

# ZARZĄDZANIE INFORMACJĄ 1

EGZAMINATOR: DR HAB. INŻ. PRZEMYSŁAW KORYTKOWSKI

---

## 1. Historia rozwoju relacyjnych baz danych

---

Pierwszy system zarządzania bazami danych został opracowany **w latach sześćdziesiątych XX wieku**. Pionierem był Charles Bachman. Wczesne opracowania Bachmana pokazywały, że jego celem było bardziej efektywne użycie nowych urządzeń bezpośredniego dostępu do składowanych danych, które wtedy zaczynały być dostępne. Jak dotąd, przetwarzanie danych było oparte na kartach dziurkowanych i taśmach magnetycznych. Oznaczało to szeregowy dostęp do danych, co pociągało za sobą użycie innych algorytmów niż dla dostępu swobodnego.

Powstały wtedy dwa kluczowe modele danych: sieciowy, opracowany przez CODASYL na bazie idei Bachmana i (być może niezależnie) hierarchiczny, użyty w systemie opracowanym przez North American Rockwell i później adoptowany przez IBM jako kamień milowy dla IMS.

W 1970 E. F. Codd zaproponował relacyjny model danych. Krytykował on istniejące modele danych za mieszanie abstrakcyjnego opisu struktury informacyjnej z opisami mechanizmów fizycznego dostępu. Jednak przez dłuższy czas model relacyjny pozostawał tylko w sferze rozważań akademickich. Podczas gdy produkty CODASYL (IDMS) i IBM (IMS) były uważane za praktyczne rozwiązania wymagające tylko dostępnych wówczas technologii, to model relacyjny musiał poczekać na odpowiedni poziom rozwoju oprogramowania i sprzętu. Jednym z pierwszych implementacji modelu relacyjnego były: Ingres Michaela Stonebrakera z Berkeley i System R z IBM. Oba były prototypami badawczymi, ogłoszonymi w roku 1976. Pierwsze komercyjne rozwiązania, Oracle i DB2 nie były dostępne aż do roku około 1980.

W latach osiemdziesiątych XX wieku aktywność badaczy skupiała się na rozproszonych bazach danych i maszynach bazodanowych (ang. database machines), ale te wysiłki nie miały większego odzwierciedlenia w ofertach rynkowych.

W latach dziewięćdziesiątych uwaga badaczy przesunęła się w kierunku obiektowych baz danych. Stosowano je z powodzeniem tam, gdzie konieczna była obsługa bardziej skomplikowanych danych niż dane, którym mogły podołać relacyjne bazy danych. Przykładem były: przestrzenne bazy danych (ang. spatial databases), dane inżynierijne i dane multimedialne.

Pierwsze lata XXI wieku są okresem dużego zainteresowania bazami danych XML. W tym czasie, podobnie jak to było w przypadku obiektowych baz danych, powstało sporo nowych firm-producentów tych baz, ale kluczowe ich elementy są wbudowywane także w istniejące relacyjne bazy danych. Celem baz danych XML jest usunięcie tradycyjnego podziału na dokumenty i dane, pozwalające na trzymanie wszystkich zasobów informacyjnych organizacji w jednym miejscu, obojętnie czy te dane są wysoce ustrukturalizowane czy nie.

---

*2. System zarządzania bazami danych – cechy, funkcje, zadania, nazwy produktów*

---

### **SYSTEM ZARZĄDZANIA BAZAMI DANYCH (Database Management System – DBMS) –**

oprogramowanie umożliwiające użytkownikom definiowanie, tworzenie i zarządzanie bazą danych oraz kontrolowanie dostępu do niej.

- duży, zintegrowany zbiór danych
- modeluje encje i relacje

**Przykłady:** MySQL, PostgreSQL, Firebird, Oracle Database, MS SQL Server

Model danych -> zbiór pojęć opisujących dane

Relacyjny model danych -> obecnie najczęściej stosowany

Główna koncepcja -> relacja = tabela

Schemat -> opis określonego zbioru danych wykorzystujący dany model danych

Np. każda relacja w relacyjnym modelu danych ma schemat opisujący typy itp.

**Schemat fizyczny** – opisuje układ danych -> relacje jako nieuporządkowane pliki, niektóre dane posortowane po indeksach

**Najważniejsze powody używania DBMS:** aplikacje nie muszą martwić się o strukturę danych i ich przechowywanie; logiczna niezależność danych -> ochrona przed zmianami w logicznej strukturze danych (nie trzeba pytać czy możemy dodać nową encję lub atrybut bez przepisywania aplikacji); Fizyczna niezależność danych -> ochrona przed zmianami układu fizycznego (nie trzeba pytać na jakich dyskach są przechowywane dane, czy dane są indeksowane).

### **Cechy i funkcje:**

- operacje na dużych i bardzo dużych zbiorach danych
- zarządzanie złożonymi strukturami
- posiada mechanizmy, które:
  - a) pozwalają administrować zbiorami danych umieszczonymi w bazie,
  - b) zapewniają bezpieczeństwo i integralność danych,
  - c) umożliwiają dostęp do danych za pomocą języka zapytań (najczęściej SQL),
  - d) zapewniają wielodostępność danych (na przykład przez transakcje) oraz pozwalają na autoryzację dostępu do danych.

Często takie systemy posiadają narzędzia do przechowywania i przetwarzania danych multimedialnych, narzędzia do konwersji danych w celu współpracy z innymi systemami baz danych, graficzne środowiska do tworzenia aplikacji oraz narzędzia do udostępniania bazy danych w internecie.

W systemach zarządzania bazą danych można wyodrębnić dwa elementy: - serwer — przechowuje dane, umożliwia ich pobieranie i aktualizowanie oraz zapewnia ich integralność i bezpieczeństwo;

- oprogramowanie pośredniczące (ang. middleware) — realizuje komunikację między użytkownikiem a serwerem. Wyposażone jest w mechanizmy pozwalające wykorzystywać pobierane dane, na przykład w narzędzia do tworzenia i obsługi formularzy oraz raportów

Systemy zarządzania bazą danych zwykle działają w trybie klient-serwer, czyli baza umieszczona na serwerze jest udostępniana klientom poprzez oprogramowanie pośredniczące (systemy bazodanowe). Istnieją również systemy, które nie uwzględniają podziału typu klient-serwer. Przykładem takiego systemu jest Microsoft Access. System zarządzania bazą danych (serwer bazodanowy), który należy do architektury klient-serwer, składa się z dwóch części, które ze sobą współpracują. Jedna część, działająca na serwerze, odpowiedzialna jest za wydajność, bezpieczeństwo, kopie zapasowe itp. Druga, działająca po stronie klienta, zapewnia interfejs użytkownika, dzięki któremu możliwe jest przeprowadzanie operacji na bazie danych. Najczęściej serwer bazodanowy komunikuje się z bazą danych za pomocą języka SQL.

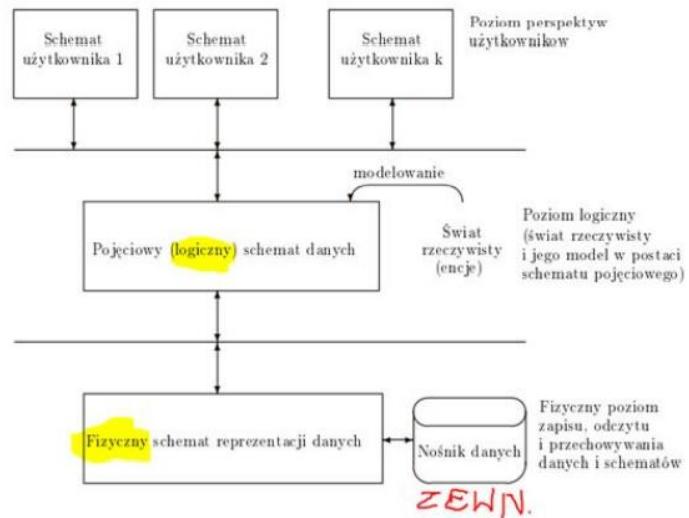
---

### *3. Schemat danych relacyjnych: fizyczny, logiczny, zewnętrzny*

---

**Model relacyjny** – model organizacji danych bazujący na matematycznej teorii mnogości, w szczególności na pojęciu relacji. Na modelu relacyjnym oparta jest relacyjna baza danych (ang. Relational Database) – baza danych, w której dane są przedstawione w postaci relacyjnej.

W najprostszym ujęciu w modelu relacyjnym dane grupowane są w relacje, które reprezentowane są przez tablice. Relacje są pewnym zbiorem rekordów o identycznej strukturze wewnętrznie powiązanych za pomocą związków zachodzących pomiędzy danymi. Relacje zgrupowane są w tzw. schematy bazy danych. Relacją może być tabela zawierająca dane teleadresowe pracowników, zaś schemat może zawierać wszystkie dane dotyczące firmy. Takie podejście w porównaniu do innych modeli danych ułatwia wprowadzanie zmian, zmniejsza możliwość pomyłek, ale dzieje się to kosztem wydajności.



### Fizyczny (PDM):

Model fizyczny opisuje układ danych:

- relacje jako nieuporządkowane pliki
- pewne dane w posortowanym porządku(indeksy)

Model logiczny jest podstawą do wygenerowania tzw. fizycznego modelu danych (PDM).

W modelu fizycznym tabele zostają uzupełnione o kolumny kluczy obcych – na podstawie odpowiednich związków istniejących pomiędzy encjami w modelu fizycznym.

### Logiczny (CDM):

Na wstępie określa się zbiory encji oraz ich atrybuty (wraz z okrešeniem typu danych, wymagalności, ograniczeń) i klucze główne. Pomiędzy tak zdefiniowanymi zbiorami encji kreśsi się relacje o określonych własnościach. Wszystko to odbywa się w trybie graficznym.

### Zewnętrzny:

Jest to model, który powstaje jako ostatni, jest to ostateczny widok wygenerowanych tabel.

---

#### 4. Projektowanie baz danych: analiza wymagań, model konceptualny, projekt: logiczny, fizyczny, bezpieczeństwa

---

### PROJEKTOWANIE BAZ DANYCH:

- ma na celu m.in. uzgodnienie struktury bazy danych przed podjęciem decyzji o konkretnej implementacji
- należy określić jakie encje będą modelowane, jak będą ze sobą połączone, jakie ograniczenia (constraints) istnieją w domenie

### **ANALIZA WYMAGAŃ:**

- biorą w niej udział zarówno osoby techniczne jak i nie-techniczne
- na tym etapie należy odpowiedzieć na pytania: Co będzie przechowywane w bazie danych? W jaki sposób będzie używane? Co będziemy robić z danymi z bazy danych? Kto powinien mieć dostęp do danych?

### **MODEL KONCEPTUALNY:**

- wysokopoziomowy opis bazy danych
- wystarczająco precyzyjny aby osoby techniczne mogły go zrozumieć
- ale nie aż tak precyzyjny by osoby nie techniczne nie mogły brać udziału w jego powstawaniu

### **PROJEKT LOGICZNY:**

Projekt logiczny to proces przekształcania (lub mapowania) koncepcyjnego schematu domeny aplikacji w schemat dla modelu danych będącego podstawą określonego DBMS (ang. Database Management System – System zarządzania bazą danych), takiego jak relacyjny lub obiektowy model danych.

To mapowanie można rozumieć jako rezultat próby osiągnięcia dwóch różnych zestawów celów:

- Cel reprezentacji: zachowanie zdolności do przechwytywania i rozróżniania wszystkich poprawnych stanów schematu konceptualnego.
- Cele zarządzania danymi: rozwiązanie kwestii związanych z ułatwieniem i kosztem zapytań do schematu logicznego, a także kosztów przechowywania i obsługi ograniczeń (constraint maintenance)

### **PROJEKT BEZPIECZEŃSTWA:**

Projektanci i inżynierowie baz danych kładą nacisk przede wszystkim na szybkość i wydajność. W rezultacie kwestie wydajnościowe czasami przyjmiewają kwestie bezpieczeństwa. W dzisiejszym świecie opartym na danych ich wycieki spowodowane naruszeniem bezpieczeństwa generują ogromne koszty dla posiadaczy baz danych oraz ogromne zyski dla hakerów, którzy te dane przejmują. Należy więc priorytetowo traktować bezpieczeństwo w projektach baz danych.

Systemy powinny być w stanie przeciwstawić się atakom takim jak SQL Injection, przepełnienie bufora, exploity uwierzytelniania oraz ataki typu „odmowa usługi” (DoS)

### **Aspekty zabezpieczania baz danych:**

Projektant bazy danych musi zapewnić bezpieczeństwo na trzech poziomach:

- Dane: hakerzy mogą tworzyć zagrożenia bezpieczeństwa poprzez uszkadzanie, manipulowanie i kradzież danych. Projektanci muszą więc przyjrzeć się wszystkich możliwych scenariuszom, w których bezpieczeństwo informacji mogłoby zostać naruszone
- Systemy: baza danych jest zależna od różnych systemów oprogramowania i sprzętu. Systemy te odgrywają tutaj bezpośrednią i pośrednią rolę. Programiści i inżynierowie baz danych muszą ocenić systemy, aby odnaleźć w nich punkty podatne na atak

- Użytkownicy: model bazy danych powinien uwzględniać procesy kontroli dostępu i autoryzacji właściwe dla klienta – odbiorcy bazy danych.

---

5. *Diagramy ER: zbiory encji, atrybuty, relacje, słabe zbiory encji, słabe relacje, podklasy, IsA*

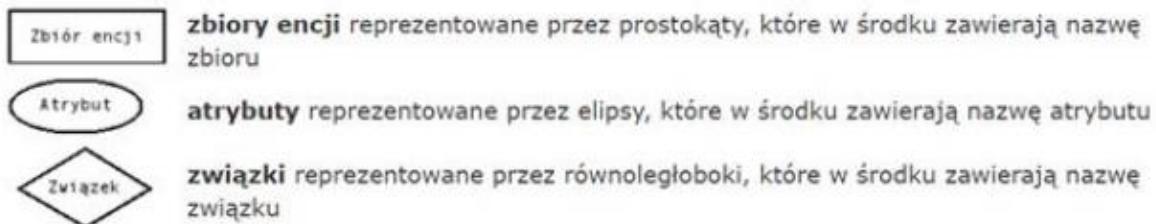
---

Związek = relacja

Diagramy ER = Diagramy związków (encji) = entity-relationship diagrams

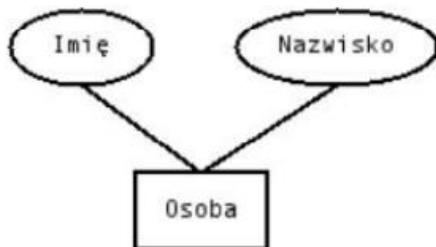
**Elementy podstawowe:**

Diagramy związków-encji pozwalają w sposób graficzny opisywać model konceptualny. Do podstawowych elementów diagramów ER należą:



**Zbiory**

Zbiory encji opisywane są za pomocą atrybutów, które łączy się z nimi za pomocą pojedynczej linii prostej:



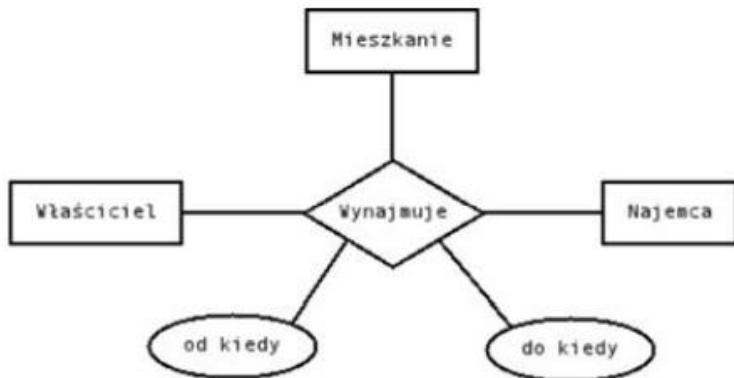
**Uwaga:** wewnętrz elipsy wprowadzana jest nazwa atrybutu, a nie jego wartość. Zatem niepoprawnym atrybutem jest np. czerwony, gdyż jest to wartość atrybutu, który powinien nazywać się kolor.

**Relacje**

Związki zachodzące pomiędzy zbiorami encji przedstawia się w postaci równolegloboku, z którego wychodzą linie proste do wszystkich zbiorów encji, należących do danego związku:



Związki zachodzące pomiędzy zbiorami encji mogą być więcej niż dwuargumentowe, mogą również posiadać atrybuty:



Związki mogą zachodzić pomiędzy encjami należącymi do jednego zbioru. Taka sytuacja wymaga jednak opisania ról w jakich występują poszczególne encje:



### Krótność związków

Krótność związków zbiorów encji jest cechą, która opisuje z iloma encjami należącymi do jednego zbioru encji może być połączona jedna encja należąca do drugiego zbioru encji.

### Istnieją trzy główne typy związków

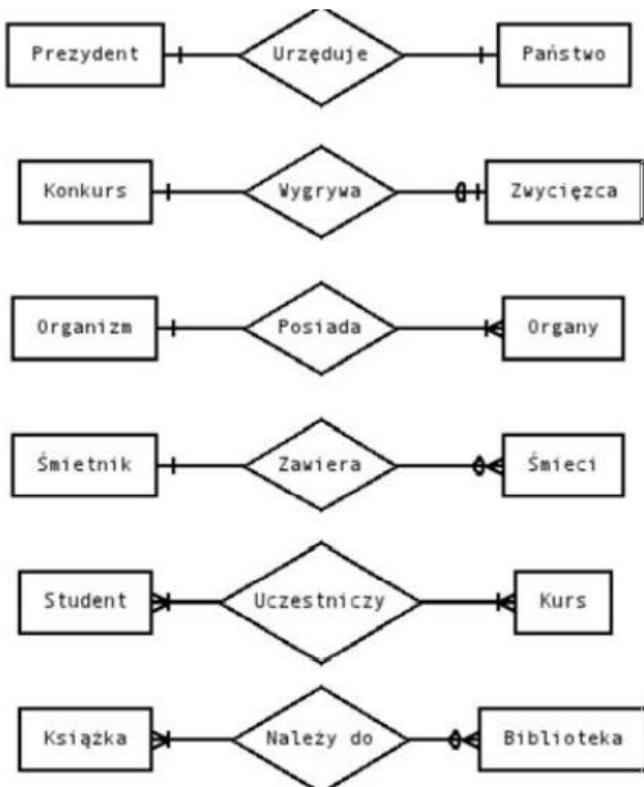
- 1 do 1
- 1 do n
- n do n

Tym niemniej wykorzystując tylko te trzy typy, nie jesteśmy w stanie wystarczająco precyzyjnie opisać przypadków z jakimi spotykamy się w rzeczywistości.

Niestety w odniesieniu do diagramów ERD nie wypracowano jednego standardu oznaczania krótności, aczkolwiek najczęściej można spotkać się z następującą notacją:

- - powiązanie jednokrotne, obowiązkowe (tylko jeden)
- - powiązanie jednokrotne, opcjonalne (jeden lub zero)
- - pozwianie wielokrotne, obowiązkowe (co najmniej jeden)
- - powiązanie wielokrotne, opcjonalne (zero lub więcej)

Różne kombinacje krotności:



**Uwaga:** symbole krotności interpretowane są zawsze w ten sposób, że jedna encja z jednego zbioru encji może być powiązana z taką liczbą encji należących do drugiego zbioru, na jaką wskazuje symbol leżący bezpośrednio przy drugim zbiorze encji.

W powyższym przykładzie – jeden organizm może posiadać jeden lub więcej organów, natomiast jeden organ musi należeć do dokładnie jednego organizmu.

### Typy atrybutów

W modelu ER wyróżnić można trzy specjalne typy atrybutów:

- kluczowe
- pochodne
- częściowe kluczowe

Jednym z ważniejszych etapów projektowania modelu konceptualnego, jest zidentyfikowanie **atributów kluczowych**, które tworzą klucz encji. Atrybuty kluczowe to atrybuty, które w sposób jednoznaczny identyfikują encję, która je posiada. Znaczy to, że nie mogą występować dwie encje, które posiadają taką samą wartość atrybutów kluczowych.

Klucz może być pojedynczym atrybutem lub zbiorem atrybutów. Encja może posiadać wiele kluczy. W takiej sytuacji jeden z nich jest wyróżniony i nazywa się go **kluczem głównym**. Pozostałe klucze, to **klucze alternatywne**. Ponadto każdy zbiór encji musi posiadać **przynajmniej jeden klucz**.

Atrybuty kluczowe wyróżnia się poprzez podkreślenie ich nazwy:



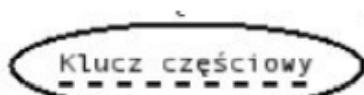
Innym ważnym typem atrybutów, są **atrybuty pochodne**. Ten typ atrybutów wykorzystywany jest do reprezentowania informacji, które zawsze mogą być obliczone na podstawie danych zgromadzonych w bazie, ale zachodzi potrzeba reprezentowanie ich w modelu. Przykładem atrybutu tego rodzaju może być np. średnia ocena publikowanego materiału, wyliczana na podstawie głosów oddanych przez czytelników.

Atrybuty pochodne reprezentowane są za pomocą owalu, którego brzeg rysowany jest linią przerywaną:



W diagramach ERD wykorzystuje się również **klucze częściowe**. Klucze częściowe pełnią tę samą rolę, co zwykłe kluczowe, z tą różnicą, że samodzielnie nie identyfikują encji. Występują wyłącznie w słabych encjach.

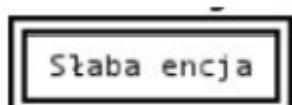
Klucze częściowe wyróżniane są za pomocą podkreślenia linią przerywaną:



## Słabe encje

Słabe encje to szczególny typ encji, których klucze składają się (przynajmniej częściowo) z atrybutów kluczowych innych encji. Jeśli encja jest słaba, to pomiędzy zbiorem encji, których klucze wykorzystywane są w jej kluczu, a zbiorem, do którego on należy, musi zachodzić związek, którego krotność po stronie przeciwej słabej encji, wynosi dokładnie jeden. W przeciwnym bowiem razie, mogłaby wystąpić sytuacja, w której słaba encja nie posiadałaby jednoznacznego identyfikatora.

Słabe encje reprezentowane są za pomocą prostokątów z podwójną ramką:



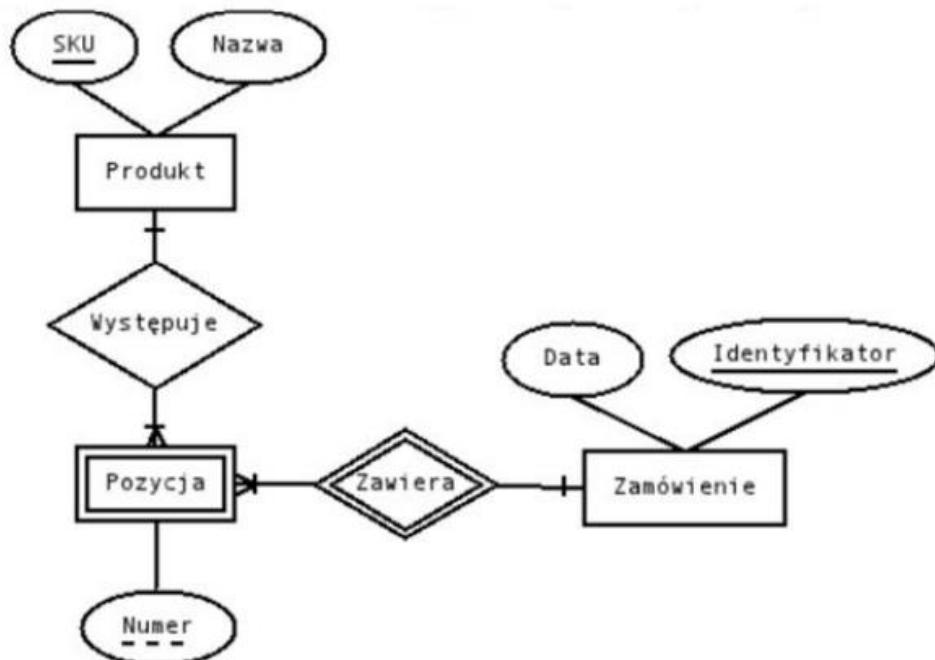
## Słabe relacje

Związki identyfikujące, które wskazują zbiory encji, których klucze wykorzystywane są przez zbiory słabych encji, reprezentowane są za pomocą podwójnego równoległoboku:



Słabe encje wykorzystywane są zwykle wtedy, gdy mamy do czynienia z kompozycją lub związkiem wiele do wiele przekształconym w zbiór encji, tzn. słabe encje zawsze występują jako elementy pewnej struktury wyższego rzędu.

Jako przykład można podać zamówienie, które posiada swój unikalny identyfikator oraz pozycje, których porządek jest ustalony przez użytkownika. Zamiast przypisywać każdej pozycji odrębny identyfikator (klucz) i dodatkowo numer porządkowy na zamówieniu, prościej jest opisać pozycję, za pomocą klucza składającego się z identyfikatora zamówienia i numeru pozycji. Sytuacja ta przedstawiona jest na poniższym diagramie:



## „Isa” i podklasy

Zbiór encji często zawiera pewne encje o specjalnych właściwościach, których nie mają wszystkie elementy tego zbioru. W takim przypadku użyteczne jest zdefiniowanie specjalnych zbiorów encji, tzw. podklas, które zawierają te dodatkowe atrybuty i/lub związki.

Zbiór encji łączymy z jego podklasami przy wykorzystaniu związku „isa” (tzn. stwierdzenie „A isa B” oznacza związek ze zbioru encji A do zbioru encji B). Związek isa jest specjalnym rodzajem związku i

aby podkreślić jego wyjątkowość, używa się dla niego specjalnego oznaczenia – każdy związek isa reprezentuje się za pomocą trójkąta.

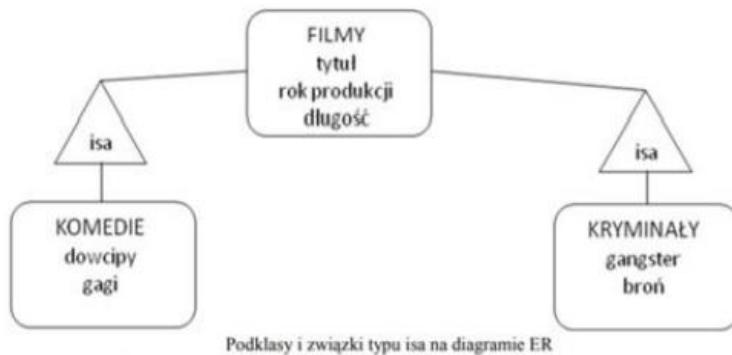


Jedna krawędź trójkąta jest połączona do podklasy, a przeciwny wierzchołek jest połączony z nadklassą. Każdy związek isa jest typu jeden-do-jednego.

Relacja isa jest jedną z podstawowych relacji występujących pomiędzy pojęciami. Typowy zwrot, który służy do jej wyrażania, to "X jest rodzajem Y" lub "X jest Y", gdzie X i Y są pewnymi pojęciami ogólnymi (np. wilk jest zwierzęciem, lazur jest rodzajem niebieskiego, van jest rodzajem samochodu).

#### Przykład:

Wśród filmów występują kryminały, komedie, przygodowe, kreskówki, itd. Dla poszczególnych typów można zdefiniować odpowiednią podkласę encji Filmy, np. wybierzmy podklasses Kryminały i Komedie.



---

#### 6. Typy relacji: 1-1, 1-N, N-M

---

#### ZWIĄZEK 1:1 (jeden do jeden)

Każdy wiersz z tabeli A może mieć tylko jednego odpowiednika w tabeli B (i na odwrót).

Ten rodzaj relacji może być postrzegany jako podzielenie tabeli na dwie (bo relacja jest jeden do jeden). Stosowany np. wtedy, gdy zbiór dodatkowych atrybutów jest określony tylko dla wąskiego podzbioru wierszy w tabeli podstawowej.

Np.: Pracownik jest osobą, ale nie każda osoba jest pracownikiem.

Innym zastosowaniem związku 1:1, jest wydzielenie pewnej grupy atrybutów które są rzadko odpytywane. Mogą być więc umiejscowione w tabeli przechowywanej na osobnym, wolniejszym nośniku danych.

Ta relacja jest nietypowa, ponieważ najczęściej informacje powiązane w ten sposób są przechowywane w jednej tabeli.

## ZWIĄZEK 1:N (jeden do wiele)

Jest to najczęściej spotykana relacja. Określamy w niej, że każdy element ze zbioru A (wiersz tabeli A), może być powiązany z wieloma elementami zbioru B. Aby w projekcie bazy danych utworzyć relację jeden-do-wielu, należy klucz podstawowy znajdujący się po stronie „jeden” relacji dodać jako pole lub pola do tabeli po stronie „wiele” tej relacji.

Np.: wiele produktów może należeć do jednej kategorii.

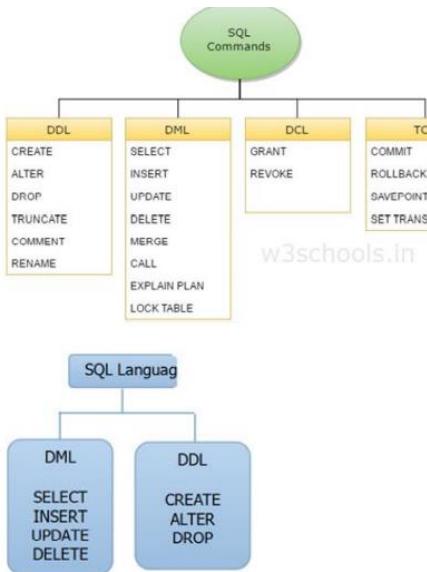
## ZWIĄZEK N:M (wiele do wiele)

Realizowana jest zawsze jako dwie relacje 1:N. Zatem jeśli chcemy między dwoma tabelami zamodelować związek N:M potrzebujemy trzecią tabelę – łącznikową. Przykładem niech będzie fragment każdego systemu zamówień. Mamy zamówienia i produkty. Każdy z produktów może być zamawiany wielokrotnie, w różnych zamówieniach. Każde z zamówień może zawierać wiele pozycji (produktów).

---

### 7. SQL: DDL, DML

---



Użycie SQL, zgodnie z jego nazwą, polega na zadawaniu zapytań do bazy danych. Zapytania można zaliczyć do jednego z czterech głównych podzbiorów:

- SQL DML (ang. Data Manipulation Language – „język manipulacji danymi”),
- SQL DDL (ang. Data Definition Language – „język definicji danych”),
- SQL DCL (ang. Data Control Language – „język kontroli nad danymi”)
- SQL DQL (ang. Data Query Language – „język definiowania zapytań”)

Instrukcje SQL w obrębie zapytań tradycyjnie zapisywane są wielkimi literami, jednak nie jest to wymóg.

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.

Dane tekstowe muszą być zawsze ujęte w znaki pojedynczego cudzysłowa ('')

```
update account set balance = 1000 where account_number = 01;
```

Do DML (– działa na danych, manipuluje danymi) zaliczamy także:

- SELECT – pobieranie z bazy danych,
- MERGE – operacja UPSERT (INSERT lub UPDATE)
- CALL – wywołaj jakiś podprogram,
- EXPLAIN PLAN – interpretacja ścieżki dostępu do danych,
- LOCK TABLE – kontrola konkurencji.

Dzięki DDL (Data Definition Language DDL – działania na strukturach danych, zmiany schematu) można operować na strukturach, w których dane są przechowywane – czyli np. dodawać, zmieniać i kasować tabele lub bazy. Najważniejsze polecenia tej grupy to:

- CREATE (np. CREATE TABLE, CREATE DATABASE, ...) – utworzenie struktury (bazy, tabeli, indeksu itp.),
- DROP (np. DROP TABLE, DROP DATABASE, ...) – usunięcie struktury,
- ALTER (np. ALTER TABLE ADD COLUMN ...) – zmiana struktury (dodanie kolumny do tabeli, zmiana typu danych w kolumnie tabeli)

```
create table account (
    account-number  char(10),
    balance integer);
```

- TRUNCATE (usuwa wszystkie rekordy z tabeli, wliczając miejsca zaalokowane na te rekordy)
- COMMENT (dodaje komentarz do słownika danych)
- RENAME (zmienia nazwę obiektu)

### Tworzenie tabeli w MySQL:

```
CREATE TABLE nazwa_tabeli struktura_tabeli
```

Przykładowe polecenie SQL:

```
CREATE TABLE pracownicy (
    imie VARCHAR(30),
    nazwisko VARCHAR(30),
    data_urodzenia DATE,
    placa DECIMAL(10,2)
);
```

Po każdej nazwie pola następuje definicja typu danych, jakie będzie ono zawierało. Istnieje możliwość określenia wartości domyślnej, jakie przyjmie każde nowe pole w momencie, kiedy nie zostanie wypełnione innymi danymi.

Wartości domyślne podaje się dla każdego pola po słowie kluczowym DEFAULT zaraz po definicji typu:

```
CREATE TABLE pracownicy (
    imie VARCHAR(30),
    nazwisko VARCHAR(30),
    data_urodzenia DATE DEFAULT '1950-01-01',
    placa DECIMAL(10,2) DEFAULT '1000.00'
);
```

### TYPY DANYCH w MySQL'u

#### Liczby całkowite (INTEGER):

- INT, INTEGER liczby całkowite z zakresu -2147483648 do 2147483647 (4 bajty)
- TINYINT liczba całkowita z zakresu -128 do 127 (1 bajt)
- SMALLINT liczba całkowita z zakresu -32768 do 32767 (2 bajty)
- MEDIUMINT liczba całkowita z zakresu -8388608 do 8388607 (3 bajty)
- BIGINT liczba całkowita z zakresu -9223372036854775808 do 9223372036854775807 (8 bajtów)

#### Liczby rzeczywiste (REAL):

- FLOAT (M,D) liczba rzeczywista z zakresu -3.402823466E+38 do -1.175494351E-38 i 0 i 1.175494351E-38 do 3.402823466E+38 (4 bajty),

- REAL, DOUBLE liczba rzeczywista z zakresu 1.7976931348623157E+308 do -2.2250738585072014E-308 i 0 i 2.2250738585072014E-308 to 1.7976931348623157E+308 (8 bajtów)
- DECIMAL (M,D) gdzie M jest ilością znaczących pozycji w liczbie a D skalą liczby np.: DECIMAL(9,2) będzie miał zakres od -9999999.99 do 9999999.99 - jest to liczba składająca się z dziewięciu pozycji i przesunięta o dwa miejsca w prawo

#### Czasowe typy danych w MySQL'u:

- DATETIME w takiej kolumnie umieszczamy datę wraz z godziną w formacie : YYYY:MM:DD:HH:mm:SS (odpowiednio rok, miesiąc, dzień, godzina, minuta i sekunda), typ ten akceptuje zakres od '1000-01-01 00:00:00' do '9999-12-31 23:59:59'
- DATE czyli data w formacie YYYY:MM:DD (rok, miesiąc, dzień), zakres od '1000-01-01' do '9999-12-31'
- TIME godzina w formacie HH:mm:SS (godzina, minuta, sekunda)
- YEAR rok YYYY (np.:2001) zakres od 1901 do 2155 (1 bajt)
- TIMESTAMP(n) jest to data z godziną, precyzję tego zapisu ustalamy sami poprzez wartość n (dla przykładu n=14 oznacza datę i godzinę w formacie YYYY:MM:DD:HH:mm:SS a n=2 YY) zakres od 1970 do 2037

#### Łańcuchowe (znakowe) typy danych:

- CHAR(n) jest to łańcuch znaków o długości n, gdzie n może przyjmować wartości od 0-255, bez względu na to jaki łańcuch zapiszemy do takiej komórki tabeli, zawsze będzie zajmował n bajtów.
- VARCHAR(n) ten typ danych pamięta łańcuch znaków oraz jego długość, wartość n określa maksymalną długość łańcucha, np.: jeśli do kolumny VARCHAR(10) zapiszemy łańcuch o długości 5 znaków to będzie on zajmował w bazie 6 bajtów
- BLOB "pamięta" dane w formie binarnej, ze względu na maksymalną długość takiego ciągu znaków dzieli się na kilka podtypów: TINYBLOB (2^8 znaków), BLOB (2^16 znaków), MEDIUMBLOB (2^24 znaków), LONGBLOB (2^32 znaków)
- TEXT dane tekstowe, podobnie jak BLOB ma odmiany ze względu na długość tekstu: TINYTEXT (2^8 - 256 znaków), TEXT (2^16 - 65535 znaków), MEDIUMTEXT (2^24 - 16777216 znaków), LONGTEXT (2^32 - 4294967296 znaków)

ENUM: Typ wyliczeniowy, dane przyjmują wartości spośród wcześniej przygotowanej przez nas listy. Lista taka może mieć maksymalnie 65535 elementów.

SET: to typ zbiorowy - pozycja taka może przyjmować 0 lub więcej wartości spośród listy przygotowanej podczas tworzenia tabeli. Przykład: dla definicji SET ("jeden" , "dwa") do komórki tabeli możemy zapisać następujące wartości:"" lub "jeden" lub "dwa" lub "jeden, dwa". Definicja SET może mieć maksymalnie 64 elementy.

**ALTER:** Do modyfikacji struktury tabeli służy polecenie ALTER TABLE. Zmodyfikowanie struktury tabeli zawierającej już jakieś dane spowoduje próbę podporządkowania istniejących danych nowemu formatowi. Dzięki temu poleceniu można dodawać, modyfikować, usuwać pola oraz manipulować indeksami. Ogólna postać polecenia przedstawia się następująco:

**ALTER TABLE nazwa\_tabeli specyfikacja\_struktury;**

Aby dodać kolejne pole do istniejącej tabeli należy użyć polecenia:

**ALTER TABLE pracownicy ADD rozmiar\_butu VARCHAR(10);**

Spowoduje to dodanie jednego pola o nazwie rozmiar\_butu na końcu struktury tabeli.

Aby zmienić typ jednego konkretnego pola użyjemy polecenia:

**ALTER TABLE pracownicy MODIFY rozmiar\_butu INT;**

Aby usunąć konkretne pole ze struktury tabeli należy użyć polecenia:

**ALTER TABLE pracownicy DROP rozmiar\_butu;**

Zmiana nazwy tabeli przedstawia się następująco:

**ALTER TABLE pracownicy RENAME zatrudnieni;**

**DROP:** jest to polecenie SQL do usuwania obiektów w systemie zarządzania relacyjną bazą danych. Przykładowo jeśli chcemy usunąć tabelę o nazwie klienci to wydajemy instrukcję:

DROP TABLE klienci;

**INSERT:** Służy do wypełniania tabeli danymi. Polecenie SQL dla przykładowej tabeli:

Służy do wypełniania tabeli danymi. Polecenie SQL dla przykładowej tabeli:

```
INSERT INTO pracownicy VALUES ('Jan', 'Kowalski', '2002-07-20', '1200.00');
```

Powyższe polecenie wstawi pojedynczy rekord do tabeli pracownicy. Aby wstawić kolejne rekordy należy ponówić powyższe zapytanie używając nowych danych:

```
INSERT INTO pracownicy VALUES ('Aleksander', 'Borowiecki', '1952-08-06',  
'1500.34');  
INSERT INTO pracownicy VALUES ('Aniela', 'Michałkowska', '1970-05-23',  
'854.29');  
INSERT INTO pracownicy VALUES ('Katarzyna', 'Kowalska', '2002-07-02',  
'1200.00');
```

Polecenia INSERT można użyć także w innej formie. Za nazwą tabeli można wyspecyfikować listę pól, które będziemy wypełniać danymi. Pozostałe pola przyjmą puste lub domyślne wartości. Zapytanie z wyspecyfikowaną listą pól wygląda następująco:

```
INSERT INTO pracownicy (imie, nazwisko) VALUES ('Izabela', 'Kwiatkowska');
```

**UPDATE:** Poleceniem INSERT można wstawiać nowe dane do istniejącej tabeli. Do modyfikacji danych już wcześniej umieszczonych w tabeli służy polecenie UPDATE. Jego składnia jest następująca:

```
UPDATE nazwa_tabeli SET nazwa_pola='nowa_wartość';
```

Po słowie kluczowym SET podajemy kolejno (po przecinku) nazwy kolumn wraz z nowymi wartościami, jakie powinny przyjąć. Po wydaniu poniższego polecenia wszystkie rekordy w polu imię będą miały wartość 'Genowefa':

```
UPDATE pracownicy SET imie='Genowefa';
```

Do określenia czego ma dotyczyć zmiana służy klauzula WHERE podawana na końcu polecenia UPDATE. W celu zmiany imienia tylko dla Izabeli Kwiatkowskiej polecenie UPDATE będzie wyglądać następująco:

```
UPDATE pracownicy SET imie='Genowefa' WHERE nazwisko='Kwiatkowska';
```

**DELETE:** Do usunięcia danych z tabeli służy polecenie DELETE.

**DELETE FROM nazwa\_tabeli;**

Można użyć warunku wyboru, dzięki któremu wyspecyfikujemy dane przeznaczone do usunięcia. Aby usunąć z przykładowej tabeli pracownicy wszystkie rekordy, w których płaca jest wyższa od 1000 należy wydać następujące polecenie:

**DELETE FROM pracownicy WHERE płaca > 1000;**

**INDEKSY:** w dużym uproszczeniu, powodują przyśpieszenie operacji wyszukiwania wykonywanych na bardzo dużych tabelach. Indeks należy zakładać dla pól, według których najczęściej jest wykonywane wyszukiwanie. Modyfikacja indeksów przebiega w podobny sposób do modyfikacji struktury tabeli. Dodanie indeksu na polu nazwisko będzie wyglądało następująco:

**ALTER TABLE pracownicy ADD INDEX nazwisko\_idx (nazwisko);**

Spowoduje to utworzenie indeksu o nazwie nazwisko\_idx opartego na kolumnie nazwisko. Usunięcie indeksu odbywa się za pomocą polecenia:

**ALTER TABLE DROP INDEX nazwisko\_idx;**

**SELECT:** Aby pobrać dane zapisane w tabeli należy użyć zapytania SELECT. Jego postać ogólna prezentuje się następująco:

```
SELECT co_zaprezentować FROM nazwa_tabeli
[WHERE warunki_wyszukiwania]
[ORDER BY sortowanie [ASC | DESC], ...]
[LIMIT [offset,] ilość_wierszy];
```

W miejscu co\_zaprezentować podaje się (po przecinku) listę kolumn, które chcemy zatrzymać w zestawieniu. W miejscu nazwa\_tabeli podajemy nazwę tabeli, z której pobieramy dane.

Wybierając trzy kolumny do zestawienia z tabeli pracownicy piszemy następująco:

```
SELECT imie, nazwisko, placa FROM pracownicy;
```

Spowoduje to wyświetlenie wszystkich rekordów, jednak w zestawieniu zostaną zaprezentowane jedynie wartości trzech pól: imię, nazwisko, płaca.

---

## *8. Zbiory i multizbiory: właściwości i operacje*

---

### ZBIORY:

1. Definicja:
  - Pojęcie pierwotne (nie definiuje się)
  - Intuicyjnie – Zestaw lub kolekcja pewnych elementów, zwykle podobnych jednego typu
  - George Cantor – Przez zbiór rozumiemy złączenie M określonych rozróżnialnych obiektów naszego doświadczenia poglądowego lub naszej wyobraźni w jedną całość
2. Właściwości:
  - Kolejność elementów w zbiorze nie jest istotna
  - Każdy element zbioru może wystąpić w nim tylko raz
  - Zbiory mogą być: skończone, nieskończoności, ciągłe, dyskretne
3. Sposoby definiowania zbiorów:  
Zbiory mogą być definiowane w następujący sposób:
  - Ekstensjonalnie – poprzez wyliczenie wszystkich elementów ( $\{0,1,2,3,4,5\}$ ,  $\{a,b,c,\dots,x,y,z\}$ )
  - Intensjonalnie – poprzez zdefiniowanie własności elementów, najczęściej z pewnego uniwersum – tzn. zadanie dziedziny i warunku ( $\{n \in N : 2|n\}$ )
  - Rekurencyjnie – poprzez zdefiniowanie pewnego elementu (elementów) bazowego i reguły produkcji ( $x_1 = 1$ ,  $x_i = x_{i-1} + x_{i-2}$  dla  $i > 3$ )

- Za pomocą operacji algebry zbiorów (zadawanie wtórne)
- Za pomocą operacji logicznych (zadawanie wtórne)

4. Operacje algebry zbiorów:

$$X \cup Y = \{x : x \in X \vee x \in Y\}$$

- Suma zbiorów:

$$X \cap Y = \{x : x \in X \wedge x \in Y\}$$

$$X \setminus Y = \{x : x \in X \wedge \neg x \in Y\}$$

$$\text{- Dopełnienie zbioru } X \text{ (do uniwersum } U\text{): } \overline{X} = U \setminus X$$

5. Prawa algebry zbiorów (przemienność sumy, przemienność iloczynu, łączność sumy, łączność iloczynu, rozdzielność sumy względem iloczynu, sumy, idempotentność, identyczność element naturalny, element przeciwny, prawo podwójnego dopełnienia, prawo De Morgana dla dopełnienia sumy, prawo De Morgana dla dopełnienia iloczynu):

- Przemienność sumy:  $X \cup Y = Y \cup X$
- Przemienność iloczynu:  $X \cap Y = Y \cap X$
- Łączność sumy:  $(X \cup Y) \cup Z = X \cup (Y \cup Z)$
- Łączność iloczynu:  $(X \cap Y) \cap Z = X \cap (Y \cap Z)$
- Rozdzielność sumy względem iloczynu:  $X \cup (Y \cap Z) = (X \cup Y) \cap (X \cup Z)$
- Rozdzielność iloczynu względem sumy:  $X \cap (Y \cup Z) = (X \cap Y) \cup (X \cap Z)$
- Idempotentność:  $X \cup X = X, X \cap X = X$
- Identyczność, element naturalny:  $X \cup \emptyset = X, X \cap U = X$
- Identyczność, element przeciwny:  $X \cup U = U, X \cap \emptyset = \emptyset$
- Prawo podwójnego dopełnienia:  $\overline{\overline{X}} = X$
- $X \cup \overline{X} = U, X \cap \overline{X} = \emptyset,$
- $\overline{U} = \emptyset, \overline{\emptyset} = U.$
- Prawo De Morgana dla dopełnienia sumy:  $\overline{X \cup Y} = \overline{X} \cap \overline{Y}$
- Prawo De Morgana dla dopełnienia iloczynu:  $\overline{X \cap Y} = \overline{X} \cup \overline{Y}$

**MULTIZBIORY (wielozbiory, zbiory z powtórzeniami):**

1. Definicja:

- Są to zbiory w których identyczne elementy mogą występować wielokrotnie
- Zbiorem z powtórzeniami M określonym nas pewnym zbiorem U nazywamy dowolny zbiór par  $\{(r_M(u), u) : u \in U, r_M(u) \in N \cup \{0\}\}$ , gdzie  $r_M$  jest funkcją typu  $r_M : U \rightarrow N \cup \{0\}$ .

2. Sposób zapisu:

$$M = \sum r_M(u) * u$$

$$M = \{(r_M(u_1), u_1), (r_M(u_2), u_2), \dots, (r_M(u_k), u_k)\}, \text{ dla } r_M(u) \neq 0$$

Liczbę  $r_M(u)$  nazywa się krotnością elementu  $u$  w zbiorze  $M$  lub współczynnikiem repetycji.

3. Operacje i prawa:

- Suma zbiorów (teoriomnogościowa):

$$M_1 \cup M_2 = \sum_{u \in U} \max(r_{M_1}(u), r_{M_2}(u)) * u$$

- Przecięcie (iloczyn teoriomnogościowy):

$$M_1 \cap M_2 = \sum_{u \in U} \min(r_{M_1}(u), r_{M_2}(u)) * u$$

- Różnica zbiorów:

$$M_1 \setminus M_2 = \sum_{u \in U} \max((r_{M_1}(u) - r_{M_2}(u)), 0) * u$$

- Dodawanie (suma arytmetyczna):

$$M_1 + M_2 = \sum_{u \in U} (r_{M_1}(u) + r_{M_2}(u)) * u$$

- Mnożenie skalarne:

$$k \cdot M = \sum_{u \in U} (k \cdot r_M(u)) * u$$

- Zawieranie:

$$M_1 \subseteq M_2 \text{ wtw. } \forall u \in U \ r_{M_1}(u) \leq r_{M_2}(u)$$

- Równość:

$$M_1 = M_2 \text{ wtw. } \forall u \in U \ r_{M_1}(u) = r_{M_2}(u)$$

---

## *9. Typy danych w SQL*

---

### Całkowitoliczbowe:

§ int

§ tinyint

§ smallint

§ bigint

### Zmiennoprzecinkowe:

- § decimal : Precyzja – jest to maksymalna ilość cyfr w liczbie zmiennoprzecinkowej zarówno przed jak i po przecinku. Maksymalnie typ decimal może mieć 38 liczb. Skala – jest to maksymalna ilość cyfr w liczbie zmiennoprzecinkowej po przecinku. Maksymalnie skala może być równa precyzji.

np.

- Liczba 18.667 posiada precyzję równą 5 i skalę równą 3
- Liczba 776378 posiada precyzję równą 6 i skalę równą 0
- Liczba 0.12387654321 posiada precyzję równą 11 i skalę równą 11
- § numeric = synonim to decimal

### Pieniężne:

- § money,

- § smallmoney

### Zmiennej dokładności:

- § real

- § float

### TRUE/FALSE:

§ bit

---

## 10. Klucz główny, klucz obcy

---

**Klucz główny** (ang. primary key) – wybrany minimalny zestaw atrybutów relacji, jednoznacznie identyfikujący każdy rekord tej relacji. To oznacza, że taki klucz musi przyjmować wyłącznie wartości niepowtarzalne i nie może być wartością pustą (null). Ponadto każda relacja może mieć najwyżej jeden klucz główny.

MySQL:

```
CREATE TABLE `nazwa_tabeli`  
{  
    `nazwa_kolumny1` typ_danych,  
    `nazwa_kolumny2` typ_danych,  
    PRIMARY KEY (`nazwa_kolumny1`)  
}
```

**Klucz obcy** – kombinacja jednego lub wielu atrybutów tabeli, które wyrażają się w dwóch lub większej liczbie relacji (tabel). Wykorzystuje się go do tworzenia powiązania pomiędzy parą tabel, gdzie w jednej tabeli ten zbiór atrybutów jest kluczem obcym, a w drugiej kluczem głównym.

Np. Jeśli w bazie "Firma" są tabele:

```
Oddział (id_oddziału, miejscowość, telefon, ...)  
Pracownik (id_pracownika, imię, nazwisko, id_oddziału, ...)
```

to kolumna Pracownik.id\_oddziału mogłaby być kluczem obcym wiążącym tę tabelę z tabelą oddział kolumną Oddział.id\_oddziału będącą w niej kluczem głównym.

---

11. Zapytania typu: *SELECT FROM WHERE GROUP BY HAVING ORDER BY*

---

```
SELECT <attributes>
FROM   <one or more relations>
WHERE  <conditions>
```

## Meaning (Semantics) of SQL Queries

```
SELECT x1.a1, x1.a2, ..., xn.ak
FROM   R1 AS x1, R2 AS x2, ..., Rn AS xn
WHERE  Conditions(x1,..., xn)
```

Almost never the fastest way  
to compute it!

```
Answer = {}
for x1 in R1 do
  for x2 in R2 do
    ...
    for xn in Rn do
      if Conditions(x1,..., xn)
        then Answer = Answer ∪ {x1.a1, x1.a2, ..., xn.ak}
return Answer
```

**Note:** this is a *multiset union*

1. Iloczyn Kartezjański dwóch lub więcej tabel
2. Zastosowanie warunków/selekacji
3. Zastosowanie projektji by otrzymać końcowy wynik.

```
SELECT PName, Price, Manufacturer
FROM   Product
WHERE  Category='gizmo' AND Price > 50
ORDER BY Price, PName
```

Słowo kluczowe ORDER BY służy do sortowania zestawu wyników w kolejności rosnącej lub malejącej. Domyślnie sortuje rekordy w porządku rosnącym. Aby posortować rekordy w kolejności malejącej, użyj słowa kluczowego DESC.

```
SELECT product,
       SUM(price * quantity) AS TotalSales
FROM   Purchase
WHERE  date > '10/1/2005'
GROUP BY product
```

GROUP BY- grupowanie na pewne kategorie

---

Dla polecenie SELECT FROM WHERE GROUP BY (kolejność wykonywanych operacji  
FROM-WHERE-GROUP BY-SELECT)

1. Najpierw będzie iloczyn Kartezjański
2. Rekordy zostaną pogrupowane przez polecenie GROUP BY
3. Na końcu zostanie wyliczone to co znajduje się w sekcji SELECT

```
SELECT product, SUM(price*quantity)
FROM Purchase
WHERE date > '10/1/2005'
GROUP BY product
HAVING SUM(quantity) > 100
```

Czym różni się HAVING od WHERE? Różnica polega na tym, że w HAVING wykorzystujemy funkcję agregującą, w WHERE nie możemy ich wykonać, ponieważ WHERE wykonuje się na samym początku na całym zbiorze i potem następuje grupowanie

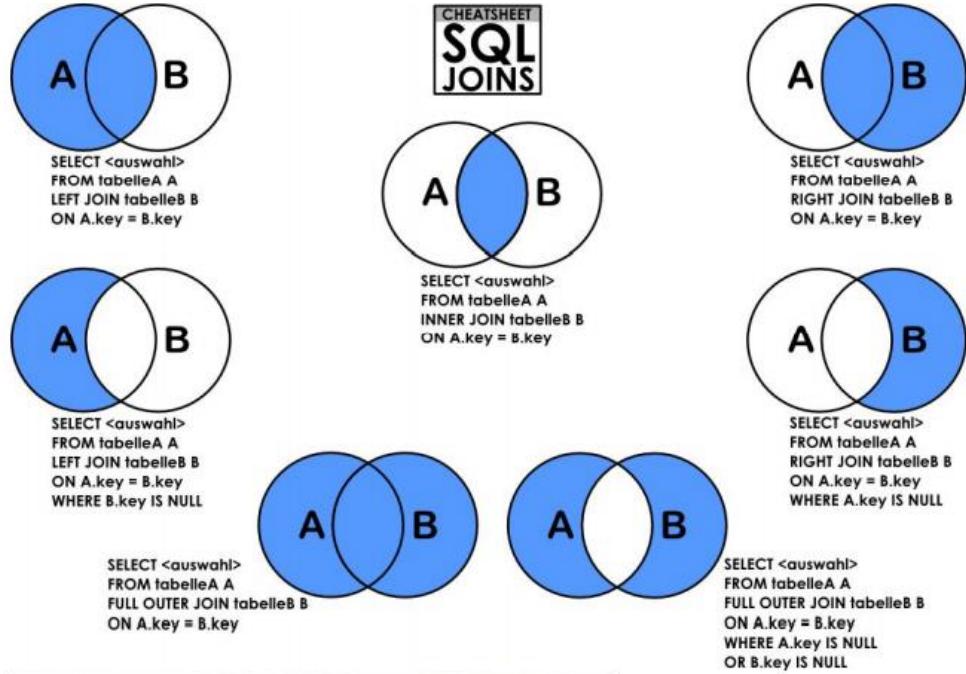
```
SELECT S
FROM R1,...,Rn
WHERE C1
GROUP BY a1,...,ak
HAVING C2
```

1. Najpierw wykonuje się część FROM-WHERE warunki C1 na atrybutach R<sub>1</sub>,...,R<sub>n</sub>
2. GROUP BY na atrybutach a<sub>1</sub>,...,a<sub>k</sub>
3. Dla każdej z tych grup sprawdzany jest warunek, który mamy w HAVING
4. Dopiero na końcu wykonywany jest SELECT

---

12. Złączenia tabel: INNER JOIN, LEFT JOIN, RIGHT JOIN, OUTER JOIN, iloczyn kartezjański

---



## INNER JOIN:

Product

name	category
Gizmo	gadget
Camera	Photo
OneClick	Photo

Purchase

prodName	store
Gizmo	Wiz
Camera	Ritz
Camera	Wiz

```
SELECT Product.name, Purchase.store
FROM Product
INNER JOIN Purchase
ON Product.name = Purchase.prodName
```

Note: another equivalent way to write an INNER JOIN!



name	store
Gizmo	Wiz
Camera	Ritz
Camera	Wiz

Słowo kluczowe INNER JOIN wybiera rekordy, które mają pasujące wartości w obu tabelach.

## LEFT OUTER JOIN:

The diagram shows two tables: **Product** and **Purchase**. The **Product** table has columns `name` and `category`, with rows: Gizmo (gadget), Camera (Photo), OneClick (Photo). The **Purchase** table has columns `prodName` and `store`, with rows: Gizmo (Wiz), Camera (Ritz), Camera (Wiz). A SQL query is shown: `SELECT Product.name, Purchase.store FROM Product LEFT OUTER JOIN Purchase ON Product.name = Purchase.prodName`. An arrow points from the query to the result table, which contains rows: Gizmo (Wiz), Camera (Ritz), Camera (Wiz), and OneClick (NULL).

Product		Purchase	
name	category	prodName	store
Gizmo	gadget	Gizmo	Wiz
Camera	Photo	Camera	Ritz
OneClick	Photo	Camera	Wiz

name	store
Gizmo	Wiz
Camera	Ritz
Camera	Wiz
OneClick	NULL

LEFT OUTER JOIN - LEFT JOIN - zwraca wszystkie rekordy z lewej tabeli Product, a dopasowane rekordy z prawej tabeli Purchase. Wynik jest NULL z prawej strony, jeśli nie ma dopasowania.

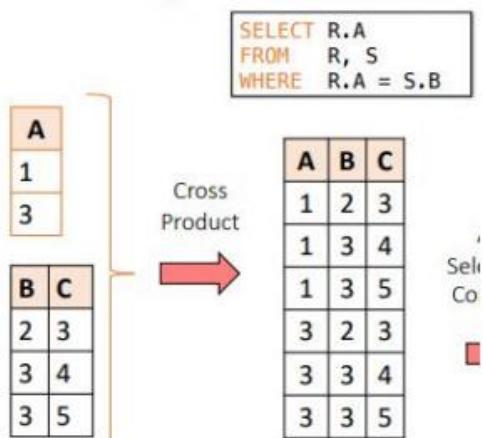
## Other Outer Joins

- Left outer join:
  - Include the left tuple even if there's no match
- Right outer join:
  - Include the right tuple even if there's no match
- Full outer join:
  - Include the both left and right tuples even if there's no match

Słowo kluczowe RIGHT JOIN zwraca wszystkie rekordy z prawej tabeli B, a dopasowane rekordy z lewej tabeli A. Wynik jest NULL z lewej strony, gdy nie ma dopasowania.

Słowo kluczowe FULL OUTER JOIN (FULL JOIN) zwraca wszystkie rekordy, gdy występuje dopasowanie w rekordach tabeli lewej A lub prawej B.

Iloczyn Kartezjański dla operacji SELECT FROM JOIN



$X=R \times S$ , Iloczyn kartezjański powoduje że dany atrybut może pojawić się wielokrotnie

---

## [13. Zapytania zagnieżdżone](#)

---

### **Podzapytania zagnieżdżone:**

Podzapytania służą do zagnieżdżania zapytań – jedno zapytanie może zawierać w klauzuli WHERE innego, zagnieżdżone pytanie. Używane są one, gdy jedno zapytanie ma bazować na wyniku drugiego zapytania.

Podzapytanie polega na umieszczeniu instrukcji SELECT wewnętrznej wewnątrz innej instrukcji SELECT. Serwer bazodanowy w pierwszej kolejności wykonuje zapytania wewnętrzne. Wynik zapytania wewnętrznego zwracany jest do zapytania zewnętrznego.

Podzapytanie zagnieżdżone SELECT znajduje się wewnętrzne zewnętrzne zapytania SELECT, np. po klauzuli WHERE, HAVING lub FROM.

Na końcu zapytania wewnętrznego nie ma średnika, zapisuje się je w nawiasach okrągłych.

Przykład: z bazy firma z tabeli emp wybrać imiona i nazwiska pracowników, którzy zarabiają więcej niż średnia płaca w firmie.

```
1 SELECT first_name AS imie, last_name AS nazwisko, salary AS plac
2 FROM emp
3 WHERE salary > (SELECT AVG(salary)
4   FROM emp)
5 ORDER BY nazwisko, imie;
```

Przykład: z bazy danych firma z tabeli emp wybrać imiona, nazwiska, wynagrodzenie, ID departamentu pracowników, którzy mają najwyższe wynagrodzenia w tym departamencie.

```
1 SELECT first_name, last_name, salary, dept_id
2 FROM emp
3 WHERE (salary, dept_id) IN (SELECT MAX(salary), dept_id
4   FROM emp
5   GROUP BY dept_id)
6 ORDER BY salary DESC;
```

---

## [14. . NULL w SQL: operatory logiczne i NULL, IS NULL, IS NOT NULL](#)

---

Błędy powstają często przez nie uwzględnienie wartości NULL. NULL- oznacza że jakaś wartość nie występuje lub że jakaś wartość nie jest znana, lub ze nas nie dotyczy.

NULL sie propaguje jeżeli na wejściu jest NULL to na wyjściu też będzie NULL

FUNKCJE:

$x \text{ IS NULL}$  - jeżeli  $x$  jest NULL to prawda (może być wartość F lub P jedynie wartości NULL nie będzie nigdy)

$x \text{ IS NOT NULL}$  - jeżeli  $x$  jest NULL to fałsz (może być wartość F lub P jedynie wartości NULL nie będzie nigdy)

## Null Values

- $p \text{ AND } q = \min(p, q)$
- $p \text{ OR } q = \max(p, q)$
- $\text{NOT } p = 1 - p$

$p$	$q$	$p \text{ OR } q$	$p \text{ AND } q$	$p = q$
True	True	True	True	True
True	False	True	False	False
True	Unknown	True	Unknown	Unknown
False	True	True	False	False
False	False	False	False	True
False	Unknown	Unknown	False	Unknown
Unknown	True	True	Unknown	Unknown
Unknown	False	Unknown	False	Unknown
Unknown	Unknown	Unknown	Unknown	Unknown

---

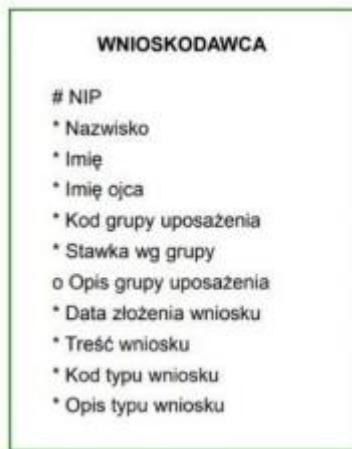
### 15. Postacie normalne relacyjnych baz danych: zerowa, pierwsza, druga, trzecia, BCNF, czwarta, piąta

---

#### Zerowa postać normalna (0PN):

Każda encja musi mieć zbiór atrybutów (związków), które w sposób unikalny określają jej wystąpienie.

Przykład:  
diagram w 0PN



#### Pierwsza postać normalna (1PN):

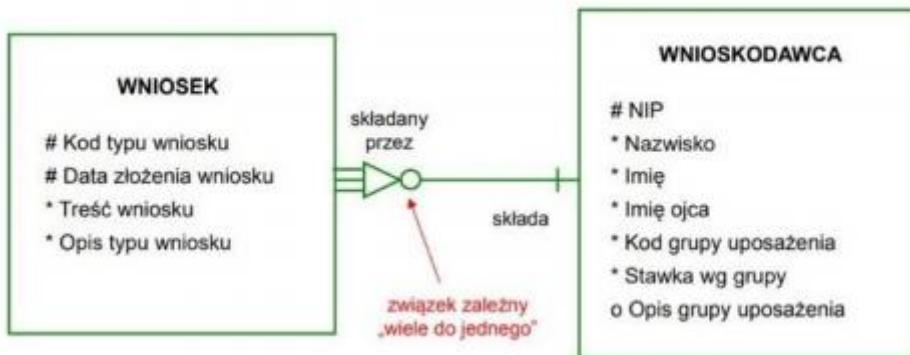
Każdy atrybut może mieć tylko jedną wartość dla każdego wystąpienia jego encji w danej chwili.

Przejście z 0PN do 1PN:

-usunięcie atrybutów wielowartościowych i utworzenie dla nich nowej encji,

- identyfikator „starej” encji wchodził będzie w skład złożonego identyfikatora „nowej” encji (związek zależny)

Diagram w 1PN



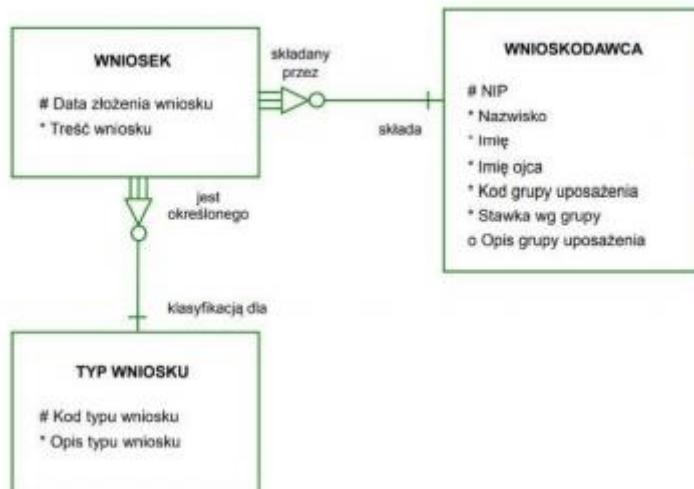
#### Druga postać normalna (2PN) (dotyczy encji z identyfikatorami złożonymi z kilku atrybutów):

Wartość każdego atrybutu encji musi zależeć od całego identyfikatora tej encji (a nie tylko od niektórych atrybutów składających się na ten identyfikator).

Przejście z 1PN do 2PN:

- usunięcie wszystkich częściowo zależnych atrybutów i utworzenie dla nich nowej encji,
- skopiowanie części identyfikatora z encji pierwotnej (od której zależne są usunięte atrybuty) do tej nowej encji).

Diagram w 2PN



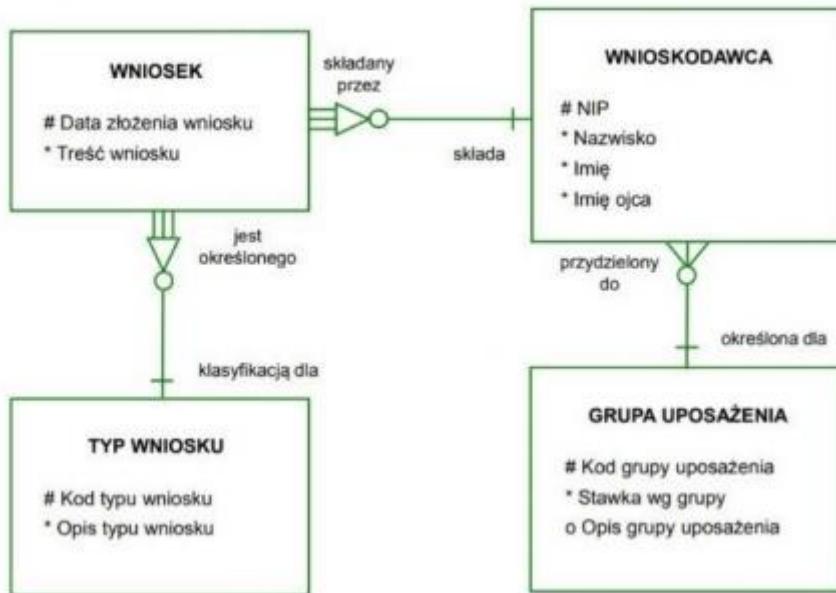
### Trzecia postać normalna (3PN):

Wartość atrybutu encji nie może zależeć od wartości innego atrybutu, nie wchodzącego w skład jej identyfikatora.

Przejście z 2PN do 3PN:

- należy usunąć atrybuty niezależne od identyfikatora i wstawić je do nowej encji,
- identyfikatorem nowej encji staje się atrybut, od którego zależą jej pozostałe atrybuty.

Diagram w 3PN



### Postać normalna Boyce'a-Codd'a (BCNF):

Relacja R jest w postaci normalnej Boyce'a-Codda, jeśli dla każdej nietrywialnej zależności funkcyjnej  $X \rightarrow A$  w  $R$   $X$  jest nadkluczem schematu relacji  $R$ .

Uwagi:

- każda relacja, która jest w BCNF, jest także w 3NF
- dekompozycja relacji powinna prowadzić przynajmniej do 3NF lub BCNF

Podstawa do porównania	3NF	BCNF
Pojęcie	Żaden atrybut inny niż główny nie może być przejściowo zależny od klucza Kandydata.	Dla każdej trywialnej zależności w relacji R powiedzmy $X \rightarrow Y$ , X powinien być super kluczem relacji R.
Zależność	3NF można uzyskać bez poświęcania wszystkich zależności.	Zależności mogą nie zostać zachowane w BCNF.
Rozkład	Bezstratny rozkład można uzyskać w 3NF.	Bezstratny rozkład trudno osiągnąć w BCNF.

Czwarta postać normalna (4PN):

Nr_stud	Przedmiot	Sport
100	Bazy danych	Tenis
100	Bazy danych	Biegi
100	Systemy informacyjne	Tenis
100	Systemy informacyjne	Biegi
200	Bazy danych	Boks

W relacji  $R = \{Nr\_stud, Przedmiot, Sport\}$  mamy do czynienia z tak zwanyimi zależnościami wielowartościowymi:

$Nr\_stud \rightarrow> Przedmiot$ ;  $Nr\_stud \rightarrow> Sport$

Schemat relacji jest w czwartej postaci normalnej, jeśli nie ma w nim zależności wielowartościowych. Powyższy schemat R nie jest więc w czwartej postaci normalnej.

Aby wyeliminować zależności wielowartościowe rozkładamy R na dwie relacje o schematach:

$\{Nr\_stud, Przedmiot\}$  i  $\{Nr\_stud, Sport\}$

Piąta postać normalna (5PN):

Rozważmy relację między dostawcami, produktami i projektami:

Nazwa_dostawcy	Nazwa_produktu	Nazwa_projektu
Kowalski	Stal	Wenus
Kowalski	Srebro	Mars
Kowalski	Stal	Mars
Jankowski	Stal	Mars
Jankowski	Papier	Wenus
Jankowski	Stal	Wenus
Misiak	Srebro	Neptun

Przyjmijmy następującą zasadę biznesową:

Jeśli

(1) dostawca X dostarcza produkt Y,

(2) dostawca X pracuje dla projektu Z,

(3) projekt Z używa produktu Y

To

(4) dostawca X dostarcza produkt Y dla projektu Z

Przy tych zasadach zapis informacji w tabeli jest redundantny, bo skoro

- (1) Kowalski jest dostawcą stali (dostarcza dla projektu Wenus),
- (2) Kowalski pracuje dla projektu Mars (dostarcza srebro),
- (3) projekt Mars używa stali (od dostawcy Jankowskiego) to redundantna jest już informacja, że:
- (4) Kowalski dostarcza stali dla projektu Mars (pokolorowany wiersz w tabelce).

Rozwiązaniem problemu jest podział relacji na trzy relacje:

- (1) Dostawcy-produkty,
- (2) Dostawcy-projekty,
- (3) Projekty-produkty

Podział na tylko dwie relacje jest niewystarczający!

W relacjach (1)-(3) nie ma już redundancji, a ich złączenie daje wyjściową relację.

---

*16. Anomalie w relacyjnych bazach danych: redundancja, aktualizacji, kasowania, wstawiania*

---

- **Redundancja** — ta sama informacja jest niepotrzebnie przechowywana w kilku krotkach.
- **Anomalia modyfikacji** — informacja zostanie zmodyfikowana w pewnych krotkach, a w innych nie. Która informacja jest wówczas prawdziwa?
- **Anomalia usuwania** — usuwanie części informacji powoduje utratę innej informacji, której nie chcielibyśmy stracić.
- **Anomalia dołączania** — wprowadzenie pewnej informacji jest możliwe tylko wtedy, gdy jednocześnie wprowadzamy jakąś inną informację, która może być obecnie niedostępna.

## Przykład

Nazwa specjalizacji	Numer lekarza	Nazwisko lekarza	Imię lekarza	Tytuł lekarza	Nr pacjenta	Nazwisko pacjenta	Imię pacjenta	Wiek pacjenta	Ubezpieczenie
internista	222	Lubicz	Jan	Lek. med.	158	Zawada	Janusz	50	NFZ
internista	222	Lubicz	Jan	Lek. med.	159	Ziobro	Jakub	41	Brak
gastrolog	555	Nowak	Anna	Dr	260	Kowal	Jan	59	NFZ
nefrolog	444	Burski	Jakub	Dr	499	Bąk	Miron	25	Brak
pediatra	777	Jarosz	Piotr	Lek. med.	248	Kmiec	Anna	12	NFZ
neurolog	333	Sum	Jerzy	Prof.	239	Nowak	Anna	31	NFZ
neurolog	666	Sum	Jerzy	Prof.	159	Ziobro	Jakub	42	Brak

1. Usunięcie pacjenta Mirona Bąka, nr 499, spowoduje utratę informacji o lekarzu Jakubie Burskim (**anomalia usuwania danych**).
2. Zmiana lekarza na stanowisku internisty spowoduje konieczność zmiany imienia i nazwiska we wszystkich wierszach, w których jego dane się powtarzają (**anomalia modyfikacji danych**).
3. Podczas zapisywania do szpitala nowego pacjenta musielibyśmy wielokrotnie powtarzać jego dane przy wszystkich lekarzach do których byłby zapisany (**anomalia wstawiania**).

---

### 17. Zależności funkcjonalne: trzy zasady Armstrong'a, domknięcie zbioru relacji, algorytm znajdowania domknięcia zbioru relacji

---

Zależność funkcjonalna: Niech A i B będą zestawami atrybutów. Zapisujemy  $A \rightarrow B$  lub mówimy, że B jest funkcjonalnie zależne od A wtedy i tylko wtedy, gdy  $t1[A] = t2[A]$  implikuje  $t1[B] = t2[B]$ . Nazywamy  $A \rightarrow B$  funkcjonalną zależnością.

Zależność funkcyjna (FD) występuje wtedy, gdy po ustaleniu wartości pewnych atrybutów relacji wartości jakichś innych atrybutów tej relacji są jednoznacznie wyznaczone (unikalne). Używa się notacji:  $A \rightarrow B$ , gdzie A i B są zbiorami atrybutów z relacji R. Mówimy, że zależność funkcyjna  $A \rightarrow B$  zachodzi w R, jeśli każde dwa wiersze mające te same wartości atrybutów z A muszą mieć te same wartości atrybutów z B.

Reguły wnioskowania Armstronga Niech A, B i C będą dowolnymi podzbiorami zbioru atrybutów relacji R, (AB oznacza sumę A i B) wówczas zachodzą związki:

- zwrotność: jeżeli  $B \subseteq A$  to  $A \rightarrow B$
- dołączenie (rozszerszanie): jeżeli  $A \rightarrow B$  to  $AC \rightarrow BC$
- przechodniość: jeżeli  $A \rightarrow B$  i  $B \rightarrow C$  to  $A \rightarrow C$

Domknięcie zbioru zależności A (oznaczane  $A^+$ ) jest to zbiór wszystkich zależności, które można wyprowadzić z A. (zbiór wszystkich zależności funkcyjnych będących konsekwencjami zbioru A)

#### Algorytm:

- zacznij od  $X = \{A_1, \dots, A_n\}$  i zależności funkcyjonalnych F

- powtarzaj, dopóki X się nie zmienia.
  - jeśli  $\{B_1, \dots, B_n\} \rightarrow C$  pociąga za sobą F i  $\{B_1, \dots, B_n\}$  zawiera się w X, dodaj C do zbioru X.
  - zwróć X jako X+
- 

#### *18. Superklucz i klucz w relacji*

---

**Superklucz (superkey)** – to kolumna lub zestaw kolumn jednoznacznie identyfikujących każda krotkę tabeli. Super klucz może zawierać kolumny, które samodzielnie mogą nie identyfikować każdej z krotek. Unikatowa identyfikacja każdej z krotek może odbywać się jedynie przez zestaw np. dwóch lub trzech atrybutów. Przedmiotem zainteresowań projektantów baz danych jest taki superklucz, który zawiera minimalny zestaw atrybutów unikalnie identyfikujących krotkę.

**Klucze** są podstawową koncepcją w teorii relacyjnych baz danych. Zapewniają tabelom możliwość skorelowania ze sobą dwóch lub więcej tabel. Nawigacja w relacyjnej bazie danych zależy od możliwości identyfikacji określonego wiersza w tabeli za pomocą klucza głównego. Artykuł ten poświęcony będzie poświęcony teorii kluczy głównych i obcych oraz zasadom ich działania

Związek pomiędzy dwoma lub większą liczbą tabel to asocjacja (w potocznym bazodanowym języku znana również jako relacja). Asocjacja jest wyrażona za pomocą wartości klucza głównego i kluczy obcych.

---

#### *19. Dekompozycja: stratna i bezstratna*

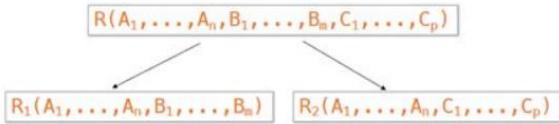
---

**Dekompozycja** – rozbicie jednej tabeli na mniejsze.

**Dekompozycja bezstratna** – rozbicie jednej tabeli na mniejsze nie tracąc informacji np. relacji między encjami.

**Dekompozycja stratna** – rozbicie jednej tabeli na mniejsze, tracąc informacje w poszczególnych tabelach

## Decompositions in General



$R_1$  = the projection of  $R$  on  $A_1, \dots, A_n, B_1, \dots, B_m$

$R_2$  = the projection of  $R$  on  $A_1, \dots, A_n, C_1, \dots, C_p$

81

## Theory of Decomposition

Name	Price	Category
Gizmo	19.99	Gadget
OneClick	24.99	Camera
Gizmo	19.99	Camera

Name	Price
Gizmo	19.99
OneClick	24.99
Gizmo	19.99

Name	Category
Gizmo	Gadget
OneClick	Camera
Gizmo	Camera

Sometimes a decomposition is "correct"

I.e. it is a Lossless decomposition

82

## 20. Relacje wielowartościowe

### Zależność wielowartościowa MVD

Niech  $R$  będzie relacją, zaś  $A, B, C$  będą dowolnymi podzbiorami zbioru atrybutów  $R$ . Mówimy, że  $B$  jest wielowartościowo zależne (MVD - multivalued dependencies) od  $A$

$$A \twoheadrightarrow B$$

wtedy i tylko wtedy, gdy zbiór wartości  $B$  odpowiadający danej parze  $(A, C)$  w  $R$  zależy od wartości  $A$  i jest niezależny od wartości  $C$ .

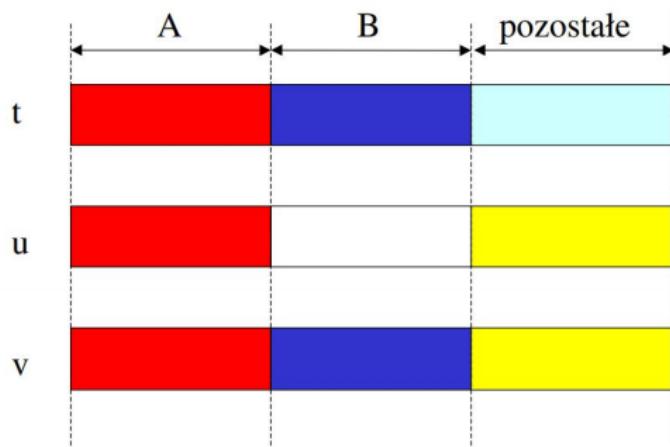
przedmiot  $\twoheadrightarrow$  wykładowca przedmiot  $\twoheadrightarrow$  podręcznik lub w skrócie przedmiot  $\twoheadrightarrow$  wykładowca | podręcznik

## Zależności wielowartościowe definicja

- Zależność wielowartościowa  $A_1A_2\dots A_n \rightarrow\rightarrow B_1B_2\dots B_m$  zachodzi w relacji R wówczas, gdy wybierając z relacji R te krotki, które są zgodne dla atrybutów typu A, zbiór wartości atrybutów typu B nie zależy od żadnych wartości tych atrybutów z R, których nie ma ani w zbiorze A, ani w B.
- Dla każdej pary krotek  $t$  i  $u$  z relacji R, które mają takie same wartości atrybutów typu A, można znaleźć w R taką krotkę  $v$ , której składowe mają wartości równe:
  - wartościom atrybutów typu A w krotkach  $t$  i  $u$
  - wartościom atrybutów typu B krotki  $t$
  - wartościom tych składowych krotki  $u$ , które nie są ani typu A, ani typu B.

## Zależności wielowartościowe

$A \rightarrow\rightarrow B$



## Wnioskowanie z zależności wielowartościowych

- Reguła zależności trywialnych
  - jeśli w pewnej relacji zachodzi zależność  $A_1A_2...A_n \rightarrow\rightarrow B_1B_2...B_m$  to wówczas, gdy  $C_1C_2...C_k$  są wszystkimi atrybutami B oraz część z nich jest typu A, zachodzi również
$$A_1A_2...A_n \rightarrow\rightarrow C_1C_2...C_k$$
- Reguła przechodniości
  - jeśli zachodzą zależności  $A_1A_2...A_n \rightarrow\rightarrow B_1B_2...B_m$  i  $B_1B_2...B_m \rightarrow\rightarrow C_1C_2...C_k$  to zachodzi również zależność
$$A_1A_2...A_n \rightarrow\rightarrow C_1C_2...C_k$$
- Zależności wielowartościowe nie spełniają reguły podziału i łączenia.

Bazy danych (studia zaoczne)

10

## Wnioskowanie z zależności wielowartościowych cd

- Każda zależność funkcyjna jest zależnością wielowartościową, czyli jeśli  $A_1A_2...A_n \rightarrow B_1B_2...B_m$  to  $A_1A_2...A_n \rightarrow\rightarrow B_1B_2...B_m$
- Reguła dopełnienia
  - jeśli  $A_1A_2...A_n \rightarrow\rightarrow B_1B_2...B_m$  zachodzi dla relacji R, to w R zachodzi również  $A_1A_2...A_n \rightarrow\rightarrow C_1C_2...C_k$ , gdzie atrybuty typu C są wszystkimi tymi atrybutami R, które nie są ani typu A, ani typu B.
- Przykład:
  - Relacja Aktorzy (nazwisko, ulica, miasto, tytuł, rok)
  - Zależności wielofunkcyjne nazwisko->-> ulica miasto, z reguły dopełnienia mamy też zależność nazwisko ->-> tytuł rok, która jest również spełniona

Bazy danych (studia zaoczne)

11

---

## 21. Denormalizacja bazy danych: cele i strategie

---

Denormalizacja to zbiór praktyk/doświadczeń, które mają służyć temu by przyspieszyć działanie danej bazy danych, konsekwencja będzie pojawić się anomali. Denormalizacje wykonuje się po jakimś czasie gdy baza danych ma pełno danych i gdy znamy charakterystykę zapytań

Cele:

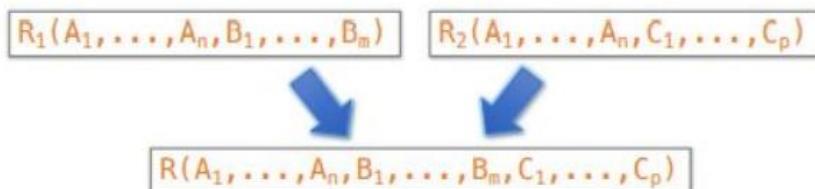
- Normalizacja nie jest działaniem, które musi być w pełni wykonane czasem w miarze upływu cyklu życia aplikacji może okazać się że wydajność aplikacji jest nie wystarczająca
- by ograniczyć korzystania z zapytania JOIN
- - trzeba się zastanowić przed denormalizacją czy większym problemem jest brak wydajności czy możliwość pojawięcia się anomalii
- zapytanie JOIN bywa bardzo kosztowne pod względem czasu przetwarzania

Strategie:

- Łączenie tabel - działanie odwrotne w stosunku do dekompozycji

## Denormalization strategies

### Collapsing Tables



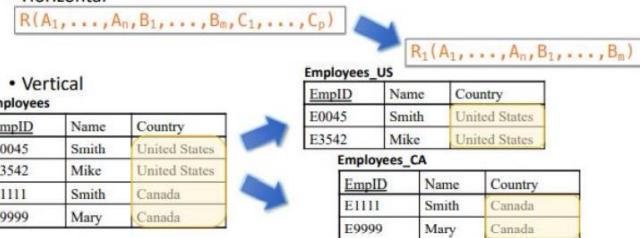
Mamy dwie relacje i z powrotem przywracamy ta relację pierwotną przed dekompozycją efekt jest taki że w przypadku zapytania o dane tej relacji R nie musimy wykonywać operacji JOIN.

- Podział tabel wertykalny/horyzontalny

### Denormalization strategies

#### Splitting Tables

- Horizontal



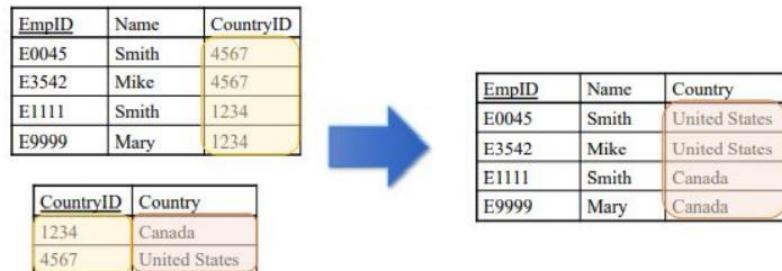
- podział w pionie - mamy tabele która zawiera bardzo dużo kolumn, część z tych kolumn jest rzadko używana w zapytaniach, możemy zatem stworzyć widok na dane które nas interesują, lub możemy stworzyć widok zmaterializowany lub podzielić relacje ze względu na jakieś id, by ilość mnożeń przy iloczynie kartezjańskim była jak najmniejsza

- podział poziomy - zostawiamy wszystkie kolumny, możemy podzielić tabele ze względu na jakąś kolumnę, przeszukiwanie tabeli będzie znacznie szybsze

- Usuwanie słowników identyfikator-nazwa (liczby mniej pamięci zajmują w pamięci, unikamy błędów np. Kanada, Canada)

### Denormalization strategies

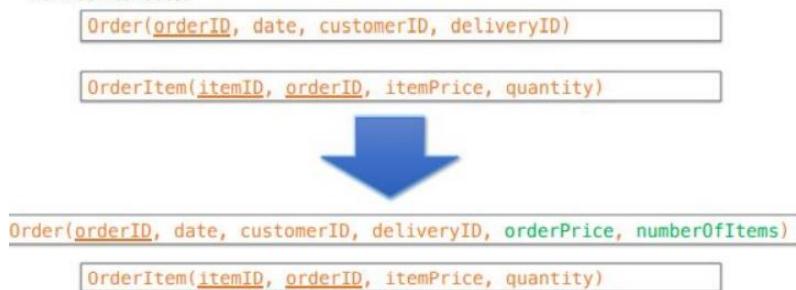
#### Removing dictionaries



- Wyliczanie atrybutów pochodnych - żeby przyśpieszyć operacje można dodać do tabeli nowe atrybuty, które są atrybutami wyliczonymi na podstawie danych zawartych w innej tabeli tak jak w przykładzie np.  $\text{orderPrice}=\text{SUM}(\text{itemPrice} * \text{quantity})$ , daje to nam, że jeżeli będziemy chcieli podsumować sprzedawcę pod koniec miesiąca i będziemy chcieli wiedzieć ile sztuk towaru sprzedaliśmy to nie będziemy musieli odpytywać w ogóle tej tabeli OrderItem wiec to nam przyspieszy odpowiedź,  $\text{orderPrice}$  - wartość statyczna do końca życia bazy danych będzie poprawna.

## Denormalization

Derived Attributes




---

### 22. Właściwości dysków HDD i SSD oraz pamięci RAM w kontekście pracy baz danych

---

#### Dysk:

- wolny (dostęp sekwencyjny – chociaż szybkie sekwencyjne odczyty)
- trwały – zakładamy, że dane na dysku są bezpieczne
- tani
- duża pojemność

#### RAM lub pamięć główna:

- szybki (losowy dostęp) -> 10x szybszy dostęp sekwencyjny, 100 000x szybszy dostęp losowy
- lotny -> dane mogą zostać utracone np. w przypadku awarii
- drogi
- ograniczona pojemność

#### Typy regionów pamięci:

- lokalny – każdy proces w DBMS ma własną pamięć lokalną, w której przechowuje wartości, które tylko „widzi”

- globalny - Każdy proces może odczytywać / zapisywać dane współdzielone w pamięci głównej
  - dysk – pamięć globalna może odczytać dane/zapisać na dysk z głównej pamięci na dysk (flushing - zapisywanie na dysk z głównej pamięci)
  - dziennik baz danych (log) – obejmuje pamięć główną i dysk
- 

### *23. Transakcje: definicja, ACID, zabezpieczenie przed awarią, wielodostęp*

---

Transakcje to jedno z podstawowych pojęć współczesnych systemów baz danych. Umożliwiają one współbieżny dostęp do zawartości bazy danych, dostarczając niezbędnych mechanizmów synchronizacji. Istotą transakcji jest integrowanie kilku operacji w jedną niepodzielną całość.

Np. W systemie bankowym jest wykonywany przelew 100 złp z konta bankowego jednego klienta (np. Kangurzycy) na konto innego klienta (np. Tygrysa). W SQL wygląda to zapewne następująco:

```
UPDATE Konta SET saldo = saldo - 100.00  
WHERE klient = 'Kangurzyca';  
  
UPDATE Konta SET saldo = saldo + 100.00  
WHERE klient = 'Tygrys';
```

Co stanie się, jeśli po wykonaniu pierwszego polecenia nastąpi awaria dysku? Podobny problem występuje nawet przy pojedynczym poleceniu SQL, jeśli modyfikuje ono wiele wierszy.

Rozwiązywanie takich problemów to transakcyjny system baz danych. Gwarantuje on zapisanie modyfikacji w sposób trwały przed zakończeniem transakcji, a jeśli się nie uda to transakcja jest wycofywana.

#### **ACID**

Transakcja to ciąg operacji do wspólnego niepodzielnego wykonania. Współbieżne wykonywanie transakcji wymaga zachowania własności ACID (Atomicity, Consistency, Isolation, Durability):

- niepodzielności: „wszystko-lub-nic”, transakcja nie może być wykonana częściowo;
- integralności: po zatwierdzeniu transakcji muszą być spełnione wszystkie warunki poprawności nałożone na bazę danych;
- izolacji: efekt równoległego wykonania dwu lub więcej transakcji musi być szeregowalny;
- trwałości: po udanym zakończeniu transakcji jej efekty na stałe pozostają w bazie danych.

#### **Zabezpieczenie przed awarią**

Bezpieczeństwo serwera baz danych to:

- zapewnienie stabilnego i w miarę możliwości bezawaryjnego działania serwera baz danych

- zapewnienie uprawnionym użytkownikom dostępu do odpowiednich baz danych
- ograniczenie dostępu do danych dla użytkowników nieuprawnionych
- zapewnienie jak najmniejszej ingerencji serwera baz danych w działanie systemu operacyjnego komputera

Bezpieczeństwo baz danych natomiast dotyczy następujących aspektów:

- umożliwienie tylko autoryzowanym użytkownikom wykonywania odpowiednich operacji na bazie danych
- zapewnienie bezpieczeństwa fizycznego bazy danych (odpowiednia strategia kopii zapasowych)

#### **Poziomy bezpieczeństwa:**

W najogólniejszym ujęciu można wyodrębnić następujące poziomy bezpieczeństwa:

- bezpieczeństwo fizyczne danych
- bezpieczeństwo sieci
- bezpieczeństwo domeny
- bezpieczeństwo maszyny lokalnej
- bezpieczeństwo serwera baz danych
- bezpieczeństwo bazy danych
- bezpieczeństwo aplikacji bazodanowe

**Wielodostęp** - system kontroli wielodostępu, który umożliwia jednoczesny dostęp do bazy dla wielu użytkowników

---

#### *24. COMMIT, ROLLBACK*

---

COMMIT i ROLLBACK są wykonywane na transakcjach. Transakcja to najmniejsza jednostka pracy wykonywana na bazie danych. Jest to sekwencja instrukcji w logicznej kolejności. Transakcja może być wykonana ręcznie przez programistę lub może zostać uruchomiona za pomocą automatycznego programu.

COMMIT jest poleceniem SQL służącym do przechowywania zmian wykonanych przez transakcję. Po wydaniu polecenia COMMIT zapisuje wszystkie zmiany od ostatniego polecenia COMMIT lub ROLLBACK.

```
1 BEGIN TRANSACTION
2 UPDATE Person.Address
3 SET AddressLine1 = 'Prosta 51'
4 WHERE AddressID = 1
5 UPDATE Person.Address
6 SET AddressLine1= 'Przyokopowa 31'
7 WHERE AddressID = 2
8 COMMIT TRANSACTION
```

ROLLBACK to polecenie SQL używane do przywracania zmian wykonanych przez transakcję. Wydanie polecenia ROLLBACK powoduje cofnięcie wszystkich zmian od ostatniego polecenia COMMIT lub ROLLBACK.

```
1 BEGIN TRANSACTION
2 UPDATE Person.Person
3 SET Title = 'TS'
4 WHERE FirstName = 'Ken' AND LastName='Sánchez'
5 IF @@ROWCOUNT = 2
6     COMMIT TRANSACTION
7 ELSE
8     ROLLBACK TRANSACTION
```

---

#### 25. Dziennik bazy danych (*log*) - typy dzienników: UNDO, REDO, UNDO/REDO

---

## Struktury danych – dziennik transakcji (*log*)

- Dziennik transakcji, DzT, (ang. *log*) – znajduje się w pamięci stałej.
- Zakładamy, że w dzienniku transakcji zapamiętane są wszystkie ostatnio zatwierdzona wartości wszystkich danych bazy danych (w praktyce stosuje się różne zabiegi optymalizacyjne).
- Dzięki DzT możliwe jest określenie ostatnio zatwierdzonych danych (należy w tym celu przeglądać dziennik z dołu do góry).
- Zawartość dziennika transakcji może być traktowana jako 3-kolumnowa tabela o następującej strukturze:

T	X	V
T <sub>1</sub>	x <sub>1</sub>	v <sub>1</sub>
T <sub>2</sub>	x <sub>2</sub>	v <sub>2</sub>
T <sub>1</sub>	x <sub>1</sub>	v <sub>3</sub>
...	...	...

gdzie (t, x, v) ∈ DzT oznacza, że transakcja t nadała danej x wartość v.

**WAŻNE! (najbardziej odpowiadający na pytanie slajd):**

## Algorytm UNDO/REDO

---

- Algorytm UNDO/REDO opisuje proces przetwarzania transakcji z uwzględnieniem możliwości odtworzenia bazy danych w przypadku awarii.
- Odtwarzanie bazy danych następuje w chwili restartu systemu po jego awarii.
- Algorytm UNDO/REDO jest stosowany wówczas, gdy w procesie odtwarzania konieczne jest wykonanie operacji UNDO dotyczącej przerwanych transakcji oraz operacji REDO dotyczącej zatwierdzonych transakcji.

Istnieją 3 rodzaje list transakcji:

- LTA - lista transakcji aktywnych,
- LTZ - lista transakcji zatwierdzonych,
- LTO - lista transakcji odrzuconych.

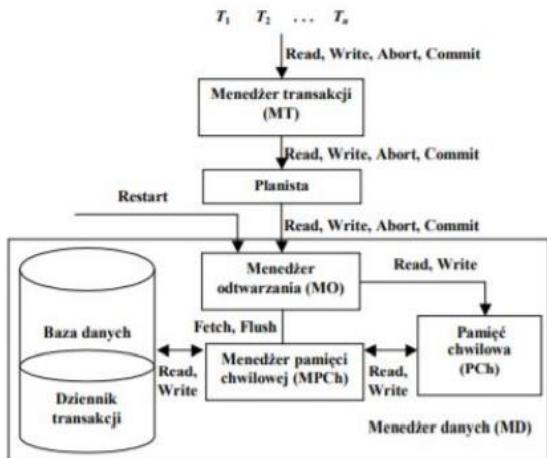
## Algorytm UNDO/REDO (c.d.)

- UNDO oznacza przywrócenie poprzednich (zatwierdzonych) wartości tym danym, które były zmieniane przez transakcje przerwane przed ich zatwierdzeniem.
- Operacja jest konieczna wtedy, gdy strategia wymiany stron w PCh prowadzona jest przez MPCh niezależnie od MO.
- Może więc się zdarzyć, że blok danych z PCh został zapamiętany w bazie danych w momencie, gdy transakcja zapisująca w nim dane nie została jeszcze zatwierdzona. W przypadku awarii należy więc te zmiany wycofać.

## Algorytm UNDO/REDO (c.d.)

- REDO oznacza ponowne wykonanie operacji aktualizacji danych wykonanych przez transakcje zatwierdzone.
- Operacja ta jest konieczna wtedy, gdy MPCh prowadzi własną strategię wymiany stron.
- Może więc się zdarzyć, że mimo iż transakcja została zatwierdzona, to wykonane przez nią zmiany nie zostały jeszcze zrzucone z PCh do bazy danych. W przypadku awarii nastąpi utrata efektów pracy transakcji.

## Architektura systemu bazy danych



### Algorytm UNDO/REDO – Write( $T, x, v$ )

1. Dołącz transakcję  $T$  do listy transakcji aktywnych (LTA), jeśli jeszcze na niej nie występuje.
2. Jeśli  $x$  nie znajduje się w PCh, to sprowadź do PCh stronę zawierającą  $x$ .
3. Dołącz  $(T, x, v)$  do dziennika transakcji (DzT).
4. Przypisz danej  $x$  w PCh wartość  $v$ .
5. Zawiadom planistę o zakończeniu operacji.

### Algorytm UNDO/REDO – Commit( $T$ )

1. Dołącz  $T$  do listy transakcji zatwierdzonych (LTZ).
2. Powiadom planistę o zatwierdzeniu transakcji  $T$ .
3. Usuń  $T$  z listy transakcji aktywnych (LTA).

### Algorytm UNDO/REDO – Abort( $T$ )

1. Dla każdej danej  $x$  zapisywanej przez  $T$ :
  - jeśli  $x$  nie znajduje się w PCh, to sprowadź tam stronę zawierającą  $x$ ,
  - przypisz w PCh danej  $x$  uprzednią wartość  $x$  względem  $T$  (*before image*).
2. Dołącz  $T$  do listy transakcji odrzuconych.
3. Powiadom planistę o odrzuceniu transakcji  $T$ .
4. Usuń  $T$  z listy transakcji aktywnych.

## Algorytm UNDO/REDO – Restart

- 
- Wyczyść wszystkie bufory w PCh, REDONE :=  $\emptyset$  i UNDONE :=  $\emptyset$ .
  - Rozpocznij od ostatniego zapisu w DzT i przesuwaj się do jego początku. Dopóki nie przeanalizowano wszystkich zapisów w DzT, wykonuj:
    1. Dla każdego zapisu  $(T, x, v)$ , takiego że  $x \notin \text{REDONE} \cup \text{UNDONE}$ :
      - jeśli  $x$  nie znajduje się w PCh, to sprowadź tam stronę zawierającą  $x$ ;
      - jeśli  $T$  jest na liście transakcji zatwierdzonych, przypisz danej  $x$  w PCh wartość  $v$  i przyjmij  $\text{REDONE} := \text{REDONE} \cup \{x\}$ ;
      - jeśli  $T$  nie jest na liście transakcji zatwierdzonych, przypisz w PCh danej  $x$  uprzednią wartość  $x$  względem  $T$  i przyjmij  $\text{UNDONE} := \text{UNDONE} \cup \{x\}$ .
    2. Dla każdej transakcji  $T$  z listy transakcji zatwierdzonych, usuń  $T$  z listy transakcji aktywnych.
    3. Powiadom planistę o zakończeniu operacji *Restart*.
- 

### 26. Write-Ahead Logging

---

**Write-Ahead Logging (WAL)** jest sposobem na zapisywanie zmian dokonywanych na bazie danych. Zapewnia on atomowość i trwałość (dwie z własności ACID). Wszystkie zmiany są zapisywane w pliku WAL, stanowiącego swoistego loga wykonywanych operacji. Zmiany mogą być wprowadzone do właściwej bazy danych dopiero w momencie, gdy ta zmiana zostanie zarejestrowana w pliku WAL. Umożliwia to innym użytkownikom korzystanie i odczytywanie danych z bazy danych w trakcie wprowadzania do niej modyfikacji oraz umożliwia odbudowanie całej bazy danych niezależnie od momentu, w którym nastąpiła awaria (np. przerwa w dostawie prądu). Takie rozwiązanie umożliwia wykonywanie wszystkich operacji w miejscu, co zmniejsza niezbędną ilość dostępnej pamięci.

---

### 27. Harmonogram serializowalny

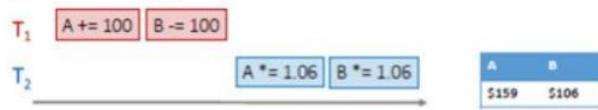
---

Harmonogram serializowalny oznacza dowolną kolejność zmian przeprowadzanych w różnych transakcjach (zmieniających te same dane), której rezultat jest zawsze taki sam jakby transakcje były wykonywane po kolei.

- Harmonogram serializowany to taki, który nie przeplata działań różnych transakcji
- A i B są równoważnymi harmonogramami, jeśli dla dowolnego stanu bazy danych wpływ wykonania A jest identyczny z efektem wykonania B.
- Serializowany harmonogram to taki, który jest równoważny z niektórymi szeregowymi realizacjami transakcji.

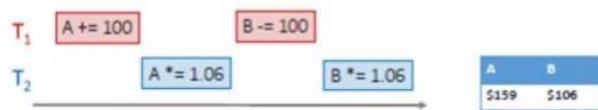
Przykład:

Serial schedule T<sub>1</sub>, T<sub>2</sub>:

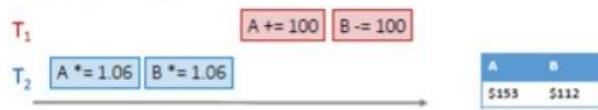


Interleaved schedule B:  
zmiany są takie same jak w harmonogramie seryjnym = serializowalny

Interleaved schedule A:

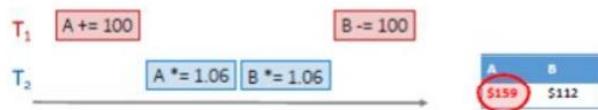


Serial schedule T<sub>2</sub>, T<sub>1</sub>:



Interleaved schedule B: zmiany nie są takie same jak w harmonogramie seryjnym = nieserializowalny

Interleaved schedule A:



## 28. Typy konfliktów: unrepeatable read, dirty read, inconsistent read (phantoms), lost, Update

**Brudny odczyt (ang. Dirty read)** to sytuacja, w której transakcja odczytuje dane zmienione przez inną transakcję, która później zostaje wycofana, a więc odczytane dane przez pierwszą transakcję są nieprawdziwe.

**Utracona modyfikacja (ang. Lost update)** to sytuacja, w której dwie transakcje równolegle próbują zmienić te same dane i zmiany wprowadzane przez jedną transakcję mogą być nadpisane przez drugą transakcję.

**Niepowtarzalny odczyt (ang. unrepeatable read)** powstaje w sytuacji gdy jedna z transakcji wykonuje kilkakrotnie te same zapytania na danych, które w między czasie są zmieniane przez inną transakcję, co powoduje, że to samo zapytanie nie generuje tych samych rezultatów.

**Fantomy (ang. Phantoms / inconsistent read)** to sytuacja podobna do niepowtarzalnego odczytu. Jedna transakcja wykonuje kilkakrotnie to samo zapytanie ale otrzymuje różne zbiory wynikowe, ponieważ inna transakcja w tym samym czasie usuwa lub dodaje nowe dane, które spełniają warunki zapytania. To powoduje, że za każdym razem wynikiem zapytania może być inna liczba wierszy.

## Classic Anomalies with Interleaved Execution

### "Unrepeatable read":

Example:



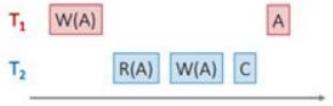
1.  $T_1$  reads some data from A
2.  $T_2$  writes to A
3. Then,  $T_1$  reads from A again  
and now gets a different / inconsistent value

*Occurring with / because of a RW conflict*

## Classic Anomalies with Interleaved Execution

### "Dirty read" / Reading uncommitted data:

Example:



1.  $T_1$  writes some data to A
2.  $T_2$  reads from A, then writes back to A & commits
3.  $T_1$  then aborts- now  $T_2$ 's result is based on an obsolete / inconsistent value

*Occurring with / because of a WR conflict*

## Classic Anomalies with Interleaved Execution

### "Inconsistent read" / Reading partial commits:

Example:



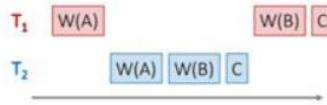
1.  $T_1$  writes some data to A
2.  $T_2$  reads from A and B, and then writes some value which depends on A & B
3.  $T_1$  then writes to B- now  $T_2$ 's result is based on an incomplete commit

*Again, occurring because of a WR conflict*

## Classic Anomalies with Interleaved Execution

### Partially-lost update:

Example:



1.  $T_1$  blind writes some data to A
2.  $T_2$  blind writes to A and B
3.  $T_1$  then blind writes to B; now we have  $T_2$ 's value for B and  $T_1$ 's value for A- not equivalent to any serial schedule!

*Occurring because of a WW conflict*

---

## 29. Poziomy izolacji w bazach danych

---

### Poziomy izolacji SQL 92

Pozim Izolacji\Anomalia	Brudny odczyt	Niepowtarzalny odczyt	Fantomy
<b>READ UNCOMMITTED</b>	możliwy	możliwy	możliwy
<b>READ COMMITTED</b>	brak	możliwy	możliwy
<b>REPEATABLE READ</b>	brak	brak	możliwy
<b>SERIALIZABLE</b>	brak	brak	brak

Standard SQL-92 definiuje cztery poziomy izolacji transakcji (kolejność od najniższego stopnia izolacji do najwyższelego):

- **READ UNCOMMITTED** - niezatwierdzony odczyt (poziom izolacji 0),
- **READ COMMITTED** - odczyt zatwierdzonych danych (poziom izolacji 1),
- **REPEATABLE READ** - powtarzalny odczyt (poziom izolacji 2),
- **SERIALIZABLE** - uszeregowany (poziom izolacji 3).

### Poziomy izolacji MS SQL 2008+

### Poziomy izolacji MS SQL 2008+

Isolation level	Dirty read	Nonrepeatable read	Phantom
<b>Read uncommitted</b>	Yes	Yes	Yes
<b>Read committed</b>	No	Yes	Yes
<b>Repeatable read</b>	No	No	Yes
<b>Snapshot</b>	No	No	No
<b>Serializable</b>	No	No	No

## Poziomy izolacji baza danych Oracle

Poziom izolacji	DR	NRR	PH	Uwagi
READ COMMITTED	NIE	NIE	NIE	Zjawisko „brudnych odczytów” nie występuje dzięki wersjonowaniu. Nigdy nie ma blokowania wierszy do odczytu, ponieważ Oracle DBMS może przywrócić wersję bazy danych sprzed rozpoczęcia transakcji i ją przez cały czas utrzymywać dla aktualnej transakcji.
SERIALIZABLE	NIE	NIE	NIE	Spójność odczytu, która jest dla pojedynczego zapytania jest zachowana dla całej transakcji. Każde zapytanie da powtarzalne wyniki podczas całego życia transakcji. Transakcja będzie cały czas widziała takie dane jakie były, kiedy się rozpoczęła.
READ ONLY	NIE	NIE	NIE	Ma wszystkie cechy SERIALIZABLE, ale nie pozwala na modyfikowanie.

DR - dirty reads, NRR - non-repeatable reads, PH - phantoms

### 30. Konflikty i metody ich rozwiązywania: porządkowanie topologiczne, blokady (wyłączna, współdzielona, aktualizacji), algorytm dwufazowego blokowania

#### PORZĄDKOWANIE TOPOLOGICZNE (SORTOWANIE TOPOLOGICZNE);

Sortowanie topologiczne skierowanego grafu acyklicznego – liniowe uporządkowanie wierzchołków, w którym jeśli istnieje krawędź skierowana prowadząca od wierzchołka x do y, to x znajdzie się przed wierzchołkiem y. Innymi słowy, każdy wierzchołek poprzedza wszystkie te wierzchołki, do których prowadzą wychodzące od niego krawędzie.

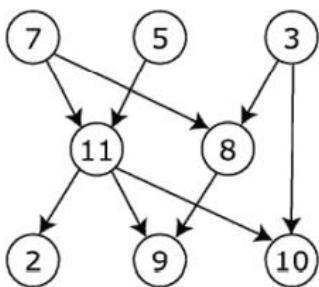
Wierzchołki w każdym grafie acyklicznym skierowanym (DAG – directed acyclic graph) można posortować topologicznie na jeden lub więcej sposobów.

Zastosowanie:

Sortowanie topologiczne pozwala na ustalenie kolejności wykonywania jakichś operacji (czynności), np. służy do ustalenia poprawnej kolejności instalacji w automatycznym uzupełnianiu zależności pakietów w systemach uniksopodobnych. Prostszym przykładem może być kolejność czynności potrzebnych do upieczenia ciasta.

Poszczególne czynności są reprezentowane jako wierzchołki, a zależności pomiędzy nimi – jako krawędzie. Jeśli krawędź prowadzi od A do B, to znaczy, że czynność A musi zostać wykonana przed czynnością B.

Zdarza się, że wykonanie jakiegoś zadania musi być poprzedzone wykonaniem innego (np. zanim obierzemy ziemniaki, musimy je kupić), ale równie dobrze czynności mogą zostać wykonane równocześnie lub w dowolnej kolejności (np. przed upieczeniem ciasta musimy kupić mąkę i jajka, choć nie ma znaczenia kolejność kupowania składników). Wynika z tego możliwość ustalenia więcej niż jednego topologicznego porządku wierzchołków dla niektórych grafów



Wierzchołki przedstawionego na rysunku grafu można posortować topologicznie na kilka sposobów, np.

- 7,5,3,11,8,2,10,9
- 7,5,11,2,3,10,8,9
- 3,7,8,5,11,10,9,2

### Algorytmy:

Najpopularniejsze algorytmy sortowania topologicznego działają w czasie  $\Theta(|V|+|E|)$ .

- Usuwanie wierzchołków „niezależnych”

Jeden z takich algorytmów polega na stopniowym usuwaniu z grafu wierzchołków o stopniu wchodząącym 0 wraz z wychodzącymi z nich krawędziami. Kolejność, w jakiej wierzchołki będą usuwane, jest poszukiwanym rozwiązańiem.

Jeśli po usunięciu wszystkich takich wierzchołków (wraz z krawędziami) graf nie pozostanie pusty, oznacza to, że zawiera cykle.

By zastosować powyższy algorytm, należy wykorzystać kontener, w którym przechowywane będą wierzchołki do usunięcia.

```

Q ← Kolejka z wierzchołkami o stopniu wchodząącym równym 0
dopóki Q jest niepusta rob
    usuń wierzchołek n z przodu kolejki Q
    wypisz n
    dla każdego wierzchołka m o krawędzi e od n do m rob
        usuń krawędź e z grafu
        jeżeli do m nie prowadzi żadna krawędź to
            wstaw m do Q
jeżeli graf ma wierzchołki to
    wypisz komunikat o błędzie (graf zawiera cykl)
  
```

Zważywszy na częste istnienie wielu rozwiązań problemu sortowania, Q nie musi być kolejką – równie dobrze może być stosem, kolejką priorytetową lub zwykłą tablicą.

- Wykorzystanie algorytmu DFS

Kolejny algorytm, którym można się posłużyć, to DFS. Wystarczy, że podczas wykonywania jego tradycyjnej wersji będziemy na początek listy dodawać aktualnie przetworzony wierzchołek, a otrzymamy listę w pożdanym porządku.

```
L ← Lista wierzchołków posortowanych w kolejności topologicznej
dla każdego wierzchołka rób
    jeśli nie był jeszcze odwiedzony, ustaw jako odwiedzony
        dla każdej krawędzi z niego wychodzącej rób
            jeśli prowadzi do nieodwiedzonego jeszcze wierzchołka
                przetwórz wierzchołek rekurencyjnie
            wstaw wierzchołek na początek listy
```

#### PODSTAWOWE RODZAJE BLOKAD:

Podstawowym mechanizmem zapobiegającym konfliktom przy współbieżnie wykonywanych transakcjach są blokady (nazywane też zamkami) zakładane na obiekty. Są dwa rodzaje blokad:

- Współdzielona, typu S (ang. shared lock) - daje transakcji współdzielony dostęp do zasobu, na przykład, kilka transakcji może jednocześnie odczytywać wiersze tej samej tabeli. Jeśli transakcja zakłada współdzieloną blokadę, inne transakcje też mogą założyć współdzieloną blokadę, ale nie mogą założyć blokady drugiego rodzaju, to jest wyłącznej blokady. Operację założenia blokady współdzielonej na obiekcie A oznaczamy przez S(A).
- Wyłączna, typu X (ang. exclusive lock) - daje transakcji wyłączne prawo do wprowadzania zmian obiektu. Tylko jedna transakcja może mieć założoną wyłączną blokadę na obiekcie i w tym czasie nie może być na nim założonej żadnej innej blokady nawet współdzielonej. Operację założenia blokady wyłącznej na obiekcie A oznaczamy przez X(A).
- Aktualizacji, typu U (ang. update lock) – ta blokada jest podobna do blokady wyłącznej, została jednak zaprojektowana tak żeby być bardziej elastyczna. Blokada aktualizacji może zostać nałożona na rekord, który ma już blokadę współdzieloną. W takim przypadku blokada aktualizacji nałoży kolejną blokadę współdzieloną na docelowy wiersz. Gdy transakcja która utworzyła blokadę aktualizacji będzie gotowa do zmiany danych, blokada aktualizacji (U) zostanie przekształcona w blokadę wyłączną (X). Ważne jest, aby zrozumieć, że blokada aktualizacji jest asymetryczna w kontekście współdzielonych blokad. Przykład - chociaż blokadę aktualizacji można nałożyć na rekord, który ma już blokadę współdzieloną, to blokada współdzielona nie może być nałożona na rekord który ma już nałożoną blokadę aktualizacji.

Dodatkowo, dopuszczana jest operacja podwyższenia blokady przez transakcję. Mianowicie, transakcja, która założyła blokadę S, może ją zmienić na X, pod warunkiem, że na obiekcie nie ma założonej innej blokady S.

Istnieje metoda zakładania blokad gwarantująca powstawanie i realizację planów wyłącznie szeregowalnych i odtwarzalnych. Jest ona powszechnie używana przez SZBD.

**Algorytm blokowania dwufazowego (ang. two-phase locking – 2PL )** jest jedną z najpopularniejszych metod implementacji transakcji. Algorytm ten należy do ogólniejszej klasy algorytmów sterowania współbieżnością, które stosują blokowanie. Działanie takich algorytmów w najprostszym przypadku polega na założeniu blokady na danej, która ma być zapisana lub odczytana, a po wykonaniu operacji blokada jest zwalniana.

Algorytm blokowania dwufazowego składa się z dwóch faz: fazy wzrostu (ang. growing phase ) oraz fazy zmniejszania (ang. shrinking phase ). W pierwszej fazie transakcja może blokować zasoby, ale nie wolno jej zwalniać zasobów, które wcześniej już zablokowała. W drugiej fazie transakcja może zwalniać zasoby, lecz nie wolno jej blokować żadnych nowych zasobów. Jeżeli proces nie chce modyfikować danych dopóki nie osiągnie momentu przed fazą zmniejszania, to w razie niepowodzenia przy zakładaniu jakiejś blokady może on zwolnić wszystkie blokady i rozpocząć ponownie algorytm.

Kluczową właściwością tego algorytmu jest fakt, że jeżeli wszystkie transakcje działają według schematu blokowania dwufazowego, to ich scenariusze utworzone na skutek przeplotu są uszeregowalne.

Istnieje kilka odmian tego algorytmu. Np. ścisłe blokowanie dwufazowe (ang. strict two-phase locking ) jest realizowane w systemach, gdzie faza zmniejszania nie występuje dopóki transakcja nie zostanie zakończona. Atutem takiego rozwiązania jest to, że transakcja zawsze czyta wartości zapisane tylko przez zatwierdzone transakcje. Ponadto operacjami blokowania i zwalniania może zająć się system, żeby nie angażować w tym celu transakcji.

**Ścisłe blokowanie dwufazowe (Strict Two-Phase Locking)** - Protokół Strict 2PL gwarantuje realizację wyłącznie planów szeregowalnych i odtwarzalnych. Nie dopuszcza do powstania zjawisk niezatwierdzonego odczytu, niepowtarzalnego odczytu i nadpisywania nie zatwierdzonych danych.

Protokół Strict 2PL nazywa się dwufazowym, ponieważ determinuje dwie fazy działania każdej transakcji związane z blokadami:

- transakcja zakłada blokady i dokonuje wymaganych odczytów i zapisów na obiektach, na których założyła blokadę;
- transakcja wykonuje COMMIT/ROLLBACK jednocześnie zwalniając wszystkie blokady

Zakładanie blokad i ich zwalnianie są oddzielonymi w czasie fazami.

Istnieją dwie modyfikacje protokołu strict-2PL, które umożliwiają wcześniejsze zwalnianie blokad.

- Protokół 2PL1: Transakcja nie może założyć żadnej nowej blokady po zwolnieniu jakiejkolwiek blokady; gwarantuje szeregowalność transakcji, ale nie odtwarzalność.
- Protokół 2PL2: Transakcja nie może wcześniej zwolnić żadnej blokady X, ale w każdej chwili może zwolnić założoną do odczytu blokadę S; gwarantuje odtwarzalność transakcji, ale nie szeregowalność.

Protokół ścisłego blokowania dwufazowego (Strict 2PL):

1. Każda transakcja musi uzyskać blokadę S (współdzieloną) na obiekcie zanim odczyta ten obiekt oraz blokadę X (wyłączną) na obiekcie przed zapisaniem go.
2. Jeśli transakcja trzyma blokadę X na obiekcie, żadna inna transakcja nie ma prawa założyć żadnej blokady (ani typu S ani X) na tym obiekcie.
3. Jeśli transakcja trzyma blokadę S na obiekcie, żadna inna transakcja nie ma prawa założyć blokady X na tym obiekcie.
4. Gdy transakcja nie może założyć blokady na obiekcie, może ustawić się w kolejce oczekujących transakcji stwarzyszonej z tym obiektem albo może zostać wycofana.
5. Wszystkie blokady trzymane przez transakcję są zwalniane jednocześnie, w chwili gdy transakcja kończy się (wycofaniem lub zatwierdzeniem).

Kolejność zwalniania blokad determinuje kolejność ustawienia wykonywanych transakcji w równoważny plan szeregowy.

---

*31. Optymistyczne zarządzanie wielodostępnem (optimistic concurrency control)*

---

**Definicja:**

OCC – jest to metoda kontroli współbieżności stosowana w systemach transakcyjnych. Zakłada, że wiele transakcji może zostać wykonane bez zakłócania się nawzajem. OCC jest ogólnie stosowane w środowiskach o niskiej rywalizacji o dane. Gdy konflikty są rzadkie, transakcje mogą być realizowane bez ponoszenia kosztów zarządzania blokadami i bez oczekiwania przez transakcje na usunięcie blokad innych transakcji, co prowadzi do wyższej przepustowości.

**Przebieg:**

- Faza rozpoczęcia – zapamiętanie czasu rozpoczęcia transakcji
- Faza modyfikacji – odczytanie wartości i wstępne zapisanie zmian
- Faza sprawdzenia – sprawdzenie czy inne transakcje zmodyfikowały dane z których korzysta transakcja. Sprawdzanie obejmuje transakcje zakończone po rozpoczęciu analizowanej transakcji oraz te które są nadal aktywne w czasie sprawdzania
- Faza zatwierdzania/cofania – Jeśli nie ma żadnego konfliktu wszystkie zmiany są zapisywane. Jeśli występuje konflikt jest on rozwiązywany zwykle poprzez przerwanie transakcji chociaż możliwe są też inne rozwiązania

---

### *32. Macierze RAID w kontekście baz danych*

---

**RAID** (ang. Redundant Array of Independent Disks, Nadmiarowa macierz niezależnych dysków) - polega na współpracy dwóch lub więcej dysków twardych w taki sposób, aby zapewnić dodatkowe możliwości, nieosiągalne przy użyciu jednego dysku.

Macierzy używa się w celu:

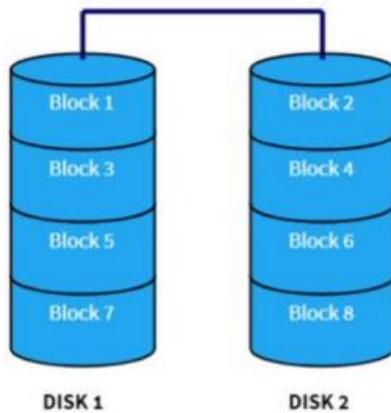
- zwiększenia niezawodności (odporności na awarie),
- przyspieszenia transmisji danych,
- powiększenia przestrzeni dostępnej jako jedna całość,
- kilku lub wszystkich powyższych łącznie.

**Przeznaczenie macierzy** implikuje wybór odpowiednich technologii w zakresie dysków, kontrolerów, pamięci cache, sposobu przesyłania danych oraz poziomu niezawodności (odpowiedniej nadmiarowości/redundancji podzespołów i połączeń). Mamy do wyboru zarówno kontrolery programowe jak i sprzętowe. Obecnie stosuje się dyski w technologii Serial ATA, jak i SAS. Coraz częściej wykorzystywane są dyski SSD.

#### **RAID 0**

- Dobra wydajność dla zapisu i odczytu.
- Jeśli jeden dysk padnie, wszystkie dane są utracone

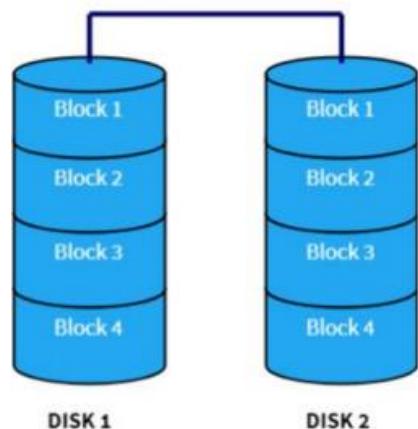
#### **RAID 0**



## RAID 1

- Dobra wydajność dla zapisu i odczytu
- Jeśli jeden dysk padnie drugi przechowuje jego dokładną kopię
- Tylko połowa dostępnego miejsca

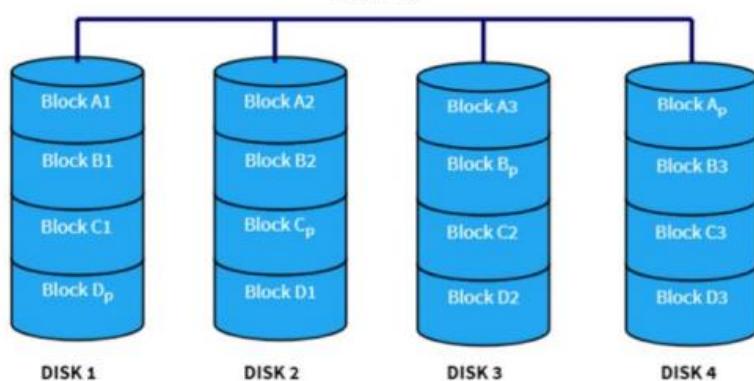
**RAID 1**



## RAID 5

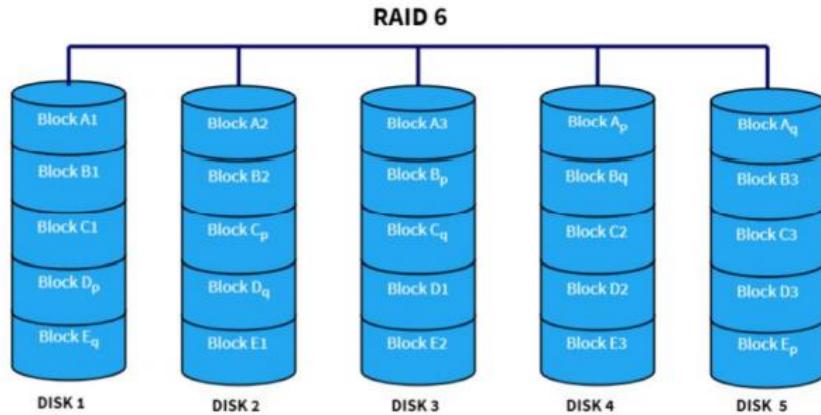
- Operacje odczytu bardzo szybkie, zapisu wolniejsze ze względu na obliczenia, które muszą być wykonane
- Jeśli jeden dysk padnie, wszystkie dane są nadal dostępne
- Zepsucie dysku ma wpływ na wydajność

**RAID 5**



## RAID 6

- Odczyt danych bardzo szybki
- Jeśli dwa dyski padną, wszystkie dane są nadal dostępne
- Zapis jest wolniejszy niż w RAID 5



---

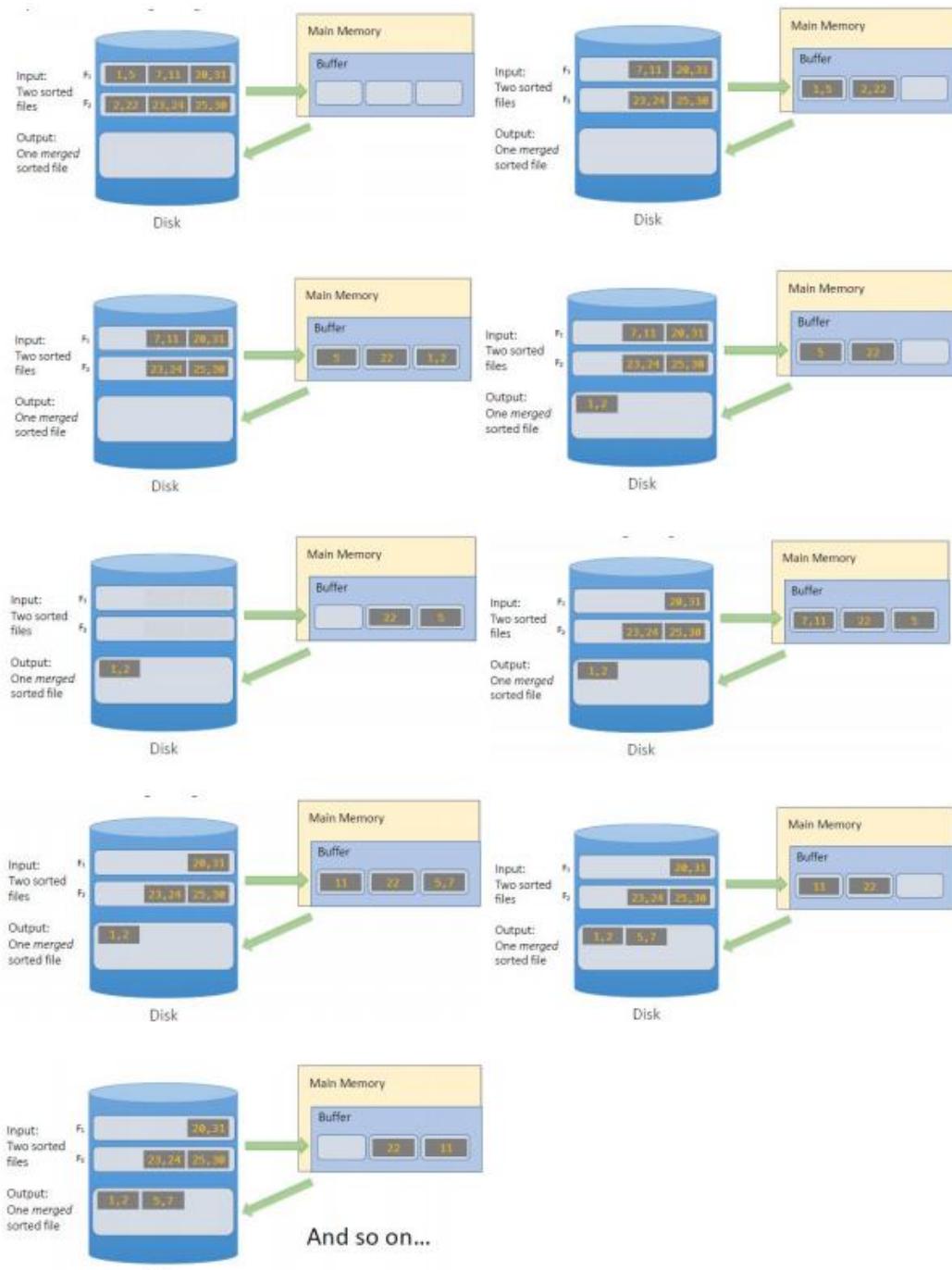
33. Algorytm zewnętrznego łączenia i sortowania: zasada działania, złożoność obliczeniowa, zastosowania, warianty: B-way merge, przepakowanie

---

## Algorytm zewnętrznego łączenia i sortowania

### Zasada działania

- Wejście: Dwie posortowane listy o długości M i N
- Wyjście: Posortowana lista o długości M+N
- Wymagane: Przynajmniej 3 strony bufora
- Liczba IO:  $2(M+N)$



### Złożoność obliczeniowa:

Dla 3 stron bufora i 6 stronnicowego pliku

- o Podział na 3 stronnicowe pliku i sortowanie w pamięci: = 1 odczyt + 1 zapis (dla każdego pliku) =  $2*(3+3) = 12$  operacji wejścia/wyjścia
  - o Łączenie każdej pary posortowanych części używając algorytmu zewnętrznego łączenia = $2*(3+3) = 12$  operacji wejścia/wyjścia
  - o Całkowity koszt =24 operacje wejścia/wyjścia

- Złożoność  $2N * (\log_2(N) + 1)$

### Zastosowania

- Ze względu na bardzo częste pobieranie posortowanych danych z bazy
- Ze względu na brak w pamięci RAM miejsca na używanie innych algorytmów np. Quicksort
- Ze względu na fakt, że posortowane dane pozwalają na wyeliminowanie duplikatów ze zbioru
- Ze względu na to, że sortowanie jest pierwszym etapem tworzenia B+ drzewa dla indeksowania
- Ze względu na to, że algorytm ten jest wykorzystywany przez algorytm Sort-Merge Join

### Wariant B+ drzewa

- Podział N stron na B+1 serii:

$$2N \left( \left\lceil \log_2 \frac{N}{B+1} \right\rceil + 1 \right)$$

- Łączenie grup:

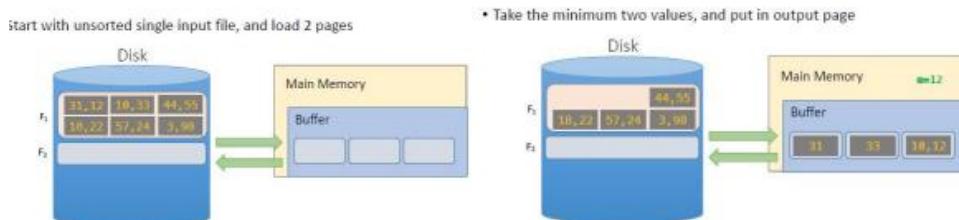
$$2N \left( \left\lceil \log_B \frac{N}{2(B+1)} \right\rceil + 1 \right)$$

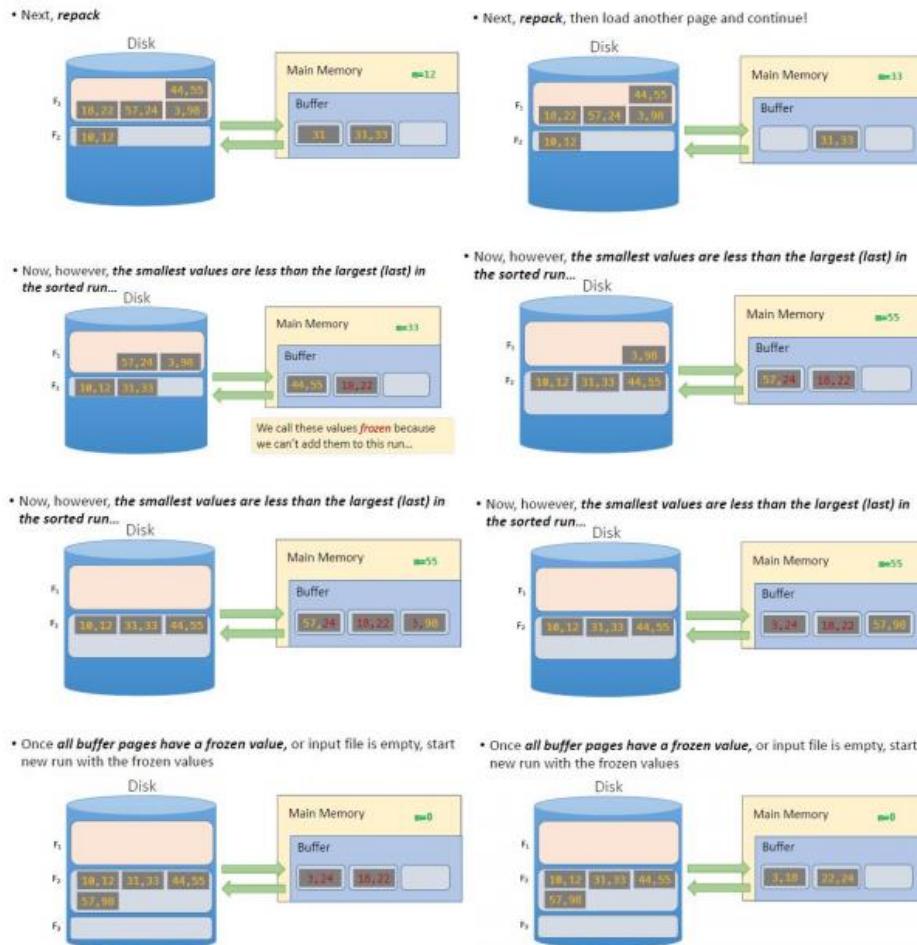
### Wariant przepakowanie

- Złożoność:

$$\sim 2N \left( \left\lceil \log_B \frac{N}{2(B+1)} \right\rceil + 1 \right)$$

- Zasada działania:





### 34. Indeksy: zalety, wady, zastosowania, B+ drzewa, tablice haszujące, bitmapy, indeks powiązany (clustered), indeks pokrywający (covered), operacje na indeksach.

Aby szybko przeszukiwać tabele po różnych atrybutach moglibyśmy utworzyć wiele kopii rekordów posortowanych według różnych atrybutów, ale to zajęłoby za dużo miejsca i byłoby głupie.

Wprowadzenie struktury danych jaką jest indeks rozwiązuje ten problem. Silniki NoSQL działają na samych indeksach. Indeksy to podstawa wszystkich systemów zarządzania bazą danych. Czasem używa się B+ drzew, czasem haszowania indeksów.

Indeks to struktura danych mapująca klucze wyszukiwania do zbioru wierszy w tabeli. Zapewnia efektywne wyszukanie i zwrócenie po wartości szukanego klucza, zwykle o wiele szybciej niż standardowe przeszukiwanie wszystkich wierszy tabeli.

Indeks może przechowywać całe rzędy, na który wskazuje (klucz główny) lub przechowywać wskaźnik do tych rzędów (secondary index, wtórny, drugorzędny).

### Operacje na indeksach:

-wyszukiwanie (search)

-wstawienie/usunięcie (insert/remove)

Indeksowanie to jedna z najważniejszych funkcji zapewnianych przez bazę danych dla jej efektywności.

### Conceptual Example

What if we want to return all books published after 1867?  
The above table might be very expensive to search over row-by-row...

Russian\_Novels

BID	Title	Author	Published	Full_text
001	War and Peace	Tolstoy	1869	...
002	Crime and Punishment	Dostoyevsky	1866	...
003	Anna Karenina	Tolstoy	1877	...

```
SELECT *
FROM Russian_Novels
WHERE Published > 1867
```

### Conceptual Example

By\_Yr\_Index

Published	BID
1866	002
1869	001
1877	003

Russian\_Novels

BID	Title	Author	Published	Full_text
001	War and Peace	Tolstoy	1869	...
002	Crime and Punishment	Dostoyevsky	1866	...
003	Anna Karenina	Tolstoy	1877	...

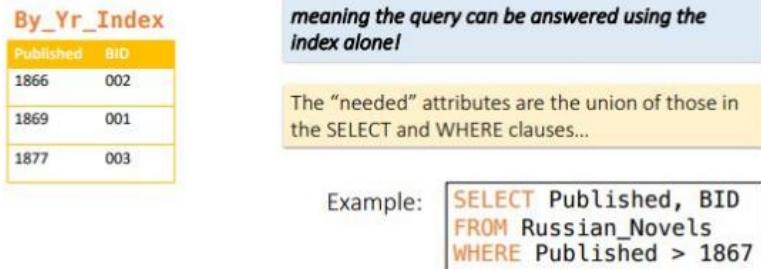
Maintain an index for this, and search over that!

Why might just keeping the table sorted by year not be good enough?

## Conceptual Example



## Covering Indexes



Mówimy że indeks jest pokrywający dla jakiegoś zapytania jeżeli ten indeks zawiera wszystkie potrzebne atrybuty - czyli że zapytanie może być zrealizowane używając tylko tego indeksu! Te potrzebne atrybuty to unia tych z zapytań SELECT i WHERE (przykład na zdjęciu)

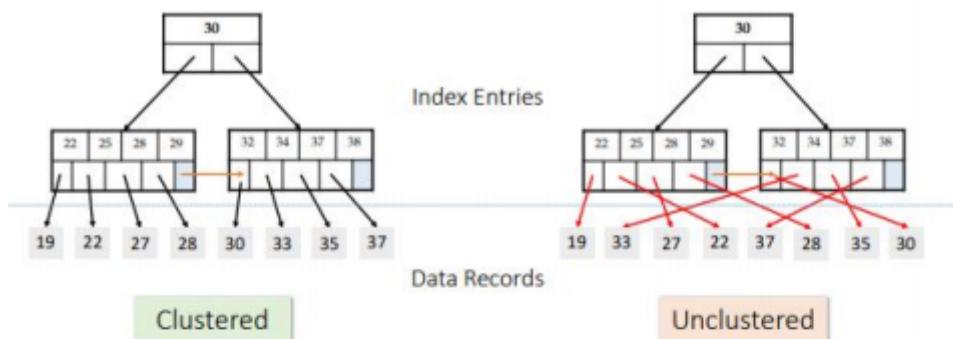
Wysokopoziomowe kategorie typów indeksów:

**B Drzewa:** -posortowane dane -szeroki wachlarz zapytań -niektóre starsze bazy danych mają je zaimplementowane tylko jak b drzewa

**Tablice haszujące:** -są ich różne warianty z różnymi kosztami IO -nazywane liniowym lub rozszerzalnym haszowaniem - IO świadome

**Indeks jest powiązany** jeśli dane są ułożone w taki sam sposób jak dane indeksu.

## Clustered vs. Unclustered Index



Indeksy powiązane a pokrywające:

- przypomnienie, że dla dysku z blokowanym dostępem sekwencyjne IO jest o wiele szybsze niż losowe IO
- dla konkretnego wyszukania nie ma różnicy między indeksem pokrywającym a powiązanym

Typy indeksów:

-B+Drzewa

-Indeksy haszowane:

- funkcja haszująca ( $\text{mod}()$  or  $(a * \text{value} + b)$ ) mapuje klucz wyszukiwania do kubelków, efektywnie dzieląc rekordy na małe grupy (kubelki). Kubelki są reprezentowane przez bloki i bloki przepelnione
  - nie wspierają range searchingu
- Indeksy bitmapowe
  - funkcja haszująca ( $\text{mod}()$  or  $(a * \text{value} + b)$ ) mapuje klucz wyszukiwania do kubelków, efektywnie dzieląc rekordy na małe grupy (kubelki). Kubelki są reprezentowane przez bloki i bloki przepelnione
    - szczególnie przydatne przy polach o małej ilości unikalnych wartości

Porady co do wybierania indeksów:

1. tworzymy indeks kiedy:
  - kolumna jest często poddawana zapytaniom
  - ograniczenie referencyjne integralności istnieje w kolumnie
  - ograniczenie integralności unikalnego klucza istnieje w kolumnie
2. kolumny z 1 lub więcej poniższymi cechami są dobrymi kandydatami do indeksowania:
  - wartości w kolumnie są unikalne lub mamy tylko parę duplikatów
  - jest szeroki wachlarz wartości w kolumnie (dobre dla B+drzewowych indeksów)
  - jest mały wachlarz wartości (dobre dla bitmapowych indeksów)
  - w kolumnie jest dużo wartości null, ale zapytania często wybierają wszystkie wiersze mające wartości.

3. powinno się umieścić najczęściej używaną kolumnę jako pierwszą w indeksie
4. Wyrzucamy (drop) indeksy już więcej niepotrzebne

---

35. B+ drzewa: struktura, właściwości, rozmiary, przeszukiwanie, wstawianie i kasowanie danych, stopień węzła, fanout, zapełnienie (fill-factor), koszt znalezienia informacji

---

Na wszystkich poziomach wyższych niż poziom liści zawierają elementy o małej pamięci takie jak: indeksy, klucze i wskaźniki na węzły potomne zaś na poziomie liści jest lista która zawiera główne dane.



#### DRZEWA WIELOKIERUNKOWE (ang.: multiway trees)

- **drzewa wielokierunkowe** – stopnie węzłów  $> 2$ ;
- **drzewa rzędu  $m$**  – stopnie węzłów  $\leq m$ ;
- **drzewa poszukiwań rzędu  $m$**  – istnieje relacja porządkująca dla składowych kluczowych, a ponadto:
  - ◆ każdy węzeł ma co najwyżej  $m$  potomków i  $(m - 1)$  kluczy (i danych nie wpływających na uporządkowanie drzewa);
  - ◆ klucze w każdym węźle są uporządkowane zgodnie z pewną liniową relacją porządkującą;
  - ◆ klucze w pierwszych  $i$  poddrzewach węzła poprzedzają w tej relacji  $i$ -ty klucz węzła (klucz ten spełnia rolę „**klucza rozdzielającego**” węzły potomne);
  - ◆ klucze w ostatnich  $(m - i)$  poddrzewach węzła następują w tej relacji po kluczu  $i$ -tego węzła.



#### ALGORYTMY 2

#### B-DRZEWA (ang.: B-trees)

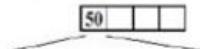
(R.Bayer, E.M.McCreight – 1972)

B-drzewo rzędu  $m$  ma następujące właściwości:

- korzeń posiada co najmniej dwa poddrzewa, (poza przypadkiem, gdy jest jedynym węzłem drzewa, czyli liściem);
- każdy węzeł nie będący liściem ani korzeniem posiada  $(k - 1)$  kluczy i  $k$  wskaźników do potomków, gdzie  $\lceil m/2 \rceil \leq k \leq m$ ;
- każdy liść posiada  $(k - 1)$  kluczy, gdzie  $\lceil m/2 \rceil \leq k \leq m$ ;
- wszystkie liście należą do tego samego poziomu.

$$\log_m [N + 1] \leq h_B \leq \log_{\lceil m/2 \rceil} [0.5(N - 1) + 1] + 1, \text{ gdzie } N - \text{liczba kluczy}$$

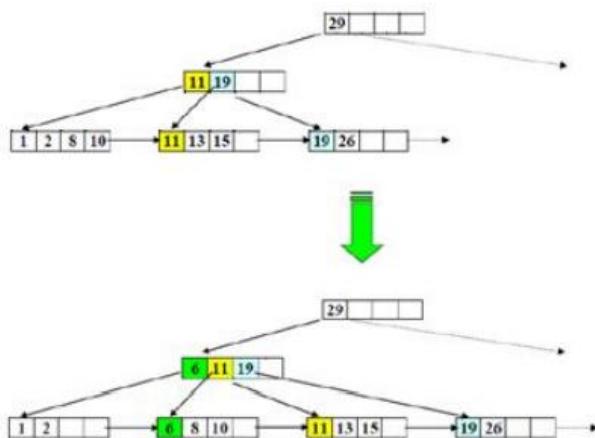
Skalę „osiagnięć” związanych ze „spłaszczeniem” drzewa dzięki zwiększeniu rzędu niech ilustruje fakt, że przy  $m = 200$  i  $N = 2000000$  całe B-drzewo mieści się na 3-ch lub 4-ch poziomach.





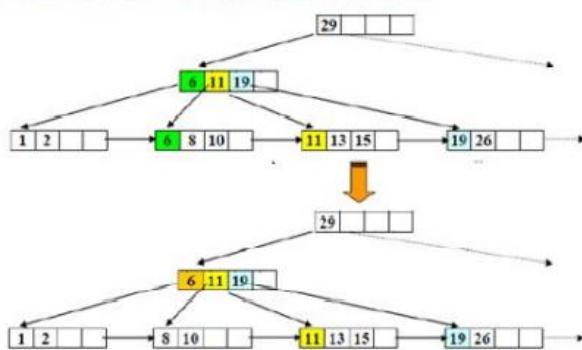
Wyszukiwanie elementu o kluczu **key** w wielokierunkowym drzewie poszukiwań

Przykład – B<sup>+</sup>-drzewo rzędu 5 – wstawienie elementu o kluczu 6



Przykład – B<sup>+</sup>-drzewo rzędu 5 – usunięcie elementu o kluczu 8

Przykład – B<sup>+</sup>-drzewo rzędu 5 – usunięcie elementu o kluczu 6



d – stopień drzewa

Każdy węzeł ma od d do 2\*d kluczy, z wyjątkiem korzenia, który może mieć 1 lub 2 klucze.

W liściach wskaźniki do rekordów bazy danych.

Ostatni element w każdym liściu zawiera wskaźnik na liść znajdujący się po jego prawej stronie.

Jak duży jest d ? (Przykład:  
 • Rozmiar klucza = 4 bajty  
 • Rozmiar wskaźnika = 8 bajtów  
 • Rozmiar bloku = 4096 bajtów  
 • Chcemy, aby każdy węzeł zmieścił się na jednym bloku / stronie:  $2d \times 4 + (2d + 1) \times 8 \leq 4096 \rightarrow d \leq 170$ )

Fanout określa liczbę przejść w drodze od korzenia do odpowiedniej wartości w liściu (w przypadku B+ drzew jest to w przedziale  $< d+1; 2*d+1 >$ ).

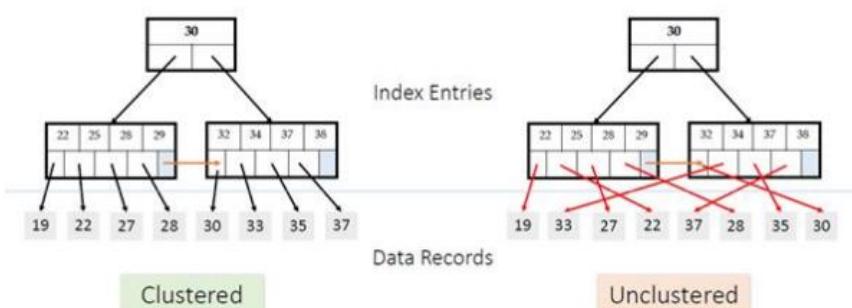
Fill-factor określa współczynnik wypełnienia drzewa. (Najczęściej koło 0.67 = 67%)

Koszt znalezienia informacji: ( $f$  - fanout;  $N$  - liczba stron, które indeksujemy;  $F$  - fill-factor)

$$\left\lceil \log_f \frac{N}{F} \right\rceil - L_B + Cost(OUT)$$

$$B \geq \sum_{l=0}^{L_B-1} f^l$$

Indeksy klastrowe i nieklastrowe:




---

### 36. Nested Loop Join (NLJ) – właściwości, koszt operacji we/wy

---

- Notacja:
  - $T(R)$  – liczba rekordów w  $R$
  - $P(R)$  – liczba stron w  $R$
- Algorytm:

Koncepcyjnie JOIN NESTED LOOPS działa jako dwie zagnieżdżone pętle po jednej i drugiej tabeli

**Compute  $R \bowtie S$  on  $A$ :**

```
for r in R:          P(R)
    for s in S:      P(R) + T(R)*P(S)
        if r[A] == s[A]: P(R) + T(R)*P(S)
            yield (r,s) P(R) + T(R)*P(S) + OUT
```

1. Pobierz rekordy z tabeli zewnętrznej
2. Pobierz rekordy z tabeli wewnętrznej
  - Wybierz pierwszy rekord tabeli zewnętrznej
  - Porównaj rekord z tabeli zewnętrznej z rekordami tabeli wewnętrznej
  - Zwróć wynik iteracji
  - Powtarzaj powyższą iterację aż, wszystkie rekordy zostaną porównane
- Złożoność:

$P(R) + T(R)*P(S) + OUT$	$P(S) + T(S)*P(R) + OUT$
--------------------------	--------------------------

Najbardziej kosztowną częścią algorytmu jest  $T(R) * P(S)$ . Oznacza to, że złożoność jest kwadratowa. Algorytm można ulepszyć dodając do pierwszej pętli tabelę o mniejszej liczbie rekordów.

- Właściwości:
  - jest podstawową metodą łączenia dwóch tabel
  - wykorzystywany jest gdy operujemy na niewielkiej ilości danych. Przy czym niewielkie tłumaczyć należy jako niewielka część danych z tabeli
  - pozwala na szybki dostęp do danych gdy potrzebujemy ich ograniczoną ilość

---

### 37. Algorytm Block Nested Loop Join (BNLJ) – właściwości, koszt operacji we/wy

---

- Algorytm:

Compute  $R \bowtie S$  on A:

```
for each B-1 pages pr of R:  
    for page ps of S:  
        for each tuple r in pr:  
            for each tuple s in ps:  
                if r[A] == s[A]:  
                    yield (r,s)
```

$P(R)$   
 $P(R) + \frac{P(R)}{B-1} P(S)$   
 $P(R) + \frac{P(R)}{B-1} P(S)$   
 $P(R) + \frac{P(R)}{B-1} P(S) + OUT$

- Dla każdej porcji stron pr z R (zostawiamy sobie dwie strony) [Odczyt z dysku]  
⇒ Dla każdej strony ps z S [Odczyt z dysku]
  - ♦ Dla każdego wiersza r z pr [Pamięć]
  - ◊ Dla każdego wiersza s z ps [Pamięć] $\Rightarrow$  Porównywanie

- Złożoność:

$$P(R) + \frac{P(R)}{B-1} P(S) + OUT$$

- BNLJ jest szybszy od NLJ o około:

$$\frac{(B-1)T(R)}{P(R)}$$

---

### 38. Algorytm Index Nested Loop Join (INLJ) – właściwości, koszt operacji we/wy

---

W relacji R:

$T(R)$ = liczba krotek w R

$P(R)$ =liczba stron w R

Koszt operacji:

$$P(R) + T(R)*L + OUT$$

gdzie L to koszt IO dostępu do wszystkich różnych wartości w indeksie zakładając, że mieszczą się na jednej stronie,  $L=3$  to dobra estymacja.

Można użyć indeksu (np. B+drzewa) celem uniknięcia liczenia cross produktu.

Compute  $R \bowtie S$  on A:

```
Given index idx on S.A:  
    for r in R:  
        s in idx(r[A]):  
            yield r,s
```

---

### *39. Algorytm Sort-Merge Join (SMJ) + wersja zoptymalizowana – właściwości, koszt operacji we/wy*

---

**SORT MERGE JOIN** jest łączeniem dwóch posortowanych zestawów danych. Ten sposób łączenia tabel składa się z dwóch operacji które składają się na operację SORT MERGE JOIN:

- **SORT JOIN** jest to operacja w której baza danych sortuje zestawy danych. Jeśli istnieją indeksy, baza danych może uniknąć sortowania pierwszego zestawu danych jednak zawsze sortuje drugi zestaw danych, niezależnie od indeksów.
- **MERGE JOIN** gdy oba zestawy danych są już posortowane baza danych rozpoczyna ich łączenie. Operacja MERGE JOIN jest specyficzny i zoptymalizowanym sposobem nested loops join.

Operacja MERGE JOIN polega na porównywaniu dwóch posortowanych zestawów. Baza danych ustawia na obu z nich znacznik na pierwszym rekordzie. Jeżeli wartości są zgodne z warunkiem złączenia to rekord jest zwracany do wyniku. Następnie indeks jednej strony przesuwa się na kolejny rekord i porównuje wartości. Jeżeli nie są zgodne to albo porusza pierwszym znacznikiem albo drugim w zależności od warunku złączenia. Dokładny opis algorytmu przedstawiłem w kolejnym punkcie.

Baza danych wybierze SORT MERGE JOIN w sytuacji gdy:

- Warunek łączenia między dwiema tabelami nie jest warunkiem równoważnym, to znaczy stosuje warunek nierówności, taki jak  $<$ ,  $\leq$ ,  $>$  lub  $=$ .
- Wykonujemy operację sortowania na kolumnie z warunku złączenia
- Istnieją indeksy na kolumnach łączenia. Nie jest to warunek konieczny ale zwiększa szansę na jego wykorzystanie.

Kroki:

1. Posortuj R, S po atrybutie A korzystając z algorytmu zewnętrznego łączenia
  2. Skanuj przesortowane pliki i połącz wynikiem
    - [Potrzebny backup jeśli występują duplikaty]
- Ważną zaletą algorytmu jest to, że często dane w tabelach są posortowane, więc pierwszy krok może być pominięty
  - Algorytm sprawdza tylko równość danych
  - Złożoność obliczeniowa:

- W przypadku kiedy niepotrzebny jest backup:  $P(R) + P(S)$
  - W przypadku pełnego backup'u każdego razu:  $P(R) * P(S)$

Lecture 15 > Section 1 > Backup

## SMJ: Total cost

- Cost of SMJ is **cost of sorting R and S...**
- Plus the **cost of scanning**:  $\sim P(R) + P(S)$ 
  - Because of *backup*: in worst case  $P(R) * P(S)$ ; but this would be very unlikely
- Plus the **cost of writing out**:  $\sim P(R) + P(S)$  but in worst case  $T(R) * T(S)$

$\sim \text{Sort}(P(R)) + \text{Sort}(P(S))$   
 $+ P(R) + P(S) + \text{OUT}$

Recall:  $\text{Sort}(N) \approx 2N \left( \left\lceil \log_B \frac{N}{2(B+1)} \right\rceil + 1 \right)$

Note: this is using repacking, where we estimate that we can create initial runs of length  $\sim 2(B+1)$

Lecture 15 > Section 1 > Backup

## Simple SMJ Optimization

Given **B+1** buffer pages

- Now, on this last pass, we only do  $P(R) + P(S)$  IOs to complete the join!
- If we can initially split R and S into **B total runs each of length approx.  $\leq 2(B+1)$** , assuming repacking lets us create initial runs of  $\sim 2(B+1)$ - then we only need  **$3(P(R) + P(S)) + OUT$**  for SMJ!
  - 2 R/W per page to sort runs in memory, 1 R per page to B-way merge / join!
- How much memory for this to happen?
  - $\frac{P(R) + P(S)}{B} \leq 2(B + 1) \Rightarrow \sim P(R) + P(S) \leq 2B^2$
  - Thus,  $\max(P(R), P(S)) \leq B^2$  is an approximate sufficient condition

If the larger of R,S has  $\leq B^2$  pages, then SMJ costs  **$3(P(R)+P(S)) + OUT$**

#### 40. Algorytm Hash Join (HJ) – właściwości, koszt operacji we/wy

1. Dziel tabele R i S na koszyczki (koszyków jest B)
  2. Bierz parę rekordów z każdego koszyczka, które mają taką samą wartość funkcji haszującej i połącz.
  3. Na koszykach można użyć BNLI
  4. Złożoność:  $P(R)+P(S)$

5. Potrzebna pamiętać:

$$B - 1 \geq \frac{P(R)}{B} \Rightarrow \sim B^2 \geq P(R)$$

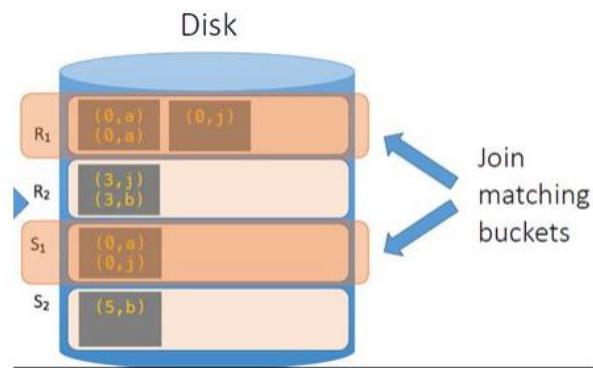
HJ faza 1 (partycjonowanie):

1. Czytamy strony R na „stronę wejściową” bufora.
2. Następnie używamy funkcji haszującej  $h_2$  do sortowania w segmentach, z których każdy mieć jedną stronę w buforze
3. Powtarzamy kroki 1 i 2 do momentu zapełnienia stron bufora
4. Przenosimy zawartość bufora na dysk.
5. Jeżeli R nie jest puste, wróć do kroku 1.

(analogicznie postępujemy dla S)

Uwaga, mogą występować kolizje (funkcja haszująca przyporządkuje dwóm różnym wartościom to samo miejsce), w takim przypadku, należy użyć funkcji  $h_2'$  aby uniknąć takiej sytuacji.

Dolaczamy pary kubełków R i S mające te same wartości funkcji haszującej, aby zakończyć łączenie.



Wydajność:

$$\sim 3(P(R) + P(S)) + OUT$$

---

*41. Porównanie powyższych algorytmów.*

---

Algorytmy:

- z zagnieżdżonymi pętlami,
- z zagnieżdżonymi pętlami i blokowaniem,
- z zagnieżdżonymi pętlami i indeksowaniem,
- zewnętrznego sortowania,
- haszujący

Pierwsze dwa algorytmy można było wykonać na join różnego typu czyli mniejszy większy ale pozostałe algorytmy działają tylko gdy mamy równa się. Algorytm zewnętrznego łączenia i haszujący jesteśmy w stanie doprowadzić do tego by ich czas trwania był liniowy w przybliżeniu 3 razy suma rozmiarów tych tabel, algorytm haszujący będzie potrzebował mniej pamięci gdy tabele będą różnych rozmiarów. Te algorytmy spowalniają gdy mają duplikaty.

---

*42. Algebra relacji: selekcja, projekcja, zmiana nazwy, iloczyn kartezjański, łączenie (natural, equi-join, theta join, semijoin, antijoin); suma, iloczyn i różnica zbiorów oraz multizbiorów.*

---

Krotka – struktura danych uporządkowanego ciągu wartości. Krotki przechowują stałe wartości o różnych typach danych.

---

## Operator selekcji (*ang. selection operator*)

---

- Operuje na pojedynczej relacji będącej jego operandem, ale zawiera także dodatkowe wyrażenia warunkowe stanowiące jego parametry.
- Operator selekcji zapisujemy w postaci:

$$\sigma_C(R)$$

R – relacja  
C – warunek

- Warunek C może zawierać stałe, jak i operandy będące atrybutami ze schematu relacji R.
- Operatorami wykorzystywanymi w warunku C są typowe wyrażenia warunkowe z języka programowania C, czyli wyrażenia złożone z porównań arytmetycznych oraz logicznych łączników.
- Wynikiem operacji jest relacja której schemat jest identyczny ze schematem relacji R.
- W relacji tej umieszczamy wszystkie krotki t z relacji R, dla których warunek C jest prawdziwy po podstawieniu za każdy atrybut A właściwej dla niej składowej krotki t.

---

### Przykład: Relacja ZSO Zajęcia-StudentID-Ocena

---

Zajęcia	Student ID	Ocena
CS101	12345	5.0
CS101	67890	4.0
EE200	12345	3.0
EE200	22222	4.5
CS101	33333	2.0
PH100	67890	3.5

#### Operator selekcji

$$\sigma_{Zajęcia = "CS101"} (ZSO)$$

Zajęcia	Student ID	Ocena
CS101	12345	5.0
CS101	67890	4.0
CS101	33333	2.0

**Projekcja – (rzutowanie) -z relacji wybieramy tylko podane kolumny. Po więcej odsyłam do linka.**

zmiana nazwy

– przemianowanie pS(R):

-zmiana nazwy relacji

-zmiana nazw jej atrybutów

-zmiana jednego i drugiego

## Operator rzutowania (ang. projection operator)

- Operuje na pojedynczej relacji będącej jego operandem, ale zawiera także dodatkowe wyrażenia warunkowe stanowiące jego dodatkowe parametry.
- Operator rzutowania zapisujemy w postaci:

$$\pi_{B_1, B_2, \dots, B_n}(R) \quad \begin{array}{l} R - \text{relacja} \\ B_1, B_2, \dots, B_n - \text{atrybuty} \end{array}$$

- Jeśli  $R$  jest relacją ze zbiorem atrybutów  $\{A_1, A_2, \dots, A_k\}$  oraz  $(B_1, B_2, \dots, B_n)$  jest listą pewnych atrybutów  $A$ , to  $\pi_{B_1, B_2, \dots, B_n}(R)$ , czyli rzutowanie relacji  $R$  na atrybuty  $B_1, B_2, \dots, B_n$  jest zbiorem krotek utworzonych przez wybranie z każdej krotki t tylko atrybutów  $B_1, B_2, \dots, B_n$ .
- Jedna lub więcej krotek może posiadać te same wartości atrybutów  $B_1, B_2, \dots, B_n$ .
- Jako wynik operacji rzutowania pojawia się tylko jedna taka krotka.

## Operator łączenia (ang. join operator)

- Umożliwia nam przechodzenie z jednej relacji do drugiej.
- Operator łączenia zapisujemy w postaci  $\&&$
- Przypuśćmy, że mamy dwie relacje  $R$  i  $S$ , których zbiory atrybutów (schematy) mają odpowiednio postać  $\{A_1, A_2, \dots, A_n\}$  oraz  $\{B_1, B_2, \dots, B_m\}$
- Z obu zbiorów wybieramy po jednym atrybutcie – powiedzmy  $A_i$  i  $B_j$  – i te atrybuty są parametrami naszej operacji złączenia, której argumentami są relacje  $R$  i  $S$ .
- Złączenie relacji  $R$  i  $S$  zapisujemy:  
$$R \underset{A_i = B_j}{\&\&} S$$

i jest utworzone w wyniku porównania każdej krotki  $r$  z relacji  $R$  z każdą krotką  $s$  z relacji  $S$ .
- Jeśli składowa  $r$  odpowiadająca atrybutowi  $A_i$  jest równa składowej  $s$  odpowiadającej atrybutowi  $B_j$  to tworzymy jedną krotkę.
- Schemat złączonej relacji jest  $\{A_1, A_2, \dots, A_n, B_1, B_2, \dots, B_{j-1}, B_{j+1}, \dots, B_m\}$
- A więc atrybut  $B_j$  się nie pojawia.
- Jeżeli atrybuty  $A_i$  i  $B_j$  mają tą samą nazwę to mówimy o złączeniu naturalnym.

Iloczyn kartezjański ( $R \times S$ ) także jest zdefiniowany klasycznie. Ponieważ jednak argumenty mogą mieć atrybuty o tych samych nazwach, nazwy kolumn w schemacie wynikowym trzeba czasem poprzedzać nazwami relacji, z których pochodzą, np. dla relacji  $R(A, B)$  i  $S(B, C)$  schematem ich iloczynu kartezjańskiego będzie  $R \times S(A, R.B, S.B, C)$ , tak jak w poniższym przykładzie

R1 =	A	B
	1	2
	3	4

R2 =	B	C
	5	6
	7	8

R1 × R2 =	A	R1.B	R2.B	C
	1	2	5	6
	1	2	7	8
	1	2	9	10
	3	4	5	6
	3	4	7	8
	3	4	9	10

Lepiej jednak w takiej sytuacji użyć przemianowania.

## Przykład

**ZDG**

Zajęcia	Dzień	Godzina
CS101	Pn	9.15
CS101	S	9.15
EE200	Pt	8.30
EE200	W	13.00
CS101	Pt	9.15

**ZK**

Zajęcia	Klasa
CS101	Aula
EE200	Hala
PH100	Laborat

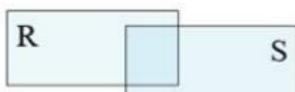
**Operacja łączenia**

**ZK**  
**Zajęcia=Zajęcia**

Zajęcia	Klasa	Dzień	Godzina
CS101	Aula	Pn	9.15
CS101	Aula	S	9.15
EE200	Hala	Pt	8.30
EE200	Hala	W	13.00
CS101	Aula	Pt	9.15

## Suma relacji

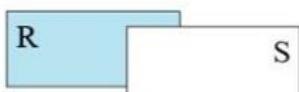
- Dwie relacje są zgodne (kompatybilne) jeśli mają taką samą liczbę atrybutów i odpowiadające sobie atrybuty mają tą samą dziedzinę.
- $R \cup S$  – zbiór wszystkich krotek, które należą do  $R$  lub  $S$ .



- $R$  – zbiór znajomych Jarka.
- $S$  – zbiór znajomych Joli
- $R \cup S$  – zbiór osób, które są znajomymi Jarka i/lub Joli

## Różnica relacji

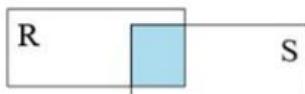
- Relacje muszą być kompatybilne.
- $R - S$  – zbiór wszystkich krotek, które należą do zbioru  $R$  i jednocześnie nie należą do zbioru  $S$ .



- $R$  – zbiór znajomych Jarka.
- $S$  – zbiór znajomych Joli
- $R - S$  – zbiór osób, które są znajomymi Jarka ale nie są znajomymi Joli

## Iloczyn relacji

- Relacje muszą być kompatybilne.
- $R \cap S$  – zbiór krotek, które należą zarówno do zbioru R i do zbioru S.
- Iloczyn relacji nie jest operacją prymitywną, bo  $R \cap S = R - (R - S) = S - (S - R)$



- R – zbiór znajomych Jarka.
- S – zbiór znajomych Joli
- R - S – zbiór tych osób, które są znajomymi Jarka i jednocześnie Joli

## Wielozbiory – suma, iloczyn

- Suma wielozbiorów daje inny wynik niż suma zbiorów. Jeśli jakaś krotka t w relacji R występuje m razy a w relacji S n razy, to w relacji będącej wynikiem sumy R i S krotka ta wystąpi m + n razy.
- Przecięcie wielozbiorów (iloczyn). Jeśli jakaś krotka t w relacji R występuje m razy a w relacji S n razy, to w relacji będącej wynikiem iloczynu R i S krotka ta wystąpi min(m,n) razy.

---

### 43. Własności (przekształcenia) algebry relacji

---

Legenda:

-Selekcja:

---

### 44. Optymalizacja zapytań: plan algebry relacji, optymalizacja planu algebry relacji, plan fizyczny.

---

**Cel użycia histogramów:**

Domyślnie optymalizator zakłada jednostajny rozkład (uniform distribution) wierszy między różnymi wartościami w kolumnie.

W przypadku kolumn zawierających skośność danych (nierównomierny rozkład danych w kolumnie) histogram umożliwia optymalizatorowi wygenerowanie dokładnych oszacowań liczności dla filtru i predykatów łączenia (join predicates), które dotyczą tych kolumn.

**Oszacowanie kosztów operacji we/wy za pomocą histogramów:**

- Dla wyboru indeksu – jaki jest koszt wyszukiwania indeksu?
- W celu podjęcia decyzji jakiego algorytmu użyć - np: aby wykonać  $R \bowtie S$ , którego algorytmu typu JOIN powinien użyć DBMS?
- Potrzeba sposobu oszacowania pośrednich rozmiarów zestawu wyników

Histogramy zapewniają sposób na skuteczne przechowywanie oszacowań tych ilości.

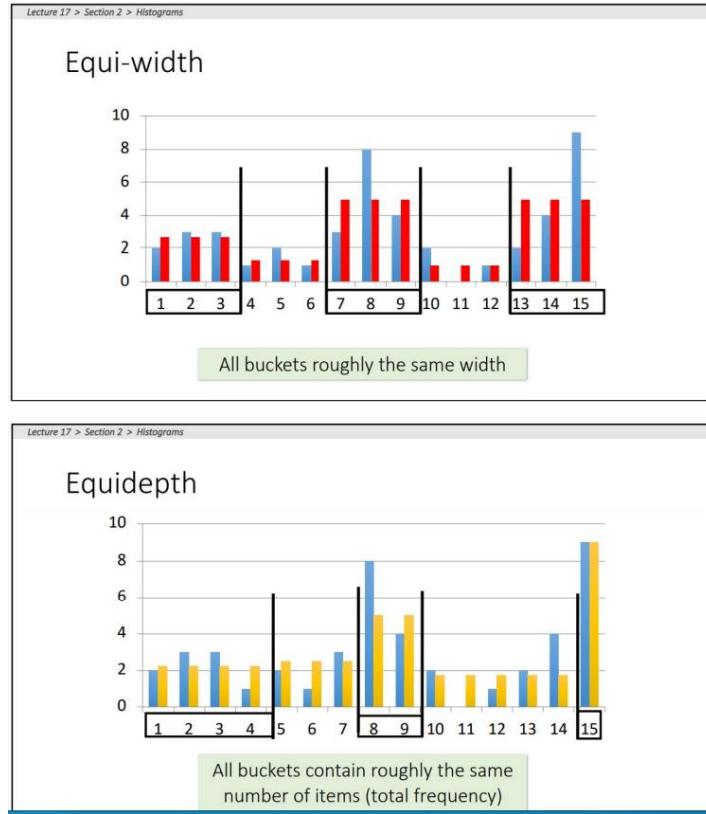
**Histogram:**

- **Definicja z prezentacji:** Histogram to zestaw zakresów wartości („segmentów”, „wiaderek” – „buckets”) i częstotliwości występowania wartości w tych segmentach.
- **Definicja z Wikipedii:** Histogram – jeden z graficznych sposobów przedstawiania rozkładu empirycznego cechy. Składa się z szeregu prostokątów umieszczonych na osi współrzędnych. Prostokąty te są z jednej strony wyznaczone przez przedziały klasowe (patrz: szereg rozdzielczy) wartości cechy, natomiast ich wysokość jest określona przez liczebności (lub częstotliwości, ewentualnie gęstość prawdopodobieństwa) elementów wpadających do określonego przedziału klasowego.
- Jeśli histogram pokazuje liczebności, a nie gęstość prawdopodobieństwa, wówczas szerokości przedziałów powinny być równe.
- Histogramy są proste, intuicyjne w użyciu i popularne
- Parametry: liczba „wiaderek” i typ
- Może obejmować wiele atrybutów (wielowymiarowy)
- Histogramy muszą być aktualizowane za pomocą wykonania/zaplanowania wykonania komendy aktualizującej statystyki dla bazy danych
- Wartości o wysokich częstotliwościach w histogramie są bardzo istotne.

Szeroko stosowane są dwa rodzaje histogramów:

- Equi-width – wszystkie wiaderka mają tę samą szerokość
- Equi-depth – wszystkie wiaderka zawierają tę samą liczbę wierszy (tj. ich szerokość jest zmienna)

Histogramy equi-depth są w stanie dostosować się do zniekształcenia danych (wysoka jednorodność).



Skompresowane histogramy:

Popularne podejście:

1. Przechowuj jawnie najczęściej występujące wartości i ich licznosci
2. Dla pozostałych wartości przygotuj histogram equi-width lub equi-depth

---

#### 46. XML: zastosowania, właściwości, składnia

---

#### DEFINICJA XML:

XML (eXtensible Markup Language) jest językiem znaczników o hierarchicznym, przyjaznym człowiekowi (human-readable) formacie. „Brat” HTML'a, zawsze parsowalny (always parseable). Koduje dokumenty i ustrukturyzowane dane. Łączy w sobie dane i strukturę (schema). Ratyfikowany przez World Wide Web Consortium (W3C), XML stał się standardem identyfikowania i opisywania danych (dokumentów) przekazywanych w sieci WWW.

Rdzeń ekosystemu:

**Core of a broader ecosystem**

- Data – XML (also RDF, Ch. 12)
- Schema – DTD and XML Schema
- Programmatic access – DOM and SAX
- Query – XPath, XSLT, XQuery
- Distributed programs – Web services

**ZASTOSOWANIA:**

XML może zostać użyty do:

- oddzielenia danych od dokumentów HTML (dane mogą być trzymane w dokumentach XML na zewnątrz HTML, lub wewnętrz w tzw. "wyspach danych", ale wciąż mając świadomość, że HTML jest używany tylko do formatowania i wyświetlania)
- wymiany danych (w tym również między różnymi, niekompatybilnymi systemami)
- wymiany danych przez Internet ( np. B2B - informacje finansowe )
- udostępniania danych (dokumenty XML są pisane w czystym tekscie co umożliwia nam niezależną od sprzętu i oprogramowania metodę udostępniania danych)
- przechowywania danych (można napisać aplikacje, które będą magazynowały i w razie potrzeby pobierały dane z pliku czy bazy danych opartej o XML

**WŁAŚCIWOŚCI:**

- Łatwo można w nim kodować relacje
- Znakomity do obsługi danych o złożonej strukturze oraz nietypowych danych
- Idealny do długotrwałego przechowywania danych i ich ponownego użycia
- Opisuje dane językiem znaczników
- Posiada tylko jeden element główny (korzeń – root)
- Dane umieszczone są w strukturze drzewa
- Jest semistrukturalny
- XML jako model danych:

**„Zestaw informacyjny” XML składa się z 7 typów węzłów:**

- Korzeń dokumentu (document root)
- Element
- Atrybut
- Instrukcja przetwarzania

- Tekst (treść)
- Przestrzeń nazw (namespace)
- Komentarz

#### SKŁADNIA:

- XML jest językiem case-sensitive, tzn. to czy użyjemy małych czy dużych liter w znacznikach itp. ma znaczenie!
- Nazwy elementów muszą podlegać następującym regułom:
  - mogą zawierać litery, liczby i inne znaki
  - nie mogą zaczynać się od liczby lub znaku interpunkcyjnego
  - nie mogą zaczynać się od liter xml (XML, Xml itd)
  - nie mogą zawierać spacji
- Instrukcje przetwarzania (PI) dostarczają specjalnych instrukcji dla aplikacji. Nie są one częścią danych znakowych dokumentu i są bez ich przetwarzania przekazywane do aplikacji. Instrukcje przetwarzania mają postać:

`<?name pidata?>`

Nazwa name identyfikuje instrukcję. Konkretna aplikacja reaguje tylko na pewne instrukcje przetwarzania, a pozostałe pomija. Dane pidata występujące po nazwie instrukcji przetwarzania są opcjonalne i mają znaczenie tylko dla aplikacji, które je rozpoznają. Nazwy zaczynające się słowem 'xml' są zarezerwowane dla standardu XML.

- `<?xml version="1.0" encoding="ISO-8859-1" ?>`

Deklaracja XML (preambuła) – przykład instrukcji przetwarzania, może występować na początku dokumentu. Służy do podania wersji standardu XML i standardu kodowania użytego do zapisania dokumentu. Można ją pominąć jeśli mamy dokument XML 1.0 z kodowaniem UTF-8

- `<dblp>`

Znacznik początku elementu (tag otwierający) – oznacza początek elementu. Musi występować w parze z odpowiadającym znacznikiem końca elementu (w tym przypadku ). Wyjątek od tej reguły stanowią elementy puste (nie zawierające żadnych danych ani innych elementów, mogą zawierać atrybuty), np.

`<tag/>` skrót dla pustych tagów (ekwiwalent `<tag></tag>`)

- <title>PRPL: A Database Workload Specification Language</title>

Element – jeden z dwóch rodzajów elementów danych w języku XML. Hierarchiczna struktura posiadająca parę: tag otwierający-tag zamykający. Może zawierać zagnieżdżone w sobie inne elementy. Jej tag otwierający może zawierać atrybuty. Wiele elementów może mieć tą samą nazwę. W przypadku elementów kolejność ma znaczenie.

- <article mdate="2002-01-03" key="tr/dec/SRC1997-018">

Atrybut – nazwana wartość (nie hierarchiczna). W pojedynczym elemencie może występować tylko jeden atrybut o danej nazwie. Kolejność atrybutów nie ma znaczenia.

- <journal>Digital System Research Center Report</journal>

Znacznik końca elementu (tag zamykający) – oznacza koniec wcześniejszej otwartego elementu. Musi występować w parze z odpowiadającym znacznikiem początku elementu.

Przykład:

```

XML Anatomy

<?xml version="1.0" encoding="ISO-8859-1" ?> ← Processing Instr.
<dblp> ← Open-tag
  <mastersthesis mdate="2002-01-03" key="ms/Brown92">
    <author>Kurt P. Brown</author>
    <title>PRPL: A Database Workload Specification Language</title> ← Element
    <year>1992</year>
    <school>Univ. of Wisconsin-Madison</school> ← Element
  </mastersthesis>
  <article mdate="2002-01-03" key="tr/dec/SRC1997-018"> ← Attribute
    <editor>Paul R. McJones</editor>
    <title>The 1995 SQL Reunion</title>
    <journal>Digital System Research Center Report</journal>
    <volume>SRC1997-018</volume> ← Close-tag
    <year>1997</year>
    <ee>db/labs/dec/SRC1997-018.html</ee>
    <ee>http://www.mcjones.org/System_R/SQL_Reunion_95/</ee>
  </article>

```

---

#### 47. DTD i XSD: zastosowania, właściwości, składnia

---

### DTD - Document Type Definition

Zastosowania:

- użycie przez różne grupy tego samego standardu do lepszej wymiany danych.
- Sprawdzenie danych przychodzących czy są prawidłowe

- Sprawdzenie danych własnych czy są prawidłowe.

```
<?xml version="1.0"?>
<!DOCTYPE note [
  <!ELEMENT note (to,from,heading,body)>
  <!ELEMENT to (#PCDATA)>
  <!ELEMENT from (#PCDATA)>
  <!ELEMENT heading (#PCDATA)>
  <!ELEMENT body (#PCDATA)>
]>
<note>
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend</body>
</note>
```

Właściwości:

!DOCTYPE note określa, że głównym elementem tego dokumentu jest uwaga(note)

!ELEMENT note określa, że element notatki musi zawierać cztery elementy: „do, od, nagłówek, treść”

!ELEMENT to definiuje element do typu „#PCDATA”

!ELEMENT from definiuje element from jako „#PCDATA” !Pozycja heading definiuje element nagłówka typu „#PCDATA” !Element body definiuje element nadwozia typu „#PCDATA”

#PCDATA oznacza analizowalne dane znakowe

### XSD – XML Schema Definition – opisuje strukturę dokumentu

Zastosowania:

**Celem schematu XML jest zdefiniowanie legalnych elementów składowych dokumentu XML:**

1. elementy i atrybuty, które mogą pojawić się w dokumencie
2. liczba (i kolejność) elementów potomnych
3. typy danych dla elementów i atrybutów
4. domyślne i stałe wartości elementów i atrybutów

Przykład:

```
<?xml version="1.0"?>
<xss:schema xmlns:xss="http://www.w3.org/2001/XMLSchema">

<xss:element name="note">
  <xss:complexType>
    <xss:sequence>
      <xss:element name="to" type="xs:string"/>
      <xss:element name="from" type="xs:string"/>
      <xss:element name="heading" type="xs:string"/>
      <xss:element name="body" type="xs:string"/>
    </xss:sequence>
  </xss:complexType>
</xss:element>

</xss:schema>
```

#### Właściwości:

- W świecie XML setki standardowych formatów XML są codziennie używane.
- Wiele z tych standardów XML jest zdefiniowanych przez schematy XML.
- Schemat XML jest opartą na XML (i bardziej wydajną) alternatywą dla DTD.

---

#### *48. XPath: zastosowania, właściwości, składnia*

---

**XPath to inaczej XML Path Language** (Język ścieżek XML). Język ten używa składni nie-XML-owej, pozwalając w elastyczny sposób wskazywać różne części dokumentu XML. Ponadto może być używany do sprawdzania, czy wskazane węzły dokumentu pasują do wzorca.

**XPath jest głównie używany w XSLT**, ale może być także wykorzystywany do znacznie bardziej wymagających zadań nawigacji po DOM dowolnego języka bazowanego na XML-u, takiego jak HTML czy XUL, zamiast opierania się na metodzie `document.getElementById`, właściwościach `element.childNodes`, itp. The following is vague: This is especially useful within extensions, particularly regarding to overlays.

**XPath używa notacji ścieżkowej** (ang. path notation) (tak jak adresy URL) do nawigacji po hierarchicznej strukturze dokumentu XML. Używa do tego nie-XML-owej składni, dzięki czemu może być stosowany w adresach URI oraz wartościach atrybutów XML.

---

#### *49. XQuery: zastosowania, właściwości, składnia*

---

XQuery (ang. XML query language) – język zapytań (jednakże posiadający pewne cechy języka programowania) służący do przeszukiwania dokumentów XML. Jest to statycznie typowany język deklaratywny oparty na wyrażeniach ścieżkowych. Jest to język wyrażeń, nie ma w nim instrukcji.

**Składnia:**

- XQuery jest case sensitive - stringi umieszczamy w apostrofach lub cudzysłowie
- zmienne definiujemy przy pomocy \$ np. \$zmienna
- komentarz otwieramy i zamykamy dwukropkiem np.: to jest komentarz studencie leniwy

**Zastosowania:**

- wyciąganie informacji z bazy danych do użytku w usługach webowych
- tworzenie podsumowań na temat danych z dokumentów XML - przeszukiwanie dokumentów tekstowych w internecie
- wybieranie i przekształcanie danych XML do XHTML celem publikacji w internecie
- wyciąganie danych z bazy celem użycia do integracji aplikacji
- rozdzielenie XMLa, który reprezentuje wiele transakcji w wiele dokumentów XML.

```
for $d in doc("dzialy.xml")//nrdzialu
let $p := doc("pracownicy.xml")//pracownik[nrdzialu = $d]
where count($p) >= 10
order by avg($p/pensja) descending
return
<ZbiorczoDzial>
{ $d,
  <zatrudnionych>{count($p)}</zatrudnionych>,
  <sredniapensja>{avg($p/salary)}</sredniapensja>
}
</ZbiorczoDzial>
```

---

## *50. Zagrożenia związane z bezpieczeństwem baz danych*

---

### **Zagrożenia dla baz danych:**

- utrata integralności – nieprawidłowe tworzenie, modyfikacja lub usunięcie danych w bazie danych,
- utrata dostępności – kiedy baza danych staje się niedostępna dla podmiotu, który powinien mieć do niej dostęp,
- utrata tajności – kiedy pewna część danych staje się dostępna dla nieuprawnionych do tego podmiotów (złamanie RODO).

---

## *51. Mechanizmy zapewnienia bezpieczeństwa baz danych: Discretionary Access Control, Role-Based Access Control, Mandatory Access Control*

---

### **Discretionary Access Control (DAC) – uznaniowa kontrola dostępu**

Podstawowy mechanizm ochrony danych w bazie. DAC jest oparty na uprawnieniach do wykonywania operacji na obiektach bazy danych. Właściciel obiektu ma do niego pełne prawa. Może część z tych praw przekazać innym, wybranym przez siebie użytkownikom.

Pseudokod pokazujący nadawanie uprawnień:

```
GRANT uprawnienie, ...
ON nazwa_obiektu
TO użytkownik, ...
[WITH GRANT OPTION]
```

Osoba przyznająca uprawnienie może to uprawnienie w przyszłości odwołać.

Pseudokod pokazujący odwołanie uprawnień:

```
REVOKE uprawnienie, ...
ON nazwa_obiektu
FROM użytkownik, ...
```

Przy stosowaniu REVOKE gdy uprawnienie zostanie odjęte od użytkownika X, zostaje również odjęte od wszystkich użytkowników, którzy dostali je wyłącznie od X.

Właściciel tabel może określić zakres dostępu do swoich danych w bazie danych poprzez perspektywy.

Zarządzanie uprawnieniami i użytkownikami jest wspomagane przez role, które odzwierciedlają sposób funkcjonowania organizacji. W trakcie działania baz danych role można dynamicznie włączać i wyłączać, np.:

SET ROLE operator;

Użytkownik aplikacji ma na stałe przyznane tylko uprawnienie CONNECT upoważniające do zalogowania się do bazy danych.

#### **Mandatory access control (MAC) – obowiązkowa kontrola dostępu**

Obowiązkowa kontrola dostępu jest mechanizmem stosowanym dodatkowo oprócz uznaniowej kontroli dostępu. Jej celem jest ochrona bazy danych przed nieuprawnionymi zmianami jakie mogą być dokonane poza bazą danych - w programach aplikacyjnych (w rodzaju kodu wprowadzającego konia trojańskiego bez wiedzy użytkownika aplikacji).

Obowiązkowa kontrola dostępu jest oparta na klasach poufności (ang. security class) przypisanych do poszczególnych obiektów w bazie danych i do użytkowników (lub programów). Przez porównanie klasy obiektu i klasy użytkownika, system podejmuje decyzję czy użytkownik może wykonać określzoną operację na obiekcie. Przede wszystkim, zapobiega to płynięciu informacji z wyższego poziomu poufności do niższego.

Obowiązkowa kontrola dostępu zapobiega użyciu koni trojańskich do dostępu do chronionych danych. Rozważmy następującą sytuację:

1. Haker tworzy tabelę MojaTabela i daje do niej uprawnienia użytkownikowi, podczas gdy użytkownik nie jest tego świadomym.
2. Haker modyfikuje kod programu klienckiego (który znajduje się poza kontrolą SZBD) wstawiając zapisywanie danych z tabeli użytkownika PoufneDane do tabeli MojaTabela.
3. Przy każdym użyciu programu przez użytkownika poufne dane zostają przepisane do tabeli hakera MojaTabela.

#### **Model Bell-LaPadula obejmuje**

- Obiekty np. tabele, perspektywy, wiersze.
- Podmioty np. użytkownicy, programy użytkowników.
- Klasy poufności przypisywane podmiotom class(P) i obiektom class(O); tworzą one liniowy porządek.  
Np.:

top secret - TS, secret - S , confidential - C, unclassified - U: TS > S > C > U

- Każdemu obiekowi i każdemu podmiotowi zostaje przypisana klasa poufności.

Odczytywanie i zapis danych przebiegają zgodnie z następującymi regułami:

- (1) Podmiot P może odczytać obiekt O tylko wtedy, gdy class(P)  $\geq$  class(O)
- (2) Podmiot P może zapisać obiekt O gdy class(P)  $\leq$  class(O)

Rozwiązywanie problemu konia trojańskiego polega na zapewnieniu, aby informacje nie płynęły od wyższej klasy poufności do niższej.

1. Użytkownik ma klasę S, program użytkownika i tabela użytkownika PoufneDane mają klasę S.
2. Haker ma klasę C, tabela hakera, MojaTabela, ma tę samą klasę co haker, czyli C.
3. Zatem program użytkownika (mający klasę poufności S) nie może wykonać zapisu do tabeli hakera (mającej klasę poufności C) bo S>C.

W zależności od posiadanej klasy poufności użytkownik może widzieć inny zbiór wierszy przy wykonywaniu instrukcji SELECT.

#### **Role-based access control (RBAC)** – kontrola dostępu oparta na rolach

RBAC polega na zdefiniowaniu tzw. ról dla różnych funkcji w organizacji, z którymi wiąże się określony zakres obowiązków. Poszczególnym roliom są centralnie przydzielane stosowne uprawnienia w systemie informatycznym. Role są przypisywane użytkownikom, przez co uzyskują oni uprawnienia do wykonywania określonych dla tych ról czynności. Użytkownik może posiadać wiele przypisanych ролей. Rola może być przypisana wielu użytkownikom.

W odróżnieniu od DAC i MAC nie przypisujemy uprawnień bezpośrednio użytkownikom.

Cechą charakterystyczną RBAC jest określanie ról i uprawnień w taki sposób, aby odzwierciedlały one rzeczywiste funkcje w organizacji.

RBAC sprawdza się szczególnie dobrze tam, gdzie ważne jest stosowanie zasady rozdziału obowiązków. Przykładowo, gdy w celu zapobiegania nadużyciom niektórym operacje wymagają akceptacji dwu niezależnych użytkowników.

---

#### [\*52. Przywileje \(privileges\)\*](#)

---

- Użytkownik ma pełne przywileje (uprawnienia) do obiektów stworzonych przez niego
- Przywileje z poziomu konta -> DBA (Database administrator) określa szczególne przywileje, które posiada każde konto niezależnie od relacji w bazie danych (CREATE SCHEMA, CREATE TABLE, CREATE VIEW)
- Z poziomu relacji (lub tabeli) -> DBA może kontrolować uprawnienia dostępu do każdej relacji lub widoku w bazie danych (ALL, SELECT, INSERT, UPDATE, DELETE, CREATE, DROP)

Polecenie **GRANT** służy do zakładania kont użytkownikom bazy danych i do nadawania im pewnych przywilejów związanych z dostępem do danych.

Polecenie **REVOKE** służy do odbierania nadanych uprawnień zadanym użytkownikom.

Składnia:

```
GRANT SELECT, INSERT, UPDATE, DELETE, CREATE, DROP  
ON table_name TO user_name  
WITH GRANT OPTION;
```

```
REVOKE INSERT, UPDATE  
ON table_name FROM user_name;
```

Przykład:

**USER A1:**

- GRANT SELECT ON EMPLOYEE TO B2 WITH GRANT OPTION;
- GRANT SELECT, INSERT ON EMPLOYEE TO B3 WITH GRANT OPTION;

**USER B2:**

- GRANT SELECT ON EMPLOYEE TO B4;

**USER B3:**

- GRANT SELECT, INSERT ON EMPLOYEE TO B4

**USER A1:**

- REVOKE SELECT ON EMPLOYEE FROM B3;



---

### 53. Polyinstantiation

---

**Polyinstantiation** - koncepcja typu (klasy, wiersza bazy danych) tworzonego w wielu niezależnych instancjach (obiektach, kopiach). Może również wskazywać, tak jak w przypadku poliinstantyzacji bazy danych, że dwa różne wystąpienia mają taką samą nazwę (identyfikator, klucz główny).

W bazach danych poliinstantacja jest terminologią SQL. Pozwala, żeby relacja zawierała wiele wierszy z tym samym kluczem głównym, to kilka instancji jest rozróżniane według poziomów bezpieczeństwa.

W zależności od ustalonego poziomu bezpieczeństwa, jeden rekord zawiera poufne informacje, a drugi nie. Użytkownik zobaczy więc informacje o rekordzie w zależności od jego stopnia poufności podyktywanego przez politykę danej firmy.

W poniższej tabeli kluczem głównym jest „Name” a  $\lambda(x)$  to poziom bezpieczeństwa:

Name	$\lambda(\text{Name})$	Age	$\lambda(\text{Age})$	$\lambda$
Alice	Secret	18	Top Secret	Top Secret
Bob	Secret	22	Secret	Secret
Bob	Secret	33	Top Secret	Top Secret
Trudy	Top Secret	15	Top Secret	Top Secret

Chociaż polinstantiacja jest przydatna, jeżeli chodzi o bezpieczeństwo bazy, to może spowodować problemy:

- moralne, wiąże się ona z kłamstwem,
- eksplozja w liczbie rzędów.

Stopniowanie poziomu bezpieczeństwa od najwyższego do najniższego:

1. Top secret TS
2. Secret S
3. Confidential C
4. Unclassified U

## Polyinstantiation

Name	Salary	JobPerformance	User clearance
The original table:			
Smith U	40 000 C	Fair S	S
Brown C	80 000 S	Good C	S

Appearance of EMPLOYEE after filtering for classification C users:

Name	Salary	JobPerformance	User clearance
Smith U	40 000 C	NULL S	C
Brown C	NULL S	Good C	C

Appearance of EMPLOYEE after filtering for classification U users:

Name	Salary	JobPerformance	User clearance
Smith U	NULL C	NULL S	U

**top secret (TS) > secret (S) > confidential (C) > unclassified (U)**

1. Oryginalna tabela. Jak widać dostęp do nazwy “Brown” mają dostęp użytkownicy z uprawnieniami na poziomie C, natomiast do “Smith” wszyscy.
2. Employee po filtrowaniu dla uprawnień poziom C (confidential – poufny). W miejscach na poziomie S (sekretne) pojawił się NULL zamiast informacji.
3. Employee po filtrowaniu dla wierszy zawierających poziom U (unclassified – nieklasyfikowany) - wszystko poza “Smith” jest NULLem.

## Polyinstantiation

Name	Salary	JobPerformance	User clearance
Smith U	40 000 C	Fair S	S
Brown C	80 000 S	Good C	S

User with security clearance level C:

```
UPDATE EMPLOYEE
SET JobPerformance = 'Excellence'
WHERE Name = 'Smith';
```

Name	Salary	JobPerformance	User clearance
Smith U	40 000 C	Fair S	S
Smith U	40 000 C	Excellent C	C
Brown C	80 000 S	Good C	S

top secret (TS) > secret (S) > confidential (C) > unclassified (U)

Security clearance = zwolnienie bezpieczeństwa

1. U góry oryginalna tabela.

2. Tutaj tworzymy kopie linijki z nazwą „Smith”. Po wykonaniu kodu zmieniamy JobPerformance na „Excellent” z poziomem C dla wierszy, gdzie nazwa to „Smith”. Ze względu na górną linijkę „user with security clearance level C” nowe User clearance jest ustawione właśnie na C. Pozostałe dane zostały wyświetcone dobrze gdyż są one na poziomie U oraz C.

Zmiana danych bez odpowiedniego poziomu tajności spowoduje stworzenie kopii danego rekordu z nowymi danymi i z innym poziomem tajności.

---

### [54. SQL injection](#)

---

**Definicja** – jest to metoda ataku komputerowego wykorzystująca lukę w zabezpieczeniach aplikacji polegającą na nieodpowiednim filtrowaniu lub niedostatecznym typowaniu danych użytkownika, które to dane są później wykorzystywane przy wykonywaniu zapytań (SQL) do bazy danych. Na taki atak podatne są wszystkie systemy przyjmujące dane od użytkownika i dynamicznie generujące zapytania do bazy danych.

#### Formy ataku:

- o Niedostateczne filtrowanie danych – opiera się na nieodpowiednim filtrowaniu znaków ucieczki z danych wejściowych, co pozawala m.in. na modyfikację zapytania albo przekazanie dodatkowych zapytań np. niszczących dane
- o Modyfikowanie zapytań

```
$q = mysql_query("SELECT * FROM uzytkownicy WHERE uzytkownik  
= '$uzytkownik');
```

- Gdy użytkownik jako \$uzytkownik poda wartość „kowalski” zapytanie spełni swoją funkcję przyjmując postać:

```
SELECT * FROM uzytkownicy WHERE uzytkownik = 'kowalski'
```

- Gdy użytkownik przekaże wartość „x’ OR ‘1’='1” zapytanie spowoduje pobranie z bazy wszystkich rekordów przyjmując postać:

```
SELECT * FROM uzytkownicy WHERE uzytkownik = 'x' OR '1'='1'
```

- Przekazywanie dodatkowych zapytań – w teorii można przekazać każde zapytanie SQL, włącznie z wykonaniem kilku zapytań na raz w praktyce może być to niemożliwe ze względu na używane biblioteki które nie pozwalają na wykonywanie dwóch zapytań jednocześnie
  - Gdy użytkownik przekaże „x'; DROP TABLE uzytkownicy; SELECT '1” zapytanie spowoduje usunięcie tabeli przyjmując postać:

```
SELECT * FROM uzytkownicy WHERE uzytkownik = 'x'; DROP TABLE  
uzytkownicy; SELECT '1'
```

- Blokowanie serwera - Korzystając z ataku typu SQL Injection można również przeprowadzić atak typu DoS

```
x' AND  
BENCHMARK(9999999, BENCHMARK(999999, BENCHMARK(999999, MD5(NOW())  
)))=0 OR '1'='1
```

W tym przypadku serwer spróbuje obliczyć skrót MD5 dla aktualnego czasu  $999999^3 = 999997000002999999$  (w zaokrągleniu trylion) razy.

- „Ślepy” atak – mówi się o nim w przypadku wykonywania ataku typu *SQL Injection* na stronie, która nie wyświetla komunikatów błędów.
  - Zapytanie powinno nic nie zwrócić:

```
SELECT * FROM uzytkownicy WHERE uzytkownik='x' AND 1=2;
```

- Zapytanie powinno zawsze zwrócić jakikolwiek wynik, bowiem 1 jest równe 1.

Badając zmiany zachodzące na stronie można stwierdzić czy w danym miejscu można dokonać ataku

- Błędy w serwerze SQL - Czasami błędy umożliwiające atak występują w samym serwerze SQL, jak było w przypadku funkcji `real_escape_chars()` z [MySQL](#).
- Skutki:
  - Nieautoryzowany dostęp w trybie odczytu lub zapisu do całej bazy danych
  - Możliwość ominięcia mechanizmu uwierzytelnienia
  - Możliwość odczytania wybranych plików (system operacyjny, na którym pracuje baza danych)
  - Możliwość tworzenia plików w systemie operacyjnym na którym pracuje baza
  - Możliwość wykonania kodu w systemie operacyjnym (uprawnienia użytkownika, na którym pracuje baza lub web serwer – w przypadku aplikacji webowych)
- Zabezpieczenia
  - Zabezpieczenie na poziomie aplikacji
    - Niedopuszczenie do nieuprawnionej zmiany wykonywanego zapytania.
      - ✓ wykonanie na każdym tekstowym parametrze wykorzystywanym do budowy zapytania wbudowanej funkcji `addslashes()`, która dodaje backslash przed znakami, takimi jak ', " czy \, dzięki czemu znaki te nie są traktowane jak znaki specjalne (PHP)
      - ✓ użycie funkcji `mysql_real_escape_string()` oferowanej przez serwer MySQL dla danych tekstowych
      - ✓ użycie funkcji `is_numeric(zmienna)`, , która sprawdza czy zmienna jest wartością numeryczną. Po sprawdzeniu zmiennej i upewnieniu się, iż jest liczbą, można bezpiecznie użyć zmiennej w zapytaniu SQL.
      - ✓ użycie mechanizmu tzw. „zaślepek”, gdzie zmienne nie są używane bezpośrednio do tworzenia zapytania, a odpowiednie dane dołączane są do zapytania w momencie jego wykonania

- ✓ zastosowanie paradygmatu security through obscurity poprzez wyłączenie wyświetlania komunikatów o błędach
- Zabezpieczenie na poziomie bazy danych
  - Udostępnienie użytkownikowi bazy tylko niezbędnych uprawnień
  - Wyłączenie niepotrzebnej funkcjonalności na poziomie samego silnika
  - Zastosowanie procedur składowanych dzięki którym zapytanie budowane jest po stronie bazy danych i aplikacja nie ma bezpośredniego wpływu na jego postać
  - Całkowite wyłączenie możliwości podawania parametrów jako części zapytania. Jest to działanie analogiczne do mechanizmu zaślepek po stronie aplikacji, jednak tym razem zastosowane w samym silniku. (Możliwość taką daje silnik H2)
- Zabezpieczenie na poziomie serwera aplikacji/www
  - Instalacja dodatkowych modułów do serwera warstwy aplikacyjnej (np. mod\_security do serwera Apache) filtrujących według zdefiniowanych reguł przychodzące żądania i blokujących te potencjalnie groźne