Zajebiste odpowiedzi typie

makaki są sztos

Sieć Aptek

i inne małpy

June 25, 2022

Spis treści

Rozdział I

Pytania - dr. hab. Grzegorz Wójcik

1 Bazy danych

1.1 Model relacyjny baz danych i języki zapytań.

Model relacyjny baz danych

Relacyjny model danych pojawił się po raz pierwszy w artykule naukowym Edgara Codda w 1970 roku. W terminologii matematycznej - baza danych jest zbiorem relacji. Stąd historycznie pochodzi nazwa relacyjny model danych i relacyjna baza danych. W matematyce definiuje się relację jako podzbiór iloczynu kartezjańskiego zbiorów wartości. Reprezentacją relacji jest dwuwymiarowa tabela złożona z kolumn i wierszy.

Założenia modelu relacyjnego:

- Liczba **kolumn/atrybutów/pól (synonimy)** jest z góry ustalona.
- Z każdą kolumną jest związana jej nazwa (np. FirstName) oraz dziedzina (np. TEXT(20)), określająca zbiór wartości, jakie mogą wystąpić w kolumnie.
- Na przecięciu **wiersza/krotki/rekordu (synonimy)** i kolumny znajduje się pojedyncza (atomowa) wartość należąca do dziedziny kolumny.
- Wiersz reprezentuje jeden rekord informacji np. osobę.
- W modelu relacyjnym abstrahujemy od kolejności wierszy (rekordów) i kolumn (pól w rekordzie).

Klucz główny: dla każdej tabeli musi być określony klucz główny, będący jednoznacznym identyfikatorem. Może to być jedna lub więcej kolumn, w których wartości jednoznacznie identyfikują cały wiersz. Klucz główny w tabeli może być tylko jeden.
Klucz jednoznaczny ma te same właściwości co klucz główny, ale ich może być w tabeli więcej niż jeden.
Klucz obcy - jedna lub więcej kolumn, których wartości występują również jako klucz główny/jednoznaczny w tej samej/innej tabeli i są interpretowane jako wskaźniki do wierszy w tej drugiej tabeli

Dwanaście postulatów Codda – jest to zestaw 13 zasad stworzonych przez Edgara F. Codda – pioniera relacyjnych baz danych. Każda relacyjna baza danych musi je spełniać:

- System musi być kwalifikowany jako relacyjny, jako baza danych i jako system zarządzania.
- **Postulat informacyjny** – dane są reprezentowane jedynie przez wartości atrybutów w wierszach tabel (w krotkach).
- **Postulat dostępu** – każda wartość w bazie danych jest dostępna poprzez podanie nazwy tabeli, atrybutu i wartości klucza podstawowego (głównego).
- **Postulat dotyczący wartości NULL** – dostępna jest specjalna wartość NULL dla reprezentacji zarówno wartości nieokreślonej, jak i nieadekwatnej, inna od wszystkich i podlegająca przetwarzaniu.
- **Postulat dotyczący katalogu** – wymaga się, aby system obsługiwał wbudowany katalog relacyjny z bieżącym dostępem dla uprawnionych użytkowników używających języka zapytań.
- **Postulat języka danych** – system musi dostarczać pełny język przetwarzania danych, który może być używany zarówno w trybie interaktywnym, jak i w obrębie programów, obsługuje operacje definiowania danych, operacje manipulowania danymi, ograniczenia związane z bezpieczeństwem i integralnością oraz operacje zarządzania transakcji.
- **Postulat modyfikowalności perspektyw** – system musi umożliwiać modyfikowanie perspektyw, o ile jest ono semantycznie realizowalne.
- **Postulat modyfikowalności danych** – system musi umożliwiać operacje modyfikacji danych, musi obsługiwać operacje INSERT, UPDATE oraz DELETE.
- **Postulat fizycznej niezależności danych** – zmiany fizycznej reprezentacji danych i organizacji dostępu nie wpływają na aplikacje.
- **Postulat logicznej niezależności danych** – zmiany wartości w tabelach nie wpływają na aplikacje.
- **Postulat niezależności więzów spójności** – więzy spójności są definiowane w bazie i nie zależą od aplikacji.
- **Postulat niezależności dystrybucyjnej** – działanie aplikacji nie zależy od modyfikacji i dystrybucji bazy.
- **Postulat bezpieczeństwa względem operacji niskiego poziomu** – operacje niskiego poziomu nie mogą naruszać modelu relacyjnego i więzów spójności.

Operacje modelu relacyjnego:

- Selekcja - `WHERE` - selekcja podzbioru wierszy które spełniają określone warunki
- Projekcja - `SELECT` - pominięcie z wyniku pewnych kolumn
- Opracje na zbiorach: iloczyn kartezjański - `FROM` (złączenie tabel), suma zbiorów - `UNION`, różnica zbiorów - `EXCEPT`
- Agregacja - funkcje agregujące `SUM`, `MIN` itd. oraz klauzula `GROUP␣BY`

Języki zapytań

SQL - Structured Query Language - jest to język strukturalny (i deklaratywny) służący do zarządzania bazą danych (CRUD). SQL jest najbardziej znanym językiem zapytań, ale istnieje także xBase.

- **DML (Data Manipulation Language)** służy do wykonywania operacji na danych – do ich umieszczania w bazie, kasowania, przeglądania oraz dokonywania zmian. Najważniejsze polecenia z tego zbioru to:

  `INSERT` – umieszczenie danych w bazie,

  `UPDATE` – zmiana danych,

  `DELETE` – usunięcie danych z bazy.

- **DDL (Data Definition Language)** - operacje na strukturach, w których dane są przechowywane – czyli np. dodawanie, zmienianie i kasowanie tabel lub baz. Najważniejsze polecenia tej grupy to:

  `CREATE` – utworzenie struktury (bazy, tabeli, indeksu itp.)

  `DROP` – usunięcie struktury

  `ALTER` – zmiana struktury

- **DCL (Data Control Language)** ma zastosowanie do nadawania uprawnień do obiektów bazodanowych. Najważniejsze polecenia w tej grupie to:

  `GRANT` – nadawanie uprawnień do pojedynczych obiektów lub globalnie konkretnemu użytkownikowi

  `REVOKE` – odbieranie wskazanych uprawnień konkretnemu użytkownikowi

  `DENY` – zabranianie wykonywania operacji

- **DQL (Data Query Language**) to język formułowania zapytań do bazy danych. W zakres tego języka wchodzi jedno polecenie – `SELECT`. Często SELECT traktuje się jako część języka DML, ale to podejście nie wydaje się właściwe, ponieważ DML z definicji służy do manipulowania danymi – ich tworzenia, usuwania i uaktualniania. Na pograniczu obu języków znajduje się polecenie SELECT INTO, które dodatkowo modyfikuje (przepisuje, tworzy) dane.

1.2 Model obiektowo-relacyjny baz danych, inne modele danych.

Model relacyjny

W relacyjnym modelu baz danych informacja jest zapisywana w tabeli w formie wierszy. Tabele tworzą między sobą powiązania zwane relacjami. Dużo dużo więcej jest napisane podrozdział wyżej

Model obiektowy

Model obiektowy łączy cechy programów komputerowych tworzonych w językach programowania obiektowego z cechami aplikacji bazodanowych. Obiekt w bazie reprezentuje obiekt w święcie rzeczywistym. Występują tutaj pojęcia klasa, obiekt, dziedziczenie, hermetyzacja.

- oprócz danych dostępne są także operacje na danych
- użytkownik może definiować własne typy danych
- aplikacja oraz BD może być tworzona przy pomocy jednego języka
- aplikacja korzystająca z bazy nie wymaga zmiany modelu relacyjnego na obiektowy
- nie występuje problem z optymalizacją zapytań
- nie występuje ujednolicony standard modelu obiektowego

Model obiektowo-relacyjny

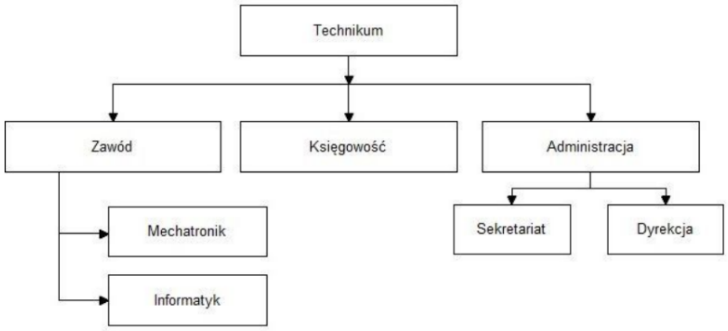
Model ten łączy koncepcję modelu relacyjnego oraz obiektowego. Jego ideą jest połączenie najlepszych cech obydwu modeli.

- występują tutaj tabele które mogą mieć pola o typie ATD (Abstract Data Type). Umożliwiają one przechowywanie w bazie np. list, kolejek, drzew - obiektów którym możemy przypisać pewne cechy i zachowanie - mają swoje własne implementacje sortowania, wyszukiwania itp
- przystosowanie do przechowywania multimediów
- ten model zwraca dane jako tabele krotek (jak w bazie relacyjnej), a nie jako kolekcję obiektów
- programista musi zamienić odpowiedź z bazy danych będącą w postaci krotek (jak w relacyjnej) na postać obiektową

Model hierarchiczny

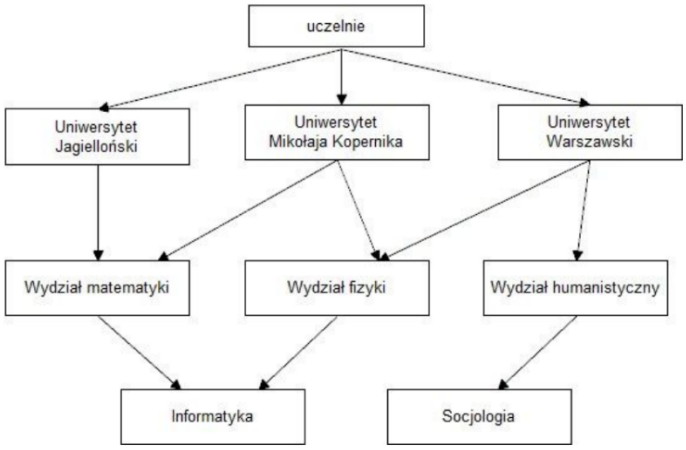
W tym modelu przechowywane dane są zorganizowane w postaci drzewa. Informacja jest zawarta w dokumentach oraz w strukturze drzewa (podobnej do drzewa folderów na dysku komputera).

- dane są przedstawione w postaci drzewa (najłatwiej to porównać do XML)
- informacje są przechowywane w strukturze drzewa
- tabele muszą być zawsze ze sobą powiązane, czyli podrzędne z nadrzędnym (np. uczeń musi być podpięty pod jakąś szkołę)
- usunięcie tabeli nadrzędnej powoduje usunięcie wszystkich tabel podrzędnych



Model sieciowy

Połączenia między dokumentami tworzą sieć. Informacja jest zawarta w dokumentach oraz w przebiegu połączeń sieci.



- jest to rozwinięcie modelu hierarchicznego gdzie tabele podrzędne mają wiele tabel nadrzędnych (skutkiem czego jest stworzenie grafu/sieci)
- można bezpośrednio odczytywać każdą tabele (nie trzeba przechodzić od korzenia do liścia jak w hierarchicznym)
- relacje/zależności między tabelami są nazywane strukturami
- w strukturze można wyłonić jedną tabelę główną z której to można przejść do tabel podrzędnych
- struktura jest relacja 1:N
- zmiany w bazie są bardzo trudne do wykonania

Model semistrukturalny

- polega na przedstawieniu danych w postaci modelu XML
- przedstawia on dane w postaci drzewiastej (czyli jest hierarchiczny)
- zapytania są w postaci ścieżki
- atrybuty znacznika przechowywują dodatkowe informacje (cechy)
- dane w postaci XML można łatwo umieścić na stronie WWW (tylko trzeba określić sposób wyświetlania)

- baza w postaci /xML może być łatwo przekonwertowana na obiekt

Model jednorodny

Wszystkie dane upchane w jednej tabeli, łatwo szukać danych po unikalnej wartości (ale ciężej po czymś nieunikalnym np nazwisko), taka forma posiada dużo duplikatów

1.3 Składnia podstawowych zapytań języka SQL.

DDL - Data Definition Language

Jak zapamiętać skrót:
Definition = DEFINIOWANIE - TWORZENIE STRUKTUR
W skład DDL wchodzą `DROP`, `CREATE` oraz `ALTER`.

- `DROP` - usunięcie struktury

  ```
  DROP TABLE tabela;
  ```

- `CREATE` - stworzenie struktury

  ```
  CREATE TABLE tabela(
  kolumna1 typ (rozmiar),
  kolumna2 typ (rozmiar),
  ...
  );
  ```

- `ALTER` - modyfikacja struktury. Obejmuje operacje takie jak np. dodanie kolumny do tabeli, zmiana typu danych w kolumnie, usunięcie kolumny.

  ```
  ALTER TABLE table
  ADD kolumna typ(dlugosc);
  ```

DML - Data Manipulation Language

Jak zapamiętać skrót:
Manipulation = MANIPULACJA - EDYCJA LUB TWORZENIE REKORDÓW
W skład DML wchodzą `INSERT`, `UPDATE` oraz `DELETE`.

- `INSERT` - dodawanie wierszy

  ```
  INSERT INTO tabela (kolumna1, kolumna2, ..., kolumna_n)
  VALUES (wartosc1, wartosc2, ..., wartosc_n)
  ```

  Nie trzeba podawać kolumn po nazwie tabeli gdy podamy po pierwsze wszystkie wartości, a po drugie w dobrej kolejności

- `UPDATE` - aktualizowanie danych, zmiana

  ```
  UPDATE tabela
  SET kolumna1 = wartosc, kolumna2 = wartosc2, ...
  WHERE warunek
  ```

  WHERE nie jest konieczny, możemy go użyć jak chcemy doprecyzować które rekordy mają się zaktualizować

- `DELETE` - usuwanie wierszy

  ```
  DELETE FROM tabela
  WHERE warunek
  ```

  WHERE nie jest konieczny, możemy go użyć jak chcemy doprecyzować które rekordy mają się skasować

DCL - Data Control Language

Jak zapamiętać skrót:
Control = KONTROLA = UPRAWNIENIA
W skład DCL wchodzą `GRANT`, `REVOKE` oraz `DENY`.

- `GRANT` - nadawanie uprawnień do pojedynczych obiektów lub globalnie konkretnemu userowi
- `REVOKE` - odbieranie uprawnień konkretnemu userowi
- `DENY` - zabranianie wykonywania operacji
- Składnia jest taka sama dla w/w poleceń:

  ```
  [GRANT/REVOKE/DENY] operacja1, operacja2, ...
  ON tabela
  TO user
  ```

  Przykład:

  ```
  GRANT SELECT, INSERT
  ON Fragment
  TO glazik
  ```

DQL - Data Query Language

Jak zapamiętać skrót:
Query = ZAPYTANIA = SELECTY
W skład DQL wchodzi jedno polecenie: `SELECT`. Pozwala wybierać wiersze z bazy danych.
Składnia:

```
SELECT kolumny
FROM tabele
WHERE warunek
GROUP BY kolumna
HAVING warunek
ORDER BY ... DESC/ASC;
```

Dodatkowe klauzule `SELECT`:

- `ORDER␣BY` - sortowanie wyników względem np. kolumny
- `ASC` oraz `DESC` - dodawane po sortowaniu, wybieramy czy ma być ascending czy descending
- `GROUP␣BY` - grupowanie wyników względem danej kolumny
- `HAVING` - filtrowanie grup, działa podobnie do WHERE. WHERE oraz HAVING mogą występować jednocześnie w zapytaniu.

1.4 Projektowanie baz danych oraz model związków encji.

Projektowanie baz danych

Etapy projektowania systemów bazodanowych:

- **Sformułowanie problemu**
- **Analiza wycinka rzeczywistości** - wywiad z ekspertem dziedzinowym, analiza wymagań funkcjonalnych (co dodawać, co usuwać itd.) i niefunkcjonalnych (tryb graficzny, platforma sprzętowa itd.)
- **Opracowanie konceptualnego modelu danych**, czyli wyodrębnienie i zdefiniowanie encji, związków między nimi oraz atrybutów (taka jakby luźna forma)
  Projektowanie modelu konceptualnego bazy danych składa się z następujących etapów:
  - **określenie zbiorów encji** - jakie encje (tabele) będzie zawierać baza?
  - **określenie atrybutów encji** - jakie cechy (kolumny) będą miały poszczególne encje?
  - **określenie dziedziny atrybutów** - jaki zakres wartości będzie miał każdy atrybut?
  - **ustalenie kluczy podstawowych** - a'la klucze główne (?)
  - **określenie związków między encjami** - a'la klucze obce (?)

  Encja to każdy element rzeczywistości, który można scharakteryzować i odróżnić od innych obiektów, np. książka (przedmiot), klient (obiekt), zamówienie (zjawisko), stan cywilny (stan).
  Atrybut to cecha encji. Zestaw atrybutów, które określamy dla encji, zależy od potrzeb bazy danych (np. książka ma atrybuty: tytuł, autor, ...)
  Dziedzina opisuje jakie wartości może przyjąć dany atrybut, np. tytuł książki to napis o długości do x znaków.
- **Opracowanie modelu logicznego**, czyli wyrażenie świata rzeczywistego za pomocą reguł modelu danych (np. relacyjnego). Transformujemy model konceptualny do logicznego. Zamiast encji są tabele, zamiast związków są wiązania poprzez klucze główne i obce itd. Następuje normalizacja relacji.
- **Opracowanie modelu fizycznego**, czyli konstruowanie modelu świata rzeczywistego za pomocą struktur danych i mechanizmów istniejących w wybranym SZBD (systemie zarządzania bazami danych), czyli np naklepanie tego w MySQL
- **Tworzenie aplikacji**
- **Testowanie systemu**

Model związków encji

Model związków encji (model ER lub ERD) to jedna z najpopularniejszych metod modelowania danych. Służy do graficznego przedstawiania koncepcji projektowanej bazy danych. Jest bardzo podobny do relacyjnego modelu. Diagram taki pokazuje obiekty (encje), ich cechy (atrybuty) oraz relacje między nimi. Elementy takie, jak encja, atrybut oraz związek zostały opisane stronę wyżej.

Typy związków encji:

- **jeden do jednego** (1:1)
  Związek ten określa, iż każde wystąpienie danego obiektu (encji) związane jest wyłącznie z pojedynczym wystąpieniem innego bytu. Na przykład: Pracownik w organizacji ma przypisane konkretne Miejsce, a to Miejsce jest przypisane do konkretnego Pracownika. Kierownik zarządza jednym Działem, a Dział zarządzany jest przez jednego Kierownika.
- **jeden do wielu** (1:N)
  Związek ten określa, iż jedno wystąpienie danego obiektu (encji) związane jest z większą ilością wystąpień innego bytu. Na przykład: Klient składa wiele Zamówień. Konkretne Zamówienie jest składane przez danego Klienta. W Firmie istnieje wiele Działów. Konkretny Dział istnieje w danej Firmie.
- **wiele do wielu** (M:N)
  Związek ten określa, iż jedno wystąpienie danego obiektu (encji) związane jest z większą ilością wystąpień innego bytu, tak samo jak jedno wystąpienie innego bytu może być powiązane z większą ilością wystąpień tego pierwszego obiektu. Przykładem takiego związku może być zależność jaka występuje pomiędzy Studentem oraz Wykładem. Student zapisany może być na wiele Wykładów, tak i na dany Wykład zapisanych może być wielu Studentów.

1.5 Problemy indeksowania baz danych, rodzaje indeksów, indeksy typu B+ drzewo.

Problemy indeksowania baz danych

Problem z przeszukiwaniem baz danych polega na tym, że... tabele w MySQL (a także w innych RDBMS) **nie są posortowane według kolumn**, dzięki którym wyciągamy odpowiednie dane. Dane ulegają aktualizacji, usuwaniu i ciągle dodawane są nowe. Jedyna „kolejność" to często klucz główny, który jest de facto indeksem. Nie jest to przydatna kolejność kiedy szukamy danych nie po kluczu a po jakimś innym polu np:

```
SELECT nazwa_produktu FROM produkty WHERE cena = 128;
```

Jeżeli mielibyśmy te nasze produkty posortowane według ceny, to znalezienie było by banalne. W przypadku kiedy dane są posortowane po kluczu głównym trzeba „sprawdzić" wszystkie rekordy i nie mamy możliwości ułatwienia sobie zadania bo produkty mogą mieć przecież różną cenę. Mówi wtedy że dokonujemy **pełnego skanu tabeli** (ang. full table scan), który działa niekorzystnie na wydajność – zabiera po prostu za wiele czasu. Sztuka optymalizacji za pomocą indeksowania polega na unikaniu tego pełnego skanu. Działanie indeksu polega na tym, że zawiera w przystępnej postaci informację, w których komórkach pamięci znajdują się produkty o cenie 128. Z tego korzysta RDBMS – najpierw uderza do indeksu a kiedy już wie skąd ma pobrać dane to od razu przechodzi do odpowiednich miejsc bez przeszukiwania całej tabeli i zwraca wynik użytkownikowi.

Zalety indeksów: znacznie przyspieszają pewne SELECTy
Wady selectów: spowalniają dodawanie, modyfikowanie i usuwanie danych (w przypadku edycji tabeli należy również zaktualizować jej indeks)

Rodzaje indeksów

- **Ze względu na liczbę wskazań indeksu (ile rekordów wskazuje indeks)**:

  indeks rzadki - zawiera rekordy tylko dla wybranych rekordów indeksowanego pliku (indeksowana jest tylko część pliku)

  indeks gęsty - zawiera rekord indeksu dla każdego rekordu indeksowanego pliku (indeksowany jest cały plik)

- **Ze względu na liczbę poziomów indeksu (indeks może wskazywać na inny indeks)**:

  indeks jednopoziomowy - do pliku z danymi tworzony jest tylko jeden indeks na dany atrybut

  indeks wielopoziomowy - do pliku z danymi tworzonych jest wiele indeksów na dany atrybut (tworzenie indeksu do indeksu)

- **Ze względu na charakterystykę indeksowanego atrybutu**:

  indeks podstawowy - - jest on założony na atrybucie (unikalnym) który umożliwia porządkowanie pliku (np. klucz główny, PESEL, dane osobowe). Jest to indeks rzadki.

  indeks zgrupowany - jest on zakładany na atrybucie który nie jest unikalny (również umożliwia porządkowanie pliku) (np. imię, wartość liczbowa). Ten indeks

wskazuje na pierwszy blok w którym występuje dana wartość (patrz wartość 2, 3, 4). Jest to indeks rzadki.

indeks wtórny - jest on zakładany na atrybucie który nie jest atrybutem uporządkowanym (czyli taki według którego plik nie jest uporządkowany). Jest to indeks gęsty.

- **Ze względu na liczbę indeksowanych atrybutów**

  indeks z kluczem prostym - indeksowany jest tylko jeden atrybut/klucz

  indeks z kluczem złożonym - indeksowane jest wiele atrybutów jednocześnie (klucz)

- **Ze względu na zarządzanie strukturą**

  indeksowanie statyczne - indeks nie ma mechanizmów które modyfikują strukturę czyli np. usuwanie pozostawia puste rekordy w indeksie, a dodawanie polega na dodawaniu rekordów do specjalnego miejsca zwanego "blokiem przepełnienia"

  indeksowanie dynamiczne - indeks ma mechanizmy które modyfikują strukturę (i nie ma tych problemów co w indeksowaniu statycznym)

Indeksy typu B+ drzewo

- to rodzaj indeksowania dynamicznego wielowarstwowego które jest przedstawione w postaci drzewa
- każdy indeks posiada wskaźnik (który nie jest liściem) wskazuje na inny indeks
- wskaźnik po lewej stronie klucza (wartości atrybutu) zawiera wartości mniejsze, a po prawej większe
- węzły tego drzewa mogą pomieścić określona ilość rekordów indeksu
- gdy rekordów w węźle jest za dużo to następuje dzielenie węzła, a gdy w dwóch węzłach jest mniej niż połowa to następuje łączenie węzłów
- B+ drzewo rośnie do góry (tak samo jak B)
- jest to drzewo zrównoważone (odległość od korzenia do liścia wszędzie jest taka sama)
- węzły które nie są liśćmi nie posiadają wartości (w B-drzewie posiadały)
- wartości kluczy które są w węzłach wewnętrznych powtarzają się w liściach (np. (2,4) C (1,2,3,4,5)
- liście są ze sobą połączone (liść o mniejszych wartościach ma wskaźnik do liścia o większych wartościach)
- Wyszukiwanie informacji w drzewie wynosi O(logn)

1.6 Przetwarzanie transakcyjne OLTP (On-Line Transaction Processing).

OLTP (Online Transaction Processing) to kategoria aplikacji klient-serwer dotyczących baz danych w ramach bieżącego przetwarzania transakcji obejmujących takie zastosowania jak systemy rezerwacji, obsługa punktów sprzedaży, systemy śledzące itp. W systemach tych klient współpracuje z serwerem transakcji, zamiast z serwerem bazy danych. Zastosowanie - np. w bankowości internetowej, w handlu detalicznym, podczas składania zamówień lub przy wysyłaniu wiadomości tekstowych. Transakcje te (tradycyjnie określane jako transakcje gospodarcze lub finansowe) są rejestrowane i zabezpieczane w taki sposób, aby przedsiębiorstwo mogło w każdej chwili uzyskać do nich dostęp w celach księgowych lub sprawozdawczych.

System OLTP:

- Umożliwia wykonywanie w czasie rzeczywistym dużej liczby transakcji bazodanowych przez dużą liczbę osób
- Wymaga błyskawicznych czasów reakcji
- Często modyfikuje małe ilości danych i zazwyczaj dokonuje tyle samo odczytów, co zapisów danych
- Używa indeksowanych danych w celu skrócenia czasu odpowiedzi
- Wymaga częstego lub równoczesnego wykonywania kopii zapasowych bazy danych
- Wymaga stosunkowo niewiele miejsca do przechowywania danych
- Zazwyczaj obsługuje proste zapytania dotyczące tylko jednego lub kilku rekordów

Transakcja to zbiór operacji na bazie danych, które stanowią w istocie pewną całość i powinny być wykonane wszystkie lub żadna z nich. Składa się zawsze z trzech etapów: **rozpoczęcia, wykonania i zamknięcia**. W systemach bazodanowych istotne jest, aby trwała jak najkrócej, ponieważ równolegle może być wykonywane wiele transakcji i część operacji musi zostać wykonana w pewnej kolejności. Każdy etap transakcji jest logowany, dzięki czemu w razie awarii można odtworzyć stan bazy sprzed transakcji.
Cechy transakcji: **ACID**
- **A**tomicity - atomowość - wykona się wszystko albo nic (zbiór operacji = całość);
- **C**onsistency - spójność - baza przed i po zmianach jest spójna;
- **I**solation - izolacja - równolegle wykonywane transakcje nie mają na siebie wpływu;
- **D**urability - trwałość - zmiany prowadzone przez transakcje są trwałe (nawet w przypadku awarii);
W SQL wyróżniamy następujące polecenia dotyczące transakcji:

- `BEGIN` lub `BEGIN␣WORK` – rozpoczęcie transakcji
- `COMMIT` – zatwierdzenie zmian wykonanych w obrębie transakcji
- `ROLLBACK` – odrzucenie zmian wykonanych w obrębie transakcji
- `SAVEPOINT␣nazwa` – zdefiniowanie punktu pośredniego o określonej nazwie
- `RELEASE␣SAVEPOINT␣nazwa` – skasowanie punktu pośredniego (nie wpływa w żaden sposób na stan transakcji)
- `ROLLBACK␣TO␣SAVEPOINT␣nazwa` – wycofanie transakcji do stanu zapamiętanego w podanym punkcie pośrednim.

2 Paradygmaty

2.1 Założenia paradygmatu programowania obiektowego.

Paradygmat obiektowy zakłada modułowość, czyli tworzenie programu z wcześniej zdefiniowanych komponentów (np. klasy).

Główne założenia paradygmatu programowania obiektowego:

1. **Abstrakcja** - to przedstawienie obiektu ze świata rzeczywistego i jego cech jako uproszczony model/szablon (tutaj klasa i obiekty) wykorzystywany w problemie programistycznym.

2. **Hermetyzacja (enkapsulacja)** - to ukrycie implementacji przed użytkownikiem (prywatne pola - publiczne metody). Cytując kogoś mądrego: "Hermetyzacja polega na tym, że klasy są zamkniętymi "czarnymi skrzynkami" - inne klasy nie muszą wiedzieć dokładnie co się dzieje w środku, mogą wchodzić z nimi w interakcje poprzez elementy, które wystawiają publicznie (publiczne pola i metody)".

3. **Dziedziczenie** - jest to tworzenie hierarchii klas. Klasa dziedzicząca otrzymuje pola i metody (te może zmieniać) "klasy matki". Tworzenie bardziej wyspecjalizowanego obiektu z bardziej ogólnego. (np. klasa jabłko dziedziczy cechy po klasie owoc)

4. **Polimorfizm** - polega na dostosowaniu zachowania obiektów do oczekiwań użytkownika. Inaczej mówiąc gdy grupa obiektów dziedziczy metodę od innej klasy. Każda klasa która odziedziczyła tę metodę to ją zmieniła. Wywołanie tej odziedziczonej metody na poszczególnych obiektach wywoła wykonanie instrukcji zaimplementowanych dla każdego obiektu. (np. dla każdej figury funkcja obliczania pola będzie inna)

2.2 Idea dziedziczenia i polimorfizmu w programowaniu.

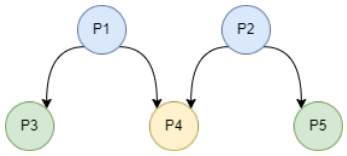
Dziedziczenie - jest to tworzenie hierarchii klas. Klasa dziedzicząca otrzymuje pola i metody (te może zmieniać) "klasy matki". Tworzenie bardziej wyspecjalizowanego obiektu z bardziej ogólnego. (np. klasa jabłko dziedziczy cechy po klasie owoc). Umożliwia ona większą czytelność kodu który jednocześnie staje się prostszy (bo po co pisać 2 razy to samo)

Polimorfizm - polega na dostosowaniu zachowania obiektów do oczekiwań użytkownika. Inaczej mówiąc gdy grupa obiektów dziedziczy metodę od innej klasy. Każda klasa która odziedziczyła tę metodę to ją zmieniła. Wywołanie tej odziedziczonej metody na poszczególnych obiektach wywoła wykonanie instrukcji zaimplementowanych dla każdego obiektu. (np. dla każdej figury funkcja obliczania pola będzie inna)

Polimorfizm umożliwia nie tylko wykonywanie odpowiednio metod dla obiektów pochodnych ale również przypisanie do bardziej ogólnego typu danych (np. Owoc jablko= new Jablko();)

2.3 Zasady programowania dynamicznego.

Programowanie dynamiczne - jest to rozszerzenie strategii "dziel i zwyciężaj". „Programowanie" oznacza w tym kontekście tabelaryczną metodę rozwiązywania problemów, a nie pisanie programów komputerowych. Wiemy, że w algorytmach typu „dziel i zwyciężaj" dzieli się problem na niezależne podproblemy, rozwiązuje je rekurencyjnie, a następnie łączy się rozwiązania wszystkich podproblemów w celu utworzenia rozwiązania pierwotnego problemu. Programowanie dynamiczne można stosować wtedy, kiedy podproblemy nie są niezależne, tzn. kiedy podproblemy mogą zawierać te same podpodproblemy. Wtedy algorytm typu „dziel i zwyciężaj" wykonuje więcej pracy niż to w istocie jest konieczne, wielokrotnie bowiem rozwiązuje ten sam podproblem.



Rysunek I.1: Przykładowy problem złożony

Jak widać na obrazku wyżej chcąc obliczyć rozwiązania problemów P1 i P2 rozwiązaniem typu "dziel i zwyciężaj", problem P4 zostałby policzony 2 razy jako problem zależny. Programowanie dynamiczne ma na celu optymalizację takich przypadków i pisząc program zapamiętalibyśmy rozwiązanie P4 gdy liczymy P1. Nastepnie obliczając drugi głowny problem - P2, znamy już rozwiązanie P4 więc nie marnujemy czasu i mocy obliczeniowej.

Proces projektowania algorytmu opartego na programowaniu dynamicznym można podzielić na cztery etapy:

1. Scharakteryzowanie struktury optymalnego rozwiązania.
2. Rekurencyjne zdefiniowanie kosztu optymalnego rozwiązania.
3. Obliczenie optymalnego kosztu metodą wstępującą (ang. bottom-up) – czyli rozpoczynając od najmniejszych podproblemów, rozwiązywać coraz większe, wykorzystując zapamiętane rozwiązania mniejszych.
4. Konstruowanie optymalnego rozwiązania na podstawie wyników wcześniejszych obliczeń.

Zasady paradygmatu programowania dynamicznego (pretty much to samo co proces projektowania) to:

- Scharakteryzować strukturę rozwiązania optymalnego.
- Zdefiniować rekurencyjnie wartość rozwiązania optymalnego, jako funkcję rozwiązań optymalnych dla podproblemów.
- Skonstruować optymalne rozwiązanie problemu na bazie wyliczonych wcześniej wielkości.

2.4 Główne paradygmaty programowania.

Paradygmat programowania - jest to wzorzec sposobu programowania, patrzenia na dane, przepływ danych oraz sposób wykonywania programu.

Programowanie imperatywne

Chcąc najkrócej scharakteryzować programowanie imperatywne, moglibyśmy napisać „zrób najpierw to, a potem tamto". Mamy zatem sekwencję poleceń zmieniających krok po kroku stan maszyny, aż do uzyskania oczekiwanego wyniku — czyli mamy stan będący funkcją czasu. Ten sposób patrzenia na programy związany jest ściśle z budową sprzętu komputerowego o architekturze von Neumanna, w którym poszczególne instrukcje to właśnie polecenia zmieniające ów globalny stan.

Imperatywne języki programowania posługują się abstrakcjami bliskimi architekturze von Neumanna, np. zmienne są abstrakcją komórek pamięci. Naturalną abstrakcją są tu też procedury (będące, notabene, zasadniczym elementem „podparadygmatu" proceduralnego). Najważniejsze — w swoim czasie — języki imperatywne to Fortran, Cobol, Pascal i C.

Programowanie proceduralne

Programowanie proceduralne – paradygmat programowania zalecający dzielenie kodu na procedury, czyli fragmenty wykonujące ściśle określone operacje. Procedury nie powinny korzystać ze zmiennych globalnych (w miarę możliwości), lecz pobierać i przekazywać wszystkie dane (czy też wskaźniki do nich) jako parametry wywołania.

Programowanie strukturalne

O programowaniu strukturalnym wspominamy z kronikarskiego obowiązku, jest to bowiem bardzo dobrze znany i powszechnie stosowany „podparadygmat" programowania imperatywnego, czy ściślej — proceduralnego. Chodzi w nim o tworzenie programów z kilku dobrze zdefiniowanych konstrukcji takich jak instrukcja warunkowa if-then-else i pętla while, za to bez skoków (go to). Powinno to sprzyjać pisaniu programów przejrzystych, łatwych w rozumieniu i utrzymaniu.

Ściślej, Dijkstra proponował użycie tylko trzech rodzajów struktur sterujących:

- Sekwencja (lub konkatenacja) — czyli po prostu wykonanie instrukcji w określonej kolejności. W wielu językach rolę „operatora konkatenacji instrukcji" spełnia niepozorny średnik...
- Wybór — czyli wykonanie jednej z kilku instrukcji zależnie od stanu programu. Przykładem jest if-then-else i switch/case.
- Iteracja, czyli powtarzanie instrukcji tak długo, jak długo spełniony (lub niespełniony) jest dany warunek. Chodzi oczywiście o pętle, np. while, repeat-until, for itp.

Programowanie obiektowe

W programowaniu obiektowym program to zbiór porozumiewających się ze sobą obiektów, czyli jednostek zawierających określone dane i umiejących wykonywać na nich

określone operacje. Najważniejsze są tu dwie cechy: po pierwsze, powiązanie danych (czyli stanu) z operacjami na nich (czyli poleceniami) w całość, stanowiącą odrębną jednostkę — obiekt; po drugie, mechanizm dziedziczenia, czyli możliwość definiowania nowych, bardziej złożonych obiektów, na bazie obiektów już istniejących.

Z tych dwóch cech bierze się zapewne wielki sukces paradygmatu obiektowego. Umożliwia on bowiem modelowanie zjawisk rzeczywistego świata w uporządkowany, hierarchiczny sposób — od idei do szczegółów technicznych. Wśród najważniejszych języków należy wymienić C++ (chociaż nie jest to język czysto obiektowy) i Javę; ze względów historycznych i poznawczych należałoby dodać Simulę 67 i Smalltalk.

Programowanie funkcyjne

Tu po prostu składamy i obliczamy funkcje, w sensie podobnym do funkcji znanych z matematyki. Nie ma stanu maszyny — nie ma zmiennych mogących zmieniać wartość. Nie ma zatem „samodzielnie biegnącego" czasu, a jedynie zależności między danymi. Nie ma efektów ubocznych. Można by wręcz sądzić, że programowanie funkcyjne zdefiniowane zostało jako poważne ograniczenie paradygmatu imperatywnego... Rzeczywiście, do pewnego stopnia zwykłe języki imperatywne zawierają w sobie „podjęzyk" funkcyjny (można przecież tworzyć, składać i wywoływać funkcje). Prawdziwe języki funkcyjne oferują jednak znacznie więcej, m.in. rekurencyjne struktury danych i możliwość operowania funkcjami wyższego rzędu.

Programowanie funkcyjne nie doczekało się takiej popularności jak dwa poprzednie paradygmaty, choć są dziedziny, gdzie jego pozycja jest przyzwoita (np. język Erlang używany w telekomunikacji oraz język K używany do obliczeń finansowych). Najsłynniejsze języki funkcyjne żyły lub żyją przede wszystkim w środowiskach akademickich: Lisp, Scheme (potomek Lispu), ML, Ocaml (potomek ML-a), Miranda, Haskell.

Programowanie w logice

Podobnie jak w programowaniu funkcyjnym, nie „wydajemy rozkazów", a jedynie opisujemy, co wiemy i co chcemy uzyskać (z tego powodu języki funkcyjne i logiczne nazywa się łącznie językami deklaratywnymi). Na program składa się zbiór zależności (przesłanki) i pewne stwierdzenie (cel). Wykonanie programu to próba udowodnienia celu w oparciu o podane przesłanki, a więc pewien rodzaj automatycznego wnioskowania; obliczenia wykonywane są niejako „przy okazji" dowodzenia celu.

Język programowania w logice (a ściślej — jego interpreter) to właściwie system automatycznego dowodzenia twierdzeń, działający w oparciu o nieco uproszczony rachunek predykatów. Kluczowym pojęciem jest tu rezolucja, realizowana m.in. przy pomocy unifikacji i nawrotów. Zastosowania programowania w logice obejmują przede wszystkim sztuczną inteligencję (np. systemy ekspertowe, rozpoznawanie obrazów) i przetwarzanie języka naturalnego. Językiem, który uzyskał największą popularność, jest Prolog. Historycznie ważny jest również Planner — wcześniejszy i bardziej złożony od Prologu.

2.5 Cechy programowania deklaratywnego.

W programowaniu deklaratywnym programista deklaruje mając jakiś element co chce z niego uzyskać (nie jest mówione jak to ma zostać wykonane, tylko jest mówione jaki ma być efekt).

Cechy:

- wzorowany na językach naturalnych
- nieliniowe wykonanie (komenda końcowa może modyfikować początek)
- bardzo częste wykonywanie rekurencji zamiast pętli
- definiuje problem i szuka odpowiedź na zadane pytanie

Do programowania deklaratywnego zalicza się:

- języki funkcyjne - zamiast instrukcji używa się funkcji (cały program to funkcja). Przykładem jest Haskell w którym Wszystko opiera się na wyrażeniach Lambda. W tym języku funkcje są sobie równe jeśli dają takie same odpowiedzi.
- języki w logice - zamiast instrukcji używa się zdań logicznych. np. w Prologu zdania logiczne na początku opisują zależności (między elementami) aby na końcu zapytać czy podane zdanie jest prawdziwe (oczywiście trzeba na koniec podać to zdanie). Tutaj wszystko jest wyrażeniem logicznym