fizycznych, takich jak temperatura, prędkość, napięcie, natężenie, itp. Natomiast często nie mają tego charakteru gromadzone łącznie dane pochodzące z pewnych obserwacji, np. wartości charakteryzujące *objawy* występujące łącznie u chorego jak {*gorączka, duszności, bóle, wysypka*}, czy też nieformalne opisy tekstowe.

Rodzaje dziedzin

Dziedziny nominalne i częściowo uporządkowane

Dziedzina atrybutu może tworzyć pewną strukturę; typowe z nich to:

- *dziedziny binarne* – dziedzina taka zawiera dokładnie dwa elementy, np. 0 i 1 lub określenia *tak* oraz *nie*; dodatkowo, może być narzucona relacja porządkująca te elementy, np. $0 \leq 1$,
- *dziedziny trójwartościowe* – dziedzina taka zawiera dokładnie trzy elementy, np. $\{tak,\ nie,\ brak\_danych\}$, $\{TRUE,\ FALSE,\ NULL\}$ lub $\{-, 0, +\}$; może być określona relacja porządku, np. $- \leq 0 \leq +$,
- *dziedziny nominalne (nieuporządkowane)* – są to skończone (lub rzadziej nieskończone) zbiory nazw, które nie są powiązane ze sobą żadną relacją porządkującą; typowymi przykładami mogą być np. zbiory dopuszczalnych kolorów, kształtów, itp.; zbiory takie można jednak porządkować leksykograficznie,
- *dziedziny częściowo uporządkowane* – podobnie jak wyżej, zbiory skończone lub nieskończone z relacją częściowego porządku – takie dziedziny występują stosunkowo rzadko; przykładem może być zbiór wartości „mieszanych" typu $\{list\_zwykły,\ list\_ekspresowy,\ list\_polecony,\ paczka\_zwykła,\ paczka\_wartościowa,\ przesyłka\_kurierska\}$ stanowiących wartości atrybutu *rodzaj_przesyłki* z częściowym porządkiem (względem czasu dostarczenia) ustalonym przez relację $list\_zwykły \leq list\_ekspresowy$, $list\_ekspresowy \leq przesyłka\_kurierska$, $list\_polecony \leq list\_ekspresowy$, $list\_ekspresowy \leq przesyłka\_kurierska$,
- *dziedziny tworzące strukturę kraty* – dziedziny takie wraz z wprowadzoną relacją częściowego porządku tworzą strukturę kraty; typowym przykładem mogą być specyfikacje typów,

Rodzaje dziedzin

Dziedziny uporządkowane

- *dziedziny uporządkowane* – skończone lub nieskończone zbiory wartości uporządkowanych (liniowo), np. $\{nb,nm,ns,z,ps,pm,pb\}$, gdzie nb – duży ujemny, nm – średni ujemny, ns – mały ujemny, z – około zera, ps – mały dodatni, pm – średni dodatni, pb – duży dodatni, z uporządkowaniem $np \leq nm \leq ns \leq z \leq ps \leq pm \leq pb$; może to być też pewien zbiór napisów (imion, nazwisk, nazw własnych) z relacją porządku alfabetycznego,
- *dziedziny liczbowe* – ze względu na liczne zastosowania i specyficzny charakter wyróżniono je jako osobny rodzaj, choć w istocie należą do klasy zbiorów uporządkowanych – wyróżnia je jednak określona arytmetyka pozwalająca realizować operacje obliczeniowe, w tym wszelkiego rodzaju przeliczenia, obliczanie wartości funkcji, obliczanie funkcji sumarycznych (takich jak: suma, średnia, wariancja, odchylenie standardowe, itp.); przykłady obejmują dowolne podzbiory zbioru liczb naturalnych, całkowitych, czy rzeczywistych, a także daty i czas, liczby zapisane w innym niż dziesiętny układzie pozycyjnym, walutę, procenty, itp. Często spotykane typy danych to:
  - *bajt*,
  - *liczba całkowita* (krótka, długa),
  - *liczba rzeczywista* (pojedynczej, podwójnej precyzji),
  - *data*,
  - *waluta*,
  - p*rocenty*

Dla dziedzin tego typu istnieje zwykle możliwość konwersji typu.

Inne typy danych

Dane *specjalizowane* nie mają charakteru atomicznego, a ich zapis i interpretacja odbywa się przy pomocy specjalnych narzędzi. Ich przetwarzanie wymaga znajomości strutury oraz specjalizowanego oprogramowania, np.:

- *dane typu tekstowego* – mogą zawierać dowolne pliki tekstowe; w bazach danych określa się je jako tzw. pola typu *memo*, przy czym nie wykonuje się na nich żadnych operacji, poza obróbką za pomocą edytora tekstu,
- *dane typu dźwięk* – zapisywane są w postaci specjalnych plików, w bazach danych jedyne ich zastosowanie polega na odtwarzaniu nagranej sekwencji; specjalistyczna obróbka możliwa jest przy zastosowaniu odpowiednich narzędzi, przy czym może ona mieć na celu np. rozpoznanie mowy i przetłumaczenie do postaci zapisu w pewnym języku symbolicznym, albo np. rozpoznanie osoby na podstawie jej wypowiedzi,
- *dane typu obraz* – zapisywane są w postaci specjalnych plików w określonym formacie, ich zastosowanie polega na odtwarzaniu (prezentacji) obrazu; specjalistyczna obróbka możliwa jest przy zastosowaniu odpowiednich narzędzi, przy czym może ona mieć na celu analizę i rozpoznawanie obrazu, porównywanie obrazów, itp. (Istnieją specjalizowane bazy danych, w których możliwe jest wyszukiwanie na podstawie wzorca,
- *dane typu sekwencja wideo* – podobie jak obrazy,
- *dane typu hyperłącze* – umożliwiają uruchomienie usługi sieciowej,
- *wiedza jako dane* – wiedza, o ile stanowi przedmiot dalszej analizy czy przetwarzania (na wyższym poziomie), może również stanowić dane wyjściowe dla tego procesu; przykładem mogą być zbiory reguł lub programy logiczne, których własności, są przedmiotem dalszej analizy. Baza danych zawierająca zbiory procedur może też być wykorzystywana do wspomagania lub automatycznej syntezy programów w oparciu o dobór istniejących komponentów o podanych specyfikacjach.

Logiczny model danych

Formuły atomowe

Reprezentacji danych za pomocą atrybutów polega na wyspecyfikowaniu wartości każdego atrybutu przyjętego do opisu pewnego obiektu. Przyjmuje się, że określona jest funkcja ρ przypisująca te wartości, taka że:

$$\rho : \; E \times \mathbf{A} \longrightarrow D_{\mathbf{A}},$$

gdzie E jest zbiorem obiektów, $\mathbf{A} = \{A_1, A_2, \ldots, A_n\}$ – zbiorem atrybutów wybranych do opisu własności tych obiektów, a $D_{\mathbf{A}}$ – łączną dziedziną wszystkich atrybutów, tj. $D_{\mathbf{A}} = \cup_{i=1}^{n} D_i$; ponadto, przyjmuje się, że dla każdego atrybutu A_i, $i = 1, 2, \ldots, n$, $\rho(e, A_i) \in D_i$, gdzie $e \in E$ jest pewnym obiektem, $A_i \in \mathbf{A}$ – atrybutem, a D_i – dziedziną atrybutu A_i.

Zamiast pisać $\rho(e, A_i) = d$, gdzie $d \in D_i$ jest pewnym elementem dziedziny atrybutu A_i, stosowane są zapisy:

$$A_i(e) = d \tag{1}$$

$$< e, A_i, d > \tag{2}$$

Oba powyższe zapisy mówią, że wartość atrubutu A_i dla obiektu e jest równa d. W przypadku gdy dokładnie wiadomo o jaki obiekt chodzi (np. zajmujemy się pojedynczym obiektem, powyższe zapisy mogą przybrać postać uproszczoną:

$$A_i = d \tag{3}$$

$$< A_i, d > \tag{4}$$

$$[A_i, d] \tag{5}$$

Powyższe wyrażenia określa się jako *formuły atomowe* (atomiczne), *atomy* lub też kompleksy. Do obu form stosuje się również określenie *fakt*. Parę (A_i, d) nazywa się *deskryptorem* kub *selektorem*.

Logiczny model danych

Formuły atomowe – model ogólny

Niech:

- $E = \{e_1, e_2, \ldots, e_m\}$ oznacza zbiór m obiektów reprezentowanych w modelu,
- $\mathbf{A} = \{A_1, A_2, \ldots, A_n\}$ oznacza zbiór n atrybutów reprezentujących wybrane własności,
- $D_1, D_2, \ldots, D_n$ oznacza dziedziny powyższych atrybutów.

Ogólna postać atomu przybiera formę:

$$A_j(e_i) = d_{ij}, \tag{6}$$

kompleksu – formę:

$$< e_i, A_j, d_{ij} > \tag{7}$$

a selektora formę:

$$[A_j(e_i) = d_{ij}] \tag{8}$$

W wyrażeniach (6), (7), (8) d_{ij} jest wartością j-tego atrybutu dla i-tego obiektu, $i = 1, 2, \ldots, m$, $j = 1, 2, \ldots, n$.

Powyższy sposób reprezentacji danych, a także w ogólniejszym przypadku – wiedzy, określa się także jako *O-A-V* (od ang. *Object-Attribute-Value*). Model ten może zostać zawężony do postaci *A-V*, zawierającej jedynie parę atrybut-wartość lub też rozszerzony o dodatkowe parametry. Typowym przykładem rozszerzenia jest uzupełnienie o tzw. *współczynnik wiarygodności* (ang. *Certainty Factor;* przybiera on wówczas postać *O-A-V-CF*.

Logiczny model danych

Przy pomocy formuł atomowych można konstruować bardziej złożone wyrażenia reprezentujące określone dane. W szczególności mogą to być formuły opisujące wartości wszystkich atrybutów z zadanego zbioru dla określonego obiektu. Formuły takie odpowiadają logicznie *koniunkcji* atomów, z których każdy opisuje wartość pojedynczego atrybutu.

Definicja 1 *Formułę postaci*

$$\phi_i = [A_1(e_i) = d_{i,1}] \wedge [A_2(e_i) = d_{i,2}] \wedge \ldots \wedge [A_n(e_i) = d_{i,n}] \tag{9}$$

nazywamy (pełną) formułą (prostą) opisującą własności obiektu e_i.

Określenie *formuła pełna* odnosi się do specyfikacji wartości wszystkich atrybutów. Jeżeli w danej formule nie wszystkie wartości atrybutów są sprecyzowane, to formuła taka nie jest formułą pełną. Określenie *formuła prosta* odnosi się do formuł o postaci koniunkcji atomów. Jeżeli wiadomo jaki obiekt jest przedmiotem opisu, specyfikację tego obiektu można pominąć; formuła (9) przyjmie wówczas postać $\phi = [A_1 = d_1] \wedge [A_2 = d_2] \wedge \ldots \wedge [A_n = d_n]$.

Jeżeli danych jest m obiektów $e_1, e_2, \ldots, e_m$ opisywanych tymi samymi atrybutami $A_1, A_2, \ldots, A_n$, to łączny opis tych obiektów (formuła opisująca relację) może przyjąć postać dysjunkcji wszystkich pełnych formuł prostych

$$\Phi = \phi_1 \vee \phi_2 \vee \ldots \vee \phi_m. \tag{10}$$

Zapis (10) jest zapisem logicznym; z uwagi na obszerność tej fomuły oraz jej strukturalny charakter wygodnie jest prezentować taką formułę w postaci tabelarycznej, w odpowiednio skonstruowanej tabeli, gdzie poszczególne wiersze zawierają pełne opisy obiektów a kolumny odpowiadają wartościom danego atrybutu dla wszystkich obiektów.

Logiczny model danych

Postać tabelaryczna

Logiczną formułę 10 opisującą informację w bazie danych można przedstawić w postaci tabelarycznej:

$$\begin{array}{cccc} [A_1(e_1) = d_{1,1}] & [A_2(e_1) = d_{1,2}] & \ldots & [A_n(e_1) = d_{1,n}] \\ [A_1(e_2) = d_{2,1}) & [A_2(e_2) = d_{2,2}] & \ldots & [A_n(e_2) = d_{2,n}] \\ . & . & \ldots & . \\ . & . & \ldots & . \\ . & . & \ldots & . \\ [A_1(e_m) = d_{m,1}] & [A_2(e_m) = d_{m,2}] & \ldots & [A_n(e_m) = d_{m,n}], \end{array} \tag{11}$$

W schemacie podstawowym przyjmuje się, że:

- obiekty są jednorodne, tzn. wszystkie atrybuty są stosowalne do opisu każdego obiektu,
- wszystkie wartości dowolnego atrybutu mogą być zastosowane do opisu każdego obiektu (chociaż oczywiście nie muszą być, w danej chwili, prawdziwe),
- żaden obiekt nie wymaga charakteryzowania poprzez dodatkowe atrybuty.

Założenia te mogą być w praktyce osłabiane.

Reprezentacja danych w modelu relacyjnym: tabele

W relacyjnych bazach danych informacja zapisywana jest w tabelach. Pierwowzór takiej tabeli może być postrzegany jako odpowiednio zaaranżowana przestrzennie formuła (11).

Interpretacja informacji w tablicy: każdy wiersz (krotka) odpowiada specyfikacji wartości wszystkich atrybutów dla danego obiektu, poszczególne wiersze zawierają opisy kolejnych obiektów, a w dowolnej kolumnie umieszczone są wartości pojedynczego atrybutu dla kolejnych obiektów. Niech:

- $A_1, A_2, \ldots, A_n$ będą wybranymi atrybutami o dziedzinach odpowiednio
- $D_1, D_2, \ldots, D_n$; ponadto niech
- $E = \{e_1, e_2, \ldots, e_m\}$ będzie zbiorem pewnych obiektów opisywanych wskazanymi wyżej atrybutami. Reprezentacja tablicowa opisu wszystkich obiektów zbioru E może mieć następującą postać:

E	A_1	A_2	$\ldots$	A_j	$\ldots$	A_n
e_1	$d_{1,1}$	$d_{1,2}$	$\ldots$	$d_{1,j}$	$\ldots$	$d_{1,n}$
e_2	$d_{2,1}$	$d_{2,2}$	$\ldots$	$d_{2,j}$	$\ldots$	$d_{2,n}$
$\vdots$	$\vdots$	$\vdots$		$\vdots$		$\vdots$
e_i	$d_{i,1}$	$d_{i,2}$	$\ldots$	$d_{i,j}$	$\ldots$	$d_{i,n}$
$\vdots$	$\vdots$	$\vdots$		$\vdots$		$\vdots$
e_m	$d_{m,1}$	$d_{m,2}$	$\ldots$	$d_{m,j}$	$\ldots$	$d_{m,n}$

(12)

Powyższy sposób reprezentacji jest oszczędniejszy niż poprzedni: zarówno nazwa każdego atrybutu jak i obiektu jest wypisana tylko jeden raz. Równocześnie, reprezentacja taka wydaje się być bardziej przejrzysta i naturalna. Ma ona jednak *model logiczny* (semantykę) i może być interpretowana i przetwarzana logicznie. Może też być oceniana wartość logiczna tablicy jako formuły i jej elementów składowych (wierszy).

Reprezentacja danych w modelu relacyjnym: relacje

Z algebraicznego punktu widzenia, tabela przedstawiająca dane reprezentuje pewną relację R_E określoną w iloczynie kartezjańskim $E \times D_1 \times D_2 \times \ldots \times D_n$:.

$$R_E \subseteq E \times D_1 \times D_2 \times \ldots \times D_n.$$

R_E jest relacją $n+1$ argumentową, przy czym poszczególne argumenty krotek relacji odpowiadają elementom zbioru E oraz wartościom kolejnych atrybutów.

W rzeczywistości, obiekty $e_1, e_2, \ldots, e_n$ istnieją w świecie rzeczywistym; w modelu są one identyfikowane poprzez pewien wyróżniony atrybut (atrybuty), będący nazwą własną danego obiektu, jego identyfikatorem symbolicznym, numerem kolejnym, etc. Powinien to być identyfikator *jednoznaczny.*

Tablicowa reprezentacja relacji przyjmuje ostatecznie jednorodną postać

A_1	A_2	$\ldots$	A_j	$\ldots$	A_n
$d_{1,1}$	$d_{1,2}$	$\ldots$	$d_{1,j}$	$\ldots$	$d_{1,n}$
$d_{2,1}$	$d_{2,2}$	$\ldots$	$d_{2,j}$	$\ldots$	$d_{2,n}$
$\vdots$	$\vdots$		$\vdots$		$\vdots$
$d_{i,1}$	$d_{i,2}$	$\ldots$	$d_{i,j}$	$\ldots$	$d_{i,n}$
$\vdots$	$\vdots$		$\vdots$		$\vdots$
$d_{m,1}$	$d_{m,2}$	$\ldots$	$d_{m,j}$	$\ldots$	$d_{m,n}$

(13)

Powyższa tablica reprezentuje pewną relację R, gdzie:

$$R \subseteq D_1 \times D_2 \times \ldots \times D_n. \tag{14}$$

W tabeli (13) (relacja R jest zadana *ekstensjonalnie.* Jeżeli opis każdej encji jest unikalny (zapewniona jest jednoznaczna identyfikacja), to $R = \pi_{2,3,\ldots,n+1}(R_E)$

Schemat relacji

Zgodnie z matematyczną definicją relacja R jest pewnym zbiorem n-krotek postaci $(d_1, d_2, \ldots, d_n)$, tzn.

$$R \subseteq \{(d_1, d_2, \ldots, d_n) : d_1 \in D_1, d_2 \in D_2, \ldots, d_n \in D_n\}.$$

W powyższym zapisie zastosowane do opisu atrybuty nie są uwidocznione. Aby móc odczytać znaczenie poszczególnych elementów składowych każdej krotki należy znać nie tylko zbiór wybranych atrybutów, ale także ich przypisanie poszczególnym składowym krotek (tzn. kolejność atrybutów). W tym celu definiuje się pojęcie *schematu relacji.*

Definicja 2 *Schematem relacji R o danych atrybutach $A_1, A_2, \ldots, A_n$, takiej że $R \subseteq D_1 \times D_2 \times \ldots \times D_n$ nazywamy* **ciąg** $(A_1, A_2, \ldots, A_n)$. *W celu jawnej specyfikacji schematu relacji R piszemy $R(A_1, A_2, \ldots, A_n)$.*

Wyrażenie $R(A_1, A_2, \ldots, A_n)$ czytamy „relacja R o schemacie $A_1, A_2, \ldots, A_n$" lub też „relacja R ma schemat $A_1, A_2, \ldots, A_n$". Znajomość schematu relacji pozwala dokonać właściwej interpretacji jej elementów, odczytać znaczenie poszczególnych argumentów krotek tworzących tą relację. W celu jawnej specyfikacji dziedzin atrybutów danej relacji jej schemat zapisuje się też jako $R(A_1 : D_1, A_2 : D_2, \ldots, A_n : D_n)$.

Uwaga:
Wielu Autorów podręczników z zakresu baz danych definiuje schemat relacji jako *zbiór* atrybutów $\{A_1, A_2, \ldots, A_n\}$; podejście takie wydaje się niewystarczające dla właściwej interpretacji relacji reprezentowanej jako zbiór krotek – dla dowolnej relacji n-argumentowej istniałoby wówczas $n!$ możliwych interpretacji jej argumentów.

Własności relacji

Tablice reprezentujące relacje posiadają następujące własności:

- liczba kolumn jest z góry ustalona (na etapie projektowania relacyjnej bazy danych) i jest taka sama dla wszystkich wierszy (krotek),
- kolumny etykietowane są atrybutami; z każdą kolumną związany jest dokładnie jeden atrybut (o określonej dziedzinie),
- liczba wierszy zależy od liczby opisywanych obiektów i jest dowolna (można zdefiniować pustą relację o zadanym schemacie),
- kolejność kolumn nie jest w zasadzie istotna, o ile przypisze się im odpowiednie atrybuty (dotyczy to tabeli; w relacji będącej zbiorem krotek, których elementy nie są etykietowane atrybutami, nie wolno zmieniać kolejności argumentów),
- kolejność wierszy nie jest istotna; relacja jest *zbiorem* krotek,
- na przecięciu wierszy i kolumn znajdują się wartości atomiczne będące elementami dziedziny właściwej dla atrybutu przypisanego danej kolumnie,
- wszystkie wiersze tabeli reprezentującej relację są różne (różnią się przynajmniej na jednej pozycji – dwa identyczne wiersze nie mogą się powtarzać w tabeli, o ile jest ona zapisem relacji); w rzeczywistej tabeli mogą wystąpić identyczne wiersze, tzw. *duplikaty* (o ile nie określono klucza ani indeksu jednoznacznego).

Wymaganie dotyczące rozróżnialności rekordów jest konsekwencją definicji relacji jako zbioru; w tabeli stanowiącej reprezentację relacji żadna krotka nie może wystąpić dwa lub więcej razy. W konsekwencji, żaden wiersz nie powinien się powtórzyć.

Problem duplikatów

W realnych bazach danych gdzie tabele zapisywane są w pewnych plikach, nie ma w zasadzie przeszkód aby pewne wiersze powtarzały się. Jeżeli przy definiowaniu schematu relacji (atrybutów i formatu tabeli) nie zdefiniuje się żadnego *klucza*, ani nie zażąda się aby rekordy były *indeksowane unikatowo*, to wprowadzenie dwóch lub więcej identycznych wierszy do tabeli jest możliwe, a nawet czasem celowe; co więcej, tabele takie mogą się pojawić w wyniku realizacji pewnych operacji algebraicznych na istniejących już tabelach bez powtórzeń, np. w wyniku zastosowania operacji projekcji. Powtarzające się wiersze tabeli nazywa się *duplikatami*. Powoduje to jednak pewne określone konsekwencje, które należy uwzględnić już na etapie projektu.

- obiekty opisywane takimi wierszami nie są rozróżnialne, a więc celowość reprezentowania więcej niż jednego takiego obiektu jest wątpliwa,
- usuwając informację o takim obiekcie z bazy należy wyraźnie sprecyzować, czy chodzi nam o usunięcie jednego jej wystąpienia, dwóch, trzech, itd., czy też wszystkich,
- mogą pojawić się trudności związane z przechowywaniem takich tabel, jeżeli dostęp do ich elementów (wierszy) realizowany jest za pomocą indeksu unikatowego,
- w wyniku realizacji pewnych operacji na takich tabelach (np. złożenia lub iloczynu kartezjańskiego) następuje niekontrolowane i nie zawsze celowe powielanie informacji,
- niektóre operacje na tabelach (np. łączenie) wymagają aby tzw. tabela główna (tabela nadrzędna) była indeksowana jednoznacznie względem pól łączących (lub aby pola te definiowały klucz) i bez spełnienia tego warunku operacje te nie mogą być realizowane; dlatego w większości przypadków, już na etapie projektowania tabel bazy danych, definiuje się dla każdej tablicy klucz podstawowy.

Problem jednoznacznej identyfikacji encji

Jeżeli żaden z atrybutów występujących w schemacie tablicy nie pozwala na jednoznaczną identyfikację obiektów, to zwykle wprowadza się (sztucznie) nowy atrybut jednoznacznie identyfikują każdy obiekt (identyfikator).

Przykładami takich identyfikatorów mogą być: numer PESEL, numer NIP, identyfikator pracownika w zakładzie pracy, sygnatura książki w bibliotece, numer rejestracyjny samochodu, itp. Wszystkie one mają pewną istotną własność, a mianowicie w sposób *jednoznaczny* identyfikują dany obiekt. Formalnie, jeżeli A jest takim wybranym atrybutem o dziedzinie D_A, to musi istnieć wzajemnie jednoznaczna *funkcja identyfikująca* μ, postaci:

$$\mu : \ E \longrightarrow D_A. \tag{15}$$

Warunek (15) oznacza, że każdy obiekt ze zbioru E może być reprezentowany przez pewien identyfikator, przy czym żądanie wzajemnej jednoznaczności funkcji identyfikującej czyni zadość wymaganiu *rozróżnialności* obiektów, tzn. mając daną wartość funkcji jesteśmy w stanie jednoznacznie zidentyfikować obiekt. Oczywiście nie wszystkim potencjalnym identyfikatorom muszą być w danej chwili przypisane relanie istniejące obiekty; zazwyczaj definicja dziedziny atrybutu identyfikującego jest tak skonstruowana, aby zawsze można było opisać nowe obiekty, jeżeli takie pojawią się i zaistnieje potrzeba ich reprezentacji w bazie danych.

Jeżeli w danym zestawie atrybutów nie istnieje (nie jest wprowadzony) żaden pojedynczy atrybut czyniący zadość wymaganiu jednoznacznej identyfikacji, można wybrać odpowiedni zestaw atrybutów $A^1, A^2, \ldots, A^k \in \{A_1, A_2, \ldots, A_n\}$ spełniających żądanie jednoznacznej identyfikacji. W takim przypadku funkcja identyfikująca przybiera postać:

$$\mu : \ E \longrightarrow D^1 \times D^2 \times \ldots \times D^k, \tag{16}$$

gdzie, $D^1, D^2, \ldots, D^k$ są odpowiednio dziedzinami atrybutów $A^1, A^2, \ldots, A^k$ wchodzących w skład identyfikatora.

Pojęcie klucza

Pojęcie klucza i definicje pochodne

- **Zbiór identyfikujący:** dowolny atrybut lub zestaw atrybutów pozwalających na jednoznaczną identyfikację obiektu w bazie danych.
- **Klucz:** każdy minimalny zbiór identyfikujący, tzn. taki, że żaden jego właściwy podzbiór nie jest wystarczający do jednoznacznej identyfikacji obiektu; klucz stanowi więc możliwie najprostszy zestaw atrybutów wystarczający do rozróżnienia wszystkich obiektów,
- **Klucz prosty:** klucz złożony z pojedynczego atrybutu,
- **Klucz złożony:** klucz składający się z więcej niż jednego atrybutu,
- **Klucz podstawowy:** wybrany klucz preferowany przez użytkownika; nazywa się go też lub *kluczem głównym*,
- **Nadklucz:** każdy zbiór atrybutów zawierający w sobie pewien klucz,
- **Podklucz:** dowolny właściwy podzbiór klucza.
- **Klucz obcy:** atrybut lub ich zestaw występujący w danej tabeli a będący kluczem określony w innekj tabeli, stosowany w celu identyfikacji elementów dla ewentualnego złączenia; klucz obcy nie musi być kluczem w tabeli dołączanej,
- **Atrybut kluczowy:** atrybut wchodzący w skład przynajmniej jednego klucza.

Klucz jest określany dla schematu relacji; co jest kluczem a co nie nie zależy od aktualnej zawartości tabeli. Pojęcie klucza jest związane z istnieniem *zależności funkcyjnych* w tabelach. Dana tabela może mieć wiele kluczy. Wyznaczenie wszystkich kluczy stanowi istotne zadanie przy normalizacji.

Realacje, tabele logiczne, tabele fizyczne

Relacje a tabele logiczne

Tabela logiczna (wirtualna) stanowi (wyidealizowaną) reprezentację relacji; może ona jednak różnić się od relacji definiowanej matematycznie:

- Dziedzina relacji (ze względu na i-ty argument; zadawana ekstensjonalnie) reprezentowanej w tabeli jest zazwyczaj podzbiorem dziedziny D_i i-tego atrybutu (zadawanej intensjonalnie),
- Dla relacji zawsze istnieje klucz; dla tabeli – niekoniecznie,
- Tabela może zawierać duplikaty (niedopuszczalne w relacji),
- W tabeli można zmieniać kolejność kolumn (przy zachowaniu etykietowania atrybutami),
- Istnieje dwoista terminologia (np. krotka – rekord, relacja – tabela).

Tabele logiczne a tabele fizyczne

Tabela fizyczna (pamiętana) stanowi zapis tabeli logicznej; może się od niej jednak istotnie różnić:

- rekord logiczny może być różny of rekordu fizycznego (przetwarzanie danych, pola wyliczane),
- dane mogą być kodowane,
- dane często mają wymiar (pomijany w tabeli fizycznej),
- dla każdego zagadnienia można zbudować wiele reprezentacji fizycznych (różna liczba i schematy tabel).

Problem wartości *NULL*

Wartości *NULL* to tzw. wartości puste (albo zerowe); reprezentują one brak wartości danego atrybutu (chwilowy lub permanentny), który może być interpretowany jako:

- brak znajomości wartości danego atrybutu (dane niedostępne, wartość nieznana; np. data urodzenia, numeru NIP czy PESEL),
- niemożliwość wyznaczenia wartości danego atrybutu w rekordzie (atrybut niestosowalny; np. pojemność skokowa dla samochodów elektrycznych),
- niepewność co do istnienia tej wartości (np. gdy nie wiadomo, czy osoba posiada telefon).

Wartości *NULL* są różne od spacji, zera czy też stringu pustego. W pewnych okolicznościach, wartości *NULL* są traktowane jak każde inne (np. wyświetalanie, porządkowanie; w innych okolicznościach nie podlegają przetwarzaniu (dają nieokreślone wartości funkcji, dwie wartości *NULL* nie są traktowane jako równe – brak połączenia). Porównanie wartości *NULL* z dowolną wartością daje wartość logiczną *UNKNOWN* lub *NULL* (różną of *TRUE* i *FALSE*). Prowadzi to do logiki trówartościowej:

AND	*TRUE*	*FALSE*	*NULL*
TRUE	*TRUE*	*FALSE*	*NULL*
FALSE	*FALSE*	*FALSE*	*FALSE*
NULL	*NULL*	*FALSE*	*NULL*

OR	*TRUE*	*FALSE*	*NULL*
TRUE	*TRUE*	*TRUE*	*TRUE*
FALSE	*TRUE*	*FALSE*	*NULL*
NULL	*TRUE*	*NULL*	*NULL*

NOT	*TRUE*	*FALSE*	*NULL*
TRUE	*FALSE*	*TRUE*	*NULL*

Operatory *IS NULL* oraz *IS NOT NULL* pozwalają sprawdzić, czy dana wartość jest *NULL*. Użycie wartości *NULL* może prowadzić do niezamierzonych efektów (np. $0 \cdot x = 0$ dla dowolnej liczby x, ale $0 \cdot NULL = NULL$).

Sortowanie i indeksowanie

Sortowanie

Operacja *sortowania* oznacza uporządkowanie rekordów w tabeli względem jednego lub kilku kryteriów (ustalonych hierarchicznie). Najczęściej ma miejsce *uporządkowanie leksykograficzne* (alfabetyczne).

Porządek leksykograficzny:

$$X_1X_2\ldots X_k < Y_1Y_2\ldots Y_m$$

wtw. gdy

- $k < m$ oraz $X_1X_2\ldots X_k = Y_1Y_2\ldots Y_k$, albo
- dla pewnego $i \leq min(k,m)$ $X_1X_2\ldots X_{i-1} < Y_1Y_2\ldots Y_{i-1}$ oraz $X_i < Y_i$.

Sortowanie może odbywać się według jednego lub kilku pól; *ASC* oznacza porządek rosnący (default), *DESC* – porządek malejący. Uwaga: porządkowanie liczb zadeklarowanych jako tekst daje niezamierzony efekt.

Indeksowanie

Indeksy to pomocnicze struktury danych pozwalające na szybkie odszukiwanie rekordów o zadanych wartościach atrybutów. Indeksy zazwyczaj mają strukturę hierarchiczną (blokową) i są realizowane za pomocą tzw. B-drzew. Użycie indeksów znakomicie przyspiesza wyszukiwanie informacji oraz operacje na danych, ale ich obsługa (aktualizacja) jest pracochłonna.

Indeksy są implementowane tak, że każdy wierzchołek B-drzewa zajmuje blok na dysku (4096 bajtów), a więc może zawierać kilkaset wskaźników do węzłów potomnych (dla typowego wyszukania wymagane jest np. tylko 3 dostępy do dysku).

Indeksy mogą być pojedyncze (oparte na jednym atrybucie) oraz wielokrotne (dla kilku atrybutów). Można je deklarować jako unikatowe (bez powtórzeń) lub jako dopuszczające powtórzenia.

Problemy modelowania danych

Dane zawarte w bazie danych powinny wiernie oddawać stan rzeczywisty modelowanego systemu, tzn. powinny one być zgodne z rzeczywistością. W praktyce modelowanie zbiorów encji w relacyjnych bazach danych napotyka szereg problemów teoretycznych i praktycznych. Najważniejsze z nich to:

1. **Problem zupełności**.
   (a) *Zupełność fizyczna:* czy wszystkie encje modelowaneo świata zostały opisane w bazie (czy nic nie zostało pominięte)?
   (b) *Zupełność logiczna:* czy wyspecyfikowano wszystkie rekordy pokrywające zadany (wymagany) obszar uniwersum U ?
2. **Problem rozróżnialności.**
   (a) *Rozróżnienie fizyczne:* czy wszystkie różne encje reprezentowane są różnymi rekordami?
   (b) *Rozróżnialność potencjalna:* czy wybrany schemat relacji (atrybuty) pozwala rozróżnić wszystkie encje na zadanym poziomie abstrakcji (wyróżnić wszystkie wymagane klasy abstrakcji)?
3. **Problem poprawności (fizycznej realizowalności).**
   (a) *Realizowalność fizyczna (poprawność potencjalna):* czy wszystkie rekordy opisują fizycznie realizowalne (potencjalnie) encje?
   (b) *Poprawność:* czy wszystkie rekordy opisują realnie istniejące encje?
4. **Spójność danych.**
   (a) *Spójność zewnętrzna (administracyjna):* czy aktualny stan bazy danych dokładnie odzwierciedla aktualny stan systemu ?
   (b) *Spójność wewnętrzna:* czy spełnione są wszystkie ograniczenia, tzw. *więzy spójności*, nałożone na schemat i instancję bazy danych?

Więzy integralności

Dla zapewnienia (względnej) poprawności danych konstruuje się różnego rodzaju *więzy integralności* i nakłada je na strukturę (schemat) oraz realizację (instancję) bazy danych. Typowe więzy spójności obejmują:

Lokalne więzy spójności:

- definiowanie dziedzin atrybutów (typu, długości),
- nakładanie ograniczeń na wartości atrubutu w rekordzie (reguły poprawności operujące na pojedynczym polu, np. `Cena >= 0`),
- nakładanie łącznych ograniczeń na wartości atrubutó w rekordzie (reguły poprawności operujące na polach każdego pojedynczego rekordu, np. `Cena_kupna < Cena_sprzdaży`),
- definiowanie masek wprowadzania oraz formatów wyprowadzania danych.

Globalne więzy spójności:

- niedopuszczenie powtarzania się wartości wybranego pola (lub grupy pól) poprzez definiowanie klucza lub indeksu bez powtórzeń,
- definiowanie więzów referencyjnych (wymuszanie więzów integralności; wartości klucza obcego muszą występować w tablicy nadrzędnej (lub być *NULL*, wymagana jest zgodność typów przy złączeniach,
- globalne ograniczenia funkcyjne (weryfikowane okresowo) (np. `Stan_magazynu = Stan_poprzedni - Suma_sprzedza.zy + Suma_dostaw`),
- globalne ograniczenia numeryczne (np. liczby wybranych rekordów), algebraiczne (np. zawieranie się wybranych zbiorów) i logiczne (spełnianie wybranych formuł definiujących pożądane własności).

Przykładowe własności pól: ACCESS

W systemie ACCESS można zdefiniować następujące własności pól tabeli:

- **Rozmiar pola** – określa maksymalny rozmiar danych przechowywanych w polu (tekst i liczba),
- **Nowe wartości** – tylko dla *autonumer*; można wybrać generowanie przyrostowe lub losowe,
- **Format** – definiuje szablon wyświetlania zawartości danego pola,
- **Miejsca dziesiętne** – definiuje ilość miejsc po przecinku (liczby),
- **Maska wprowadzania** – definiuje szablon dla wprowadzania danych,
- **Tytuł** – pozwala podać etykietę kolumny tabeli,
- **Wartość domyślna** – pozwala zdefiniować standardową wartość wprowadzaną automatycznie,
- **Reguła poprawności** – definiuje warunek, jaki muszą spełniać wszystkie wartości w danym polu,
- **Komunikat o błędzie** – pozwala zdefiniować tekst wyświetlany przy naruszeniu reguły poprawności,
- **Wymagane** – określa czy dane pole musi zostać wypełnione, czy też może pozostać puste,
- **Zerowa długość dozwolona** – pozwala zapisać w polu łańcuch zerowej długości (zamienia puste pola tekstowe na łańcuchy zerowej długości),
- **Indeksowane** – pozwala wyspecyfikować żądanie indeksowania pola (możliwe jest indeksowanie z powtórzeniami lub bez),
- **Odnośnik** – pozwala wybrać typ formantu (dla typów tekst, liczba, logiczne) jako listę, pole wyboru, lub listę rozwijalną (pole kombi) (osiągalne przez zakładkę; może być tworzone przy pomocy kreatora).

Format pola: ACCESS

Dla pól tekstowych:

<Szablon>;<String zerowej długości>;<Null>

- *<Szablon>* – sekcja definiująca sposób wyprowadzania tekstu,
- *<String zerowej długości>* - sekcja definiująca co zostanie wyprowadzone w przypadku, gdy pole zawiera string zerowej długości (""),
- *<Null>* – sekcja definiująca co zostanie wyprowadzone w przypadku, gdy pole zawiera wartość *Null*.

Na przykład:

```
@;"Brak telefonu"[Czerwony];"Telefon nieznany"[Niebieski]
```

spowoduje wypisanie numeru telefonu zawartego w polu i odpowiednich komunikatów w przypadku jego braku lub nieznajomości.

Dla liczb obowiązuje format:

<Szablon dodatnich>;<Szablon ujemnych>;<Szablon zera>;<Null>

- *<Szablon dodatnich>* – definiuje sposób wyprowadzania liczb dodatnich,
- *<Szablon ujemnych>* – definiuje sposób wyprowadzania liczb ujemnych,
- *<Szablon zera>* – definiuje, co będzie wypisane gdy w polu jest wartość zero,
- *<Null>* – definiuje, co będzie wypisane gdy pole jest puste.

Na przykład:

```
0,00;"Ujemna cena!"[Czerwony];"Zero"[Zielony];"Brak"[Niebieski]
```

wyświetli cenę z dokładnością do dwóch miejsc po przecinku, oraz wskazane komunikaty w sytuacjach wyjątkowych.

Format danych: ACCESS

Znaczenie symboli

- `"TEKST"` – wyświetla podany w cudzysłowach tekst,
- `(spacja)` – wyświetla spację,
- `!` – wymusza lewostronne wyrównanie danych w polu,
- `*` – wypełnia wole pole podanym znakiem,
- `\` – powoduje wyświetlenie następującego po nim znaku,
- `[kolor]` – powoduje wyświetlanie zawartości pola w zadanym kolorze (Black, Blue, Green, Cyan, Red, Magenta, Yellow, White),
- `,` – separator części dziesiętnej,
- `0` – cyfra lub zero; wystąpienie obowiązkowe,
- `#` – cyfra lub spacja,
- `%` – wyświetla liczbę w postaci procentu,
- `E-, e-, E+, e+` – notacja ekponencjalna (dla liczb),
- `@` – cały tekst, pojedynczy znak lub spacja,
- `&` – pojedynczy znak lub nic,
- `>` – powoduje wypisanie tekstu dużymi literami,
- `<` – powoduje wypisanie tekstu małymi literami,

Dla typu *data* istnieje szereg symboli specjalnych (vide: Help).

Maska wprowadzania: ACCESS

Schemat i przykłady masek

Maska wprowadzania stanowi szablon i filtr użytczny przy wprowadzaniu danych do pola. Schemat maski wprowadzania jest następujący:

<Szablon maski>;*<Przechowanie znaku>*;*<Znak maski>*

- *<Szablon maski>* – definiuje postać maski,
- *<Przechowanie znaku>* – 0 oznacza, że znaki maski mają być przechowywane w tabeli; 1 oznacza, że znaki maski nie będą przechowywane w tabeli (standardowo 1),
- *<Znak maski>* – znak reprezentujący w masce pojedynczy symbol (standardowo _.

Na przykład, dla przechowywania numerów ISBN moż posłużyć się maską:

```
ISBN 00\-00000\-00\-0;1;\_
```

a dla numerów telefonów można posłużyć się maską:

```
(9\-99) 000\-00\-00!;0;\_
```

Maska wprowadzania może pełnić rolę (i) szablonu, (ii) filtru poprawności oraz (iii) być zastosowana do przekształcania (formatowania) wprowadzanych symboli.

Typowe maski wprowadzania dogodnie jest tworzyć przy wykorzystaniuu *Kreatora masek.*

Maska wprowadzania: ACCESS

Znaczenie symboli używanych w maskach wprowadzania:

- `0` – cyfra, pozycja wymagana,
- `9` – cyfra lub spacja, pozycja nie wymagana,
- `#` – cyfra lub spacja, pozycja nie wymagana (puste miejsca konwertowane na spacje, dozwolone + i -),
- `L` – litera, pozycja wymagana,
- `?` – litera, pozycja opcjonalna,
- `A` – litera lub cyfra, pozycja wymagana,
- `a` – litera lub cyfra, pozycja opcjonalna,
- `&` – dowolny znak lub spacja, pozycja wymagana,
- `C` – dowolny znak lub spacja, pozycja opcjonalna,
- `!` – wyświetlanie znaków od prawej do lewej,
- `Password|Hasło` – powoduje utajnienie wprowadzanych znakó (są zastąpione $*$),
- `>` – powoduje, że wprowadzane za tym symbolem znaki będą konwertowane na duże,
- `<` – powoduje, że wprowadzane za tym symbolem znaki będą konwertowane na małe,
- `.,:;-/` – znak dziesiętny, separatory tysięcy, daty i czasu; zależne od ustawiń w *Panelu sterowania*,
- `\` – literalne cytowanie następnego znaku.