

Podstawy Baz Danych

Notatki i opracowanie pytań z egzaminu 2016/17

// Generalnie chyba niezłym pomysłem byłoby wstawianie komentarzy, jak się coś modyfikuje/dodaje, bo to ma jednak sporo stron i trochę ciężko potem ogarnąć.

x

[E] - Pytanie z egzaminu 2016 (i 2017)

[E2017] - Nowe pytanie z egzaminu 2017

[Dodatkowo] - Nie pojawiło się, ale warto znać

Bazy

Spis treści

1. Wykład - zasady oceniania	11
2. Wykład - Wprowadzenie w problematykę baz danych	11
2.1. [E2017] System Zarządzania Bazami Danych SZBD (DBMS) m	11
2.2. [E] Po co tworzymy bazy danych?	11
2.3. [E] Niedogodności plików do gromadzenia danych	11
2.3.1. Wymagana spora ilość programów do obsługi plików	11
2.3.2. Nadmiarowość i niespójność danych	12
2.3.3. Trudności w efektywnym dostępie do danych	12
2.3.4. Izolacja danych	12
2.3.5. Problemy z integralnością	12
2.3.6. Problemy z atomowością	12
2.3.7. Równoległy dostęp przez wielu użytkowników	12
2.3.8. Problemy z bezpieczeństwem (logowanie, prawa dostępu)	12
2.3.9. Trudności z optymalizacją	12
2.4. [E] Warstwy logiczne i fizyczne bazy danych	13
2.4.1. Poziom fizyczny:	13
2.4.2. Poziom logiczny	13
2.4.3. [Dodatkowo] Poziom widoku:	13
2.4.4. [Dodatkowo] Zależność pomiędzy tymi poziomami abstrakcji	14
2.5. [E] Co to instancja? Co to jest schemat?	14
2.5.1. Instancja	14
2.5.2. Schemat	14
2.5.3. [Dodatkowo] Instancje i schematy	15
2.6. [E] Czym jest fizyczna izolacja danych?	15
2.6.1. Fizyczna niezależność danych	15
2.7. [E] Co to jest model danych, podaj jego elementy i przykłady modeli. Co to jest model konceptualny? Czym różni się od implementacyjnego?	16
2.7.1. Model danych	16
2.7.2. Przykłady modeli danych	16
2.7.3. Model konceptualny i implementacyjny	17
2.8. [E] Języki baz danych/Rodzaje języków	17
2.8.1. Język definicji danych Data Definition Language (DDL)	17
2.8.2. Język manipulacji danymi Data Manipulation Language (DML)	18
2.9. [E] Funkcjonalne komponenty systemu BD, za co odpowiadają	18
2.9.1. Menedżer składowania	19
2.9.2. [E2017] Procesor przetwarzania zapytań zawiera komponenty:	19
2.10. [E2017] Architektura baz danych	20

2.10.1. Architektura systemów baz danych	20
2.10.2. Architektura aplikacji	21
3. Wykład - Wstęp do modelu relacyjnego	21
3.1. [Dodatkowo] Relacyjny model danych	21
3.1.1. Cechy modelu relacyjnego	21
3.1.2. Cechy atrybutów	22
3.1.3. Schemat relacji i instancja	23
3.1.4. Baza danych	23
3.2. [E] Opowiedzieć o kluczach (w kontekście modelu relacyjnego)	23
3.2.1. [Dodatkowo] Wstęp	23
3.2.2. Superklucz	24
3.2.3. Klucz kandydujący (potencjalny)	24
3.2.4. Klucz główny/podstawowy (primary key, PK)	24
3.2.5. Klucz obcy relacji (foreign key FK)	24
3.3. [E] Integralność referencyjna	25
3.4. [E] Czy w praktyce klucz obcy może być NULLem?	25
3.5. [E] Zawężenie dziedziny, checki	26
4. Wykład - Algebra relacji	27
4.1. [E2017] Algebra relacji	27
4.1.1. Podstawowe operatory:	27
4.1.1.1. Operator selekcji (select): σ	27
4.1.1.2. Operator projekcji (project): Π	29
4.1.1.3. Operator unii (union): \cup	30
4.1.1.5. Operator różnicy zbiorów (set difference difference):	31
4.1.1.6. Operator produktu kartezjańskiego (Cartesian product): \times	32
4.1.1.7. Operator przemianowania (assignment): ρ	33
4.1.2. Dodatkowe operatory:	33
4.1.2.1. Przecięcie zbiorów (set intersection): \cap	34
4.1.2.2. [E2017] Złączenie naturalne (natural join) i złączenie theta:	35
Złączenie naturalne:	35
Złączenie naturalne i złączenie theta:	36
4.1.2.5. Przypisanie (assignment): \leftarrow	37
4.1.2.6. Złączenie zewnętrzne (outer join)	38
4.1.3. Operatory rozszerzonej algebry relacji	39
4.1.3.1. [E2017] Uogólniona projekcja	40
4.1.3.2. [E2017] Operacje agregacji	40
4.1.4. [Dodatkowo] SQL i algebra relacji	42
4.2. [E] Przetwarzanie zapytań	42
4.2.1. Podstawowe kroki w przetwarzaniu zapytań:	42
5. Wykład - Projektowanie bazy danych, Model związków-encji E-R	43
5.1. [Dodatkowo] Model związków encji ER	43

5.1.1. Projekt związków encji	43
5.2. [E] Czym jest zbiór związków encji?	44
5.2.1. Ze slajdów: Zbiory związków	44
5.2.2. Z poprzedniego opracowania:	45
5.3. [E] Cechy związków encji	45
5.3.2. Ograniczenie krotności (poniższe dotyczą związków binarnych)	46
5.3.3. Istnienie	47
5.3.4. Udział zbioru encji w zbiorze związków	47
5.4. [E] Czym jest klucz kandydujący w ER?	48
5.4.1. [Dodatkowo] Klucze dla związków encji	48
5.4.2. [E] Klucz kandydujący	48
5.4.3. [Dodatkowo] Klucz główny	48
5.5. [E] Jak się wyznacza klucz główny w zbiorach związków?	49
5.6. [E2017] Zbiory słabych encji. Co to są słabe encje, objaśnić przekształcanie ich do modelu relacyjnego.	49
5.6.1. Encja słaba	49
5.6.2. Identyfikator encji słabej	50
5.6.3. Z wykładu: Zbiór słabych encji	50
5.6.4. Przekształcanie słabych encji do modelu relacyjnego	51
5.7. [E] Co to jest agregacja?	51
5.7.1. Agregacja	51
5.7.2. Przykład agregacji	51
5.7.3. Przykład z wykładu	52
5.8. [E] Zamiana agregacji w ER na postać relacyjną (na model relacyjny?)	55
5.9. [E] Generalizacja i specjalizacja w ER	56
5.10. [E] Ograniczenia na generalizację/specjalizację w ER	57
5.11. [E2017] Mapowanie generalizacji i specjalizacji z modelu związków encji na model relacyjny. Klucze główne w relacjach niższych i wyższych poziomów. Generalizacja w ER	58
5.11.1. Generalizacja i specjalizacja	58
5.11.2. Reprezentacja generalizacji/specjalizacji w schemacie (z wykładu - 2 metody/schematy)	58
5.11.3. metody w skrócie:	58
5.11.4. Opis metod z Database System Concepts: 7.8.6.1 Representation of Generalization	58
5.12. [E2017] Rodzaje atrybutów w ER (5 atrybutów) + mapowanie tych atrybutów na model relacyjny	60
5.12.1. Rodzaje atrybutów	60
5.12.1.1. Atrybuty proste	60
5.12.1.2. Atrybuty złożone	60
5.12.1.3. Atrybuty jednowartościowe	61
5.12.1.4. Atrybuty wielowartościowe	61
5.12.1.5. Atrybuty pochodne	61
5.12.2. Mapowanie atrybutów opisane w następnym podpunkcie	61
5.13. [E] Transformacja atrybutów związków encji na model relacyjny	61

5.13.1. Atrybuty proste	61
5.13.2. Atrybuty złożone	62
5.13.3. Atrybuty wielowartościowe	62
5.13.4. Atrybuty pochodne	62
5.14. [E2017] Zamiana związków ternarnych na binarne	63
5.14.1. Zagadnienia związane z projektowaniem Binarne vs nie-binarne związki	63
5.14.2. Przykład	63
6. Wykład - Dodatkowe konstrukcje SQL	65
6.1. [E] Definicja, rodzaje i zastosowanie widoków	65
6.1.1. Widok	65
6.1.2. Zalety widoków	65
6.1.3. Rodzaje widoków:	65
6.1.4. Wady i zalety widoków zmaterializowanych:	65
6.2. [E] Jak odświeżamy dane w widoku zmaterializowanym, typy refresh'y	66
6.3. [E] Wstawianie do widoków/modyfikacja widoków (czego nie można używać?, "with check option") + [E2017] Dlaczego nie powinno się modyfikować widoków	66
6.3.1. Wstawianie/modyfikowanie widoków	66
6.3.2. [E2017] Dlaczego nie należy wstawać do widoków	67
6.4. [E2017] Jakie mogą wystąpić problemy przy usuwaniu widoku?	67
6.5. [E] Co to jest indeks?	67
6.5.1. Indeks	67
6.5.2. Tworzenie indeksu (wykład 6):	67
6.5.3. [Dodatkowo] Więcej informacji	68
6.6. [E] Czym są warunki integralnościowe?	68
6.7. [E2017] Blokowanie constraintów (Które można blokować) i jak się odblokowuje, po co się to robi.	69
6.7.1. Po co blokować	69
6.7.2. Blokowanie na istniejących danych	69
6.7.3. Blokowanie przy ładowaniu nowych danych	69
6.8. [E] Zależności referencyjne, jak można je modyfikować, kiedy się je definiuje	70
6.8.1. Zależność referencyjna	70
6.8.2. Warunek zależności referencyjnej	70
6.8.3. Integralność referencyjna (to samo co zależność referencyjna?) wiele do	70
6.9. [E] Usuwanie kaskadowe w zależnościach referencyjnych	70
6.9.1. Z poprzedniego opracowania	70
6.9.2. Usuwanie wierszy z tabeli powiązanej	71
6.10. [E2017] Klauzula With	71
6.10.1. Klauzula WITH w podzapytaniu	71
6.10.2. Przykład:	71
6.10.3. [E] O co chodzi z klauzulą with w procedurze?	72
6.11. [E] Co to jest procedura skalarna?	72
6.11.1. [Dodatkowo] Podzapytanie skalarne	72

6.12. [E] Co to jest procedura składowana? Zalety? Wady?	73
6.13. [E] Duże pliki danych (blob/clob)	73
6.13.1. Blob	73
6.13.2. Clob	74
6.14. [E2017] Co zwraca zapytanie o clob/blob?	74
6.15. [E] Różnice między własnymi typami, a zawęźaniem dziedziny	74
6.15.1. [E] Różnice pomiędzy typami, a dziedzinami	74
6.15.2. [Dodatkowo] Typy własne - typy zdefiniowane przez użytkownika - User-Defined Types	
75	
6.15.2.1. Structured data types	75
6.15.2.2. Distinct types	75
6.15.3. [Dodatkowo] Dziedziny, domains	75
6.15.4. [Dodatkowo] Wsparcie dla typów i dziedzin	76
6.15.5. Poprzednie opracowanie	76
6.16. [E] Czym jest rozszerzenie tabeli	76
6.16.1. Tworzenie rozszerzeń tabeli	77
6.16.2. Rozszerzenie tabeli vs widok	77
6.16.3. Rozszerzenie tabeli vs zmaterializowany widok	77
6.17. [E] Autoryzacje/Uprawnienia na widokach	78
6.17.1. Uprawnienia na widokach	78
6.17.2. Formy autoryzacji (uprawnienia) na częściach bazy danych :	78
6.18. [E] Po co są role i jak je definiujemy?	78
6.19. [E] Kto ma uprawnienia do modyfikacji schematu?	79
6.19.1. Autoryzacja na schematach	79
6.20. [E] Formy autoryzacji do modyfikowania schematu bazy	79
6.21. [E] REFERENCES permission, ALL permission	79
6.21.1. Opracowanie z zeszłego roku	79
6.21.2. Autoryzacja na schematach c.d.	79
7. Wykład - Procedury,	79
7.1. [E] 2 sposoby dostępu do bazy danych pod względem języków programowania	81
7.2. [E] Różnica między SQL dynamicznym i osadzonym	81
7.2.1. Osadzony SQL:	81
7.2.2. Dynamiczny SQL:	83
7.2.2.1. JDBC (Java Database Connectivity)	83
7.2.2.2. ODBC (Open Database Connectivity)	84
7.3. [E] Funkcje tablicowe	84
7.4. [E2017] Co to jest PSM - Persistent storage module?	85
7.4.1. Konstrukcje proceduralne	85
7.5. [E] Konstrukcja obsługi wyjątku	85
7.5.1. Przykład z wykładu:	85
7.6. [E] Case - co robi?	85
7.7. [E2017] Funkcje zewnętrzne - wady zalety, jak radzimy sobie z wadami?	86

7.7.1. Zewnętrzne funkcje/procedury	86
7.7.2. Korzyści i wady z zewnętrznych funkcji/procedur	86
7.7.3. Bezpieczeństwo przy zewnętrznych procedurach/funkcjach	86
7.8. [E] Sandboxing	86
7.9. [E] Czym jest trigger (definicja), jak się go definiuje?	87
7.9.1. [E] Trigger - definicja	87
7.9.2. [E] Po co triggery	87
7.9.3. [Dodatkowo] Cechy triggerów	87
7.9.4. [Dodatkowo] Zdarzenia i akcje triggera	88
7.10. [E2017] Triggery na poziomie wierszy i instrukcji	88
7.10.1. Wyzwalacze na poziomie wierszy	88
7.10.2. Wyzwalacze na poziomie polecenia	88
7.11. [E] Kiedy nie używać triggerów?	89
7.11.1. Kiedy nie używać triggerów	89
7.11.2. Problemy z triggerami (z wykładu)	90
8. Wykład - Projektowanie relacyjnych baz danych - Postaci Normalne	91
8.1. [E] Jak projektujemy relacyjną bazę danych?	91
8.2. [E] Anomalie i rodzaje anomalii (omówić przykład z wykładu)	91
8.2.1. Przykład 1	91
8.2.2. Przykład 2	92
8.3. [Dodatkowo] Wstęp do formalnego modelu relacyjnych baz danych	93
8.3.1. Wstęp	93
8.3.2. Schemat relacji	93
8.3.3. Dziedzina atrybutu	93
8.3.4. Dziedzina relacji	94
8.3.5. Definicja relacji	94
8.3.6. Operacja ograniczenia krotki	94
8.4. [E] Zależności funkcyjne	95
8.4.1. Zależność funkcyjna:	95
8.4.1.1. Z wykładu: Zależności funkcyjne – twierdzenie uogólniające ideę kluczy relacji.	95
8.4.1.2. Dalszy ciąg ze strony: http://edu.pjwstk.edu.pl/wyklad/rbd/scb/wyklad5/norm.htm	95
8.4.2. Identyfikacja zależności funkcyjnych	96
8.4.3. Opracowanie z zeszłego roku	96
8.5. [E2017] Definicje kluczy w kontekście zależności funkcyjnych	96
8.5.1. Superklucz (nadklucz)	97
8.5.2. Klucz kandydujący + [E] Czym jest klucz kandydujący w kontekście zależności funkcyjnych?	97
8.5.3. [Dodatkowo] Klucze kandydujące i klucze główne	98
8.6. [Dodatkowo] Wykorzystanie zależności funkcyjnych	98
8.6.1. Z wykładu:	98
8.6.2. [Dodatkowo] Znaczenie zależności funkcyjnych	98

8.7. [E] Legalność relacji	99
8.7.1. Co to znaczy, że relacja jest legalna	99
8.8. [E] Domknięcie zbioru zależności (F+)	100
8.8.1. Domknięcie zbioru zależności funkcyjnych	100
8.8.2. [E] Jak wyznaczyć domknięcie zbioru zależności F+?	100
8.8.2.1. [E] Podać 3 aksjomaty Armstronga:	100
8.8.2.2. [E] Wyznaczenie domknięcia zależności funkcyjnych przez użycie domknięcia atrybutów.	101
8.9. [Dodatkowo] Pierwsza postać normalna (1NF)	101
8.10. [Dodatkowo] Druga postać normalna (2NF) - nieużywana	102
8.11. [E2017] Trzecia postać normalna (3NF)	103
Definicja z wykładu:	103
8.11.2. Przykład dekompozycji do 3NF:	104
8.11.2.1. Zależności w tabeli Table_book_detail	105
8.11.2.2. Po normalizacji 3NF:	105
8.12. [E2017] Po co stworzono 3 postać normalną (3NF)?	106
8.12.1. Wikipedia:	106
8.12.2. database_system_concepts_6th: Comparison of BCNF and 3NF	106
8.12.3. Ze slajdów:	106
8.13. [E] Postać normalna Boyce-Codda (BCNF)	107
8.13.1. Definicja z wykładu + Opracowanie z zeszłego roku	107
8.13.1.1. Definicja BCNF z wykładu	107
8.13.1.2. Co zrobić jeżeli nie spełnia BCNF?	107
8.13.2. Trochę bardziej opisowo ze strony http://edu.pjwstk.edu.pl/wyklady/rbd/scb/wyklad5/norm.htm	110
8.13.2.1. Postać normalna Boyce'a-Codda	110
8.13.2.2. Przykłady schematów w postaci normalnej Boyce'a-Codda:	110
8.13.2.3. Schemat nie dający się sprowadzić do postaci normalnej Boyce'a-Codda	111
8.14. [E2017] Różnice między BCNF a 3NF	111
8.14.1. Porównanie definicji	111
8.14.2. database_system_concepts_6th: Comparison of BCNF and 3NF:	111
8.14.3. Przykład	111
8.15. [E] Domknięcie atrybutów? 3 zastosowania? Jak go użyć w tych zastosowaniach?	112
8.15.1. Domknięcie zbioru atrybutów - definicja	112
8.15.2. Zastosowania:	112
8.16. [E] Algorytm sprawdzania czy dany zbiór atrybutów jest superkluczem - wykorzystaj zbiór domknięć atrybutów	114
8.17. [E] Domknięcie atrybutów co to jest, pseudokod	114
8.17.1. Algorytm do obliczania α^+ , domknięcia zbioru α nad F:	114
8.18. [E] Co to są nadmiarowe atrybuty i jak się je znajduje?	114
8.18.1. Nadmiarowe atrybuty	114
8.18.2. Efektywne testowanie czy atrybut jest nadmiarowy:	114

8.19. [E] Kanoniczne pokrycie zbioru zależności funkcyjnych	115
8.20. [E2017] Co to jest dekompozycja, cechy dobrej dekompozycji	116
8.20.1. Co to jest dekompozycja	116
8.20.2. Cechy dobrej dekompozycji	116
8.21. [E] Dekompozycja bezstratna + warunki	116
8.22. [E] Kiedy dekompozycja zachowuje zależność?	117
8.23. [E2017] Zależności wielowartościowe	118
8.23.1. Zależności wielowartościowe (czwarta postać normalna) na przykładzie Z wykładu - zależności wielowartościowe	118
8.23.3. Wykorzystanie wielowartościowych zależności	119
8.24. [E] Czwarta postać normalna (4NF)	120
8.24.1. [Dodatkowo] Restrykcje wielowartościowych zależności	120
8.24.2. [Dodatkowo] Algorytm dekompozycji 4NF	120
8.25. [E2017] Modelowanie danych w czasie	121
8.26. [E] Czasowe zależności klucza obcego od głównego	122
9. Wykład - Organizacja plików	123
9.1. [E2017] PCTFREE and PCTUSED	123
9.2. [E2017] Freelist i w jakiej strukturze	123
9.3. Rekordy o stałej długości	124
9.3.1. Podejście:	124
9.4. [E2017] Rekordy o zmiennej długości	124
9.4.1. Reprezentacja rekordów o zmiennej długości	124
9.5. [E] W rekordzie są 3 pola o zmiennym rozmiarze atrybutów i jeden o stałym, jak to jest fizycznie zorganizowane	125
9.5.1. Opracowanie z zeszłego roku:	126
9.6. [E] Przechowywanie rekordów o zmiennej długości - struktura slotowa, slotted page	126
9.7. [E2017] Dlaczego istnieją rekordy zmiennej długości?	127
9.8. [Dodatkowo] Kiedy używać rekordów o stałej długości	127
9.9. [E2017] Rodzaje organizacji rekordów w plikach	127
9.9.1. Nieuporządkowane	127
9.9.2. Uporządkowane (sekwencyjne) - organizacja pliku sekwencyjnego	127
9.9.2.1. [E2017] - Co to jest overflow block, czy to jest korzystne i co trzeba robić?	128
9.9.2.2. Reorganizacja pliku sekwencyjnego	128
9.9.3. Haszowe	128
9.9.4. Klastrowanie wielotablicowe. Multitable clustering file organization	128
9.10. [E] Słownik danych co to jest, co przechowuje, jak jest fizycznie zorganizowany	130
9.11. [E] Buffer manager	130
9.11.1. [E] Zarządzaniem buforem/Zarządca bufora (buffer manager)	130
9.12. [E] Buforowanie bloków danych	130
9.13. [Dodatkowo] Techniki wykorzystywane przez zarządcę bufora	131
9.13.1. LRU strategy	131
9.13.2. Przypięty blok (pinned block)	131

9.13.3. Wymuszony zapis (forced output)	131
9.13.4. Strategia natychmiastowego podrzucania (toss-immediate)	131
9.13.5. Strategia ostatniego używanego (most recently used MRU)	131
10. Wykład - Przetwarzanie transakcyjne	132
10.1. [E2017] Transakcje - co to jest (dodatkowo - ACID)	132
10.2. [E] Diagram stanów transakcji	133
10.3. [E] Wielodostęp, problemy wielodostępu	133
10.3.1. Utrata zmian	133
10.3.2. Niezatwierdzone zależności ("brudne dane")	134
10.3.3. Analiza niespójności	134
10.3.3.1. Analiza niespójności- Mylące odczyty	134
10.3.3.2. Analiza niespójności - Odczyty fantomowe	135
10.4. [E] Co to jest fantom?	135
10.5. [E] Szeregowalność	135
10.6. [E] Jak mają być ułożone transakcje w harmonogramie?	135
10.7. [E] Harmonogram sekwencyjny i niesekwencyjny	135
10.6.1. Harmonogram sekwencyjny	135
10.6.2. Harmonogram niesekwencyjny	135
10.6.3. [Dodatkowo] Harmonogram szeregowalny	136
10.7. [E] Harmonogram odtwarzalny	136
10.8. [E2017] Co to jest graf pierwszeństwa?	136
10.8.1. Z wykładu:	137
10.8.2. Bardziej szczegółowo ze strony: http://math.uni.lodz.pl/~cybula/bd/pbd/wyklad_11_12_podstawy_baz_danych_2009_2010.pdf	137
10.9. [E] Jak szeregować harmonogram	137
10.10. [E] Prawo pierwszeństwa w harmonogramach	139
10.11. [E] Harmonogramy równoważne	140
10.12. [E] Techniki kontroli wielodostępu, blokady - podejście pesymistyczne i optymistyczne.	140
10.12.1. Metody pesymistyczne (zachowawcze)	140
10.12.1.1. Blokady	140
10.12.1.2. Znaczniki czasowe	140
10.12.2. Metody optymistyczne	141
10.13. [E] Rodzaje blokad	141
10.14. [E] Co to jest rozszerzenie/redukcja blokady?	142
10.15. [E] Kiedy transakcja spełnia warunki blokowania dwufazowego?	142

Budzę się o szóstej rano, bo ktoś napierała dzwonkiem do domu. Wstaje z łóżka nie wiem co się dzieje. Nie doszedłem jeszcze do drzwi, a już widzę kto jest tak popierdolony – Zygmunt. Nie wiem co robić, za dwie godziny egzamin w D17, planowałem jeszcze przy śniadaniu dowiedzieć się trochę o postaciach normalnych, ale co tam – mogłem już o tym zapomnieć. Nie zdążyłem nawet powiedzieć dzień dobry a Zygmunt już wleciała do mojego salonu. Dogoniłem ją, a ta zapytała mnie czy wiem, jak przechowuje się atrybuty o zmiennej długości? Odpowiedziałem, że no na dysku. Zadała jeszcze parę innych pytań, na każde odpowiedziałem źle, śmiała się bardzo. Popłakałem się. Ojciec słysząc mój płacz wstał z łóżka, widząc Zygmunt głośno śmiejącą się z mojej niewiedzy mocno się oburzył. Zygmunt popatrzyła mu się prosto w oczy i powiedziała – „on chuja wie proszę pana”, po czym wyleciała przez okno. „Zygmunt to kurwa, Zygmunt to kurwa jebana”...

1. Wykład - zasady oceniania

2. Wykład - Wprowadzenie w problematykę baz danych

2.1. [E2017] System Zarządzania Bazami Danych SZBD (DBMS)

System Zarządzania Bazami Danych to kolekcja powiązanych ze sobą danych i zbiór programów, które umożliwiają dostęp do tych danych. Kolekcja danych, przeważnie nazywana bazą danych, zawiera informacje związane z danym przedsiębiorstwem. Głównym celem SZBD jest przechowywanie danych w sposób wygodny i efektywny.

SZBD dostarcza **efektywne, niezawodne, wygodne i bezpieczne** środowisko do przechowywania i dostępu **wielu użytkownikom do ogromnych ilości trwałych** danych.

Aplikacje bazodanowe:

- Bankowość
- Linie lotnicze
- Biuro dziekanatu
- Sprzedaż internetowa
- Produkcja przemysłowa
- Kadry

Bazy danych mogą być bardzo duże – BIG DATA

2.2. [E] Po co tworzymy bazy danych?

Bazy danych tworzymy po to, aby w sposób efektywny przechowywać duże ilości danych (wyeliminować niedogodności w używaniu plików do gromadzenia danych - [punkt następny](#)).

Przechowywanie danych w bazie danych pozwala nam na:

- proste kierowanie zapytań i modyfikowanie danych
- przechowywanie danych przez długi czas
- zapewnienie trwałości danych
- sterowanie dostępem do danych przez wielu użytkowników jednocześnie

2.3. [E] Niedogodności plików do gromadzenia danych

Slajd 4: Wady użycia systemu plików do przechowywania danych:

Database System Concepts strona 3: Purpose of Database Systems

2.3.1. Wymagana spora ilość programów do obsługi plików

Aby użytkownicy mogli operować na danych, system musiał posiadać wiele programów do manipulowania tymi plikami

2.3.2. Nadmiarowość i niespójność danych

Nadmiarowość w stosunku do tego, co konieczne lub zwykłe np. student na 2 kierunkach ma swoje dane zduplikowane w 2 plikach.

Niespójność - student zmienia adres, adres zmienia się tylko w systemie (w jednym pliku), w innych miejscach dane są niepoprawne - niespójne.

2.3.3. Trudności w efektywnym dostępie do danych

Problemy np. z wyszukiwaniem studentów po rejonie zamieszkania.

2.3.4. Izolacja danych

Ciężko jest odnaleźć odpowiednie dane, które są rozrzucone po różnych plikach w różnych formatach.

2.3.5. Problemy z integralnością

Integralność danych - funkcja bezpieczeństwa polegająca na tym, że dane nie zostały zmienione, dodane lub usunięte w nieautoryzowany sposób.

W bazie danych jesteśmy w stanie zapewnić integralność na poziomie samej bazy. Przy korzystaniu z plików każda z aplikacji mająca do nich dostęp musi zadbać o integralność na własnym poziomie.

Odpowiedzialność związana z integralnością danych spada na twórców aplikacji.

2.3.6. Problemy z atomowością

Operacja atomowa - operacja, która przelewa \$500 z konta A na konto B musi być operacją atomową - nie może być tak, że w trakcie przelewu nastąpi awaria systemu i \$500 zostanie usunięte z konta A a nie zostanie dodane na konto B. Cieżko jest zapewnić atomowość w systemach opartych na plikach.

2.3.7. Równoległy dostęp przez wielu użytkowników

Np. Przy zapisach na dany przedmiot może pojawić się **hazard**:

Student A i student B zapisują się z powodzeniem na ten kurs w tym samym czasie. Dwa programy pobrały aktualną ilość uczestników np. 39, studenci zapisują się na zajęcia, programy jeden po drugim nadpisują ilość uczestników na 40. Powoduje niepoprawne zwiększenie ilości o 1, mimo, że dwóch studentów z powodzeniem zapisała się na kurs i ilość uczestników to 41.

żeby temu zapobiec system musi mieć swego rodzaju nadzór nad wykonywaniem operacji równoległych. Jest to problematyczne m. in. dlatego, że dostęp do danych mogą mieć różne aplikacje, które nie zostały wcześniej skoordynowane.

2.3.8. Problemy z bezpieczeństwem (logowanie, prawa dostępu)

Różni użytkownicy powinni mieć dostęp do różnych danych. W systemie opartym o pliki jest to trudne do uzyskania, bo aplikacje uzyskują dostęp do wszystkich plików.

2.3.9. Trudności z optymalizacją

2.4. [E] Warstwy logiczne i fizyczne bazy danych

Database System Concepts strona 6: Data Abstraction

2.4.1. Poziom fizyczny:

- operacje na plikach rekordów
- opisuje jak rekord jest przechowywany fizycznie (np. bloki na dysku)

Physical level. The lowest level of abstraction describes how the data are actually stored. The physical level describes complex low-level data structures in detail.

2.4.2. Poziom logiczny

- operacje na tabelach stored
- opisuje jakie dane są przechowywane i zależności między nimi

Logical level. The next-higher level of abstraction describes what data are stored in the database, and what relationships exist among those data. The logical level thus describes the entire database in terms of a small number of relatively simple structures. Although implementation of the simple structures at the logical level may involve complex physical-level structures, the user of the logical level does not need to

be aware of this complexity. This is referred to as **physical data independence**. Database administrators, who must decide what information to keep in the database, use the logical level of abstraction.

2.4.3. [Dodatkowo] Poziom widoku:

- najwyższy poziom abstrakcji
- ułatwia przeciętnym użytkownikom interakcję z systemem
- ukrywa typy danych
- pozwala ukrywać dane z powodów bezpieczeństwa (np. poufne dane jak płace pracowników)

View level. The highest level of abstraction describes only part of the entire database. Even though the logical level uses simpler structures, complexity remains because of the variety of information stored in a large database. Many users of the database system do not need all this information; instead, they need to access only a part of the database. The view level of abstraction exists to simplify their interaction with the system. The system may provide many views for the same database.

2.4.4. [Dodatkowo] Zależność pomiędzy tymi poziomami abstrakcji

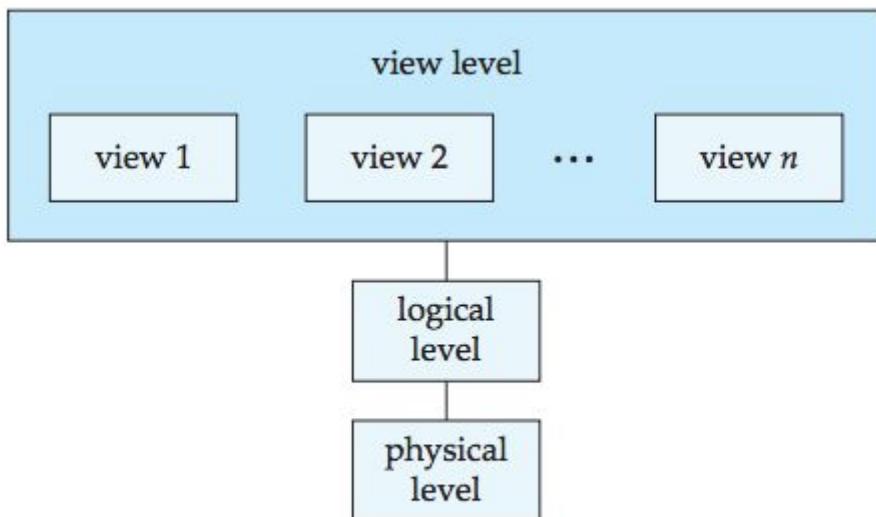


Figure 1.1 The three levels of data abstraction.

At the **physical level**, an instructor, department, or student record can be described as a block of consecutive **storage locations**. The compiler hides this level of detail from programmers. Similarly, the database system hides many of the lowest-level storage details from database programmers. Database administrators, on the other hand, may be aware of certain details of the physical organization of the data.

At the **logical level**, each such record is described by a type definition, as in the previous code segment, and the interrelationship of these record types is defined as well. **Programmers using a programming language work at this level of abstraction.** Similarly, database administrators usually work at this level of abstraction.

Finally, at the **view level**, computer **users see a set of application programs** that hide details of the data types. At the view level, several views of the database are defined, and a database user sees some or all of these views. In addition to hiding details of the logical level of the database, the views also provide a security mechanism to prevent users from accessing certain parts of the database. For example, clerks in the university registrar office can see only that part of the database that has information about students; they cannot access information about salaries of instructors.

2.5. [E] Co to instancja? Co to jest schemat?

Wykład 2 slajd 9:

Database System Concepts strona 8: Instances and Schemas

2.5.1. Instancja

Instancja - stan danych (konkretnie wartości w danej relacji) aktualna zawartość bazy danych w określonym punkcie w czasie.codec

2.5.2. Schemat

Schemat - logiczna struktura bazy danych (zmieniana bardzo rzadko lub w ogóle)

- **Schemat logiczny**: logiczna struktura bazy danych
 - schemat na którym pracują programiści
 - schemat na którym bazują aplikacje
- **Schemat fizyczny**: projekt bazy danych na poziomie fizycznym
- (opcjonalnie) schemat na poziomie widoku
 - ułatwia przeciętnym użytkownikom interakcję z systemem

2.5.3. [Dodatkowo] Instancje i schematy

The concept of database schemas and instances can be understood by analogy to a program written in a programming language. A **database schema corresponds to the variable declarations** (along with associated type definitions) in a program. Each variable has a particular value at a given instant. **The values of the variables in a program at a point in time correspond to an instance** of a database schema.

Database systems have several schemas, partitioned according to the levels of abstraction. The **physical schema** describes the database design at the physical level, while the **logical schema** describes the database design at the logical level. A database may also have several schemas at the view level, sometimes called subschemas, that describe different views of the database.

Of these, the **logical schema is by far the most important**, in terms of its effect on application programs, since programmers construct applications by using the logical schema.

2.6. [E] Czym jest fizyczna izolacja danych?

Wykład 2 slajd 9.

Database System Concepts strona 8: Instances and Schemas

2.6.1. Fizyczna niezależność danych

Fizyczna niezależność danych - zdolność do modyfikowania schematu fizycznego, bez modyfikacji schematu logicznego.

Na slajdzie nazywało się to "fizyczną niezależnością danych" chodzi o to, że dane z bazy są na dysku oddzielone od programu, nie są fizycznie jego częścią. Umożliwia to łatwą zmianę w warstwie fizycznej bez ingerencji w warstwę logiczną.

The physical schema is hidden beneath the logical schema, and can usually be changed easily without affecting application programs. Application programs are said to exhibit **physical data independence** if they do not depend on the physical schema, and thus need not be rewritten if the physical schema changes.

2.7. [E] Co to jest model danych, podaj jego elementy i przykłady modeli. Co to jest model konceptualny? Czym różni się od implementacyjnego?

Wykład 2 slajd 10.

Database System Concepts strona 8: Data Models

2.7.1. Model danych

Pod spodem struktury bazy danych jest **model danych** czyli zbiór narzędzi konceptualnych do opisu:

- danych
- operacji na danych (np. selekcja, projekcja)
- struktury danych
- zależności między danymi
- semantyki danych (znaczenia danych)
- ograniczeń dla danych, ograniczeń integralnościowych

Model danych umożliwia opisanie projektu bazy danych na poziomie fizycznym, logicznym i widoku.

2.7.2. Przykłady modeli danych

Różnorodność modeli:

- **Relacyjny (Ted Codd, 1970)** - używa zbioru tabeli do reprezentacji danych i zależności pomiędzy nimi. Każda tabela ma kolumny, każda z kolumn ma niepowtarzalną nazwę. Model relacyjny to przykład modelu opartego na rekordach - record-based model. W takim modelu baza danych ma strukturę rekordów różnych typów o niezmiennym formacie.
bn
Każda z tabeli zawiera rekord z konkretnym typem. Każdy rekord posiada niezmienną ilość pól lub atrybutów. Znaczna większość dzisiejszych baz danych jest oparta na relacyjnym modelu danych
- **Związków encji (Entity-Relationship)** - używa zbioru podstawowych obiektów nazywanych **encjami** i **związków** pomiędzy tymi obiektami. Encja to "rzecz" lub "obiekt" w realnym świecie, który jest rozróżnialny na tle innych obiektów. Model związków encji jest używany przy projektowaniu baz danych.
- **Obiektowy (również obiektowo-relacyjny)** - podobny do związków encji, dodatkowo związany z następującymi pojęciami: enkapsulacja, metody i tożsamość obiektu. Motywacją do stworzenia tego modelu była popularność języków obiektowych.
- **Semistrukturalny (XML)** - dane tego samego typu mogą mieć różne zbiory atrybutów (w przeciwieństwie do poprzednich modeli, w których każdy element danego typu musiał mieć taki sam zestaw atrybutów). Wykład 2 slajd 23: XML stał się podstawą dla wszystkich nowej generacji formatów wymiany danych (zdolność do specyfikowania nowych tagów-takie jak <p> czy <h2>, tworzenia zagnieżdzonych struktur tagów), a nie tylko dokumentów.

- Inne starsze modele:
 - Hierarchiczny
 - Sieciowy (raport CODASYL, 1971)

2.7.3. Model konceptualny i implementacyjny

Model konceptualny:

Reprezentacja obiektów w uniwersalnym modelu niezależnym od modelu implementacyjnego in(związków – encji, model UML). Model konceptualny jest podstawą **schematu bazy danych**.

Model implementacyjny:

Wykorzystywany do implementacji modeli konceptualnych. Odzwierciedla on model konceptualny w konkretne struktury danych. Wśród modeli danych najpopularniejszymi obecnie są:

- relacyjny,
- obiektowy,
- obiektowo-relacyjny
- semistrukturalny.

2.8. [E] Języki baz danych/Rodzaje języków

Wykład 2 slajdy 13, 14

Database System Concepts strona 36: Database Languages

System bazodanowy dostarcza języka definicji danych **DDL** i języka manipulacji danymi **DML**. W praktyce nie są to 2 osobne języki - są to części jednego języka bazy danych np. powszechnie używanego języka SQL.

2.8.1. Język definicji danych Data Definition Language (DDL)

Pozwala na specyfikację schematu bazy danych - zbioru relacji i informacji o każdej z nich

- definicja schematu dla każdej relacji
- ustalenie dziedziny wartości dla każdego atrybutu
- warunki integralnościowe (integralność referencyjna)
- inne informacje takie jak:
 - zestaw indeksów dla każdej relacji ([\[E\] Co to jest indeks?](#))
 - warunki dostępu / autoryzacji (bezpieczeństwo)
 - fizyczne struktury (dyskowe)

Dzięki DDL (Data Definition Language) 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).
- **ENABLE TRIGGER**

Przykład: **create table** instructor (

```
ID char(5),
name varchar(20),
dept_name varchar(20),
salary numeric(8,2))
```

Interpreter DDL generuje jako wynik zestaw szablonów tabel przechowywanych w **data dictionary** (słowniku danych)

[\[E\] Słownik danych](#) zawiera metadane (tzn. dane o danych)

2.8.2. Język manipulacji danymi Data Manipulation Language (DML)

Język dla dostępu i manipulowania danymi zorganizowanymi zgodnie z odpowiednim modelem danych. Typy dostępu:

- Odczytanie informacji z bazy danych
- Wstawianie informacji do bazy danych
- Usunięcie informacji z bazy danych
- Modyfikacja informacji z bazy danych

Istnieją 2 typy języków do manipulacji danymi (DMLs):

- **Proceduralne** DMLs - użytkownik musi sprecyzować *jakie* dane są potrzebne i w *jaki sposób* je uzyskać
- **Deklaratywne** DMLs (nieproceduralne) - użytkownik musi sprecyzować *jakie* dane są potrzebne *bez podawania sposobu* w jaki je uzyskać

Inny podział (ze slajdów):

- Pewna grupa „czystych” języków
 - algebra relacji
 - tuple relational calculus (wnioskowanie na krotkach, logika predykatów), domain relational calculus (wnioskowanie dziedzinowe)
- Komercyjne

2.9. [E] Funkcjonalne komponenty systemu BD, za co odpowiadają

Slajd 25

Funkcjonalne komponenty systemu bazodanowego mogą być podzielone na:

- **menedżer składowania** (storage manager)
- **przetwarzanie zapytań** (query processing)
- (wg. wykładu jest to jeszcze **menedżer transakcji**, wg dbs concepts menedżer transakcji to komponent menedżera składowania)

2.9.1. Menedżer składowania

Moduł programowy, który dostarcza interfejs między danymi niskiego poziomu przechowywanymi w bazie danych a programami aplikacji i pytaniami wysyłanymi do systemu.

Menedżer ten jest odpowiedzialny za:

- Interakcje z menedżerem plików
- Efektywne składowanie, pobieranie i uaktualnianie danych w bazie danych

Składa się z :

- **Menedżera autoryzacji i integralności** (warunki integralnościowe, dostęp dla odpowiednich użytkowników)
- **Menedżera plików** (jak rozłożone są dane na dysku)
- **Menedżera buforów** (przenoszenie danych z dysku do pamięci głównej)
- **Menedżera transakcji** (zapewnia, że baza danych pozostanie spójna pomimo awarii i transakcji współbieżnych)

Uwaga: Według slajdów 16/17 jest to komponent SZBD, ale według Silberschatza (s .21) to część menedżera składowania

- moduł kontroli współbieżności
- recovery manager

2.9.2. [E2017] Procesor przetwarzania zapytań zawiera komponent vcfdy:

- **DDL interpreter** (Data Definition Language) - interpretuje polecenia DDL i zapisuje je w słowniku danych
- **DML kompilator** (Data Manipulation Language) - tłumaczy polecenia DML (w języku zapytań) do poleceń niskiego poziomu z wykorzystaniem optymalizatora zapytań (query optimizer)
- **Silnik wykonywania zapytań** - wykonuje polecenia niskiego poziomu dostarczone przez DML kompilator

Przetwarzanie zapytań

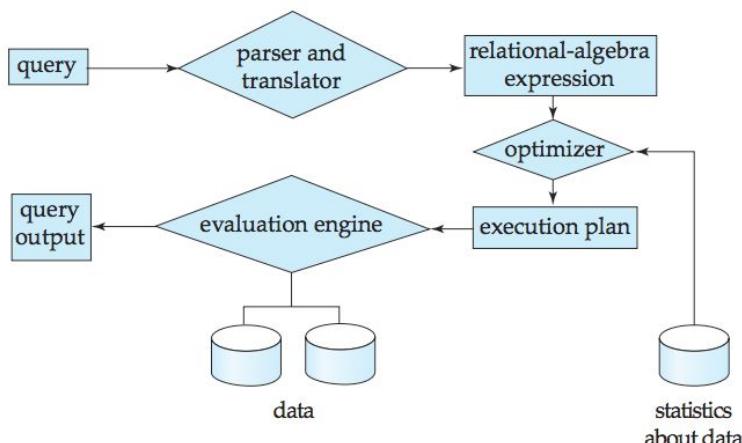


Figure 12.1 Steps in query processing.

Wewnętrzna struktura SZBD (Systemu zarządzania bazą danych)

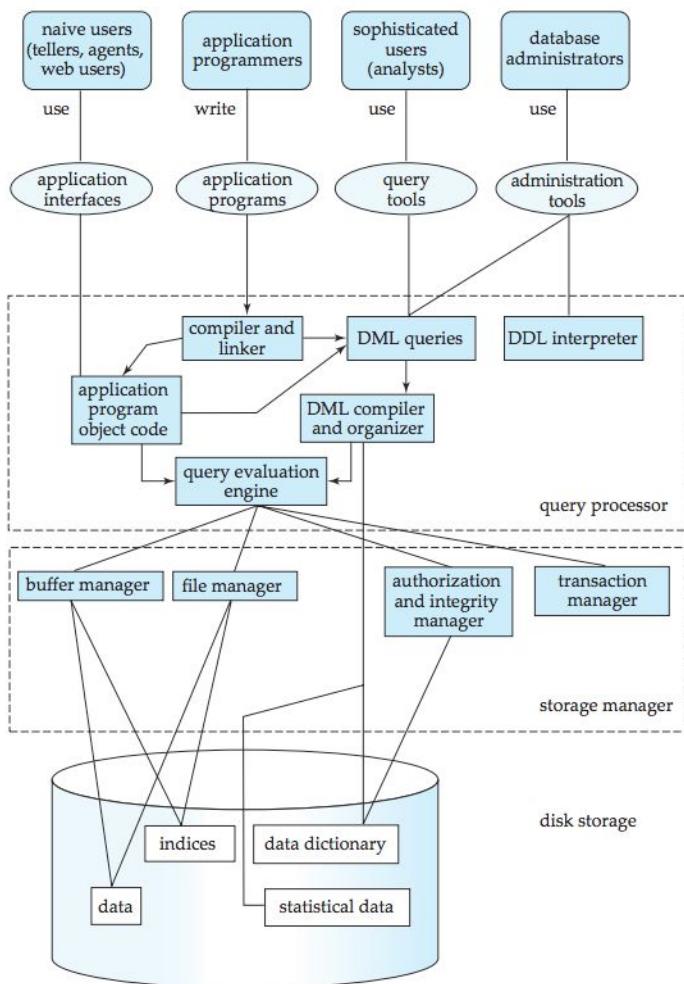


Figure 1.5 System structure.

2.10. [E2017] Architektura baz danych

2.10.1. Architektura systemów baz danych

Architektura systemów baz danych jest ściśle powiązana z systemem komputerowym, na którym działa:

- **Scentralizowana** - wszystkie komponenty systemu zainstalowane są na pojedynczej maszynie fizycznej
- **Klient - serwer** - podział ról: serwer wykonuje usługi dla klientów, klienci zgłaszają żądania do serwera
- **Równoległa (wieloprocesorowa architektura)** - charakteryzuje się obecnością wielu procesorów i dysków połączonych szybką siecią iii
- **Rozproszona (geograficznie)** - jest zbiorem współpracujących ze sobą baz danych, z których każda znajduje się na innym serwerze. Bazy te dalej będziemy nazywać lokalnymi. Z punktu widzenia użytkownika wszystkie te bazy logicznie stanowią jedną rozproszoną bazę danych.

2.10.2. Architektura aplikacji

Aplikacje bazodanowe są zazwyczaj podzielone na 2 lub 3 części.

- **dwuwarstwowa** – front end bezpośrednio komunikuje się z bazą danych po stronie back endu. Aplikacja znajdująca się po stronie klienta korzysta z funkcjonalności bazy danych za pośrednictwem języków zapytań (użycie standardów takich jak ODBC, JDBC)
- **trójwarstwowa** – back end jest rozbity na bazę danych i aplikację serwera. Maszyna klienta nie wykonuje żadnych bezpośrednich zapytań do bazy danych, komunikuje się tylko z aplikacją serwera. Aplikacja serwera komunikuje się z bazą danych w celu dostępu do danych.

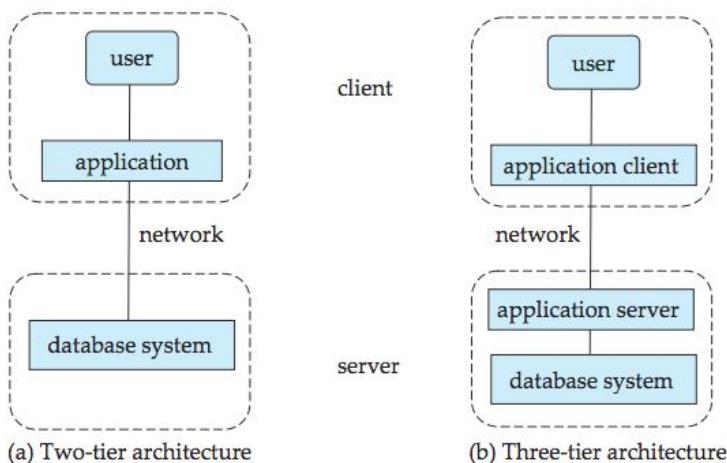


Figure 1.6 Two-tier and three-tier architectures.

3. Wykład - Wstęp do modelu relacyjnego

3.1. [Dodatkowo] Relacyjny model danych

3.1.1. Cechy modelu relacyjnego

- struktury danych: relacje (tabele)
- operacje na danych:
 - **selekcia** - rodzina operatorów parametryzowanych warunkiem logicznym. Jako argument przyjmuje relację (tabelę), na wyjściu zwraca relację zawierającą tylko te krotki (wiersze), dla których warunek logiczny był prawdziwy
 - **projekcja** - przeznaczenie - wyodrębnienie wybranych atrybutów relacji
 - **połączenie (JOIN)** - iloczyn kartezjański
 - **operacje na zbiorach**
 - suma (UNION)
 - część wspólna (INTERSECTION)
 - różnica (MINUS, DIFFERENCE)
- ograniczenia integralnościowe (reguły poprawności danych):
 - klucz podstawowy (primary key),
 - klucz obcy (foreign key),

- zawężenie dziedziny (check),
- unikalność (unique),
- wartość pusta/niepusta (NULL/NOT NULL)

Mimo ograniczeń w dalszym ciągu preferowany, bo dostarcza:

- prostych, ograniczonych sposobów tworzenia struktur danych, a jednocześnie jest wystarczająco uniwersalny, aby można było wszystko zamodelować
- ograniczonej, ale użytecznej kolekcji operacji na danych

Ograniczenia te pozwalają implementować języki, np. SQL, które dają programistom możliwość wyrażania zamiarów na bardzo wysokim poziomie

Przykład relacji *instructor*

<i>ID</i>	<i>name</i>	<i>dept_name</i>	<i>salary</i>
10101	Srinivasan	Comp. Sci.	65000
12121	Wu	Finance	90000
15151	Mozart	Music	40000
22222	Einstein	Physics	95000
32343	El Said	History	60000
33456	Gold	Physics	87000
45565	Katz	Comp. Sci.	75000
58583	Califieri	History	62000
76543	Singh	Finance	80000
76766	Crick	Biology	72000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000

Relacja = tabela
Krotka = wiersz
atrybut = kolumna

3.1.2. Cechy atrybutów

- Każdy atrybut relacji ma unikalną nazwę
- Zbiór możliwych wartości dla każdego atrybutu to domena (dziedzina)
- Pożądaną cechą atrybutów jest atomowość (niepodzielność)
- Specjalna wartość NULL

3.1.3. Schemat relacji i instancja



Schemat relacji i instancja

- A_1, A_2, \dots, A_n są *atrybutami*
- $R = (A_1, A_2, \dots, A_n)$ jest *schematem relacji*

Np. :

instructor = (*ID*, *name*, *dept_name*, *salary*)

- Formalnie, mając dane zbiory D_1, D_2, \dots, D_n **relacja r** jest podzbiorem:

$$D_1 \times D_2 \times \dots \times D_n$$

Czyli relacja jest zbiorem n -krotek (a_1, a_2, \dots, a_n) gdzie każde $a_i \in D_i$

- Bieżące wartości (**instancja relacji**) relacji są określone przez tabelę

3.1.4. Baza danych

Schemat bazy danych = projekt logiczny bazy (zbiór schematów relacji)

Instancja bazy danych = snapshot bazy w konkretnym momencie czasu

Informacja o organizacji jest podzielona na części, instructor student advisor

Zły projekt: univ (instructor -ID, name, dept_name, salary, student_Id, ..)

Dane rozdziela się na kilka relacji

e3.2. [E] Opowiedzieć o kluczach (w kontekście modelu relacyjnego)

3.2.1. [Dodatkowo] Wstęp

Zdarza się, że baza danych nie przechowuje pojedynczej tabeli, lecz wiele tabel, pomiędzy którymi występują powiązania. Aby pomiędzy tabelami można było tworzyć związki, powinny one mieć przynajmniej jeden atrybut (kolumnę), który musi spełniać dwa podstawowe warunki. Musi być unikatowy (oznacza to, że jego wartości nie mogą się powtarzać). Drugim warunkiem jest jego atomowość (musi być minimalny, tzn. powinien zawierać elementarne, niepodzielne wartości). Możemy rozróżnić następujące rodzaje kluczy:

- **klucz prosty** – np. *primary key* (*ID*) to taki, który jest jednoelementowy (składa się z jednej kolumny),
- **klucz złożony / połączeniowy** – np. *primary key* (*ID*, *course_id*, *sec_id*, *semester*, *year*), to taki, który jest kilkuelementowy (składa się z więcej niż jednej kolumny).

Kluczem może być zatem jedna lub kilka kolumn, które wspólnie będą w stanie jednoznacznie zidentyfikować pozostałe dane w tabeli (relacji). Kolumny, które należą do kluczy (używa się ich do jednoznacznej identyfikacji wierszy w tabeli), nazywamy **atrybutami podstawowymi**. Kolumny nienależące do kluczy (zawierają dane, które w określonej relacji są przedmiotem opisu) nazywamy **atrybutami opisowymi**.

Do łączenia dwóch tabel (np. A i B) za pomocą związków używa się klucza. Klucz pochodzący z obcej tabeli B (w której jest on kluczem głównym), używany do łączenia tej tabeli z tabelą A, będzie dla tabeli A **kluczem obcym**.

3.2.2. Superklucz

To kolumna lub zestaw kolumn jednoznacznie identyfikujących każdą 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.

Niech $K \subseteq R$. K jest **superkluczem** (superkey) R jeżeli wartości K są wystarczające aby zidentyfikować unikalność krotki w każdej możliwej relacji $r(R)$

- Superklucz może zawierać nadmiarowe atrybuty
- Interesujący jest minimalny zestaw atrybutów, który jest superkluczem => potencjalne/kandydujące klucze (candidate keys)

Definicja: Każdy podzbiór K atrybutów relacji R, taki że dla każdych dwóch krotek ze zbioru $r(R)$ zachodzi $t1[S] \neq t2[S]$ jest superkluczem (superkey) R

3.2.3. Klucz kandydujący (potencjalny)

Superklucz o najmniejszej ilości atrybutów (może być ich kilka).

Własność klucza kandydującego: usunięcie dowolnego atrybutu A z K powoduje, że $K' = K - A$ nie jest już superkluczem.

3.2.4. Klucz główny/podstawowy (primary key, PK)

Wybrany potencjalny klucz do identyfikowania krotek w obrębie danej relacji. Żadne 2 krotki w danej relacji nie mogą mieć takich samych wartości w atrybutach klucza w danym momencie

Klucz główny nie może zawierać wartości null - gwarantowane przez SZBD (System zarządzania bazą danych)

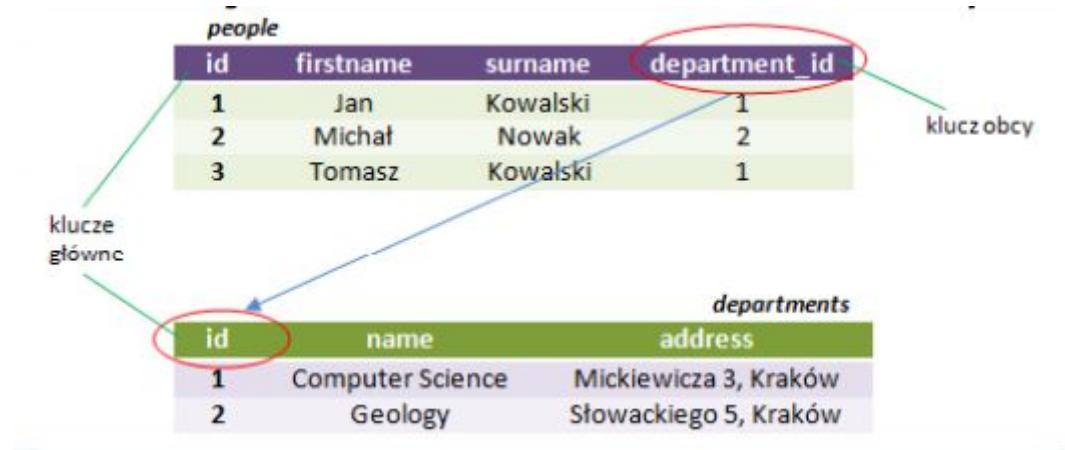
3.2.5. Klucz obcy relacji (foreign key FK)

- atrybut (lub zbiór atrybutów), który wskazuje na klucz podstawowy
- służy do reprezentowania powiązań między danymi (łączenia relacji)
- dziedziną wartości FK jest dziedzina wartości PK na który FK wskazuje

Ograniczenie klucza obcego (foreign key constraint FK):

Relacja r1 może zawierać wśród swoich atrybutów klucz główny innej relacji r2. Ten atrybut to klucz obcy z r1 odwołujący się do r2. Wówczas r1 to referencing relation (relacja odwołująca się), a r2 to referenced relation tego klucza obcego.

(za Silberschatz str. 46)



3.3. [E] Integralność referencyjna

Z wykładu:

Integralność referencyjna (referential integrity constraint) wartości pojawiające się w kluczu obcym w jakiejś krotce relacji (w jakimś wierszu tabeli) muszą pojawić się w atrybutach jakiejś krotki w relacji (kolumnie innej tabeli), do których się ten klucz odwołuje.

Opis z Database System Concepts: There are cases where we wish to ensure that a value that appears in one relation for a given set of attributes also appears in a certain set of attributes in another relation (referential integrity). For example, the department listed for each course must be one that actually exists. More precisely, the dept name value in a course record must appear in the dept name attribute of some record of the department relation. Database modifications can cause violations of referential integrity. When a referential-integrity constraint is violated, the normal procedure is to reject the action that caused the violation.

3.4. [E] Czy w praktyce klucz obcy może być NULLem?

Sprzeczne opinie:

- Oczekiwana odpowiedź to zdecydowanie "NIE".
- Tak. (Przykładem może być kolumna "ReportsTo" w bazie "Northwind", gdzie "szef" miał wartość NULL)

Zgodnie z książką "Systemy baz danych. Kompletny podręcznik" (przykład 7.1) FK może mieć wartość NULL i oznacza to, że, do której się odnosimy nie istnieje (jak w przykładzie z "ReportsTo").

Może jej chodzi o to, że wartość PK w tabeli 2, który jest naszym FK w tabeli 1 nie może być NULlem?

Wartość na którą FK wskazuje nie może być nullem, ale sam FK nullem może być na pewno.(+208)

3.5. [E] Zawężenie dziedziny, checki

Pozwalają ograniczyć wartości danego atrybutu. Można je nazywać. Umieszczamy je w definicji tabeli.

constraint <nazwa> check <atrybut> <wyrażenie> (np. atrybut > 5)

Edycja poprzez usunięcie i dodanie nowej definicji.

alter table add/drop constraint <nazwa> check <atrybut> <wyrażenie> (np. atrybut > 5)

Zawężenie dziedziny (ograniczenie domeny) atrybutu (check) – ograniczenie dozwolonych wartości do pewnego podzbioru przez wyrażenie logiczne określające przedział lub za pomocą wyliczeniowej listy wartości.

Przykładami tego typu ograniczenia są np.

- ograniczenie dopuszczalnych wartości atrybutu płeć do: K, M, nieznana, N/A (zgodnie ze standardem ISO),
- zagwarantowanie dodatkowych wartości atrybutu pensja,
- ograniczenie dopuszczalnych wartości atrybut kolor_oczu do trzech wartości: niebieskie, szare, piwne.

4. Wykład - Algebra relacji

4.1. [E2017] Algebra relacji

Algebra relacji jest proceduralnym językiem zapytań. Składa się ze zbioru operatorów, które na wejściu biorą jedną lub dwie relacje i produkują nową relację jako wynik.

// na pewno nie więcej niż 2? coś mi się nie wydaje.

// tak, bo np. join: a join b join c = (a join b) join c, nie ma trójargumentowego joina (+2)

Sześć podstawowych operatorów:

- selekcja (select): σ
- projekcja (project): Π
- unia (union): \cup
- różnica zbiorów (set difference): $-$
- produkt kartezjański (Cartesian product): \times
- przemianowanie (rename): ρ

Dodatkowe operatory:

- przecięcie zbiorów (set intersection): \cap
- złączenie naturalne (natural join): \bowtie
- przypisanie (assignment): \leftarrow

Operatory rozszerzonej algebry relacji

- Uogólniona projekcja
- Funkcje agregujące

4.1.1. Podstawowe operatory:

4.1.1.1. Operator selekcji (select): σ

Termin *selekcja* (select) ma inne znaczenie w algebrze relacji, a inne w SQL. *Selekcja* w relacyjnej algebrze odpowiada *where* w SQL.

Operator selekcji

- Notacja: $\sigma_p(r)$
- p zwane **predykatem selekcji**
- Definicja:

$$\sigma_p(r) = \{t \mid t \in r \text{ and } p(t)\}$$

gdzie p jest formułą w rachunku zdań złożoną z **pojęć** połączonych : \wedge (**and**), \vee (**or**), \neg (**not**)

Każde **pojęcie** jest jednym z:

<attribute> op <attribute> or <constant>

gdzie op jest jednym z: $=, \neq, >, \geq, <, \leq$

Przykłady selekcji

$$\sigma_{dept_name="Physics"}(instructor)$$

ID	name	dept_name	salary
10101	Srinivasan	Comp. Sci.	65000
12121	Wu	Finance	90000
15151	Mozart	Music	40000
22222	Einstein	Physics	95000
32343	El Said	History	60000
33456	Gold	Physics	87000
45565	Katz	Comp. Sci.	75000
58583	Califieri	History	62000
76543	Singh	Finance	80000
76766	Crick	Biology	72000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000

ID	name	dept_name	salary
22222	Einstein	Physics	95000
33456	Gold	Physics	87000

$$\sigma_{salary>9000}(instructor)$$

$$\sigma_{dept_name="Physics"} \wedge salary>9000(instructor)$$

$$\sigma_{dept_name=building}(department)$$

4.1.1.2. Operator projekcji (project): Π



Operator projekcji

- Notacja: $\Pi_{A_1, A_2, \dots, A_k}(r)$

gdzie A_1, A_2 są nazwami atrybutów i r jest nazwą relacji.

- Wynik: relacja z k kolumnami uzyskana w wyniku usunięcia tych niewymienionych
- Zduplikowane wiersze usunięte z wynikowej relacji, bo relacje są zbiorami
- Przykład: eliminacja atrybutu $dept_name$ z $instructor$

$$\Pi_{ID, name, salary}(instructor)$$

Przykład projekcji:

$$\Pi_{ID, name, salary}(instructor)$$

ID	$name$	$dept_name$	$salary$
10101	Srinivasan	Comp. Sci.	65000
12121	Wu	Finance	90000
15151	Mozart	Music	40000
22222	Einstein	Physics	95000
32343	El Said	History	60000
33456	Gold	Physics	87000
45565	Katz	Comp. Sci.	75000
58583	Califieri	History	62000
76543	Singh	Finance	80000
76766	Crick	Biology	72000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000

ID	$name$	$salary$
10101	Srinivasan	65000
12121	Wu	90000
15151	Mozart	40000
22222	Einstein	95000
32343	El Said	60000
33456	Gold	87000
45565	Katz	75000
58583	Califieri	62000
76543	Singh	80000
76766	Crick	72000
83821	Brandt	92000
98345	Kim	80000

4.1.1.3. Operator unii (union): \cup



Operator unii

- Notacja: $r \cup s$
- Definicja:

$$r \cup s = \{t \mid t \in r \text{ or } t \in s\}$$
- Muszą być spełnione warunki:
 - r, s muszą mieć ten sam **stopień** (tę samą liczbę atrybutów)
 - dziedziny atrybutów muszą być **kompatybilne**
- Przykład: znajdź wszystkie kursy prowadzone w semestrze zimowym (Fall) 2009 lub letnim (Spring) 2010 lub w obydwu

$$\Pi_{course_id} (\sigma_{semester="Fall" \wedge year=2009} (section)) \cup \\ \Pi_{course_id} (\sigma_{semester="Spring" \wedge year=2010} (section))$$

Przykład unii:

$$\Pi_{course_id} (\sigma_{semester="Fall" \wedge year=2009} (section)) \cup \\ \Pi_{course_id} (\sigma_{semester="Spring" \wedge year=2010} (section))$$

course_id	sec_id	semester	year	building	room_number	time_slot_id	course_id
BIO-101	1	Summer	2009	Painter	514	B	CS-101
BIO-301	1	Summer	2010	Painter	514	A	CS-315
CS-101	1	Fall	2009	Packard	101	H	CS-319
CS-101	1	Spring	2010	Packard	101	F	CS-347
CS-190	1	Spring	2009	Taylor	3128	E	FIN-201
CS-190	2	Spring	2009	Taylor	3128	A	HIS-351
CS-315	1	Spring	2010	Watson	120	D	MU-199
CS-319	1	Spring	2010	Watson	100	B	PHY-101
CS-319	2	Spring	2010	Taylor	3128	C	
CS-347	1	Fall	2009	Taylor	3128	A	
EE-181	1	Spring	2009	Taylor	3128	C	
FIN-201	1	Spring	2010	Packard	101	B	
HIS-351	1	Spring	2010	Painter	514	C	
MU-199	1	Spring	2010	Packard	101	D	
PHY-101	1	Fall	2009	Watson	100	A	

4.1.1.5. Operator różnicy zbiorów (set difference difference):



Operator różnicy zbiorów

- Notacja $r - s$
- Definicja:

$$r - s = \{t \mid t \in r \text{ and } t \notin s\}$$
- Relacje muszą być **kompatybilne**.
 - r i s muszą mieć ten sam stopień
 - dziedziny atrybutów r i s muszą być takie same
- Przykład: znajdź wszystkie kursy prowadzone w semestrze Fall 2009 semester, ale nie w Spring 2010

$\Pi_{course_id} (\sigma_{semester='Fall' \wedge year=2009} (section)) -$

$\Pi_{course_id} (\sigma_{semester='Spring' \wedge year=2010} (section))$

Przykład operatora różnicę

$\Pi_{course_id} (\sigma_{semester='Fall' \wedge year=2009} (section)) -$
 $\Pi_{course_id} (\sigma_{semester='Spring' \wedge year=2010} (section))$

course_id	sec_id	semester	year	building	room_number	time_slot_id
BIO-101	1	Summer	2009	Painter	514	B
BIO-301	1	Summer	2010	Painter	514	A
CS-101	1	Fall	2009	Packard	101	H
CS-101	1	Spring	2010	Packard	101	F
CS-190	1	Spring	2009	Taylor	3128	E
CS-190	2	Spring	2009	Taylor	3128	A
CS-315	1	Spring	2010	Watson	120	D
CS-319	1	Spring	2010	Watson	100	B
CS-319	2	Spring	2010	Taylor	3128	C
CS-347	1	Fall	2009	Taylor	3128	A
EE-181	1	Spring	2009	Taylor	3128	C
FIN-201	1	Spring	2010	Packard	101	B
HIS-351	1	Spring	2010	Painter	514	C
MU-199	1	Spring	2010	Packard	101	D
PHY-101	1	Fall	2009	Watson	100	A

course_id
CS-347
PHY-101

4.1.1.6. Operator produktu kartezańskiego (Cartesian product): \times



Operator produktu kartezańskiego

- Notacja $r \times s$

- Definicja:

$$r \times s = \{t q \mid t \in r \text{ and } q \in s\}$$

- Założenie, że atrybuty $r(R)$ i $s(S)$ są rozłączne.
(tzn. $R \cap S = \emptyset$).
- Jeżeli atrybuty $r(R)$ i $s(S)$ nie są rozłączne, należy użyć przemianowania.
- Przykład:
 - $r = \text{instructor} \times \text{teaches}$
 - powstaje relacja $r=(\text{instructor.ID}, \text{name}, \text{dept_name}, \text{salary}, \text{teaches.ID}, \text{course_id}, \text{sec_id}, \text{semester}, \text{year})$

■ Relacje r, s

A	B
a	1
b	2

r

C	D	E
a	10	a
b	10	a
b	20	b
c	10	b

s

■ $r \times s$

A	B	C	D	E
a	1	a	10	a
a	1	b	10	a
a	1	b	20	b
a	1	c	10	b
b	2	a	10	a
b	2	b	10	a
b	2	b	20	b
b	2	c	10	b

Produkt kartezański - przykład:

4.1.1.7. Operator przemianowania (assignment): ρ



Operator przemianowania

- Pozwala na nazwanie, a tym samym odniesienie się do wyników wyrażeń algebry relacji.
- Pozwala odnosić się do relacji za pomocą więcej niż jednej nazwy.
- Przykład:

$$\rho_X(E)$$

zwraca wynik wyrażenia E pod nazwą X

- Jeżeli wyrażenie algebry relacji E ma licznosć n , to wtedy

$$\rho_{x(A_1, A_2, \dots, A_n)}(E)$$

zwraca wynik wyrażenia E pod nazwą X z atrybutami przemianowanymi na A_1, A_2, \dots, A_n .



Przykładowe zapytanie

- Znajdź najwyższą pensję na uniwersytecie
 - Krok 1: znajdź pensje, które są mniejsze od dowolnych innych pensji (tzn. nie są maksymalne)
 - użyj kopii $instructor$ pod nową nazwą d
 - $\Pi_{instructor.salary} (\sigma_{instructor.salary < d.salary} (instructor \times \rho_d(instructor)))$
 - Krok 2: Znajdź najwyższą pensję
 - $\Pi_{salary}(instructor) - \Pi_{instructor.salary} (\sigma_{instructor.salary < d.salary} (instructor \times \rho_d(instructor)))$

4.1.2. Dodatkowe operatory:

Nie dodają nowych możliwości, ale upraszczają zapytania.

- Przecięcie zbiorów (Set intersection)
- Złączenie naturalne (Natural join)
- Przypisanie (Assignment)
- Złączenie zewnętrzne (Outer join)

4.1.2.1. Przecięcie zbiorów (set intersection): \cap



Operator przecięcia zbiorów

- Notacja: $r \cap s$
- Definicja:
$$r \cap s = \{ t \mid t \in r \text{ and } t \in s \}$$
- Założenie:
 - r, s mają ten sam stopień
 - atrybuty relacji r i s są kompatybilne
- Uwaga: $r \cap s = r - (r - s)$

■ Relacje r, s	r	s	$r \cap s$																		
	<table border="1"><thead><tr><th>A</th><th>B</th></tr></thead><tbody><tr><td>α</td><td>1</td></tr><tr><td>α</td><td>2</td></tr><tr><td>β</td><td>1</td></tr></tbody></table>	A	B	α	1	α	2	β	1	<table border="1"><thead><tr><th>A</th><th>B</th></tr></thead><tbody><tr><td>α</td><td>2</td></tr><tr><td>β</td><td>3</td></tr></tbody></table>	A	B	α	2	β	3	<table border="1"><thead><tr><th>A</th><th>B</th></tr></thead><tbody><tr><td>α</td><td>2</td></tr></tbody></table>	A	B	α	2
A	B																				
α	1																				
α	2																				
β	1																				
A	B																				
α	2																				
β	3																				
A	B																				
α	2																				

4.1.2.2. [E2017] Złączenie naturalne (natural join) i złączenie theta:

Złączenie naturalne:



- Notacja: $r \bowtie s$
- Niech r i s będą relacjami o schematach R i S .
To $r \bowtie s$ jest relacją o schemacie $R \cup S$ uzyskanym następująco:
 - Rozważ każdą parę krotek t_r z r i t_s z s .
 - Jeżeli t_r i t_s mają tę samą wartość na każdym atrybutie z $R \cap S$, dodaj krotkę t do wyniku, gdzie
 - o t ma taką samą wartość jak t_r w r
 - o t ma taką samą wartość jak t_s w s
- Przykład:
 - $R = (A, B, C, D)$
 - $S = (E, B, D)$
 - Wynikowy schemat = (A, B, C, D, E)
 - $r \bowtie s$ jest zdefiniowane jako:
 $\Pi_{r.A, r.B, r.C, r.D, s.E} (\sigma_{r.B = s.B \wedge r.D = s.D} (r \times s))$

Naturalne złączenie – przykład



■ Relacje r, s				r			s		
	A	B	C	D	B	D	E		
α	1	α	a		1	a	α		
β	2	γ	a		3	a	β		
γ	4	β	b		1	a	γ		
α	1	γ	a		2	b	δ		
δ	2	β	b		3	b	ϵ		

■ Relacje $r \bowtie s$					
	A	B	C	D	E
α	1	α	a		α
α	1	α	a		γ
α	1	γ	a		α
α	1	γ	a		γ
δ	2	β	b		δ

Złączenie naturalne i złączenie theta:



Złączenie naturalne i złączenie theta

- Znajdź nazwiska wszystkich wykładowców z wydziału informatyki wraz z nazwami prowadzonych przez nich kursów
 - $\Pi_{name, title} (\sigma_{dept_name='Comp. Sci.'} (instructor \bowtie teaches \bowtie course))$
- Złączenie naturalne jest łączne
 - $(instructor \bowtie teaches) \bowtie course$ jest równowiązne $instructor \bowtie (teaches \bowtie course)$
- Złączenie naturalne jest przemienne
 - $instruct \bowtie teaches$ jest równowiązne $teaches \bowtie instructor$
- **Złączenie theta** $r \bowtie_\theta s$ jest definiowane jako:
 - $r \bowtie_\theta s = \sigma_\theta (r \times s)$

Złączenie naturalne vs złączenie theta - przykład:

Złączenie naturalne

- Zwierzaki

gatunek	imię	waga
Papuga	Kropka	3,50
Papuga	Lulu	5,35
Papuga	Hipek	3,50
Lis	Fufu	6,35
Krokodyl	Czako	75,00

- Zwierzaki \bowtie Gatunki

gatunek	imię	waga	kontynent
Papuga	Kropka	3,50	Ameryka
Papuga	Lulu	5,35	Ameryka
Papuga	Hipek	3,50	Ameryka
Lis	Fufu	6,35	Europa
Krokodyl	Czako	75,00	Afryka

Złączenie theta

- Zwierzaki

gatunek	imię	waga
Papuga	Kropka	3,50
Papuga	Lulu	5,35
Papuga	Hipek	3,50
Lis	Fufu	6,35
Krokodyl	Czako	75,00

- Zwierzaki \bowtie Gatunki

gatunek	imię	waga	kontynent	nazwa	kontynent
Papuga	Kropka	3,50	Ameryka	Papuga	Ameryka
Papuga	Lulu	5,35	Ameryka	Papuga	Ameryka
Papuga	Hipek	3,50	Ameryka	Papuga	Ameryka
Lis	Fufu	6,35	Europa	Lis	Europa
Krokodyl	Czako	75,00	Afryka	Krokodyl	Afryka

4.1.2.5. Przypisanie (assignment): \leftarrow



Operator przypisania

- Operator przypisania (\leftarrow) dostarcza wygodnego sposoby wyrażania złożonych zapytań.
 - Zapisz zapytanie jako program sekwencyjny składający się z
 - o serii przypisań
 - o a następnie przez wyrażenie, którego wartość jest wyświetlna jako wynik zapytania.
 - Przypisanie musi być zawsze robione do relacji tymczasowej (niewidocznej).
- Naturalne złączenie $r \bowtie s$ może zostać zapisane:
 - $temp1 \leftarrow R \times S$
 - $temp2 \leftarrow \sigma_{r.A1 = s.A1 \wedge r.A2 = s.A2 \wedge \dots \wedge r.An = s.An} (temp1)$
 - $result = \Pi_{R \cup S} (temp2)$
- Przypisanie do stałej relacji oznacza modyfikację

4.1.2.6. Złączenie zewnętrzne (outer join)



Złączenie zewnętrzne (outer join)

- Rozszerzenie operatora złączenia, który zapobiega utracie danych.
- Wylicza złączenie, a następnie dodaje do wyniku krotki z jednej z relacji, które nie pasują do krotek w drugiej relacji.
- Używa wartości *null*:
 - *null* oznacza, że wartość jest nieznana lub nie istnieje
 - wszystkie porównania z *null* są (generalnie) z definicji **false**.



Przykład – złączenie zewnętrzne

- Relacja *instructor*

<i>ID</i>	<i>name</i>	<i>dept_name</i>
10101	Srinivasan	Comp. Sci.
12121	Wu	Finance
15151	Mozart	Music

- Relacja *teaches*

<i>ID</i>	<i>course_id</i>
10101	CS-101
12121	FIN-201
76766	BIO-101

Przykład – złączenie zewnętrzne

- Złączenie

instructor \bowtie *teaches*

<i>ID</i>	<i>name</i>	<i>dept_name</i>	<i>course_id</i>
10101	Srinivasan	Comp. Sci.	CS-101
12121	Wu	Finance	FIN-201

- Left Outer Join

instructor $\bowtie\bowtie$ *teaches*

<i>ID</i>	<i>name</i>	<i>dept_name</i>	<i>course_id</i>
10101	Srinivasan	Comp. Sci.	CS-101
12121	Wu	Finance	FIN-201
15151	Mozart	Music	null

Przykład – złączenie zewnętrzne

- Right Outer Join

instructor $\bowtie\subset$ *teaches*

<i>ID</i>	<i>name</i>	<i>dept_name</i>	<i>course_id</i>
10101	Srinivasan	Comp. Sci.	CS-101
12121	Wu	Finance	FIN-201
76766	null	null	BIO-101

- Full Outer Join

instructor $\bowtie\bowtie\subset$ *teaches*

<i>ID</i>	<i>name</i>	<i>dept_name</i>	<i>course_id</i>
10101	Srinivasan	Comp. Sci.	CS-101
12121	Wu	Finance	FIN-201
15151	Mozart	Music	null
76766	null	null	BIO-101

4.1.3. Operatory rozszerzonej algebry relacji

Dostarczają możliwości pisania zapytań, które nie mogą być wyrażone przy pomocy podstawowych operatorów algebry relacji.

4.1.3.1. [E2017] Uogólniona projekcja



Uogólniona projekcja

- Rozszerza operację projekcji zezwalając aby operatory arytmetyczne były użyte w liście projekcji.

$$\prod_{F_1, F_2, \dots, F_n} (E)$$

- E jest dowolnym wyrażeniem algebry relacji
- Każdy F_1, F_2, \dots, F_n jest wyrażeniem arytmetycznym ze stałymi i atrybutami ze schematu E .

4.1.3.2. [E2017] Operacje agregacji



Funkcje i operacje agregujące

- Relacja r :

A	B	C
α	α	7
α	β	7
β	β	3
β	β	10

Funkcja agregująca bierze zbiór wartości i zwraca w wyniku pojedynczą wartość:
avg, min, max, sum, count

$$G_{\text{sum}(\text{salary})} (\text{instructor})$$

$$\blacksquare G_{\text{sum}(c)}(r)$$

$$\boxed{\text{sum}(c)}$$

$$\boxed{27}$$



Operacje agregujące – eliminacja duplikatów

- Znajdź wszystkich instruktorów, którzy prowadzą kursy w semestrze wiosennym 2010

$$G_{\text{count-distinct}(ID)} (\sigma_{\text{semester}=\text{"Spring"} \wedge \text{year}=2010}(\text{teaches}))$$

Operacje agregujące - przykład

- Znajdź średnią pensję na każdym wydziale

$\text{dept_name } G \text{ avg(salary) (instructor)}$

ID	name	dept_name	salary
76766	Crick	Biology	72000
45565	Katz	Comp. Sci.	75000
10101	Srinivasan	Comp. Sci.	65000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000
12121	Wu	Finance	90000
76543	Singh	Finance	80000
32343	El Said	History	60000
58583	Califieri	History	62000
15151	Mozart	Music	40000
33456	Gold	Physics	87000
22222	Einstein	Physics	95000

dept_name	avg_salar
Biology	72000
Comp. Sci.	77333
Elec. Eng.	80000
Finance	85000
History	61000
Music	40000
Physics	91000

Operacje agregacji G

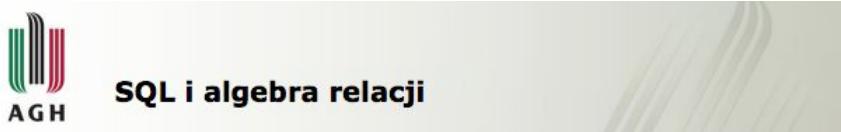
- Operacja agregacji** w algebrze relacji

$$G_{G_1, G_2, \dots, G_n} G_{F_1(A_1), F_2(A_2), \dots, F_n(A_n)}(E)$$

E jest dowolnym wyrażeniem algebry relacji

- G_1, G_2, \dots, G_n lista atrybutów po której ma być grupowanie (może być pusta)
- Każde F_i jest funkcją agregującą
- Każde A_i jest nazwą atrybutu

4.1.4. [Dodatkowo] SQL i algebra relacji



- **select A₁, A₂, .. A_n**
from r₁, r₂, ..., r_m
where P

jest równoważne następującemu wyrażeniu w algebrze relacji wielozbiorów

$$\Pi_{A_1, \dots, A_n} (\sigma_P (r_1 \times r_2 \times \dots \times r_m))$$

- **select A₁, A₂, **sum**(A₃)**
from r₁, r₂, ..., r_m
where P
group by A₁, A₂

jest równoważne następującemu wyrażeniu w algebrze relacji wielozbiorów

$$A_1, A_2 \mathcal{G} \text{sum}(A_3) (\Pi_{A_1, \dots, A_n} (\sigma_P (r_1 \times r_2 \times \dots \times r_m)))$$

4.2. [E] Przetwarzanie zapytań

4.2.1. Podstawowe kroki w przetwarzaniu zapytań:

1. **Parsowanie i translacja** na wyrażenie algebry relacji
2. **Optymalizacja**
3. **Ewaluacja**

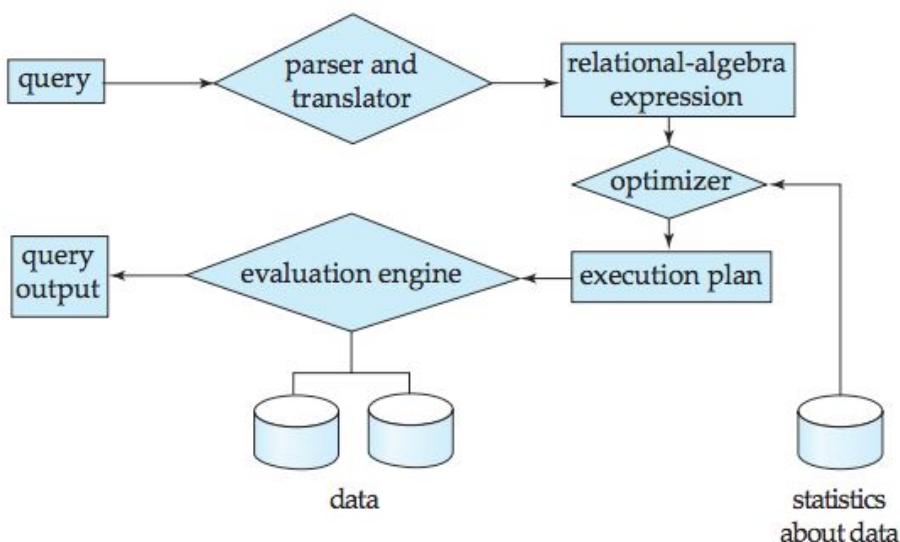


Figure 12.1 Steps in query processing.

5. Wykład - Projektowanie bazy danych, Model związków-encji E-R

5.1. [Dodatkowo] Model związków encji ER

- **Model związków encji** – całościowa logiczna reprezentacja rzeczywistości – mapowanie znaczeń i zależności
- **Encja** reprezentuje rzeczywisty obiekt rozróżnialny od innych
 - Obiekty materialne (np. książka, instruktor, student)
 - Obiekty abstrakcyjne (np. rezerwacja hotelu, kurs)
- Encje posiadają **atrybuty** (np. tytuł książki, imię instruktora)
- **Zbiór encji** (entity set) – zbiór obiektów tego samego typu, mających te same atrybuty (odpowiednik tabeli-relacji z modelu relacyjnego)

Zbiory encji - przykład



Zbiory encji - przykład

instructor_ID	instructor_name
76766	Crick
45565	Katz
10101	Srinivasan
98345	Kim
76543	Singh
22222	Einstein

instructor

student-ID	student_name
98988	Tanaka
12345	Shankar
00128	Zhang
76543	Brown
76653	Aoi
23121	Chavez
44553	Peltier

student

5.1.1. Projekt związków encji

Dobry projekt ER nie powinien posiadać redundancji

Etapy projektowania w modelu E-R:

- identyfikacja zbiorów encji
- wybór odpowiednich atrybutów:
- ustalanie związków między zbiorami encji

5.2. [E] Czym jest zbiór związków encji?

5.2.1. Ze slajdów: Zbiory związków



Zbiory związków

- **Związek** (*relationship*) jest powiązaniem między kilkoma encjami

Przykład:

44553 (Peltier) advisor 22222 (Einstein)
encja *student* związek encja *instructor*

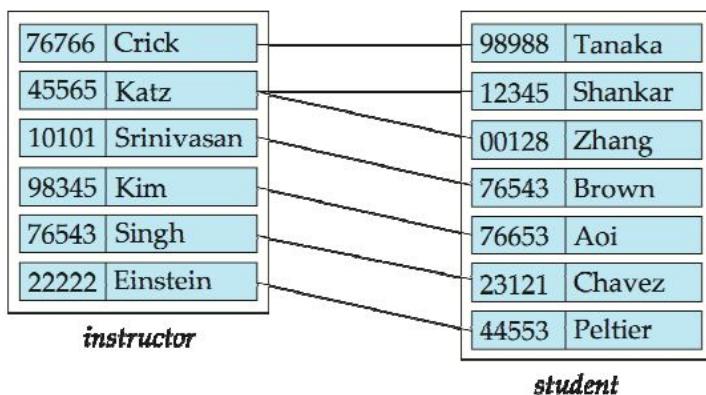
- **Zbiór związków** (*relationship set*) jest matematyczną relacją między $n \geq 2$ encjami, każdy z nich jest ze zbioru encji

$$\{(e_1, e_2, \dots, e_n) \mid e_1 \in E_1, e_2 \in E_2, \dots, e_n \in E_n\}$$

gdzie (e_1, e_2, \dots, e_n) jest związkiem

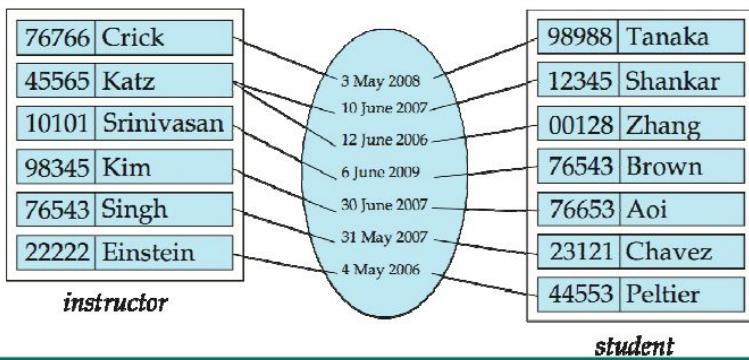


Zbiór związków *advisor*



Zbiory związków c.d.

- Zbiory związków również mogą mieć **atrybuty**.
- Zbiór związków *advisor* m. zb. encji *instructor* a *student* ma atrybut *date*



5.2.2. Z poprzedniego opracowania:

Zbiór związków to matematyczna relacja między dwoma lub więcej encjami. Zapisujemy ją: (e1, e2, ..., en), gdzie każda encja ei należy do zbioru encji Ei. (tak na chłopski rozum: dla zbioru związków o stopniu 2 odpowiednikiem jest zbiór wszystkich krotek połączonych danym kluczem obcym natomiast jeden związek to jedna para krotek)

modeluje przedsiębiorstwo jako zbiór encji i związków (zależności) reprezentowany schematycznie przez diagram związków encji

Encja to abstrakcyjny obiekt pewnego rodzaju. Kolekcja podobnych encji tworzy zbiór encji. Encja ~ krotka, Zbiór encji ~ relacja (przykład 4.1 Systemy baz danych. Kompletny podręcznik)

Związek ~ jest pomiędzy dwoma konkretnymi krotkami np. instruktor Andrzej jest w związku advisor ze studentem Wojtkiem

Zbiór związków ~ jest pomiędzy całymi relacjami

5.3. [E] Cechy związków encji

[E2017] - Związki w modelu konceptualnym(wymienić 3 cechy)

// KTOŚ COŚ?

no chyba po prostu: stopień, ograniczenie krotności, pewność istnienia/udział

5.3.1. Stopnie związków encji

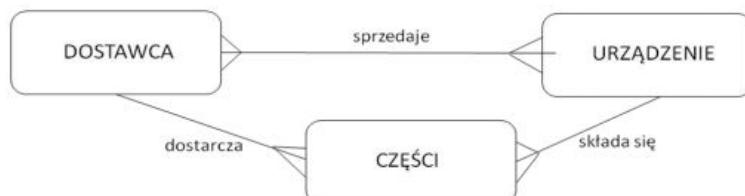
Stopień związku - ilość encji biorących udział w związku:

- **unarny** (binarny rekursywny - tak jak w Northwind w pracownikach ReportsTo)(np. 1:1 **pracownik** może być przełożonym wielu **pracowników**),

- **binarny** (np. M:N **student** może zapisać się na wiele **przedmiotów**, na przedmiot może zapisać się wielu studentów),



- **ternarny** (np. **kierowca** zostaje złapany przez **policjanta** za **wykroczenie**),



Przykład związku ternarnego: DOSTAWCA sprzedaje URZĄDZENIE.
 DOSTAWCA dostarcza CZĘŚCI do URZĄDZENIA.
 URZĄDZENIE składa się z CZĘŚCI.

- n - arny... inaczej arność związku)

ad//The number of entity sets that participate in a relationship set is the degree of the relationship set. A binary relationship set is of degree 2; a ternary relationship set is of degree 3.

5.3.2. Ograniczenie krotności (poniższe dotyczą związków binarnych)

- jeden-do-jeden (1:1)
- jeden-do-wiele (1:M)
- wiele-do-jeden(M:1)
- wiele-do-wiele (M:N)

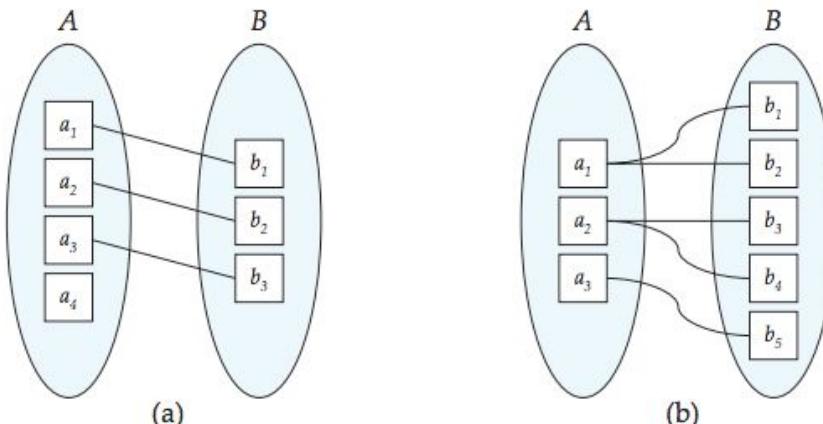


Figure 7.5 Mapping cardinalities. (a) One-to-one. (b) One-to-many.

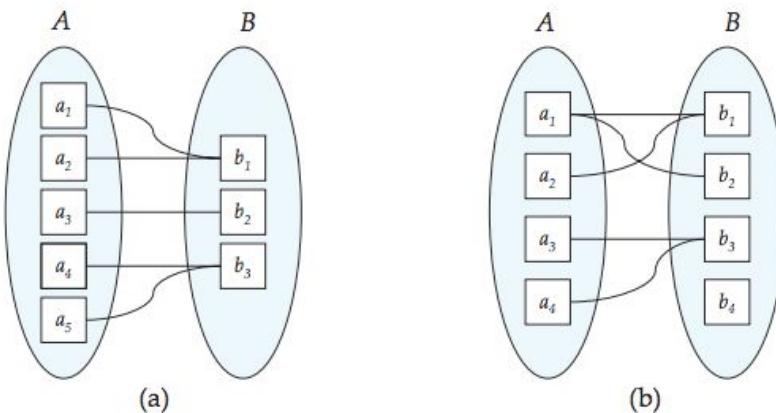
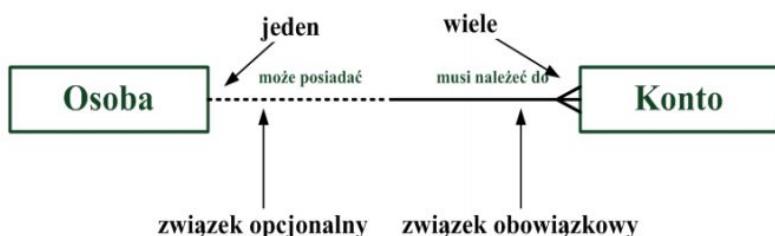


Figure 7.6 Mapping cardinalities. (a) Many-to-one. (b) Many-to-many.

5.3.3. Istnienie

- opcjonalny (osoba może posiadać 0 lub wiele kont". (0..*)
- obowiązkowy (jedno konto musi należeć tylko do jednej osoby)



Uwaga: Zaproponowane tutaj oznaczenie może być trochę mylące. Na wykładzie liczność była trochę inaczej wyrażana albo za pomocą zapisu np. 1..* albo za pomocą strzałek. Poza tym oznaczenie może być również mylące, ponieważ w visual paradigm linia przerywana wyraża relację nieidentyfikującą, a ciągła identyfikującą. Opcjonalność wyraża się za pomocą pustego kółka, a obowiązkowość prostopadłą linią. Nie wiem czy to jest charakterystyczne tylko dla tego programu czy ogólnie dla notacji tzw. "kruczej stopki" - która jest chyba najbardziej popularną notacją.

5.3.4. Udział zbioru encji w zbiorze związków

- Całkowity udział (total participation)
- Częściowy udział (partial participation)

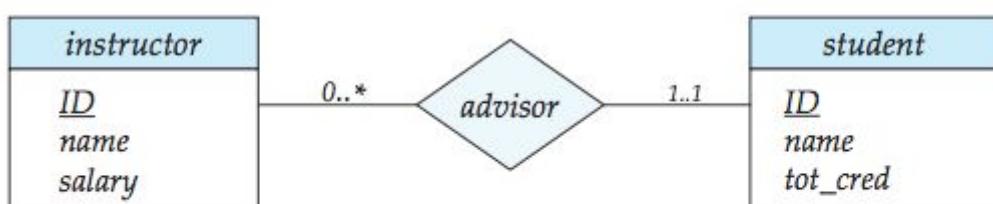


Figure 7.10 Cardinality limits on relationship sets.

For example, consider Figure 7.10. The line between advisor and student has a cardinality constraint of 1..1, meaning the minimum and the maximum cardinality are both 1. That is, each student must have exactly one advisor. The limit 0..* on the line between advisor and instructor indicates that an instructor can have zero or more students. Thus, the relationship advisor is one-to-many from instructor to student, and further the participation of student in advisor is **total**, implying that a student must have an advisor.

It is easy to misinterpret the 0..* on the left edge and think that the relationship advisor is many-to-one from instructor to student—this is exactly the reverse of the correct interpretation.

If both edges have a maximum value of 1, the relationship is one-to-one. If we had specified a cardinality limit of 1..* on the left edge, we would be saying that each instructor must advise at least one student.

The E-R diagram in Figure 7.10 could alternatively have been drawn with a double line from student to advisor, and an arrow on the line from advisor to instructor, in place of the cardinality constraints shown. This alternative diagram would enforce exactly the same constraints as the constraints shown in the figure.

5.4. [E] Czym jest klucz kandydujący w ER?

5.4.1. [Dodatkowo] Klucze dla związków encji

Sposób na rozróżnienie encji w kontekście atrybutów

- Zapewniają unikalność związków
- Kombinacja (suma) kluczy głównych zbiorów encji biorących udział w związku tworzy superklucz dla zbioru związków.
- Przy podjęciu decyzji o PK dla zbioru związku należy wziąć pod uwagę krotność

5.4.2. [E] Klucz kandydujący

Minimalny zbiór atrybutów K ze zbioru R, taki że klucz złożony z atrybutów K jednoznacznie określa unikalność encji w zbiorze encji, a usunięcie dowolnego atrybutu z K usuwa jednoznaczność

Każdy zbiór encji może mieć więcej niż jeden klucz kandydujący.

Klucze kandydujące dzielą się na:

- klucze proste – składające się z dokładnie jednego atrybutu,
- klucze złożone / połączeniowe – składające się z więcej niż jednego atrybutu.

5.4.3. [Dodatkowo] Klucz główny

To klucz kandydujący, który został wybrany do jednoznacznej identyfikacji każdego z wystąpień encji w zbiorze encji. Każdy zbiór encji może mieć dokładnie tylko jeden klucz główny.

5.5. [E] Jak się wyznacza klucz główny w zbiorach związków?

Jeśli po którejś stronie związku jest krotność 1, to wystarczy wziąć ten jeden PK (np. związek: klient ma wiele zamówień, zamówienie zostało złożone przez jednego klienta - jako klucz główny związku wybieramy PK zamówienia).

Jeśli wszędzie jest 'wiele', to kombinację wszystkich kluczy głównych wchodzących w skład związku.np. związek: klient zamówił wiele produktów, produkt został zamówiony przez wielu klientów - kombinacja PK klienta i produktu)

- For a binary many-to-many relationship, the union of the primary-key attributes from the participating entity sets becomes the primary key.
- For a binary one-to-one relationship set, the primary key of either entity set can be chosen as the primary key. The choice can be made arbitrarily.
- For a binary many-to-one or one-to-many relationship set, the primary key of the entity set on the "many" side of the relationship set serves as the primary key.
- For an n -ary relationship set without any arrows on its edges, the union of the primary key-attributes from the participating entity sets becomes the primary key.
- For an n -ary relationship set with an arrow on one of its edges, the primary keys of the entity sets not on the "arrow" side of the relationship set serve as the primary key for the schema. Recall that we allowed only one arrow out of a relationship set.

5.6. [E2017] Zbiory słabych encji. Co to są słabe encje, objaśnić przekształcanie ich do modelu relacyjnego.

5.6.1. Encja słaba

Encja słaba (weak entity)

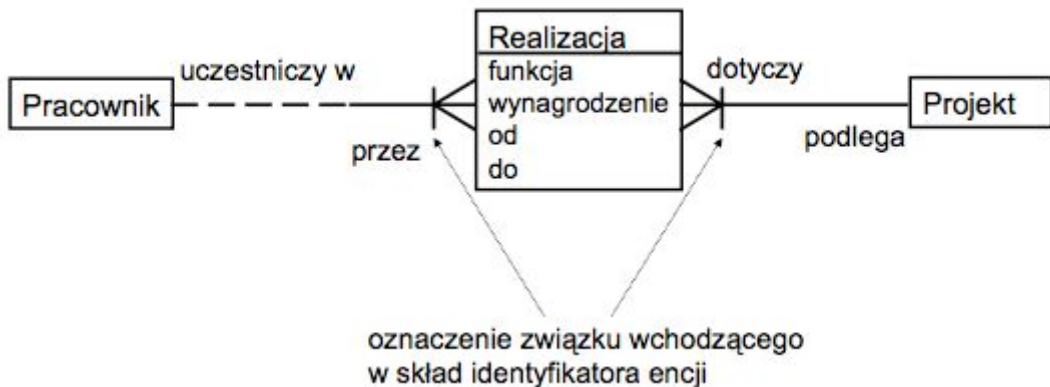
- nie posiada swojego identyfikatora
- wystąpienia encji mogą istnieć tylko w kontekście wystąpień encji powiązanych z encją słabą
- konkretne wystąpienie encji Realizacja może wystąpić wyłącznie w kontekście konkretnego pracownika i konkretnego projektu



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.

5.6.2. Identyfikator encji słabej

Identyfikatorem encji słabej są wszystkie związki, w które wchodzi ta encja. W przykładzie ze slajdu, w skład identyfikatora encji Realizacja wchodzą związkki z encją Pracownik i Projekt.



5.6.3. Z wykładowu: Zbiór słabych encji

Zbiór encji nie posiadający klucza głównego określany jest jako zbiór słabych encji
Istnienie zbioru słabych encji oparte jest na istnieniu **zbiorów identyfikujących encje**

Dyskryminator zbioru słabych encji

- klucz częściowy zbioru encji
- pozwala na rozróżnienie encji w zbiorze słabych encji
- atrybuty z przerywaną linią na diagramie

Związek w którym uczestniczą zbiory słabych encji – w podwójnym rombie.

PK dla section – (course_id, sec_id, semester, year)

(Z książki database system concepts)

The discriminator of a weak entity set is a set of attributes that allows this distinction to be made.
The primary key of a weak entity set is formed by the primary key of the identifying entity set, plus the weak entity set's discriminator. In the case of the entity set section, its primary key is {course id, sec id, year, semester}, where course id is the primary key of the identifying entity set, namely course, and {sec id, year, semester} distinguishes section entities for the same course.

Bardziej szczegółowy opis przykładu course i section w Database System Concepts: 7.5.6 Weak Entity Sets.

5.6.4. Przekształcanie słabych encji do modelu relacyjnego

TODO

// Chyba wtedy "związek" staje się kluczem obcym, a PK słabej encji tworzy się z tego klucza obcego i atrybutów encji: https://en.wikipedia.org/wiki/Weak_entity

Reprezentacja zbiorów encji z prostymi atrybutami:

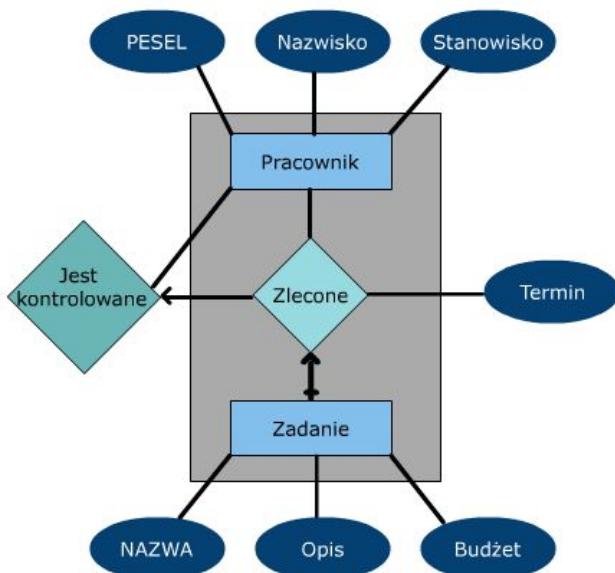
- **Zbiór silnych encji** course(course_id, title, credits)
- **zbiór słabych encji** section(course_id, sec_id, semester, year)

5.7. [E] Co to jest agregacja?

5.7.1. Agregacja

Agregacja (odpowiednik kompozycji) - abstrakcja, przez którą zależności są traktowane jako encje wyższego poziomu.

5.7.2. Przykład agregacji



Agregacja: związek **Zlecone** jest argumentem związku **Jest kontrolowane**. Związek, który jest argumentem innego związku jest obejmowany prostokątem wraz ze swoimi argumentami.

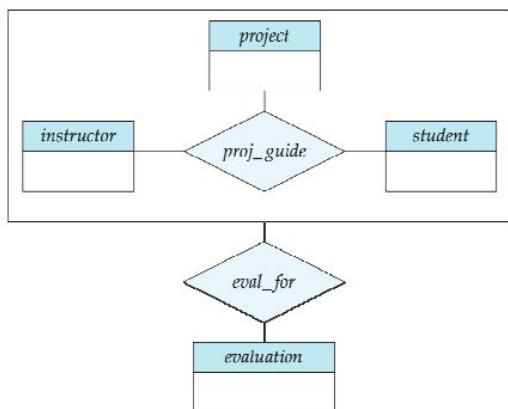
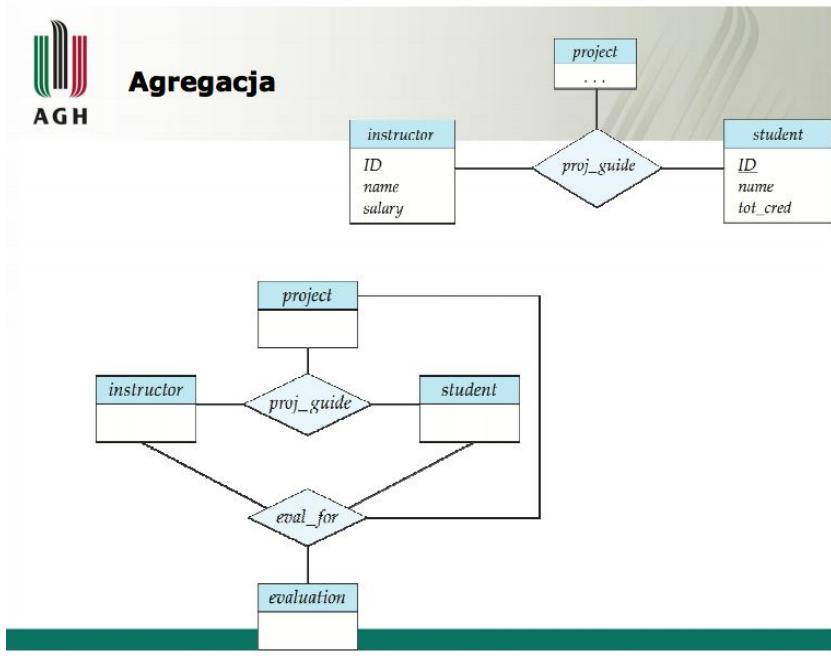
Powstawanie diagramu ilustruje prezentacja [Flash..](#)

5.7.3. Przykład z wykładow

UWAGA: Zygmunt przy omawianiu tego przykładu stwierdza, że go nie rozumie i prosi o podanie innego. Dla wielu osób skończyło się to oblany egzaminem.

Zbiory związków eval_for i proj_guide przedstawiają pokrywające się informacje – Każdy związek eval_for odpowiada związkowi proj_guide

Eliminacja redundancji poprzez wprowadzenie agregacji – Potraktować zbiór związków (proj_guide) jako zbiór pojęć wyższego poziomu proj_guide



Dokładny opis przykładu z Database System Concepts: 7.8.5 Aggregation

One limitation of the E-R model is that it cannot express relationships among relationships. To illustrate the need for such a construct, consider the ternary relationship proj_guide, which we saw earlier, between an instructor, student and project (see Figure 7.13).

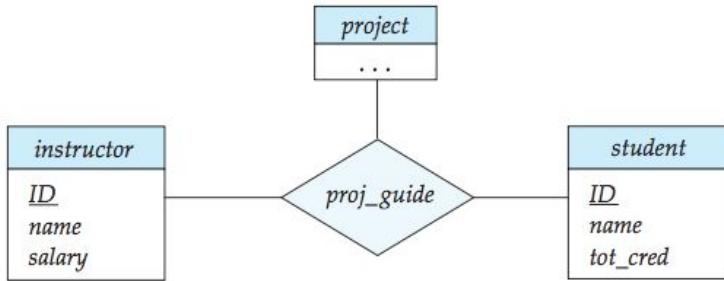
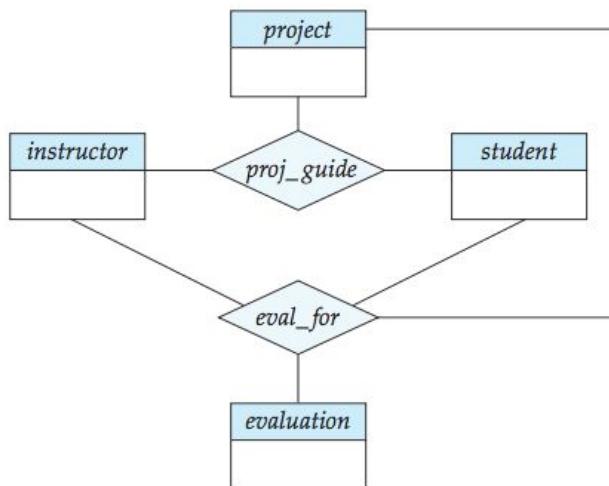


Figure 7.13 E-R diagram with a ternary relationship.

Now suppose that each instructor guiding a student on a project is required to file a monthly evaluation report. We model the evaluation report as an entity evaluation, with a primary key evaluation id. One alternative for recording the (student, project, instructor) combination to which an evaluation corresponds is to create a quaternary (4-way) relationship set eval_for between instructor, student, project, and evaluation. (A quaternary relationship is required—a binary relationship between student and evaluation, for example, would not permit us to represent the (project, instructor) combination to which an evaluation corresponds.) Using the basic E-R modeling constructs, we obtain the E-R diagram of Figure 7.22. (We have omitted the attributes of the entity



sets, for simplicity.)

Figure 7.22 E-R diagram with redundant relationships.

It appears that the relationship sets proj guide and eval for can be combined into one single relationship set. Nevertheless, we should not combine them into a single relationship, since some instructor, student, project combinations may not have an associated evaluation.

There is redundant information in the resultant figure, however, since every instructor, student, project combination in eval for must also be in proj guide. If the evaluation were a value rather than a entity, we could instead make evaluation a multivalued composite attribute of the relationship set proj guide. However, this alternative may not be an option if an evaluation may also be related to other entities; for example, each evaluation report may be associated with a secretary who is responsible for further processing of the evaluation report to make scholarship payments.

The best way to model a situation such as the one just described is to use aggregation. **Aggregation** is an abstraction through which relationships are treated as higher-level entities. Thus, for our example, we regard the relationship set proj guide (relating the entity sets instructor, student, and project) as a higher-level entity set called proj guide. Such an entity set is treated in the same manner as is any other entity set. We can then create a binary relationship eval for between proj_guide and evaluation to represent which (student, project, instructor) combination an evaluation is for. Figure 7.23 shows a notation for aggregation commonly used to represent this situation.

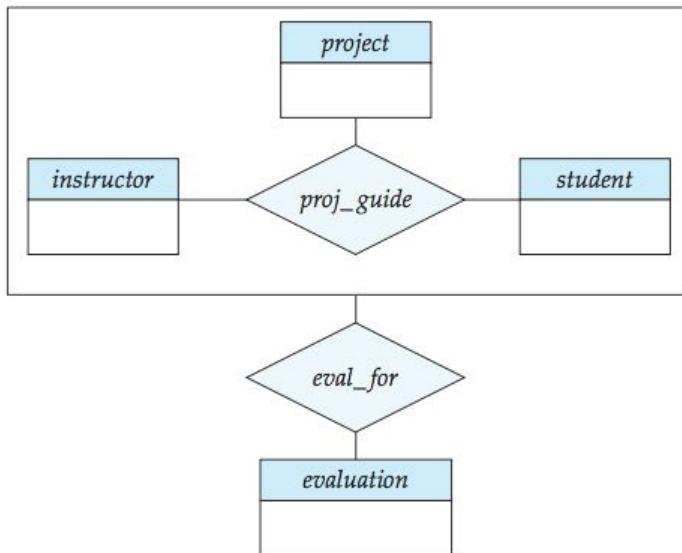


Figure 7.23 E-R diagram with aggregation.

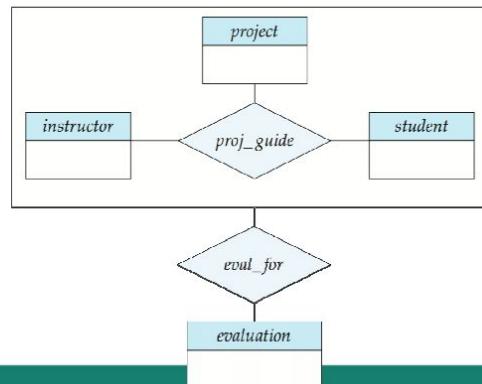
5.8. [E] Zamiana agregacji w ER na postać relacyjną (na model relacyjny?) Do reprezentowania agregacji, stwórz schemat zawierający:

- **PK zagregowanego związku** (w odniesieniu do [poprzedniego przykładu](#): PK proj_guide)
- **PK połączonego zbioru encji** (w odniesieniu do [poprzedniego przykładu](#): PK evaluations)
- **Atrybuty opisowe** - kolumny nienależące do kluczy (zawierają dane, które w określonej relacji są przedmiotem opisu - w odniesieniu do [poprzedniego przykładu](#): atrybuty opisowe związku eval_for)

Schematy odpowiadające agregacji

- Do reprezentowania agregacji, stwórz schemat zawierający:

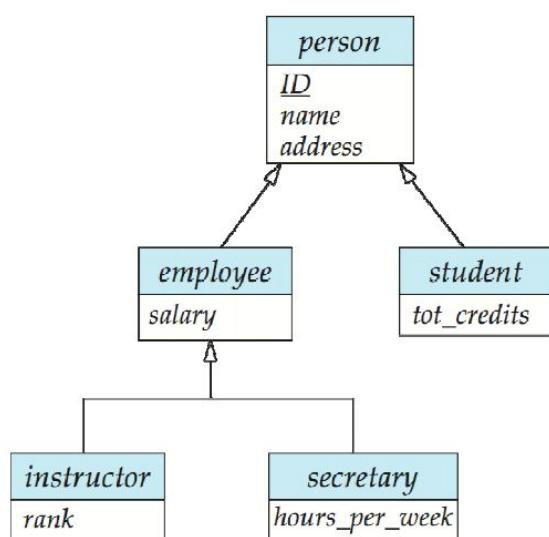
- PK zagregowanego związku
- PK połączonego zbioru encji
- atrybuty opisowe



5.9. [E] Generalizacja i specjalizacja w ER

Specjalizacja - coś a'la dziedziczenie //ale nie mowic tak!!!, przykład z wykładu: zbiory encji instruktorów i studentów mogą być specjalizacją zbioru encji person. Stworzone za pomocą techniki zwanej TOP-DOWN.

Specjalizacja przykład



Generalizacja działa w drugą stronę, person jest generalizacją studentów i instruktorów. Można wyróżnić specjalizację/generalizację częściową/całkowitą na podstawie tego, czy każdy element ze zbioru wyższego rzędu ma odpowiednik w zbiorach niższych rzędów, a także zachodzącą/rozłączną kryterium podziału jest tutaj czy encja ze zbioru wyższego rzędu może mieć kilka specjalizacji.

Tworzymy ją za pomocą techniki zwanej BOTTOM-UP.

Warto tutaj także wspomnieć o ograniczeniach: zdefiniowanych przez użytkownika (user-defined) i przez określony warunek (condition-defined). Więcej: [Ograniczenia na generalizację/specjalizację w ER](#)

5.10. [E] Ograniczenia na generalizację/specjalizację w ER

Ograniczenie określające które encje mogą być członkami danego zbioru encji niższego poziomu. Przynależność może być:

- **zdefiniowana przez ograniczenie (condition defined)** - np. encja wyższego rzędu *student* ma atrybut *student_type*. Tylko encje spełniające warunek *student_type="graduate"* mogą należeć do zbioru encji niższego poziomu *graduate*. Since all the lower-level entities are evaluated on the basis of the same attribute (in this case, on student type), this type of generalization is said to be **attribute-defined**.
- **zdefiniowana przez użytkownika (user defined)** - użytkownik przypisuje encję do danego zbioru encji np. pracownicy uniwersytetu są przypisywani przez użytkowników do odpowiednich zespołów badawczych. Zespoły są reprezentowane jako zbiory encji niższego poziomu dla zbioru encji wyższego poziomu *pracownicy*. Pracownik nie jest przypisywany do danego zespołu automatycznie na podstawie ograniczenia. Zamiast tego odpowiedzialny za to użytkownik przypisuje zespoły do konkretnych pracowników.

Ograniczenie czy encje mogą należeć do więcej niż jednego zbioru encji niższego poziomu w pojedynczej generalizacji

- **Rozłączne** (encja nie może należeć do więcej niż jednego zbioru encji niższego poziomu) - np. student może należeć tylko do jednej encji niższego poziomu *graduate* lub *undergraduate*
- **Zachodzące** (Jednocześnie) - np. encja pracownik może należeć do wielu zbiorów encji niższego poziomu typu *zespół badawczy*

Ograniczenie kompletności określa czy zbiór pojęć ze zbioru encji na wyższym poziomie musi należeć do przynajmniej jednego zbioru encji na niższym poziomie

- **Zupełne (total)** Np. każdy ze studentów musi należeć do jednego ze zbiorów: *undergraduate* lub *graduate*
- **Częściowe (partial)** Np. pracownik uniwersytetu może, ale nie musi należeć do zespołu badawczego

5.11. [E2017] Mapowanie generalizacji i specjalizacji z modelu związków encji na model relacyjny. Klucze główne w relacjach niższych i wyższych poziomów. Generalizacja w ER

5.11.1. Generalizacja i specjalizacja

Generalizacja łączy kilka zbiorów encji, które mają te same cechy, w zbiór encji wyższego poziomu. Specjalizacja to inwersja generalizacji.

Związek "IS A", zwany również superclass - subclass, oznacza przynależność encji niższego poziomu do encji wyższego poziomu (przykład z dbbook: instructor "is a" person).

5.11.2. Reprezentacja generalizacji/specjalizacji w schemacie (z wykładu - 2 metody/schematy)

Silberschatz str. 302

5.11.3. metody w skrócie:

Metoda 1:

- Stwórz schemat dla zbioru encji na wyższym poziomie
- Stwórz schemat dla każdego zbioru encji niższego poziomu włączając PK ze zbioru pojęć wyższego poziomu oraz lokalnych atrybutów
- Dla każdego schematu niższego poziomu stwórz FK
- Wada: wyszukanie informacji o encji niższego poziomu wymaga dostępu do dwóch relacji, odpowiadających encjom niższego i wyższego poziomu

Metoda 2:

- Stwórz schemat dla każdego zbioru encji ze wszystkimi lokalnymi i dziedzicznymi atrybutami
- Jeżeli specjalizacja jest rozłączna (przecięcie zbiorów obiektów podklas jest zbiorem pustym) i zupełna (total participation), schemat dla bardziej ogólnego zbioru pojęć nie wymaga przechowywania informacji redundantnej
- Wada: może wystąpić redundancja, jeśli obiekt należy do więcej niż jednej encji niższego poziomu (czyli jeśli generalizacja będzie zachodząca (overlapping))

5.11.4. Opis metod z Database System Concepts: 7.8.6.1 Representation of Generalization

There are two different methods of designing relation schemas for an E-R diagram that includes generalization. Although we refer to the generalization in Figure 7.21 in this discussion, we simplify it by including only the first tier of lower-level entity sets—that is, employee and student. We assume that ID is the primary key of person.

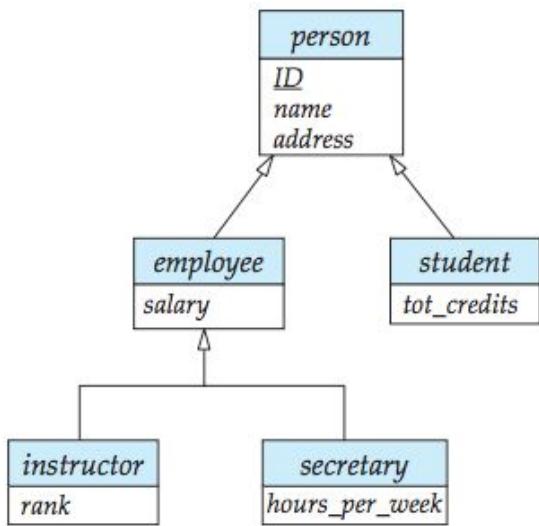


Figure 7.21 Specialization and generalization.

1. Create a schema for the higher-level entity set. For each lower-level entity set, create a schema that includes an attribute for each of the attributes of that entity set plus one for each attribute of the primary key of the higher-level entity set. Thus, for the E-R diagram of Figure 7.21 (ignoring the instructor and secretary entity sets) we have three schemas:
 - person (ID, name, street, city)
 - employee (ID, salary)
 - student (ID, tot cred)
 The primary-key attributes of the higher-level entity set become primary-key attributes of the higher-level entity set as well as all lower-level entity sets. These can be seen underlined in the above example. In addition, we create foreign-key constraints on the lower-level entity sets, with their primary-key attributes referencing the primary key of the relation created from the higher-level entity set. In the above example, the ID attribute of employee would reference the primary key of person, and similarly for student.
2. An alternative representation is possible, if the generalization is disjoint and complete—that is, if no entity is a member of two lower-level entity sets directly below a higher-level entity set, and if every entity in the higher-level entity set is also a member of one of the lower-level entity sets. Here, we do not create a schema for the higher-level entity set. Instead, for each lower-level entity set, we create a schema that includes an attribute for each of the attributes of that entity set plus one for each attribute of the higher-level entity set. Then, for the E-R diagram of Figure 7.21, we have two schemas:
 - employee (ID, name, street, city, salary)
 - student (ID, name, street, city, tot cred)

Both these schemas have ID, which is the primary-key attribute of the higher-level entity set person, as their primary key

One drawback of the second method lies in defining foreign-key constraints. To illustrate the problem, suppose we had a relationship set R involving entity set person. With the first method, when we create a relation schema R from the relationship set, we would also define a foreign-key constraint on

R, referencing the schema person. Unfortunately, with the second method, we do not have a single relation to which a foreign-key constraint on R can refer. To avoid this problem, we need to create a relation schema person containing at least the primary-key attributes of the person entity.

If the second method were used for an overlapping generalization, some values would be stored multiple times, unnecessarily. For instance, if a person is both an employee and a student, values for street and city would be stored twice.

If the generalization were disjoint but not complete—that is, if some person is neither an employee nor a student—then an extra schema

person (ID, name, street, city)

would be required to represent such people. However, the problem with foreign-key constraints mentioned above would remain. As an attempt to work around the problem, suppose employees and students are additionally represented in the person relation. Unfortunately, name, street, and city information would then be stored redundantly in the person relation and the student relation for students, and similarly in the person relation and the employee relation for employees. That suggests storing name, street, and city information only in the person relation and removing that information from student and employee. If we do that, the result is exactly the first method we presented.

5.12. [E2017] Rodzaje atrybutów w ER (5 atrybutów) + mapowanie tych atrybutów na model relacyjny

[E2017] Typy atrybutów w modelu ER i sposób, w jaki każdy z nich jest przechowywany w schemacie.
UWAGA: *Niewymienienie jednego z nich często skutkuje oblaniem egzaminu.*

5.12.1. Rodzaje atrybutów

Silberschatz str. 267

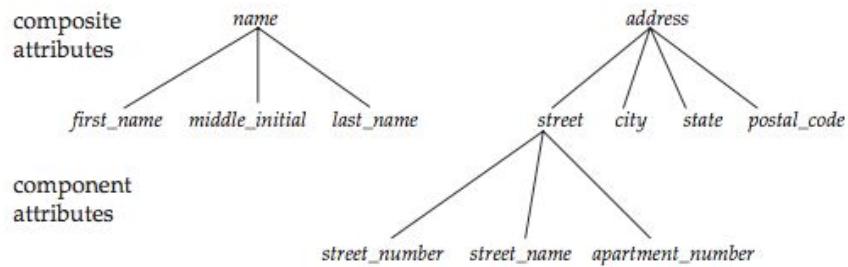
5.12.1.1. Atrybuty proste

Są niepodzielne na mniejsze części (np. wartość pensji instruktora)

5.12.1.2. Atrybuty złożone

Mogą być podzielone na mniejsze części. Np. atrybut *name* może składać się z *first_name*, *middle_initial* i *last_name*. Note also that a composite attribute may appear as a **hierarchy**. In the composite attribute address, its component attribute street can be further divided into *street_number*,

`street_name`, and `apartment_number`. Figure 7.4 depicts these examples of composite attributes for



the `instructor` entity set.

Figure 7.4 Composite attributes `instructor name` and `address`.

5.12.1.3. Atrybuty jednowartościowe ¶¶

Mają jedną wartość dla konkretnej encji. Np. student ma tylko jedną wartość `student_ID`

5.12.1.4. Atrybuty wielowartościowe

Mają zbiór wartości dla danej encji. Np. instruktor z atrybutem `phone_number` może mieć 0, 1 lub kilka numerów telefonów.

5.12.1.5. Atrybuty pochodne

Mogą zostać wywnioskowane na podstawie innych atrybutów. Np. `wiek` instruktora może być obliczony z jego `daty urodzenia` (`wiek` - atrybut pochodny, `data urodzenia` - atrybut bazowy, przechowywany). Nie są przechowywane w DB. Wyliczane są z innych atrybutów.

5.12.2. Mapowanie atrybutów opisane w następnym podpunkcie

5.13. [E] Transformacja atrybutów związków encji na model relacyjny

5.13.1. Atrybuty proste

- W zbiorze silnych encji dla każdego atrybutu prostego tworzymy osobną kolumnę, PK relacji to PK zbioru silnych encji. Np. **course(course_id, title, credits)**
- W zbiorze słabych encji dla każdego atrybutu prostego tworzymy osobną kolumnę, dodatkowo tworzymy kolumnę z PK encji identyfikującej. PK relacji to dyskryminator ([więcej o dyskryminatorze](#)) zbioru słabych encji + PK encji identyfikującej. Np. **section(course_id, sec_id, sem, year)**

Przykład: encja silna - course, encja słaba - section



5.13.2. Atrybuty złożone

Atrybuty złożone rozbijamy na mniejsze, atomowe części i dla każdej tworzymy nową kolumnę.

Złożone i wielowartościowe atrybuty

instructor
<code>ID</code>
<code>name</code>
<code>first_name</code>
<code>middle_initial</code>
<code>last_name</code>
<code>address</code>
<code>street</code>
<code>street_number</code>
<code>street_name</code>
<code>apt_number</code>
<code>city</code>
<code>state</code>
<code>zip</code>
<code>{ phone_number }</code>
<code>date_of_birth</code>
<code>age()</code>

- Atrybuty złożone są rozbijane poprzez stworzenie osobnego atrybutu dla każdego składowego atrybutu
- Ignorowanie wielowartościowych atrybutów
- Rozszerzony schemat:
 - `instructor(ID, first_name, middle_initial, last_name, street_number, street_name, apt_number, city, state, zip_code, date_of_birth)`

5.13.3. Atrybuty wielowartościowe

Atrybuty wielowartościowe umieszczamy w osobnej tabeli, której kolumny to klucz główny tabeli, która miała atrybut wielowartościowy oraz kolumna na nazwę atrybutu wielowartościowego. Każdą wartość wpisujemy do osobnej krotki.

Wielowartościowe atrybuty

- Wielowartościowy atrybut M encji E jest reprezentowany przez osobny schemat EM
 - Schemat EM ma atrybuty odpowiadające kluczowi głównemu z E i atrybutowi odpowiadającemu wielowartościowemu atrybutowi M
 - Każda wartość wielowartościowego atrybutu mapuje się do osobnej krotki relacji o schemacie EM

5.13.4. Atrybuty pochodne

Atrybuty pochodne odpowiadają procedurom bezargumentowym, wiec **nie trzymamy ich w tabeli** tylko tworzymy procedurę dla ich wyliczania. Procedurę taką można włączyć do widoku.

5.14. [E2017] Zamiana związków ternarnych na binarne

5.14.1. Zagadnienia związane z projektowaniem Binarne vs nie-binarne związki

Każdy n-arny zbiór związków można zaprezentować jako pewną ilość binarnych

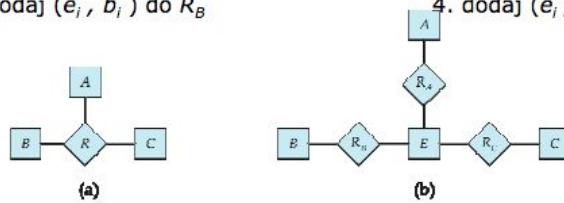
Pewne zależności nie-binarne lepiej zaprezentować jako binarne

Konieczność odwzorowania ograniczeń

- Konwersja wszystkich ograniczeń może nie być możliwa
- Mogą być instancje w przekształconym schemacie, które nie odpowiadają żadnej instancji w R



- Każdy nie-binarny związek może być zaprezentowany przy użyciu binarnych przez stworzenie sztucznego zbioru encji.
 - Zamień R między zbiorami encji A , B , C na zbiór encji E i utwórz trzy zbiory związków:
 1. R_A , odpowiadający E i A
 2. R_B , odpowiadający E i B
 3. R_C , odpowiadający E i C
 - Stwórz specjalny identyfikujący atrybut dla E
 - Dodaj wszystkie atrybuty z R do E
 - Dla każdego związku (a_i, b_i, c_i) w R , utwórz
 1. nową encję e_i w zbiorze związków E
 2. dodaj (e_i, a_i) do R_A
 3. dodaj (e_i, b_i) do R_B
 4. dodaj (e_i, c_i) do R_C

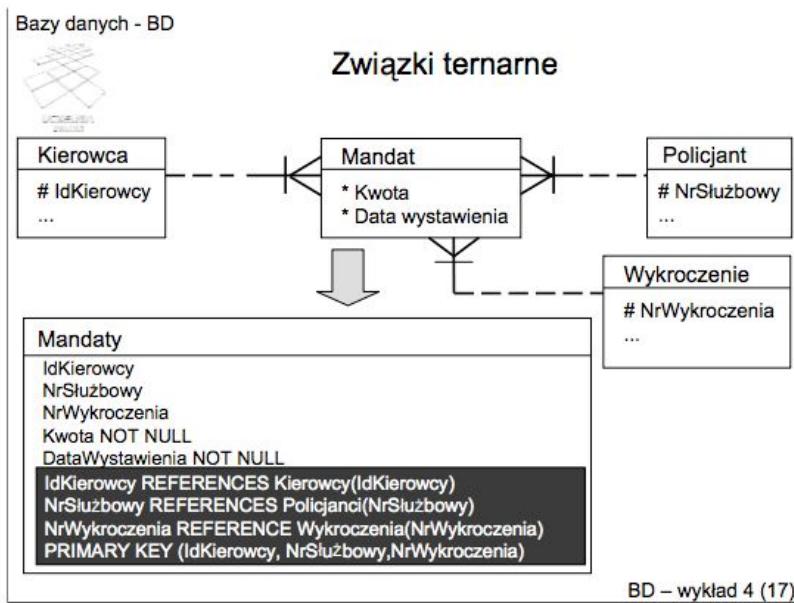


5.14.2. Przykład

Związek ternarny transformuje się w sposób identyczny jak związek 1:M. W przykładzie ze slajdu, z encji Mandat powstaje tabela Mandaty. Zawiera ona 3 klucze obce: IdKierowcy, NrSłużbowy, NrWykroczenia wskazujące odpowiednio na: IdKierowcy w tabeli Kierowcy, NrSłużbowy w tabeli Policjanci, NrWykroczenia w tabeli Wykroczenia.

Te trzy klucze obce wchodzą w skład klucza podstawowego tabeli Mandaty, ponieważ

transformowana **encja** Mandat była słabą.



// emmm, ale ten przykład jest niedokw w modelu ER i sposób, w jaki każdy z nich jest przechowywany w schemacie.

UWAGA: Niewymienienie jednego z nich często sończony bo związek dalej jest ternarny
// albo trzeba zrobić 3 tabele na oddzielne relacje albo cała ta idea jest kompletnie bez sensu

// ^ to znaczy to ze strony 48?

// chyba ze ja czegos nie rozumiem, bo czemu to rozbijac na binarne jak i tak przy przechodzeniu na model relacji dostajemy to samo

6. Wykład - Dodatkowe konstrukcje SQL

6.1. [E] Definicja, rodzaje i zastosowanie widoków

6.1.1. Widok

Dowolna relacja, która nie jest częścią modelu koncepcjonalnego, ale jest widoczna dla użytkownika jako "wirtualna relacja"

Widok dostarcza mechanizmu do ukrywania pewnych danych przed określonym użytkownikiem.

6.1.2. Zalety widoków

- Umożliwiają składowanie w bazie danych predefiniowanych zapytań
- Z pewnymi wyjątkami - niemal dowolne zapytanie typu SELECT może zostać nazwane i składowane w bazie danych w formie widoku
- Pokazują użytkownikom „właściwe dane”
- Ukrywają złożoność bazy danych
- Zapytania stają się prostsze, bardziej naturalne
- Upraszczają zarządzanie uprawnieniami

6.1.3. Rodzaje widoków:

- zmaterializowane
- niezmaterializowane

Materializowanie widoku: stworzenie fizycznej relacji zawierającej wszystkie krotki będące wynikiem zapytania definiującego widok; jeżeli relacja użyta w zapytaniu zostanie zmodyfikowana, widok zmaterializowany staje się nieaktualny.

Uwaga: Most database systems perform immediate view maintenance; that is, incremental view maintenance is performed as soon as an update occurs, as part of the updating transaction. Some database systems also support deferred view maintenance, where view maintenance is deferred to a later time; for example, updates may be collected throughout a day, and materialized views may be updated at night.

6.1.4. Wady i zalety widoków zmaterializowanych:

- zalety:
 - przydatne dla aplikacji wymagających szybkich odpowiedzi na pytania wyliczających agregaty na dużych zbiorach danych
 - dobre, gdy widok często wykorzystywany
 - szybszy dostęp do złożonych złączeń
- wady:
 - koszty wydajnościowe zarządzania widokiem
 - koszt przechowywania danych na dysku

Składnia - przykład:

```
CREATE MATERIALIZED VIEW mv_Customer  
AS SELECT * FROM Customer  
...
```

6.2. [E] Jak odświeżamy dane w widoku zmaterializowanym, typy refresh'y

Jak dużo odświeżamy:

1. **całkowite odświeżenie** czyli ponowne zmaterializowanie widoku (wyznaczenie danych i zapis na dysk). Kosztowne, ale dobre rozwiązanie jeśli dane nie muszą być aktualne. np. można dane odświeżać co noc kiedy baza nie jest używana, a analitykom wystarczą dane "z wczoraj"
2. **przyrostowo** do widoku dodawane są nowe dane na bieżąco. Dzięki temu nie ma potrzeby materializowania widoku na nowo (od zera). Czas odświeżenia może być krótki

Kiedy odświeżamy:

- **natychmiast** - tuż po modyfikacji danych
- w sposób **opóźniony**, leniwy - dopiero, gdy potrzebny jest dostęp do widoku

6.3. [E] Wstawianie do widoków/modyfikacja widoków (czego nie można używać?, "with check option") + [E2017] Dlaczego nie powinno się modyfikować widoków

6.3.1. Wstawianie/modyfikowanie widoków

Operacje wstawiania, modyfikowania usuwania rekordów nie zawsze są możliwe (np. w sytuacji gdy widok udostępnia część kolumn dwóch tabel tb_A oraz tb_B bez kolumny z kluczem głównym tabeli tb_B).

Generalnie nie należy wstawać do widoków. SQL udostępnia formalną definicję sytuacji, w których modyfikowanie widoku jest dozwolone. Widok jest modyfikalny (dozwolone są operacje **update**, **insert** i **delete**) jeśli jego definicja spełnia warunki:

- w klauzuli SELECT nie wolno aliasować atrybutów (select ID as 'numer'), nie wolno używać agregatów i distinct
- atrybuty, których nie ma w poleceniu SELECT mogą być nullami
- w klauzuli FROM może występować tylko 1 relacja - tabela (i tylko 1 raz)
- brak klauzuli GROUP BY i HAVING

Co się stanie, jeżeli wstawimy ('25566', 'Brown', 'Biology', 100000) do history_instructors?

Domyślnie SQL pozwoli na wstawienie do relacji *instructor*, ale będzie niewidoczny przez widok (nie spełnia warunków where). Żeby temu zapobiec definiujemy widok z klauzulą **with check option**.

Jeżeli widok zostanie utworzony z klauzulą **WITH CHECK OPTION**, to nie będzie można wstawić do niego żadnego wiersza, który nie byłby później widoczny w tym widoku.

Przykład z wykładu:

```
create view history_instructors as
select *
from instructor
where dept_name= 'History';
```

6.3.2. [E2017] Dlaczego nie należy wstawać do widoków

"Powiedziałam, że jak się będzie dodawać rekord do widoku, to te atrybuty, których tam nie ma, będą nullowane i jeśli nie ma tam primary key, to będzie on nullowany i to będzie problem, pewnie też o atrybutach z różnych tabel można powiedzieć."

Uwaga: Primary key nie może być nullowany. "Klucz główny nie może zawierać wartości null - gwarantowane przez SZBD (System zarządzania bazą danych)"

Więcej w poprzednim podpunkcie.

6.4. [E2017] Jakie mogą wystąpić problemy przy usuwaniu widoku?

[E2017] Dlaczego nie należy usuwać widoków

Na podstawie jednego widoku można tworzyć inne obiekty - wówczas przy usuwaniu widoku, takiego widoku może wystąpić problem. Poza tym z widoku mogą korzystać aplikacje oparte na bazie danych. Usunięcie takiego widoku po wdrożeniu systemu może powodować konieczność wprowadzenia zmian w takich aplikacjach.

Składnia PostgreSQL:

```
DROP VIEW name [, ...] [ CASCADE | RESTRICT ]
```

Cascade - usunięcie kaskadowe zależności

Restrict - zablokowanie możliwości usuwania w przypadku wykrycia zależności innego widoku od kasowanego.

// a nie "usuwaniu **Z** widoku"? nie

6.5. [E] Co to jest indeks?

6.5.1. Indeks

Jest to struktura danych wykorzystywana w celu przyspieszenia dostępu do rekordów o określonych wartościach dla atrybutów indeksu.

6.5.2. Tworzenie indeksu (wykład 6):

```
create table student
(ID varchar (5),
name varchar (20) not null,
dept_name varchar (20),
```

```
tot_cred numeric (3,0) default 0,  
primary key (ID))  
  
create index studentID_index on student(ID)
```

np.

```
select *  
from student  
where ID = '12345'
```

6.5.3. [Dodatkowo] Więcej informacji

Indeks w bazie danych jest odpowiednikiem spisu treści w książce. Po co kartkować całą książkę dla znalezienia jednej interesującej nas informacji, jeśli możemy zatrzymać się na stronie, na której znajduje się to, czego szukamy? Zaoszczędzimy w ten sposób cenny czas choćby dlatego, że spis treści jest zwykle zorganizowany w sposób alfabetyczny, co znacznie upraszcza wyszukiwanie frazy, która nas interesuje. // chyba chodzi o indeks w książce, a nie o spis treści - spis treści jest odpowiednikiem indeksu klastrowego (kolejność przechowywania), a indeks jest odpowiednikiem indeksu nieklastrowego (wskaźniki)

Indeks w bazie danych wykorzystuje się przy zapytaniach typu SQL (**SELECT**), które mają na celu wyszukiwanie odpowiednich wartości w bazie danych. Podczas realizowania zapytania optymalizator (SQL Server) najpierw przeszukuje indeks, który jest uporządkowany, a następnie na podstawie indeksu odczytuje odpowiednie rekordy. Indeks posiada strukturę logiczną i fizyczną niezależną od tabeli, do jakiej się odwołuje. Posiada również własną przestrzeń dyskową oraz jest automatycznie utrzymywany przez system zarządzania bazą danych.

Definicja tworzenia indeksu:

```
CREATE INDEX name_index ON user (name);  
lub  
ALTER TABLE user ADD INDEX (name);
```

Indeks może być CLUSTERED albo NONCLUSTERED.

CLUSTERED - tylko jeden na tabelę i tylko jedna kolumna, sortuje dane po tej kolumnie na której jest indeks - więc zmienia fizyczny rozkład danych. Najczęściej stosowane na PK.

NONCLUSTERED - dowolna ilość na tabelę, może dotyczyć jednej lub wielu kolumn, nie zmienia sposobu składowania danych, tworzy jakąś strukturę (najprawdopodobniej B+tree).

6.6. [E] Czym są warunki integralnościowe?

Warunki integralnościowe, reguły poprawności danych - zabezpieczają bazę danych przed przypadkowym uszkodzeniem, poprzez zapewnienie, że zmiany wprowadzone przez autoryzowanych użytkowników nie spowodują utraty spójności danych.

Można je definiować na poziomie pojedynczego atrybutu lub całej relacji.

Rodzaje:

- wartość niepusta/pusta - not null,
- klucz główny - primary key,

- klucz obcy - foreign key
- unikalność danych - unique,
- zawężenie dziedziny - check -

6.7. [E2017] Blokowanie constraintów (które można blokować) i jak się odblokowuje, po co sie to robi.

Przy dodawaniu nowego constrainta do istniejącej tabeli, SZBD automatycznie sprawdza, czy istniejące dane spełniają warunki constrainta

6.7.1. Po co blokować

- Blokowanie sprawdzania ze względów wydajnościowych, np. przy ładowaniu dużej ilości danych
- Stare dane mają spełniać inne warunki (np. dotyczące długości kodu pocztowego) niż nowe

6.7.2. Blokowanie na istniejących danych

- Zastosowanie do constraintów **CHECK** i **FOREIGN KEY** – inne constrainty muszą być usunięte i utworzone od nowa
- Użycie opcji **WITH NOCHECK**, gdy dodawany nowy constraint
- Używany, gdy istniejące dane nie ulegną zmianie

Zamiast blokowania można zmienić istniejące dane przed dodaniem constraintów

Przykład:

```
USE Northwind
ALTER TABLE dbo.Employees
WITH NOCHECK
    ADD CONSTRAINT FK_Employees_Employees
    FOREIGN KEY (ReportsTo)
    REFERENCES dbo.Employees(EmployeeID)
```

6.7.3. Blokowanie przy ładowaniu nowych danych

Zastosowanie do constraintów **CHECK** i **FOREIGN KEY**

Używany gdy:

- Dane są zgodne z constraintami (ale chcemy przyspieszyć proces ładowania)
- Ładujesz nowe dane, które nie są zgodne z constraintami
- Potem należy uruchomić frazę modyfikującą dane zgodnie z definicją constrainta i uaktywnić constraint

Przykład:

```
USE Northwind
ALTER TABLE dbo.Employees
NOCHECK
    CONSTRAINT FK_Employees_Employees
```

```
Blokowanie wszystkich: NOCHECK CONSTRAINT ALL  
ALTER TABLE dbo.Employees CHECK CONSTRAINT FK_Employees_Employees
```

6.8. [E] Zależności referencyjne, jak można je modyfikować, kiedy się je definiuje

6.8.1. Zależność referencyjna

Gwarantuje, że wartość, która występuje w jednej relacji dla danego zestawu atrybutów występuje także dla określonego zestawu atrybutów w innej relacji.

Przykład: Jeżeli "Biology" jest nazwą departamentu, który występuje w jednej z krotek relacji instructor, to istnieje krotka w relacji departamentu dla "Biology".

Niech A będzie zbiorem atrybutów. Niech R i S będą dwoma relacjami zawierającymi atrybuty A i A jest kluczem głównym w S. Mówiąc, że A jest kluczem obcym w R jeżeli dla dowolnej wartości A występującej w R te wartości występują również w S.

6.8.2. Warunek zależności referencyjnej

Dane relacje r1 i r2 o zbiorach atrybutów odpowiednio R1 i R2 i PK K1 i K2. Podzbiór α zbioru R2 jest kluczem obcym odwołującym się do K1 w relacji r1, to dla każdej krotki t2 w r2 musi istnieć krotka t1 w r1, taka że $t1.K1=t2.α$.

6.8.3. Integralność referencyjna (to samo co zależność referencyjna?) wiele do

Referencja - zgodność wartości klucza głównego jednej relacji z wartościami klucza obcego innej relacji.

Integralność referencyjna pomaga zapewnić poprawność danych, pozwala uniknąć przypadkowego usunięcia powiązanych danych.

Referencje (związki) - zależności między relacjami określające wzajemne powiązania.

Typy:

- Jednojednoznaczne (jeden do jeden),
- Jednoznaczne (wiele do jeden lub jeden do wiele),
- Wieloznaczne (wiele do wiele). E2017

6.9. [E] Usuwanie kaskadowe w zależnościach referencyjnych

6.9.1. Z poprzedniego opracowania

Przy tworzeniu tabeli można zdefiniować klucz obcy jako "on delete cascade".

W momencie usunięcia krotki, która była kluczem obcym dla innej z klauzulą "On delete cascade" ta krotka jest usuwana automatycznie.

6.9.2. Usuwanie wierszy z tabeli powiązanej

Usunięcie wiersza tabeli klucza głównego jest zabronione, dopóki nie zostaną usunięte (lub zmodyfikowane) wiersze z tabel, których wartości kluczów obcych stałyby się wskutek tej operacji nieaktualne.

Usuwanie **kaskadowe** powoduje, że usunięcie wiersza z tabeli po stronie klucza głównego powoduje automatyczne usunięcie z tabel powiązanych wszystkich wierszy, dla których wartości kluczów obcych stałyby się nieaktualne.

Z wykładu: Akcje kaskadowe w zależności referencyjnej

Przy naruszeniu zależności referencyjnej standardowo wycofywana akcja powodująca naruszenie

```
create table course (
    course_id char(5) primary key,
    title varchar(20),
    dept_name varchar(20) references department)
```

Ale definiując **foreign key** można wskazać aby zamiast odrzucenia akcji system podjął kroki przywracające constraint

```
create table course (
    ...
    dept_name varchar(20),
    foreign key (dept_name) references department
        on delete cascade
        on update cascade,
    ...
)
```

Przy użyciu **on delete cascade** w momencie usuwanie krotki department system nie wycofa akcji powodującej naruszenie. Zamiast tego usuwanie propaguje się do relacji course - usuwana jest krotka z course, która odwoływała się do usuwanej krotki z department.

Alternatywne działania do kaskady: set null, set default

```
ON {DELETE | UPDATE} { NO ACTION | CASCADE | SET NULL | SET DEFAULT }
```

6.10. [E2017] Klauzula With

6.10.1. Klauzula WITH w podzapytaniu

Klauzula WITH pozwala na bardziej przejrzyste budowanie kodu. Pozwala podzapytaniu nadać nazwę i dzięki temu możemy później odwoływać się do niego poprzez tę nazwę, bez potrzeby wielokrotnego wstawiania tego samego kodu.

Klauzula with jest przydatna do pisania złożonych zapytań, wspierana przez większość systemów bazodanowych, z drobnymi różnicami syntaktycznymi.

6.10.2. Przykład:

Bez With:

```
select last_name,department_name from employees
join (
    select * from departments join locations
    using(location_id) where city='Seattle'
)
using(department_id);
```

```
with departamenty_seattle as (
    select * from departments join locations
    using(location_id) where city='Seattle'
)
select last_name,department_name from employees
join departamenty_seattle using(department_id);
```

Z użyciem with (większa czytelność):

6.10.3. [E] O co chodzi z klauzulą with w procedurze?

Definiuje wartość, którą można użyć w procedurze. Przykład:

```
with max_budget (value) as (select max(budget) from department)
```

6.11. [E] Co to jest procedura skalarna?

Procedura zwracająca skalar.

6.11.1. [Dodatkowo] Podzapytanie skalarne

Jest to podzapytanie zwracające jeden atrybut. Skalarne podzapytanie mogą wystąpić w klauzulach select, where i having. Przykład:

```
select dept name,
       (select count(*)  <-podzapytanie skalarne
        from instructor
        where department.dept name = instructor.dept name)
       as num instructors
  from department;
```

Technicznie **podzapytanie zawsze zwraca relację** (tabelę), ale jeśli podzapytanie skalarne jest używane w wyrażeniu, które oczekuje wartości, SQL niejawnie wydobędzie i zwróci tę wartość z pojedynczego atrybutu pojedynczej relacji.

6.12. [E] Co to jest procedura składowana? Zalety? Wady?

Procedura składowana jest umiejscowiona bezpośrednio w systemie bazy danych, a nie po stronie klienta.

Cechy:

- przechowywane w bazie danych jako część schematu
- można używać w zapytaniach SQL

Zalety:

- zmniejszenie liczby kroków wymiany danych pomiędzy klientem a systemem zarządzania bazą danych, co może przyczynić się do wzrostu wydajności systemu
- bardziej przejrzysty interfejs pomiędzy bazą danych a aplikacjami z niej korzystającymi
- pozwalają zewnętrznym aplikacjom na operowanie na bazie danych bez znajomości wewnętrznych szczegółów
- pozwalają, aby logika biznesowa była przechowywana w bazie danych i uruchamiana z poziomu wyrażenia SQL

Wady:

- W bazie danych istnieje jedna instancja danej procedury. Jej zmiana wpływa natychmiast na działanie wszystkich elementów systemu (w bazie i poza nią), które z danej procedury korzystają. Prześledzenie wszystkich tego typu zależności jest trudne i wręcz nie zawsze możliwe.
- procedury składowane stosowane są jako metoda nadawania uprawnień do bazy danych. Jednak najczęściej użytkownikami bazy danych są programiści, którzy pozbawieni możliwości wykonania właściwych zapytań, zmuszeni są stosować udostępnione im procedury, które często nie są zoptymalizowane do realizacji celu stawianego przed aplikacją.
- Zysk wydajnościowy płynący ze stosowania procedur składowanych jest również ograniczony, gdyż dynamiczny SQL (we współpracy z niektórymi systemami baz danych) pozwala na wykonywanie wielopoleceniowych zapytań

6.13. [E] Duże pliki danych (blob/clob)

Duże obiekty lub bardzo duże są przechowywane jako *large object*:

- **blob:** binary large object
- **clob:** character large object

Przykładowo

- book_review clob(10KB)
- image blob(10MB)
- movie blob(2GB) // nie róbcie tego nigdy w prawdziwej bazie xd (+xd)(+xd)(+xd)(+turbo XD)

6.13.1. Blob

Blob (ang. binary large object) – typ danych, który umożliwia przechowywanie dużych ilości danych binarnych jako pojedynczy obiekt w bazie danych, stosowany w szczególności do przechowywania danych multimedialnych, takich jak grafika, muzyka czy filmy.

Ze względu na rozmiar i wielką różnorodność typów danych binarnych bloby są zwykle traktowane odmiennie przez systemy baz danych niż inne typy, np. **nie są wyświetlane przez polecenie SELECT** języka SQL.

6.13.2. Clob

Dane typu CLOB różnią się od danych typu BLOB tym, że posiadają określone kodowanie znaków.

CLOB – kolekcja danych znakowych w systemach zarządzania bazą danych. Obiekty CLOB są najczęściej przechowywane w wydzielonym miejscu, do którego odwołanie znajduje się w tabeli. Przykładowo systemy zarządzania bazą danych Oracle oraz DB2 dostarczają typu nazwanego wprost CLOB. Wiele innych systemów zarządzania bazą danych obsługuje w jakiejś formie ten typ danych często oznaczany jako text lub też long character.

6.14. [E2017] Co zwraca zapytanie o clob/blob?

database_system_concepts_6th: 4.5.4 Large-Object Types:

For result tuples containing large objects (multiple megabytes to gigabytes), it is inefficient or impractical to retrieve an entire large object into memory. Instead, an **application would usually use an SQL query to retrieve a “locator” for a large object** and then use the locator to manipulate the object from the host language in which the application itself is written.

For instance, the JDBC application programKur interface permits a locator to be fetched instead of the entire large object; the locator can then be used to fetch the large object in small pieces, rather than all at once, much like reading data from an operating system file using a read function call.

6.15. [E] Różnice między własnymi typami, a zawęźnieniem dziedziny

Przetłumaczone z: database_system_concepts_6th: SQL Data Types and Schemas, 4.5.5 User-Defined Types

6.15.1. [E] Różnice pomiędzy typami, a dziedzinami

- **Dziedziny** mogą posiadać ograniczenia takie jak NOT NULL, wartości domyślne zdefiniowane dla zmiennych typu tej dziedziny. Podczas, gdy **typy użytkownika** nie mogą mieć zdefiniowanych ograniczeń lub wartości domyślnych. Typy użytkownika zostały zaprojektowane nie tylko do określenia typów atrybutów, ale również do użytku w rozszerzeniach proceduralnych do SQL (gdzie wymuszenie ograniczeń może być niemożliwe)

- **Dziedziny nie są silnie typowane**, przez co wartości jednej dziedziny mogą być przypisane do wartości drugiej dziedziny, jeśli ich typy podstawowe są kompatybilne.
 - Np będąemy w stanie przypisać wartości typu Dollars do zmiennej typu Pounds (w przeciwieństwie do typów użytkownika - **silnie typowanych**, gdzie wystąpiłby błąd komilacji)
 - **create type Dollars as numeric (12,2) final** - typ użytkownika
 - **create type Pounds as numeric (12,2) final** - typ użytkownika

Poniżej znajduje się więcej informacji o typach własnych i dziedzinach.

6.15.2. [Dodatkowo] Typy własne - typy zdefiniowane przez użytkownika - User-Defined Types

6.15.2.1. Structured data types

Złożone typy z zagnieżdżoną strukturą rekordów, tablicami i wielozbiorami.

6.15.2.2. Distinct types

Pomagają w wychwytywaniu błędów.

np. tworzymy 2 następujące typy: Dollars i Pounds

- **create type Dollars as numeric (12,2) final;**
- **create type Pounds as numeric (12,2) final;**
- (klauzula final nie jest za bardzo znacząca w tym kontekście, ale wymagane przez SQL 1999)

Nowo stworzone typy mogą zostać użyte jako typy atrybutów w relacjach (tabelach). Np. możemy zadeklarować tabelę department wykorzystując typ własny Dollars:

```
create table department
```

```
(dept_name varchar (20),
building varchar (15),
budget Dollars);
cast (department.budget to numeric(12,2))
```

Próba przypisania wartości typu Dollars do zmiennej typu Pounds powoduje **błąd komplikacji**, mimo, że obydwa typy są zdefiniowane jako *numeric (12,2)*. Tego typu przypisanie jest najczęściej spowodowane błędem programisty. Deklarowanie różnych typów dla różnych walut pomaga w **wychwytywaniu takich błędów**.

Ze względu na *strong type checking* wyrażenie *(department.budget+20)* nie zostanie zaakceptowane, ponieważ atrybut *budget* i *integer constant 20* mają różne typy.

SQL zawiera klauzule **drop type** i **alter type**, które usuwają lub modyfikują typy które zostały stworzone wcześniej.

6.15.3. [Dodatkowo] Dziedziny, domains

Przed typami własnymi (dodanymi w SQL 1999) SQL posiadał podobne, ale jednak nieco inne podejście zwane dziedzinami (domains) - dodane w SQL 92.

Dziedziny mogły nakładać warunki integralnościowe na podstawowe typy. Np. mogliśmy zdefiniować dziedzinę DDollars w ten sposób:

```
create domain DDollars as numeric(12,2) not null;
```

Domena DDollars może zostać użyta tak samo jak typ Dollars, ale występują tutaj 2 **różnice pomiędzy typami a dziedzinami** (opisane na początku tego paragrafu).

6.15.4. [Dodatkowo] Wsparcie dla typów i dziedzin

- **PostgreSQL** – wspiera **create domain**, a **create type** – inna składnia i interpretacja
- **Microsoft SQL Server** - implementuje wersję **create type**, która wspiera dziedzinowe ograniczenia (podobne do **create domain**)
- **IBM DB2 IBM DB2** – wspiera wersję **create type (create distinct type)**, nie wspiera **create domain**
- **Oracle** – nie wspiera

6.15.5. Poprzednie opracowanie

Cieźko coś znaleźć na ten temat (slajd 77 basic_intermediate_sql daje tylko przykład, który po napisaniu w mssql BEZ słowa FINAL nie rzuca błędem).

Wydaje mi się, że może chodzić o to, że więzy check możemy nazywać, dzięki czemu w dowolnym momencie można zlikwidować ograniczenie dziedziny lub je zmodyfikować*, a w przypadku własnego typu trzeba by zmienić całą jego definicję.

* modyfikacja odbywa się poprzez usunięcie i ponowne dodanie nowej definicji.

np. create constraint kwota_nieujemna check (kwota >= 0) < w definicji tabeli lub jak niżej alter table tabela drop constraint kwota_nieujemna

alter table tabela create constraint kwota_nieujemna check (kwota > 100)

// czy o to chodzi?

Zapytałem mniej więcej o to Zygmunt:

Nie rozumiem wiec co oferuje nam definiowanie własnych typów czego nie oferuje nam zawężanie dziedziny przy użyciu więzów check/(not) null/pk/fk/unique?

np do definiowania proceduralnych rozszerzeń gdzie nie można wymusić ograniczeń (było szczegółowo omawiane na wykładzie)

6.16. [E] Czym jest rozszerzenie tabeli

Przetłumaczone z: database_system_concepts_6th: SQL Data Types and Schemas, 4.5.6 Create Table Extensions

// Całe pojęcie jest wg. mnie błędem tłumacza, a w oryginalu chodziło o "rozszerzenia polecenia create table". Niemniej Zygmunt używa tego pojęcia, więc trzeba je znać.

6.16.1. Tworzenie rozszerzeń tabeli

Przy złożonych zapytaniach wygodnie jest przechowywać wynik zapytania jako nową tabelę (tabelę chwilową [temporary]). Tabela zawierająca wyniki zapytania:

```
CREATE TABLE temp AS (SELECT ...) with data;  
lub  
create table temp_instructor like instructor;
```

Powyższe wyrażenie tworzy nową tabelę temp_instructor, która ma taki sam schemat jak tabela instructor.

Tworzenie tabeli zawierającej wyniki zapytania (SQL:2003):

```
create table t1 as  
  (select *  
   from instructor  
   where dept_name = 'Music')  
   with data;
```

Domyślnie nazwy i typy kolumn są wnioskowane na podstawie wyniku zapytania. Nazwy kolumn mogą być jawnie podane poprzez wylistowanie ich po nazwie relacji (tabeli).

Standard SQL:2003 mówi, że jeśli klauzula **with data** zostanie pominięta, tabela zostanie stworzona, ale nie zostanie wypełniona danymi. Jednak wiele implementacji wypełnia tabelę nawet jeśli klauzula **with data** zostanie pominięta.

6.16.2. Rozszerzenie tabeli vs widok

Powyższe wyrażenie *create table ... as* przypomina wyrażenie *create view*, obydwa są zdefiniowane poprzez zapytania. Najważniejszą różnicą jest jednak to, że:

- **zawartość rozszerzonej tabeli jest umieszczana w momencie tworzenia tabeli**, podczas gdy
- **zawartość widoku zawsze odzwierciedla aktualny wynik zapytania**.

6.16.3. Rozszerzenie tabeli vs zmaterializowany widok

Wydaje mi się, że rozszerzenie tabeli będzie trzymało stan rozszerzanej tabeli tylko i wyłącznie z momentu utworzenia tej tabeli a przy widoku zmaterializowanym można ustalić jak się ma odświeżać. (zmaterializowane widoki mogą też przechowywać dane z kilku tabel). Jest też znacząca różnica w uprawnieniach jakie dostaje twórca tabeli i twórcy widoku.

TODO

6.17. [E] Autoryzacje/Uprawnienia na widokach

6.17.1. Uprawnienia na widokach

Twórca widoku niekoniecznie dostaje do niego wszystkie uprawnienia. Dostaje tylko te uprawnienia, które nie wymagają żadnych dodatkowych autoryzacji poza te, które ma aktualnie.

Np. użytkownik, który tworzy widok nie może mieć uprawnienia do modyfikacji (**update**) na widoku, bez uprawnienia do modyfikacji (**update**) relacji (tabeli), które zostały użyte do zdefiniowania tego widoku. Do stworzenia widoku konieczne jest, aby // aby co? // ABY CO!?! // co jest surykatki?!

Przyznawanie uprawnień do widoku:

grant select on <nazwa widoku> to <grupa uzytkownikow>

Np.

```
create view geo_instructor as
(select *
from instructor
where dept_name = 'Geology');
```

Przyznanie uprawnień:

grant select on geo_instructor to geo_staff

6.17.2. Formy autoryzacji (uprawnienia) na częściach bazy danych :

- **Select**: wymagane do odczytu krotek z relacji
- **Insert** – możliwość wstawiania krotki (można podać atrybuty, reszta na null lub default)
- **Update** – pozwala na modyfikacje, ale nie na usuwanie danych. Możliwość aktualniania dowolnej krotki (wszystkich lub wybranych atrybutów)
- **Delete** – pozwala na usuwanie danych. Możliwość usuwania krotki
- **all privileges**: krótka forma dla wszystkich możliwych uprawnień

6.18. [E] Po co są role i jak je definiujemy?

Definiujemy je poprzez "create role", służą do łatwiejszego autoryzowania użytkowników do wykonywania pewnych operacji w bazie, np. po utworzeniu roli admin można zrobić

"**grant <rodzaj operacji> to admin**"

6.19. [E] Kto ma uprawnienia do modyfikacji schematu?

6.19.1. Autoryzacja na schematach

W standardzie SQL są prymitywne mechanizmy autoryzacji schematów: **Tylko właściciel schematu** może przeprowadzać modyfikacje schematu, takie jak:

- dodawanie/usuwanie relacji,
- dodawanie/usuwanie atrybutów relacji (kolumn tabeli)
- dodawanie/usuwanie indeksów

Tylko właściciel może modyfikować schemat. (NIE administrator tylko właściciel)

6.20. [E] Formy autoryzacji do modyfikowania schematu bazy

Index tworzenie i usuwanie indeksów

Resources tworzenie nowych relacji

Alteration dodawanie i usuwanie atrybutów w relacji

Drop usuwanie relacji

6.21. [E] REFERENCES permission, ALL permission

6.21.1. Opracowanie z zeszłego roku

Uprawnienie references na tabeli daje możliwość deklarowania kluczy obcych do tej tabeli **GRANT references ON <table> TO <grupa użytkowników>** (np. public dla wszystkich)

Chcąc natomiast nadać komuś wszystkie uprawnienia używamy konstrukcji **GRANT all privileges ...**

6.21.2. Autoryzacja na schematach c.d.

Uprawnienie references do deklarowania kluczy obcych:

grant reference (dept_name) on department to Mariano

Powyższe wyrażenie **grant** pozwala użytkownikowi Mariano na tworzenie relacji, które odnoszą się do klucza *dept_name* relacji (*tabeli*) *department* jako klucz obcy.

Dalszy opis przykładu:

Initially, it may appear that there is no reason ever to prevent users from creating foreign keys referencing another relation. However, recall that foreign key constraints restrict deletion and update operations on the referenced relation. Suppose Mariano creates a foreign key in a relation *r* referencing the *dept_name* attribute of the *department* relation and then inserts a tuple into *r* pertaining to the Geology department. It is no longer possible to delete the Geology department from the *department* relation without also modifying relation *r*. Thus, the definition of a foreign key by Mariano restricts future activity by other users; therefore, there is a need for the references privilege.

Continuing to use the example of the *department* relation, the references privilege on *department* is also required to create a check constraint on a relation *r* if the constraint has a subquery referencing

department. This is reasonable for the same reason as the one we gave for foreign-key constraints; a check constraint that references a relation limits potential updates to that relation.

7. Wykład - Procedury,

7.1. [E] 2 sposoby dostępu do bazy danych pod względem języków programowania

Obszerniej wyjaśnione w następnym podpunkcie

Chodzi o osadzony i dynamiczny SQL, trzeba wspomnieć, że w dynamicznym zapytaniu są wysyłane podczas wykonywania, a w osadzonym zapytanie jest optymalizowane (nawet przed procesem komplikacji), tworzony jest plan wykonania tego zapytania - a nie jest wykonywane. "Embedded SQL is SQL statements written within an application programming language such as C. These statements are preprocessed by a SQL preprocessor, which is database dependent, before the application is compiled. In the preprocessing stage, the database creates the access plan for each SQL statement. During this time, the SQL was embedded and, typically, always static." - źródło

<http://www.informit.com/articles/article.aspx?p=1330223&seqNum=2>

7.2. [E] Różnica między SQL dynamicznym i osadzonym

Z technicznego punktu widzenia, SQL jest podjęzykiem danych. Oznacza to, że jest on wykorzystywany wyłącznie do komunikacji z bazą danych. Nie posiada on cech pozwalających na tworzenie kompletnych programów. Jego wykorzystanie może być trojakie i z tego względu wyróżnia się trzy formy SQL-a: SQL

- Interakcyjny,
- Statyczny kod SQL
 - Osadzony SQL
 - Język modułów
- Dynamiczny kod SQL

7.2.1. Osadzony SQL:

Embedded SQL (Osadzony SQL)

Z wykładu:

Standard SQL definiuje wbudowanie SQL w różne języki programowania: C, C++, Java, Pascal, Fortran, PL/1, i inne. Jest to język, w którym zapytania SQL są wbudowane jest określony jako język macierzysty, a struktury SQL dozwolone w języku macierzystym tworzą osadzony SQL.

Cechy Osadzonego SQL:

- Wyrażenie **EXEC SQL** używanie jest do identyfikacji żądania osadzonego SQL do preprocesora
 - **EXEC SQL <polecenie> END_EXEC**
choć składnia może się różnić np. EXECSQL w COBOLu
- Przed wykonaniem dowolnego wyrażenia SQL, program musi się najpierw połączyć z bazą danych:
 - **EXEC SQL connect to SERWER user UZYTKOWNIK using HASŁO**

- W osadzonym SQL mogą być użyte zmienne języka macierzystego, zmienne muszą być deklarowane w sekcji DECLARE:
 - EXEC SQL BEGIN DECLARE SECTION;
int credit-amount ;
EXEC SQL END DECLARE SECTION;
- Każdą instrukcję SQL, która nie zwraca wyniku można osadzić w programie w języku macierzystym
- Ze względu na niezgodność impedancji nie można bezpośrednio osadzić **SELECT-FROM-WHERE**
- Zapytania są wysyłane w momencie komplikacji programu.

Mechanizmy łączenia wyników zapytań z programem w języku macierzystym:

- Jednowierszowe instrukcje SELECT
- KURSORY

Żeby pisać zapytania w osadzonym SQL deklarujemy kursor, który będzie identyfikował nasze zapytania.

Przykład Kursora z wykładowi

Osadzony SQL c.d

- Do zapisu osadzonego zapytania SQL, używa się wyrażenie:

declare c cursor for <SQL query>

zmienna c do identyfikacji zapytania

- Przykład:

```
EXEC SQL
  declare c cursor for
    select ID, name
    from student
    where tot_cred > :credit_amount
  END_EXEC
```

Instrukcje osadzonego SQL

- **Instrukcja open** EXEC SQL open c;
- **Instrukcja fetch** EXEC SQL fetch c into :si, :sn;
 - Instrukcja FETCH pobiera kolejno wiersze z aktywnego zbioru. Za każdym razem wykonywania instrukcji FETCH kursor jest przesuwany do następnego wiersza aktywnego zbioru.
 - Pojedynczy fetch zwraca 1 krotkę (1 wiersz)
- **Instrukcja close** EXEC SQL close c;
 - Uwaga: szczegóły zależą od języka

Modyfikacje poprzez osadzony SQL:

Modyfikacje poprzez osadzony SQL

EXEC SQL

```
declare c cursor for
  select *
    from instructor
   where dept_name = 'Music'
  for update
```

EXEC SQL

```
update instructor
  set salary = salary + 100
 where current of c
```

7.2.2. Dynamiczny SQL:

Dynamiczny kod SQL (Dynamic SQL) generowany jest w trakcie pracy aplikacji. Wykorzystuje się go w miejsce podejścia statycznego, jeżeli w chwili pisania aplikacji nie jest możliwe określenie treści potrzebnych zapytań – powstaje ona w oparciu o decyzje użytkownika. Tę formę SQL generują przede wszystkim takie narzędzia jak graficzne języki zapytań. Utworzenie odpowiedniego zapytania jest tu odpowiedzią na działania użytkownika.

Jest to API pozwalające programowi na współpracę z serwerem BD.

Aplikacja zgłasza żądanie:

- połączenia z serwerem bazodanowym
- przesyła komendy SQL (jako string) do serwera bazodanowego w czasie runtime
- pobiera krótki wynikowe jedna po drugiej do zmiennych programu

zapytania są wysyłane podczas wykonywania

Dwa standardy dynamicznego SQL: JDBC, ODBC

//dwa główne standardy, bo jest ich dużo więcej

7.2.2.1. JDBC (Java Database Connectivity)

Javowe API do komunikacji z systemami bazodanowymi wspierające SQL.

- obsługuje różne funkcje do wydawania zapytań i modyfikacji danych oraz do pobierania wyników zapytań.
- obsługuje pobieranie metadanych, takich jak zapytania dotyczące relacji w bazie danych jak również nazwy i typy atrybutów relacji.

Model komunikacji:

1. nawiąż połączenie.
2. stwórz obiekt “statement”.
3. wykonaj zapytania używając obiektu do przesyłania zapytań oraz odbierania wyników.
4. mechanizm radzenia sobie z błędami.

7.2.2.2. ODBC (Open Database Connectivity)

Otwarty standard do komunikacji programu z serwerem bazodanowym.

Pozwala na:

- otwieranie połączenia z bazą danych
- przesyłanie zapytań i modyfikacji
- otrzymywanie wyników

Aplikacje takie jak GUI, arkusze, etc. mogą używać tego samego ODBC API do połączenia do serwera bazodanowego wspierającego ODBC.

Model komunikacji:

1. nawiąż połączenie.
2. wyślij zapytania.
3. odbierz wyniki.

7.3. [E] Funkcje tablicowe

Funkcje tablicowe jako wynik zwracają tablicę. Można o nich myśleć jak o spараметryzowanych widokach.

Dają w wyniku kolekcję krotek. Mogą też przyjmować kolekcję krotek jako parametr

Konstrukcja: create function ... returns table (...),

Przykład:

```
create function instructors_of (dept_name varchar(20)
    returns table ( ID varchar(5),
                    name varchar(20),
                    dept_name varchar(20),
                    salary numeric(8,2))
    return table
        (select ID, name, dept_name, salary
         from instructor
         where instructor.dept_name = instructors_of.dept_name)
```

Użycie:

```
select *
from table (instructors_of ('Music'))
```

7.4. [E2017] Co to jest PSM - Persistent storage module?

7.4.1. Konstrukcje proceduralne

SQL wspiera konstrukcje, które dają mu podobne możliwości jakie mają języki programistyczne ogólnego przeznaczenia. Cześć standardu SQL odpowiedzialna za te konstrukcje nazywana jest **Persistent Storage Module (PSM)**. PSM pozwala na:

- Deklarowanie zmiennych: **declare**, przypisanie: **set**
- Wyrażenie złożone: **begin ... end**,
- Wyrażenia: **while** i **repeat** (SQL:1999)

7.5. [E] Konstrukcja obsługi wyjątku

7.5.1. Przykład z wykładow:

Deklaracja bloku obsługi wyjątków:

```
DECLARE <gdzie przejść> HANDLER FOR <lista warunków>
    <instrukcja>
```

```
declare out_of_classroom_seats condition
declare EXIT handler for out_of_classroom_seats
```

```
begin
sequence of statements
end
```

EXIT - wyjście z wyrażeń między begin..end

Dostępne do wyboru miejsca

- CONTINUE
- EXIT
- UNDO

The statements between the **begin** and the **end** can raise an exception by executing **signal** `out_of_classroom_seats`. The handler says that if the condition arises, the action to be taken is to **exit** the enclosing begin end statement. Alternative actions would be **continue**, which continues execution from the next statement following the one that raised the exception. In addition to explicitly defined conditions, there are also predefined conditions such as `sqlexception`, `sqlwarning`, and `not found`.

(db concepts str. 178)

7.6. [E] Case - co robi?

W SQL case rozpatruje przypadki i w każdym przypadku należy zdefiniować warunek

```

case
  when pred1 then result1
  when pred2 then result2
  ...
  when predn then resultn
  else result0
end

```

7.7. [E2017] Funkcje zewnętrzne - wady zalety, jak radzimy sobie z wadami?

7.7.1. Zewnętrzne funkcje/procedury

SQL:1999 pozwala na użycie funkcji i procedur napisanych w innych językach, takich jak Java, C#, C lub C++. Przykład deklaracji zewnętrznych procedur i funkcji (syntaktyka zależna od konkretnego DBMS):

```

create procedure dept_count_proc(in dept_name varchar(20),
                                out count integer)
language C
external name '/usr/avi/bin/dept_count_proc'

create function dept_count(dept_name varchar(20))
returns integer
language C
external name '/usr/avi/bin/dept_count'

```

7.7.2. Korzyści i wady z zewnętrznych funkcji/procedur

Korzyści:

- funkcje zewnętrzne są bardziej efektywne dla wielu operacji i większa siła ekspresji.
- programiści nie muszą uczyć się nowego języka dla każdego systemu bazodanowego

Wady:

- **Podatność na błędy:** kod do implementacji funkcji może wymagać ładowania do systemu bazy danych i wykonywany w przestrzeni adresowej systemu bazy danych. **Błąd** w programie zewnętrznym może uszkodzić wewnętrzne struktury bazy danych.
- **Problemy z bezpieczeństwem:** program zewnętrzny może obejść kontrolę dostępu bazy danych. Wykonywanie bezpośrednio w przestrzeni systemu bazodanowego wykorzystywane jest, gdy efektywność jest ważniejsza niż bezpieczeństwo.

7.7.3. Bezpieczeństwo przy zewnętrznych procedurach/funkcjach

Rozwiązywanie problemów bezpieczeństwa:

- Technika **sandbox** - procedury/funkcje zewnętrzne mają własną pamięć, ale ze względu na bezpieczeństwo nie mają dostępu do pamięci procesów i plików BD
- **Uruchomienie zewnętrznych funkcji/procedur w osobnym procesie**, bez dostępu do pamięci procesów bazodanowych - wykorzystywane w systemach, w których bezpieczeństwo jest ważniejsze niż efektywność

7.8. [E] Sandboxing

Sandbox to “bezpieczne miejsce” do wykonywania zewnętrznych skryptów (funkcji/procedur). Procedury zewnętrzne wykonywane w safe access sandbox mają własną pamięć, ale ze względów bezpieczeństwa nie mają dostępu do pamięci procesów i plików BD.

Wiele SZBD wspiera wykonywanie zewnętrznych funkcji/procedur uruchamianych w sandbox wewnątrz procesu wykonywanego zapytania

- Oracle, IBM DB2 Oracle, IBM DB2 – pozwalają funkcjom w Java być uruchamianym jako część procesu bazodanowego
- MS SQL pozwala wykonywać się procedurom (C#, VB) skompilowanym do Common Language Runtime (CLR) wewnątrz procesu bazodanowego
- PostgreSQL pozwala na funkcje pisane np. w Perl, Python, Tcl

7.9. [E] Czym jest trigger (definicja), jak się go definiuje?

7.9.1. [E] Trigger - definicja

Trigger jest wyrażeniem wykonywanym automatycznie przez system jako efekt modyfikacji bazy danych.

Uwaga: W SQL server (TSQL) mamy oprócz DML triggerów i DDL triggerów również LOGON triggery uruchamiające się na zdarzenia związane z autoryzacją.

[https://msdn.microsoft.com/pl-pl/library/ms189799\(v=sql.110\).aspx](https://msdn.microsoft.com/pl-pl/library/ms189799(v=sql.110).aspx)

Projektując mechanizm triggera należy zdefiniować:

- kiedy trigger ma się uruchomić
- jakie akcje ma wykonać trigger

Po zdefiniowaniu triggera w bazie, SZBD (System zarządzania bazą danych) odpowiedzialny za uruchomienie triggera jeżeli zajdzie określone zdarzenie i spełniony zostanie określony warunek. Triggery wprowadzono do SQL:1999, ale były już we wcześniejszych wersjach w niestandardowych rozszerzeniach w większości baz danych.

7.9.2. [E] Po co triggery

- Implementacja pewnych warunków integralnościowych
- Mechanizm do powiadamiania albo uruchamiania pewnych zadań automatycznie jeżeli zajdą pewne warunki
- Triggery nie mogą wykonywać modyfikacji poza bazą danych

7.9.3. [Dodatkowo] Cechy triggerów

- podobnie jak procedury i funkcje, są nazwanymi blokami PL/SQL zawierającymi sekcje deklaracji, wykonania i obsługi wyjątków;

- nie akceptują argumentów;
- muszą być składowane jako samodzielne obiekty w bazie danych i nie mogą występować lokalnie w bloku lub w pakiecie;
- są wykonywane niejawnie, o ile wystąpi zdarzenie wyzwalające, którym może być, np. operacja DML (INSERT, UPDATE lub DELETE) wykonywana dla tabeli lub odpowiedniego rodzaju widoków.

7.9.4. [Dodatkowo] Zdarzenia i akcje triggera

Typy triggerów/akcje triggera - Istnieje kilka typów wyzwalaczy (triggerów).

- **BEFORE** - wykonywane przed instrukcją generującą zdarzenie.
- **AFTER** są wykonane po instrukcji generującej zdarzenie.
- **[E2020] INSTEAD OF** (w niektórych SBD) - są one wykonywane zamiast instrukcji generującej zdarzenie. Często wykorzystywane do modyfikacji danych przy widokach niemodyfikowalnych (zamiast nich).

Zdarzenie powodujące wykonanie triggerów/zdarzenia triggera - Istnieją trzy typowe zdarzenia powodujące wykonanie wyzwalaczy:

- **INSERT** - dopisanie nowego rekordu do bazy danych w wyniku wykonania instrukcji INSERT,
- **UPDATE** - zmiana zawartości rekordu w wyniku wykonania instrukcji UPDATE. W tym wypadku triggery mogą być ograniczone do konkretnego atrybutu (konkretniej kolumny tabeli)
- **DELETE** - usunięcie rekordu w wyniku wykonania instrukcji DELETE.

Do wartości atrybutów przed i po można się odwoływać

- referencing old row as
- referencing new row as

7.10. [E2017] Triggery na poziomie wierszy i instrukcji

Różnica między wyzwalaczami na poziomie wierszy i instrukcji: ile razy wykona się kod triggera i w jakim momencie.

7.10.1. Wyzwalacze na poziomie wierszy

Wykonują się przed lub po modyfikacji każdego wiersza. Użycie klauzuli FOR EACH ROW oznacza typ wyzwalacza na poziomie wierszy. W innym przypadku jest to wyzwalacz poziomu instrukcji.

Jeśli na żadnym wierszu nie wykonamy instrukcji UPDATE trigger w ogóle się nie wykona, a w przypadku aktualizacji 10 wierszy, wyzwalacz typu ROW będzie uruchamiany 10 razy.

7.10.2. Wyzwalacze na poziomie polecenia

Wykonują się raz przed lub po całym poleceniu, niezależnie od tego ile wierszy zostanie zmodyfikowanych (nawet jeśli nie zmodyfikujemy żadnego wiersza trigger i tak zostanie wywołany).

Slajd z wykładu (Statement Level Triggers):

Statement Level Triggers (wyzwalacze na poziomie wiersza->instrukcji)

- **for each statement** zamiast **for each row**
- **referencing old table** lub **referencing new**
- Może być wydajniejszy gdy modyfikacje dużych ilości wierszy
- Tabele przejściowe nie mogą być użyte z triggerami **before**

Wyzwalacze na poziomie instrukcji vs na poziomie wierszy:

Wyzwalacz z poziomu instrukcji - przykład

```
CREATE OR REPLACE TRIGGER test
BEFORE INSERT ON employees
BEGIN
IF (TO_CHAR(SYSDATE,'DY') IN ('SO','N'))
THEN
RAISE_APPLICATION_ERROR(-20500,'Nie
wstawiaj do tabeli w weekend');
END IF;
END;
```

Wyzwalacz z poziomu wierszy - przykład

```
CREATE OR REPLACE TRIGGER test
BEFORE INSERT OR UPDATE ON employees
FOR EACH ROW
BEGIN
:new.job_id:=upper(:new.job_id);
END;
```

Rodzaje wyzwalaczy - podsumowanie

Kategoria	Wartości	
Instrukcja	INSERT, UPDATE, DELETE	Pojedyncze instrukcje powodujące uruchamianie wyzwalacza lub ich kombinacje
Moment wykonania	BEFORE, AFTER	Wyzwalacz jest uruchamiany przed lub po wykonaniu instrukcji,
Poziom wykonania	Wiersze lub instrukcje	Wyzwalacz działający dla wierszy uruchamia się raz dla każdego z nich użytego przez instrukcję (za jego utworzenie odpowiada klauzula FOR EACH ROW), wyzwalacze z poziomu instrukcji są uruchamiane jeden raz przed lub po jego wykonaniu,

7.11. [E] Kiedy nie używać triggerów?

7.11.1. Kiedy nie używać triggerów

Opracowanie z zeszłego roku:

- zarządzanie danymi sumarycznymi
- replikacja BD
- kaskadowe usuwanie
- gdy sprawdzenie danych można wykonać za pomocą check constraints

Ponieważ BD posiadają do trzech powyższych wbudowane mechanizmy (oraz "gdy chcemy wiedzieć co się dzieje w środku, bo nie mamy kontroli nad tym co się dzieje w triggerze" // *co* słowa Pani Doktor)

7.11.2. Problemy z triggerami (z wykładu)

- **Wydajność** - zarządzanie triggerami i ich wykonywaniem pociąga za sobą dodatkowe obciążenie systemu
- triggery przesyłają dane w momencie ich np. modyfikacji, nie wykorzystują jednak informacji o transakcyjnym charakterze wykonywanych zmian
- nie ma możliwości planowania kolejności realizacji triggerów
- błąd wykonania triggera powoduje niepowodzenie wykonania jego akcji
- **Złożoność** - akcje jednego triggera mogą powodować uruchomienie innego triggera
- Podczas debugowania łatwo o nich zapomnieć

Zalecane używanie alternatywy, np. procedur

8. Wykład - Projektowanie relacyjnych baz danych - Postaci Normalne

8.1. [E] Jak projektujemy relacyjną bazę danych?

- Zebranie wymagań użytkownika i opracowanie struktury BD spełniającej te wymagania
- Wybór modelu danych i "przetłumaczenie" tych wymagań na konceptualny model BD

Proces przechodzenia z modelu konceptualnego do implementacyjnego składa się z dwóch głównych faz:

- Logiczny projekt
- fizyczny projekt

// slajd 2 z normalizacji:

Kilka sposobów podejścia, np.:

- wysokopoziomowe notacje opisu struktury danych + sposoby konwersji wysokopoziomowych projektów na relacje
- analiza wymagań + definiowanie relacji bezpośrednio (bez wykonywania etapu pośredniego w języku wysokopoziomowym)

• Niezależnie od podejścia jest miejsce na usprawnienia – zwłaszcza poprzez eliminowanie redundancji

8.2. [E] Anomalie i rodzaje anomalii (omówić przykład z wykładu)

Anomalie: tracenie informacji, podatność na brak spójności lub niemożliwość wykonania operacji na danych spowodowane złym schematem bazy danych.

- **redundancja** (gdy dane powtarzają się, nadmiarowość danych)
- **anomalia wprowadzania danych** (np. gdy mamy tabele 'klient' i 'towary', które są reprezentowane jako jedna tabela i chcemy wstawić klienta, to musimy uzupełnić jeszcze inne pola)
- **anomalia usuwania danych** (np. usuwając klienta, musimy usunąć też powiązane w danej krotce towary)
- **anomalia aktualizacji danych** (np. aktualizując towary, musimy zrobić to dla każdego klienta)

8.2.1. Przykład 1

Dostawcy = {Nazwa_dostawcy, Adres, Nazwa_towaru, Cena}

Nazwa_dostawcy	Adres	Nazwa_towaru	Cena
Kowalski	Violinowa 7	Telewizor	1500
Kowalski	Violinowa 7	Radio	500
Jaworski	Mozarta 5	Telewizor	1800
Jaworski	Mozarta 5	Komputer	5000
Kowalski	Violinowa 7	Baterie	5
Marciniak	Warszawska 140	Magnetowid	1000

Oto zestaw wad łatwych do zidentyfikowania w tym schemacie:

- **Redundancja:** adres dostawcy powtarza się dla każdego dostarczanego towaru.
- **Anomalie przy modyfikacji:** uaktualniony adres w jednym wierszu pozostaje niezmieniony w innych.
- **Anomalie przy wstawianiu:** trudno wstawić dostawcę bez towarów; towar wchodzi w skład klucza - nie może być NULL.
- **Anomalie przy usuwaniu:** usuwając informacje o wszystkich towarach dostarczanych przez dostawcę (który może zmienić profil produkcji) usuwamy информацию o samym dostawcy.

Przyczyna: złączenie w jednej encji dwóch różnych rodzajów obiektów (encji):

- Dostawcy = {Nazwa_dostawcy, Adres}
- Towary = {Nazwa_dostawcy, Nazwa_towaru, Cena}

Poprawienie schematu polega na rozbiciu początkowego schematu na dwie tabele każda reprezentująca osobny typ obiektów czyli odpowiednio dostawców i towary.

8.2.2. Przykład 2

Pracownicy = {Id_prac, Nazwisko, Nazwa_uczelnii, Adres_uczelnii}

Id_prac	Nazwisko	Nazwa_uczelnii	Adres
101	Kowalski	PJWSTK	Koszykowa 86
123	Kalinowski	WSI	Zamiany 15
109	Jaworski	WSI	Zamiany 15
102	Makowski	PJWSTK	Koszykowa 86
105	Rudziak	WSI	Zamiany 15

Znowu możemy zaobserwować podobne wady jak poprzednio mimo, że tym razem klucz główny jest jednoelementowy.

- **Redundancja:** adres uczelni powtarza się dla każdego zatrudnionego w niej pracownika.

- **Anomalie przy modyfikacji:** uaktualniony adres uczelni w jednym wierszu pozostaje niezmieniony w innych.
- **Anomalie przy wstawianiu:** trudno wstawić uczelnię bez pracownika; Id_prac stanowi klucz i nie może być NULL.
- **Anomalie przy usuwaniu:** usuwając wszystkich pracowników usuwamy uczelnię.

Przyczyna: złączenie w jednej encji dwóch różnych rodzajów obiektów (encji):

- Pracownicy = {Id_prac, Nazwisko, Nazwa_uczelni}
- Uczelnie = {Nazwa_uczelni, Adres}

Poprawienie schematu polega na rozbiciu początkowego schematu na dwie tabele każda reprezentująca osobny typ obiektów czyli odpowiednio pracowników i uczelnie.

8.3. [Dodatkowo] Wstęp do formalnego modelu relacyjnych baz danych

Pomaga przy zrozumieniu definicji zależności funkcyjnych.

8.3.1. Wstęp

- Relacja jest abstrakcyjnym, matematycznym pojęciem zawierającym w sobie istotę modelu relacyjnego. Relacyjna baza danych to zbiór relacji.
- Tabela jest konkretną reprezentacją relacji – jedna relacja ma wiele różnych reprezentacji za pomocą tabel.
- W relacji kolejność wierszy i kolejność kolumn są nieistotne.
- Dwa wiersze w tabeli zawierające te same wartości są uznawane za identyczne to znaczy za ten sam element relacji.

8.3.2. Schemat relacji

Schematem relacji nazywamy listę:

$$R = \{A_1, A_2, \dots, A_n\}$$

gdzie A₁, A₂, ..., A_n są atrybutami (nazwami kolumn).

Na przykład,

$$\begin{aligned} \text{Loty} &= \{\text{Numer}, \text{Skąd}, \text{Dokąd}, \text{Odlot}, \text{Przylot}\} \\ \text{Pracownik} &= \{\text{Idprac}, \text{Imię}, \text{Nazwisko}, \text{Iddept}, \text{Zarobki}, \text{Stanowisko}\} \\ \text{Departament} &= \{\text{Iddept}, \text{Nazwa}, \text{Miejsce}\} \end{aligned}$$

8.3.3. Dziedzina atrybutu

Każdemu atrybutowi A przyporządkowana jest dziedzina oznaczana przez Dom(A) - zbiór dopuszczalnych wartości. Np.

$$\text{Dom}(\text{Numer}) = \text{NUMBER}(3)$$

$$\text{Dom}(\text{Skąd}) = \text{CHAR}(15)$$

$$\text{Dom}(\text{Dokąd}) = \text{CHAR}(15)$$

$\text{Dom}(\text{Odlot}) = \text{CHAR}(5)$

$\text{Dom}(\text{Przylot}) = \text{CHAR}(5)$

8.3.4. Dziedzina relacji

Dziedziną relacji o schemacie $R = \{A_1, A_2, \dots, A_n\}$ nazywamy sumę dziedzin wszystkich atrybutów relacji

$$\text{Dom}(R) = \text{Dom}(A_1) + \text{Dom}(A_2) + \dots + \text{Dom}(A_n)$$

gdzie $+$ oznacza tutaj sumę zbiorów

Wersja alternatywna: iloczyn kartezjański ? // raczej iloczyn kartezjański, co wynika z 8.3.5

Definicja relacji

8.3.5. Definicja relacji

Relacją o schemacie $R = \{A_1, A_2, \dots, A_n\}$ - co oznaczamy $r(R)$ - nazywamy skończony zbiór $r = \{t_1, t_2, \dots, t_m\}$ odwzorowana

$$t_i : R \rightarrow \text{Dom}(R)$$

takich, że dla każdego j , $1 \leq j \leq n$,

$$t_i(A_j) \text{ należy do dziedziny } \text{Dom}(A_j)$$

Każde takie odwzorowanie t nazywa się krotką (lub wierszem).

Przykład krotki (elementu relacji)

Krotka odpowiada wierszowi (rekordowi) w tabeli. Można ją formalnie określić przez podanie wartości dla poszczególnych atrybutów np.

$t(\text{Numer}) = 83$, $t(\text{Skąd}) = \text{"Warszawa"}$, $t(\text{Dokąd}) = \text{"Moskwa"}$,
 $t(\text{Odlot}) = \text{"11:30"}$, $t(\text{Przylot}) = \text{"13:43"}$

Graficznie:

Numer	Skąd	Dokąd	Odlot	Przylot
83	Warszawa	Moskwa	11:30	13:43

8.3.6. Operacja ograniczenia krotki

// coś w stylu SELECT foo, bar zamiast SELECT *

Ograniczeniem krotki t relacji r o schemacie R do zbioru atrybutów X z R nazywamy odwzorowanie będące ograniczeniem t do zbioru atrybutów X

$$t|X : X \rightarrow \text{Dom}(R)$$

to znaczy $t|X(x) = t(x)$ dla x w X a dla x w $R-X$ wartość $t|X(x)$ jest nieokreślona.

Na przykład, gdy

$$X = \{\text{Skąd, Dokąd}\},$$

to dla krotki t z poprzedniego przykładu

$$t|X(\text{Skąd}) = \text{"Warszawa"}, t|X(\text{Dokąd}) = \text{"Moskwa"}$$

Graficznie:

Skąd	Dokąd
------	-------

8.4. [E] Zależności funkcyjne

8.4.1. Zależność funkcyjna:

8.4.1.1. Z wykładu: **Zależności funkcyjne – twierdzenie uogólniające ideę kluczy relacji.**



Zależności funkcyjne – c.d.

- Niech $r(R)$ będzie schematem relacji i niech
 $\alpha \subseteq R$ i $\beta \subseteq R$
- **Zależność funkcyjna**
 $\alpha \rightarrow \beta$
zachodzi wtedy i tylko wtedy gdy dla każdej legalnej relacji $r(R)$, jeżeli dowolne dwie krotki t_1 i t_2 w r są zgodne w atrybutach α , są również zgodne w atrybutach β . Czyli,
 $t_1[\alpha] = t_2[\alpha] \Rightarrow t_1[\beta] = t_2[\beta]$
- $A \rightarrow B$ oznacza, że „jeżeli dwie krotki są zgodne na A to są zgodne na B ” (A – **wyznacznik**)

8.4.1.2. Dalszy ciąg ze strony: <http://edu.pjwstk.edu.pl/wykłady/rbd/scb/wykład5/norm.htm>

Symbole:

- A_1, A_2, \dots, An są atrybutami (nazwami kolumn).
- Relacja r (tabela) o schemacie R to skończony zbiór $r = \{t_1, t_2, \dots, t_m\}$ (t_1, t_2, \dots, t_i - krotka-wiersz)
- $t|X: X \rightarrow Dom(R)$ - Operacja ograniczenia krotki (opisana w poprzednim punkcie)

Zależność funkcyjna:

Relacja r o schemacie $R = \{A_1, A_2, \dots, An\}$ spełnia zależność funkcyjną

$X \rightarrow Y$ (X, Y - podzbiory R)

jeśli dla każdych dwóch krotek t, u relacji r zachodzi warunek:

jeśli $t|X = u|X$ to $t|Y = u|Y$

tzn. w ramach krotek relacji r wartości atrybutów zbioru X determinują jednoznacznie wartości atrybutów zbioru Y .

W przykładowej relacji Loty, wartości atrybutu Numer jednoznacznie identyfikują cały lot a więc w szczególności jednoznacznie identyfikują wartości wszystkich atrybutów tej relacji:

{Numer} \rightarrow {Skąd, Dokąd, Odlot, Przylot}

Prześledźmy jeszcze jeden przykład relacji z zależnościami funkcyjnymi między jej atrybutami mianowicie relację Znaki Zodiaku o schemacie

{Id, Imię, Nazwisko, DzienUrodzenia, ZnakZodiaku}.

Id	Imię	Nazwisko	DzienUrodzenia	ZnakZodiaku
1	Agnieszka	Kowalska	23.01	Wodnik
2	Mariusz	Malewicz	1.04	Baran
3	Krzysztof	Zalewski	23.04	Byk
4	Ilona	Zawadzka	13.04	Baran
5	Marek	Gajęcki	31.07	Icon
6	Roman	Gerlich	5.09	Panna
7	Doman	Rębski	Just 1.09	Lambda
8	Sylwia	Frymus	13.04	Baran

Mamy do czynienia z zależnością funkcyjną

{DzienUrodzenia} -> {ZnakZodiaku}

to znaczy, temu samemu dniu urodzenia odpowiada zawsze ten podanej powyżej definicji.
(Warto zauważyć, że odwrotność nie zawsze jest prawdziwa, jest to implikacja)

8.4.2. Identyfikacja zależności funkcyjnych

W procesie projektowania dla każdego schematu relacji identyfikujemy zbiór spełniających ją zależności funkcyjnych (zależny od konkretnego zastosowania).

Na przykład dla relacji Loty identyfikujemy następujący zbiór zależności funkcyjnych między jej atrybutami:

{Numer} -> {Skąd, Dokąd, Odlot, Przylot}

{Skąd, Dokąd, Odlot} -> {Numer, Przylot}

{Skąd, Dokąd, Przylot} -> {Numer, Odlot}

8.4.3. Opracowanie z zeszłego roku

- zależność funkcyjna występuje wtedy gdy po ustaleniu wartości pewnych atrybutów relacji wartości jakichś innych atrybutów tej relacji są jednoznacznie wyznaczone (unikalne)
- notacja: $X \rightarrow Y$, gdzie X i Y są zbiorami atrybutów z relacji R
- zależność funkcyjną, której prawa strona zawiera kilka atrybutów ($X \rightarrow A_1 A_2 \dots$) można zastąpić zbiorem zależności o pojedynczych prawych stronach ($X \rightarrow A_1, X \rightarrow A_2, \dots$) operacja jest odwracalna

8.5. [E2017] Definicje kluczy w kontekście zależności funkcyjnych

8.5.1. Z wykładu: sam znak zodiaku.

W rzeczywistości mamy tutaj do czynienia z czymś więcej, mianowicie ze znaną funkcją f: DzieńUrodzenia -> ZnakZodiaku przyporządkowującą dniu urodzenia znak zodiaku. Jednak ta funkcja nie daje się wyrazić za pomocą funkcyjnej zależności w sensie



Definicje kluczy w kontekście zależności funkcyjnych

- $K \subset R$ jest **superkluczem** schematu relacji $r(R)$ wtedy i tylko wtedy gdy zależność funkcyjna $K \rightarrow R$ zachodzi w $r(R)$
- K jest kluczem **kandydującym** dla $r(R)$ wtedy i tylko wtedy gdy
 - zachodzi zależność funkcyjna $K \rightarrow R$ i
 - dla żadnego $\alpha \subset K$, $\alpha \rightarrow R$
- Zależności funkcyjne pozwalają na wyrażanie ograniczeń, które nie mogłyby być wyrażone przy użyciu superkluczy

g

8.5.1. Superklucz (nadklucz)

Superkluczem relacji r o schemacie $R = \{A_1, A_2, \dots, A_n\}$ nazywamy dowolny zbiór atrybutów K z R taki, że zachodzi zależność funkcyjna $K \rightarrow R$ - inaczej mówiąc, wartość każdego atrybutu ma być jednoznacznie zdeterminowana przez wartości atrybutów zbioru K . Jednym z nadkluczy(superkluczy) jest zawsze trywialny zbiór wszystkich atrybutów R .

8.5.2. Klucz kandydujący + [E] Czym jest klucz kandydujący w kontekście zależności funkcyjnych?

Klucz kandydujący to minimalny zbiór atrybutów potrzebny do stworzenia superklucza.

Kluczem kandydującym relacji r o schemacie $R = \{A_1, A_2, \dots, A_n\}$ nazywamy każdy minimalny superklucz (nie zawierający w sobie (bez równości) żadnego innego superklucza), tzn. zbiór atrybutów X jest kluczem kandydującym, jeśli wartość każdego atrybutu w R jest jednoznacznie zdeterminowana przez wartości atrybutów zbioru X i żaden **podzbiór właściwy** zbioru X nie ma już tej własności.

Zawsze istnieje co najmniej jeden superklucz - całe R , stąd wynika, że istnieje co najmniej jeden minimalny superklucz czyli klucz kandydujący, a może być kluczy kandydujących więcej jak to pokazuje przykład relacji Loty. Zależności funkcyjne schematu Loty określają trzy klucze kandydujące:

{Numer}
 {Skąd, Dokąd, Odlot}
 {Skąd, Dokąd, Przylot}

8.5.3. [Dodatkowo] Klucze kandydujące i klucze główne

Wyróżniony klucz kandydujący nazywa się kluczem głównym. Wchodzące w jego skład atrybuty są podkreślone lub pogrubiane.

Dla relacji *Loty* wybieramy jako klucz główny klucz Numer:

Loty = {Numer, Skąd, Dokąd, Odlot, Przylot}

8.6. [Dodatkowo] Wykorzystanie zależności funkcyjnych

8.6.1. Z wykładu:

Wykorzystanie zależności funkcyjnych:

- Zależności funkcyjne pozwalają wykryć błędy w projekcie ER,
- sprawdzenie czy instancje relacji spełniają dany zbiór F zależności funkcyjnych,
- specyfikowanie ograniczeń na zbiorze **legalnych relacji** (instancja legalnej relacji spełnia wszystkie nałożone na nią zależności funkcyjne)

Wykorzystanie wielowartościowych zależności funkcyjnych:

- Wykorzystywane w 4NF

<i>ID</i>	<i>dept_name</i>	<i>street</i>	<i>city</i>
22222	Physics	North	Rye
22222	Math	Main	Manchester

Figure 8.15 An illegal r_2 relation.

nadklucz

- Do testowania relacji w celu zweryfikowania czy są legalne pod względem wielowartościowych zależności funkcyjnych

<i>ID</i>	<i>dept_name</i>	<i>street</i>	<i>city</i>
22222	Physics	North	Rye
22222	Physics	Main	Manchester
12121	Finance	Lake	Horseneck

Figure 8.14 An example of redundancy in a relation on a BCNF schema.

- Do specyfikowania ograniczeń na zbiorze legalnych relacji

Uwaga: konkretne instancje schematu relacji mogą spełniać zależności funkcyjne nawet jeżeli zależność funkcyjna nie zachodzi na wszystkich legalnych instancjach.

8.6.2. [Dodatkowo] Znaczenie zależności funkcyjnych

Zależność od czegokolwiek innego niż klucz wprowadza wewnętrzną zależność między atrybutami tabeli. Powoduje możliwość determinowania wartości jednych atrybutów przez inne (redundancję). Pokazuje to tabelka dla zależności X-> Y:

X	...	Y	
	...		+A
x	...	y	...
x	...	?	...
	...		

Jeśli X nie jest nadkluczem, to przedstawiona w tabelce sytuacja oznacza redundancję. Wartość w polu oznaczonym przez "?" jest już jednoznacznie zdeterminowana – musi to być y. Natomiast, gdy X jest nadkluczem, to przedstawiona sytuacja jest niemożliwa. Nie mogą być dwa różne wiersze z tą samą wartością klucza.

8.7. [E] Legalność relacji

8.7.1. Co to znaczy, że relacja jest legalna

Legalna instancja relacji: relacja, która spełnia wszystkie nałożone zależności funkcyjne.

ID	dept_name	street	city
22222	Physics	North	Rye
22222	Math	Main	Manchester

Figure 8.15 An illegal r_2 relation.

Przykład relacji nielegalnej:

Z wykładu:



Przykład

- Relacja r_2 :

dept_name	ID	street	city
Physics	22222	North	Rye
Physics	22222	Main	Manchester
Finance	12121	Lake	Horseneck

- Jeżeli instruktor o ID 2222 jest związany z wydziałem Physics, chcemy aby ten wydział był związany ze wszystkimi adresami tego instruktora.
- Dlatego r_2 jest nie jest legalna:

dept_name	ID	street	city
Physics	22222	North	Rye
Math	22222	Main	Manchester

- Aby była legalna, należy do niej dodać:
 - (Physics, 2222, Main, Manchester)
 - (Math, 2222, North, Rye)
- Zależność wielowartościowa $ID \rightarrow\!\!\!> street, city$ ma zachodzić

If we wish to constrain ourselves to relations on schema $r(R)$ that satisfy a set F of functional dependencies, we say that F **holds on** $r(R)$.

8.8. [E] Domknięcie zbioru zależności (F^+)

8.8.1. Domknięcie zbioru zależności funkcyjnych

Mając dany zbiór zależności funkcyjnych F na $r(R)$, są pewne inne zależności funkcyjne, które są logicznie implikowane przez zbiór F .

Dany jest schemat relacji $r(R)$ – zależność funkcyjna f na R jest logicznie implikowana przez zbiór zależności funkcyjnych F na r , jeżeli każda instancja r (R), która spełnia F , spełnia również f

Zbiór wszystkich zależności funkcyjnych logicznie wynikających ze zbioru F nazywany jest **domknięciem zbioru F** . Domknięcie F oznaczane **F^+** .

8.8.2. [E] Jak wyznaczyć domknięcie zbioru zależności $F^+?$

Podejścia:

- Aksjomaty Armstronga
- Przez użycie domknięcia atrybutów

8.8.2.1. [E] Podać 3 aksjomaty Armstronga:

Można znaleźć F^+ (domknięcie zbioru zależności funkcyjnych F), przez wielokrotne stosowanie Aksjomatów Armstronga (α - zbiór atrybutów):

1. reguły zwrotności: if $\beta \subseteq \alpha$, then $\alpha \rightarrow \beta$
2. reguła rozszerzalności: if $\alpha \rightarrow \beta$, then $\gamma\alpha \rightarrow \gamma\beta$
3. reguła przechodniości: if $\alpha \rightarrow \beta$, and $\beta \rightarrow \gamma$, then $\alpha \rightarrow \gamma$

Reguły te są: pewne i kompletne

Uwaga: γ to nowy zbiór, a $\gamma\alpha$ to suma zbiorów: $\gamma \cup \alpha$

Dodatkowe reguły, które można wyprowadzić z poprzednich:

- Unia - Jeżeli zachodzi $\alpha \rightarrow \beta$ i $\alpha \rightarrow \gamma$, to zachodzi $\alpha \rightarrow \beta \vee \gamma$
- Dekompozycja - Jeżeli zachodzi $\alpha \rightarrow \beta \gamma$, to zachodzi $\alpha \rightarrow \beta$ i $\alpha \rightarrow \gamma$
- Pseudoprzechodniość - Jeżeli zachodzi $\alpha \rightarrow \beta$ i $\gamma \beta \rightarrow \delta$, to zachodzi $\alpha \gamma \rightarrow \delta$

Procedura/algorytm wyliczania F +

Algorytm wynik := F

powtarzaj

dla każdej zależności funkcyjnej f w wyniku:

zastosuj zasady zwrotności i rozszerzania na f

dodaj wynikowe zależności funkcyjne do wyniku

dla każdej pary zależności funkcyjnych f1 i f2 w wyniku:

jeśli f1 i f2 można połączyć zasadą przechodniości

to dodaj wynikową zależność funkcyjną do wyniku

dopóki (ostatnia iteracja nie zmieniła wyniku)

zwróć wynik (domknięcie)

Z poprzedniego opracowania (Algorytm wyliczania dopełnienia ogólnie, opisowo):

8.8.2.2. [E] Wyznaczenie domknięcia zależności funkcyjnych przez użycie domknięcia atrybutów.

Dla każdego $\gamma \subseteq R$ znajdujemy domknięcie γ^+ i dla każdego $S \subseteq \gamma^+$ wypisujemy zależność $\gamma \rightarrow S$.

8.9. [Dodatkowo] Pierwsza postać normalna (1NF)

Relacja jest w pierwszej postaci normalnej, jeśli:

- opisuje jeden obiekt,
- wartości atrybutów są elementarne (atomowe, niepodzielne) – każda kolumna jest wartością skalarną (atomową), a nie macierzą lub listą czy też czymkolwiek, co posiada własną strukturę,
- nie zawiera kolekcji (powtarzających się grup informacji)
- kolejność wierszy może być dowolna (znaczenie danych nie zależy od kolejności wierszy).

Tabela przed normalizacją

Płeć	Imię

Męska	Jan, Piotr, Zenon
Żeńska	Anna, Maria, Zofia

Pierwsza postać normalna

Płeć	Imię
Męska	Jan
Męska	Piotr
Męska	Zenon
Żeńska	Anna
Żeńska	Maria
Żeńska	Zofia

Właściwości, które muszą zaistnieć w 1 formie:

1. Jest zdefiniowany klucz relacji.
2. Wszystkie atrybuty niekluczowe są w zależności funkcyjnej od klucza.

8.10. [Dodatkowo] Druga postać normalna (2NF) - nieużywana

Relacja jest w drugiej postaci normalnej wtedy i tylko wtedy, gdy jest w I postaci normalnej i żadna kolumna niekluczowa nie jest częściowo funkcyjnie zależna od jakiegokolwiek [klucza potencjalnego](#).

Przykład tabeli „Pracownicy” przed normalizacją

Imię	Nazwisko	Płeć	Stanowisko	Stawka za godzinę
Antoni	Anonim	Męska	Stolarz	10 zł
Natalia	Niewiadoma	Żeńska	Sekretarka	20 zł
Alina	Enigma	Żeńska	Sekretarka	20 zł

Klucz potencjalny składa się tu z dwóch pól: "Imię" oraz "Nazwisko". Przy założeniu, że każde imię ma przypisaną jedną płeć, czyli, że płeć zależy tylko od jednego z atrybutów klucza potencjalnego, tabela nie spełnia warunków na drugą postać normalną.

Przykład tabeli „Pracownicy” po normalizacji do 2NF

Imię	Nazwisko	Stanowisko	Stawka za godzinę
Antoni	Anonim	Stolarz	10 zł
Natalia	Niewiadoma	Sekretarka	20 zł
Alina	Enigma	Sekretarka	20 zł

Każdy atrybut niekluczowy zależy od całego klucza potencjalnego.

Nowa tabela „Płeć imienia” po normalizacji

Imię	Płeć
Antoni	Męska
Natalia	Żeńska
Alina	Żeńska

8.11. [E2017] Trzecia postać normalna (3NF)

Slajdy nieaktualne, patrz wykład 2k17/18

Zasadniczo to nowe slajdy pokazują ~to samo

3.4 Trzecia postać normalna (3NF) (1)

Definicja:

» FD: $X \rightarrow Z$ w schemacie relacji R jest **przechodnią zależnością funkcyjną**

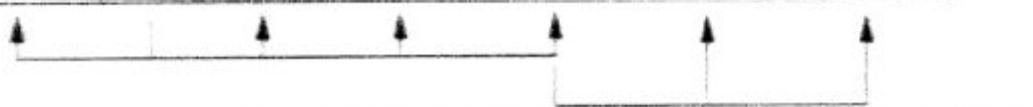
- jeżeli istnieje zbiór atrybutów Z w R, który nie jest ani kluczem kandydującym, ani podzbiorem żadnego klucza w R, że zachodzi zarówno $X \rightarrow Y$ jak i $Y \rightarrow Z$

Przykłady:

- Ssn \rightarrow Dmgr_ssn **przechodnia** FD bo zachodzą: Ssn \rightarrow Dnumber i Dnumber \rightarrow Dmgr_ssn i Dnumber nie jest ani kluczem ani nie zawiera się w kluczu
- Ssn \rightarrow Ename **nie jest przechodnia** bo nie ma żadnego zbioru atrybutów X takiego, że SSN \rightarrow X i X \rightarrow ENAME

EMP DEPT

Ename	Ssn	Bdate	Address	Dnumber	Dname	Dmgr_ssn



UWAGA: Wymagana jest dokładna definicja z wykładu (w szczególności warunek 3, który bywa różnie definiowany)

Definicja z wykładu:



Trzecia postać normalna

- Schemat relacji R jest w **trzeciej postaci normalnej (3NF)** w odniesieniu do zbioru zależności funkcyjnych F , jeżeli dla wszystkich zależności funkcyjnych:
 $\alpha \rightarrow \beta$ w F^+ (gdzie $\alpha \subseteq R$ i $\beta \subseteq R$) zachodzi **przynajmniej jeden z warunków**:
 - $\alpha \rightarrow \beta$ jest trywialną zależnością (tzn. $\beta \subseteq \alpha$)
 - α jest superkluczem w R
 - każdy atrybut A w $\beta - \alpha$ znajduje się w kluczu kandydującym dla R (*atrzybut podstawowy*).**(Uwaga:** każdy atrybut A w $\beta - \alpha$ może być w różnym kluczu kandydującym)
- Jeżeli relacja jest w BCNF jest w 3NF

Algorytm bezstratnej dekompozycji do 3NF (database_system_concepts_6th):

Figure 8.12 shows an algorithm for finding a dependency-preserving, lossless decomposition into 3NF. The set of dependencies F_c used in the algorithm is a canonical cover for F . Note that the

algorithm considers the set of schemas R_j , $j = 1, 2, \dots, i$; initially $i = 0$, and in this case the set is empty.

```

let  $F_c$  be a canonical cover for  $F$ ;
i := 0;
for each functional dependency  $\alpha \rightarrow \beta$  in  $F_c$ 
  i := i + 1;
   $R_i := \alpha \beta$ ;
if none of the schemas  $R_j$ ,  $j = 1, 2, \dots, i$  contains a candidate key for  $R$ 
  then
    i := i + 1;
     $R_i :=$  any candidate key for  $R$ ;
/* Optionally, remove redundant relations */
repeat
  if any schema  $R_j$  is contained in another schema  $R_k$ 
    then
      /* Delete  $R_j$  */
       $R_j := R_i$ ;
      i := i - 1;
  until no more  $R_j$ 's can be deleted
return ( $R_1, R_2, \dots, R_i$ )

```

Figure 8.12 Dependency-preserving, lossless decomposition into 3NF.

8.11.2. Przykład dekompozycji do 3NF:

TABLE_BOOK_DETAIL

Book ID	Genre ID	Genre Type	Price
1	1	Gardening	25.99
2	2	Sports	14.99
3	1	Gardening	10.00
4	3	Travel	12.99
5	2	Sports	17.99

8.11.2.1. Zależności w tabeli Table_book_detail

[Book ID] → [Genre ID] dozwolone, bo pkt.2 definicji ([Book ID] jest superkluczem)

[Genre ID] → [Genre Type] **niedozwolone**, bo nie spełnia:

- pkt 1. 3NF: [Genre ID] → [Genre Type] to nie jest zależność trywialna i
- pkt 2. 3NF: [Genre ID] nie jest superkluczem i
- pkt 3. 3NF: [Genre Type] nie jest atrybutem kluczem kandydującym

[Book ID] → [Genre ID] → [Genre Type] (zachodzi przechodnio): zależności przechodnie **nie są dozwolone** w 3NF

8.11.2.2. Po normalizacji 3NF:

TABLE_BOOK			TABLE_GENRE	
Book ID	Genre ID	Price	Genre ID	Genre Type
1	1	25.99	1	Gardening
2	2	14.99	2	Sports
3	1	10.00	3	Travel
4	3	12.99		
5	2	17.99		

Teraz wszystkie nie kluczowe atrybuty są funkcyjnie zależne w całości tylko na kluczu głównym.

Tabela TABLE_BOOK:

[Book ID] -> [Genre ID] **dozwolone**, bo pkt.2 definicji ([Book ID] jest superkluczem)

[Book ID] -> [Price] **dozwolone**, bo pkt.2 definicji ([Book ID] jest superkluczem)

Tabela [TABLE_GENRE]:

[Genre ID]-> [Genre Type] **dozwolone**, bo pkt.2 definicji ([Genre ID] jest superkluczem)

8.12. [E2017] Po co stworzono 3 postać normalną (3NF)?

W odróżnieniu od 3NF sprowadzenie do BCNF nie gwarantuje zachowania zależności funkcyjnych przy rozkładzie.

8.12.1. Wikipedia:

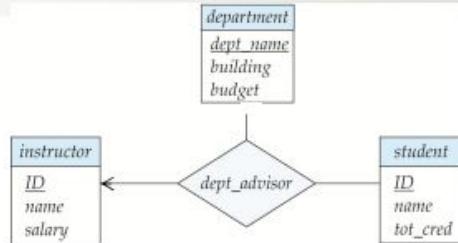
3NF was designed to improve database processing while minimizing storage costs. 3NF data modeling was ideal for online transaction processing (OLTP) applications with heavy order entry type of needs.

8.12.2. database_system_concepts_6th: Comparison of BCNF and 3NF

Zaletą 3NF jest to, że **zawsze** możliwe jest zdekomponowanie schematu do 3NF w sposób bezstratny z zachowaniem wszystkich zależności. **Wadą** 3NF jest to, że możemy być zmuszeni do użycia wartości NULL do reprezentacji niektórych bezsensownych relacji (co jest problemem z nadmiarowością danych)

8.12.3. Ze slajdów:

Wymuszenie jej trudne obliczeniowo - projekt nie zachowuje zależności -> wprowadzenie słabszej postaci normalnej 3NF



schemat: $\text{dept_advisor}(s_ID, i_ID, \text{dept_name})$
 zależności funkcyjne zachodzące na dept_advisor :

$i_ID \rightarrow \text{dept_name}$

$s_ID, \text{dept_name} \rightarrow i_ID$

po dekompozycji::

(s_ID, i_ID)

$(i_ID, \text{dept_name})$

$s_ID, \text{dept_name} \rightarrow i_ID$

Wymuszenie jej trudne obliczeniowo – projekt **nie zachowuje zależności**

Wprowadzenie słabszej postaci normalnej (3NF)

8.13. [E] Postać normalna Boyce-Codda (BCNF)

8.13.1. Definicja z wykładu + Opracowanie z zeszłego roku

8.13.1.1. Definicja BCNF z wykładu

Schemat relacji R jest w BCNF w odniesieniu do zbioru zależności funkcyjnych F jeżeli dla wszystkich zależności funkcyjnych w F+ (domknięcie zbioru F) postaci

$$\alpha \rightarrow \beta$$

gdzie $\alpha \subseteq R$ i $\beta \subseteq R$, zachodzi przynajmniej jeden z warunków:

1. $\alpha \rightarrow \beta$ jest trywialną zależnością (tzn. $\beta \subseteq \alpha$)
2. α jest superkluczem w R

8.13.1.2. Co zrobić jeżeli nie spełnia BCNF?

Z poprzedniego opracowania:

Jeżeli schemat nie w BCNF to dekomponujemy do mniejszych schematów, które będą w BCNF.

Przykład ze slajdów:

inst_dept (ID, name, salary, dept_name, building, budget) (nie jest w BCNF bo zależność funkcyjna $\text{dept_name} \rightarrow \text{budget}$

zachodzi na inst_dept, ale dept_name nie jest superkluczem) dekomponujemy do:

- instructor z zależnością funkcyjną: $ID \rightarrow name, dept_name, salary$
- department z zależnością funkcyjną: $dept_name \rightarrow building, budget$

One są już w BCNF

Ze strony apohllo.pl

Aby przekształcić relację nie spełniając warunku BCNF należy dokonać jej **dekompozycji** na szereg relacji $R_1, R_2 \dots R_n$, z których każda spełnia warunek BCNF. Poszczególne etapy dekomponowania polegają na:

1. Sprawdzeniu, czy w zbiorze otrzymanych relacji nie występuje relacja **łamiąca warunek BCNF**. Jeśli nie, to algorytm kończy działanie.
2. Dla każdej relacji łamiącej warunek BCNF należy zidentyfikować **zależność funkcyjną, która powoduje niespełnienie tego warunku** (tzn. nietrywialnej zależność postaci: $A_1 A_2 \dots A_n \rightarrow B_1 B_2 \dots B_m$, w której $A_1 A_2 \dots A_n$ nie jest superkluczem tej relacji)
3. **Zdekomponować wszystkie relacje łamiące warunek BCNF** do dwóch relacji zawierających:
 1. atrybuty $A_1 A_2 \dots A_n B_1 B_2 \dots B_m$
 2. atrybuty $A_1 A_2 \dots A_n$ oraz atrybuty nie należące do zbioru $B_1 B_2 \dots B_m$
4. Powrocie do pierwszego punktu algorytmu.

Przykład 1

Niech $R(A, B, C, D, E)$ będzie relacją, w której spełnione są zależności funkcyjne

- $A B C \rightarrow D$,
- $A \rightarrow E$.

Kluczem relacji jest zbiór atrybutów (A, B, C) . Zależność $A \rightarrow E$ łamie warunek BCNF. Relacja R powinna zostać zdekomponowana na relacje $S(A, B, C, D)$ oraz $T(A, E)$.

W relacji S zachodzi zależność funkcyjna $A B C \rightarrow D$, ponieważ $(A, B, C)^+ = (A, B, C, D)$ i D nie należy do (A, B, C) .

W relacji T zachodzi zależność funkcyjna $A \rightarrow E$, ponieważ $(A)^+ = (A, E)$ i E nie należy do (A) .

Przykład 2 (na prawdziwych danych):

Let's take a look at this table, with some typical data. The table is not in BCNF.

Author	Nationality	Book title	Genre	Number of pages
William Shakespeare	English	The Comedy of Errors	Comedy	100
Markus Winand	Austrian	SQL Performance Explained	Textbook	200
Jeffrey Ullman	American	A First Course in Database Systems	Textbook	500
Jennifer Widom	American	A First Course in Database Systems	Textbook	500

The functional dependency:

book title → genre, number of pages

is one FD violating the BCNF rules. We split our relation into two relations:

the ones in the functional dependency (book title, genre, number of pages)

the rest: (author, nationality, book title). Note that the left hand-side of the FD (book title) stays in the relation!

The example data look like this. We select the values of columns from the original relation and we eliminate the duplicate rows:

Book title	Genre	Number of pages
The Comedy of Errors	Comedy	100
SQL Performance Explained	Textbook	200
A First Course in Database Systems	Textbook	500

Author	Nationality	Book title
William Shakespeare	English	The Comedy of Errors
Markus Winand	Austrian	SQL Performance Explained
Jeffrey Ullman	American	A First Course in Database Systems
Jennifer Widom	American	A First Course in Database Systems

Are we done? Nope. The (book title, genre, number of pages) table is in BCNF. But (book title, author, nationality) isn't. We have the dependency:

author → nationality

Together with the trivial dependency

book title → book title,

the pair (book title, author) is the key of the relation.

We have to decompose the table one more time. This time we decompose into:

- columns forming the functional dependency: (author, nationality)
- the remaining columns: (author, book title)

This time every table is in BCNF.

Author	Book title
William Shakespeare	The Comedy of Errors
Markus Winand	SQL Performance Explained

Jeffrey Ullman	A First Course in Database Systems
Jennifer Widom	A First Course in Database Systems

Author	Nationality
William Shakespeare	English
Markus Winand	Austrian
Jeffrey Ullman	American
Jennifer Widom	American

The functional dependencies for this schema are the same as before:

author → nationality

book title → genre, number of pages

The key of the first table is {author}. The key of the second table is {book title}. The key of the third table is {author, book title}. There are no functional dependencies violating the BCNF rules, so the schema is in Boyce-Codd normal form.

8.13.2. Trochę bardziej opisowo ze strony

<http://edu.pjwstk.edu.pl/wyklady/rbd/scb/wyklad5/norm.htm>

8.13.2.1. Postać normalna Boyce'a-Codda

Relacja o schemacie R znajduje się w postaci normalnej Boyce'a-Codda jeśli nie zawiera zależności nie od klucza tj. dla każdej zależności X → A w schemacie relacji R (gdzie X podzbiór R, A atrybut w R) zachodzi albo

1. A należy do X (zależność trywialna), albo
2. X jest superkluczem.

Jeśli schemat relacji znajduje się w postaci normalnej Boyce'a-Codda, nie można w tabeli przewidzieć jednych wartości w oparciu o inne, chociaż jak to będzie pokazane dalej nie mamy gwarancji, że nie będzie innego rodzaju redundancji niż zależność funkcyjna.

8.13.2.2. Przykłady schematów w postaci normalnej Boyce'a-Codda:

(1) R = {Id_prac, Nazwisko, Funkcja, Stanowisko},
F: Id_prac → Nazwisko Funkcja Stanowisko

(2) R = {Numer, Skąd, Dokąd, Odlot, Przylot}
F: Numer → Skąd Dokąd Odlot Przylot
Skąd Dokąd Odlot → Numer Przylot
Skąd Dokąd Przylot → Numer Odlot

8.13.2.3. Schemat nie dający się sprowadzić do postaci normalnej Boyce'a-Codda

Nie każdy schemat tabeli da się sprowadzić do zbioru schematów tabel w postaci normalnej Boyce'a-Codda - bez utraty zawartych w tabelach informacji i z zachowaniem zależności funkcyjnych. Na przykład schematem takim jest MUK = {Miasto, Ulica, Kod} z zależnościami:

Miasto Ulica -> Kod

Kod -> Miasto

Są dwa klucze:

- {Miasto, Ulica}
- {Kod, Ulica}

Ze względu na zależność Kod -> Miasto schemat MUK nie jest w postaci normalnej Boyce'a-Codda. Tego schematu nie daje się rozłożyć z zachowaniem zależności funkcyjnych (bo jedna z zależności funkcyjnych obejmuje wszystkie atrybuty).

Atrybut kluczowy jest to atrybut wchodzący w skład jednego z kluczy tabeli.

8.14. [E2017] Różnice między BCNF a 3NF

8.14.1. Porównanie definicji

BCNF: Schemat relacji R jest w BCNF w odniesieniu do zbioru zależności funkcyjnych F jeżeli dla wszystkich zależności funkcyjnych w F^+ (domknięcie zbioru F) postaci

$$\alpha \rightarrow \beta$$

gdzie $\alpha \subseteq R$ i $\beta \subseteq R$, zachodzi przynajmniej jeden z warunków:

1. $\alpha \rightarrow \beta$ jest trywialną zależnością (tzn. $\beta \subseteq \alpha$), albo
2. α jest superkluczem w R

3NF: Schemat relacji R jest w trzeciej postaci normalnej (3NF) w odniesieniu do zbioru zależności funkcyjnych F, jeżeli dla wszystkich zależności funkcyjnych:

$$\alpha \rightarrow \beta \text{ w } F^+ \text{ (gdzie } \alpha \subseteq R \text{ i } \beta \subseteq R\text{)}$$

zachodzi przynajmniej jeden z warunków:

1. $\alpha \rightarrow \beta$ jest trywialną zależnością (tzn. $\beta \subseteq \alpha$), albo
2. α jest superkluczem w R, albo
3. **(dodatkowo względem BCNF):** $\beta - \alpha$ jest atrybutem kluczowym

Jeżeli relacja jest w BCNF jest w 3NF (BCNF jest silniejsza niż 3NF)

W odróżnieniu od 3NF sprowadzenie do BCNF nie gwarantuje zachowania zależności funkcyjnych przy rozkładzie.

8.14.2. database_system_concepts_6th: Comparison of BCNF and 3NF:

Zaletą 3NF jest to, że **zawsze** możliwe jest zdekomponowanie schematu do 3NF w sposób bezstratny z zachowaniem wszystkich zależności.

Wadą 3NF jest to, że możemy być zmuszeni do użycia wartości NULL do reprezentacji niektórych bezsensownych relacji (co jest problemem z nadmiarowością danych)

8.14.3. Przykład

Na przykład, zależność funkcyjna "Kod -> Miasto" w schemacie

$$MUK = \{Miasto, Ulica, Kod\}$$

wskazuje, że schemat MUK nie jest w postaci normalnej Boyce'a-Codda, ale jest w trzeciej postaci normalnej, bo **atrybut Miasto jest atrybutem kluczowym** – należy do jednego z kluczy {Miasto, Ulica}.

Natomiast następujący schemat R, nie jest w trzeciej postaci normalnej i nie jest w BCNF.

$$R = \{A, B, C, D\}, F = AB \rightarrow C; B \rightarrow D; BC \rightarrow A$$

Kluczami są AB i BC. Istnieje zależność B -> D a D nie jest atrybutem kluczowym.

Przykład z DBSC:

dept_advisor (s ID , i ID , dept name) - relacja ta jest zgodna z 3NF, a nie jest zgodna z BCNF.

Wysłanie:

Now, let us again consider the dept advisor relationship set, which has the following functional dependencies:

$$i \text{ ID} \rightarrow \text{dept name}$$

$$s \text{ ID}, \text{dept name} \rightarrow i \text{ ID}$$

In Section 8.3.3 we argued that the functional dependency “ $i \text{ ID} \rightarrow \text{dept name}$ ” caused the dept advisor schema not to be in BCNF. Note that here $a = i \text{ ID}$, $b = \text{dept name}$, and $b - a = \text{dept name}$. Since the functional dependency $s \text{ ID}, \text{dept name} \rightarrow i \text{ ID}$ holds on dept advisor, the attribute dept name is contained in a candidate key and, therefore, dept advisor is in 3NF.

(Database System Concepts - str. 336)

8.15. [E] Domknięcie atrybutów? 3 zastosowania? Jak go użyć w tych zastosowaniach?

8.15.1. Domknięcie zbioru atrybutów - definicja

Mając dany zbiór atrybutów α , definiuje się domknięcie zbioru α nad F (oznaczane jako α^+) jako zbiór atrybutów, które są funkcyjnie określone przez α nad F.

6.2.3. Domknięcie zbioru atrybutów

Definicja 6.4 Domknięciem zbioru atrybutów Y (względem danego zbioru zależności) nazywamy taki zbiór atrybutów Y^+ , że

1. $Y \subseteq Y^+$
2. Dla dowolnej zależności $U \rightarrow W$, jeśli $U \subseteq Y^+$, to $W \subseteq Y^+$

Jeśli $U \subseteq V^+$, to zachodzi zależność funkcyjna $V \rightarrow U$. Klucz jest to więc taki minimalny zbiór atrybutów, że jego domknięcie jest zbiorem wszystkich atrybutów relacji.

Dla przykładu policzymy domknięcie atrybutu A z naszego przykładu

1. Krok bazowy: $A^+ := A$
2. Indukcja: ponieważ $A \rightarrow B$ i $A \subseteq A^+$, to $A^+ := A^+ \cup B = AB$
3. Indukcja: ponieważ $B \rightarrow C$ i $B \subseteq A^+$, to $A^+ := A^+ \cup C = ABC$

Wniosek: ponieważ $C \subseteq A^+$, więc zachodzi $A \rightarrow C$.

Spróbujmy innego przykładu. Dla zbioru zależności

$$AB \rightarrow C, BC \rightarrow AD, D \rightarrow E, CF \rightarrow B$$

domknięciem $\{A, B\}^+$ jest $\{A, B, C, D, E\}$, ponieważ

$$\begin{array}{ll} AB \rightarrow C & C \in \{A, B\}^+ \\ BC \rightarrow AD & D \in \{A, B\}^+ \\ D \rightarrow E & E \in \{A, B\}^+ \end{array}$$

8.15.2. Zastosowania:

- testowanie superklucza
- testowanie zależności funkcyjnych
- alternatywne wyliczenie domknięcia F^+

Testowanie superklucza: Dla domniemanego superklucza wyliczamy domknięcie. Jeżeli jest nim cała tabela, to faktycznie mamy superklucz.

Testowanie zależności funkcyjnych:

Sprawdzamy czy $\alpha \rightarrow \beta$ zachodzi przez sprawdzenie czy $\beta \subseteq \alpha^+$.

Alternatywne wyliczenie domknięcia F^+ :

Dla każdego $A \subseteq R$ obliczamy domknięcie A^+ , a następnie dla każdego $S \subseteq A^+$, dodajemy do F^+ zależność $A \rightarrow S$

// imo zamiast A powinno być alfa, bo chodzi o zbiór atrybutów, tak jest w tamtej książce

8.16. [E] Algorytm sprawdzania czy dany zbiór atrybutów jest superkluczem - wykorzystaj zbiór domknięć atrybutów

Powiązane z poprzednim podpunktrem.

Założenia: mamy jakąś relację R o zadanych atrybutach oraz zbiór atrybutów S, który będziemy testować.

1. Wyliczamy domknięcie zbioru atrybutów S⁺.
2. Jeżeli S⁺ to wszystkie atrybuty relacji, to S jest superkluczem.

8.17. [E] Domknięcie atrybutów co to jest, pseudokod

Atrybut B jest funkcyjnie zależny (określony) od α jeżeli $\alpha \rightarrow B$

Mając dany zbiór atrybutów α, definiuje się domknięcie zbioru α nad F (oznaczane jako α⁺) jako zbiór atrybutów, które są funkcyjnie określone przez α nad F.

8.17.1. Algorytm do obliczania α⁺, domknięcia zbioru α nad F:

wynik := α

powtarzaj

 dla każdej zależności funkcyjnej z F:

 jeśli atrybuty będące po lewej stronie zależności należą do wyniku

 to do wyniku dodaj atrybuty będące po prawej stronie zależności

dopóki wynik się nie zmienia [w iteracji nie został dodany żaden nowy atrybut]

8.18. [E] Co to są nadmiarowe atrybuty i jak się je znajduje?

8.18.1. Nadmiarowe atrybuty

Dla zależności $\alpha \rightarrow \beta$ w F

* Atrybut **A** ∈ α jest nadmiarowy w α jeżeli

 F logicznie implikuje $(F - \{\alpha \rightarrow \beta\}) \cup \{(\alpha - A) \rightarrow \beta\}$

* Atrybut **B** ∈ β jest nadmiarowy w β jeżeli

$(F - \{\alpha \rightarrow \beta\}) \cup \{\alpha \rightarrow (\beta - B)\}$ logicznie implikuje F

(ten zbiór różni się od F tym, że z zależności funkcyjnej $\alpha \rightarrow \beta$ zostaje wykluczony atrybut nadmiarowy)

8.18.2. Efektywne testowanie czy atrybut jest nadmiarowy:



Efektywne testowanie czy atrybut jest nadmiarowy

- Rozważmy schemat relacji R , zbiór funkcyjnych zależności F zachodzących w R i zależność funkcyjną $\alpha \rightarrow \beta$ w F .
- Aby sprawdzić czy atrybut $A \in \beta$ jest nadmiarowy w β
 1. rozważ zbiór $F' = (F - \{\alpha \rightarrow \beta\}) \cup \{\alpha \rightarrow (\beta - A)\}$
 2. sprawdź czy $\alpha \rightarrow A$ może być wywnioskowane z F' :
 - wylicz α^+ (domknięcie α) nad F'
 - jeżeli α^+ zawiera A , to A jest nadmiarowe w β
- Aby sprawdzić czy atrybut $A \in \alpha$ jest nadmiarowy w α
 1. niech $\gamma = \alpha - \{A\}$
 2. sprawdź, czy $\gamma \rightarrow \beta$ może być wyprowadzone z F
 - wylicz γ^+ (domknięcie γ) nad F
 - jeżeli γ^+ zawiera wszystkie atrybuty w β , to A jest nadmiarowe w α

* Aby sprawdzić czy atrybut $A \in \alpha$ jest nadmiarowy w α definiujemy $\gamma = \alpha - \{A\}$ sprawdź, czy $\gamma \rightarrow \beta$ może być wyprowadzone z F :

- wylicz γ^+ (domknięcie γ) nad F
- jeżeli γ^+ zawiera wszystkie atrybuty w β , to A jest nadmiarowe w α

* Aby sprawdzić czy atrybut $B \in \beta$ jest nadmiarowy w β rozważ zbiór $F' = (F - \{\alpha \rightarrow \beta\}) \cup \{\alpha \rightarrow (\beta - B)\}$

sprawdź czy $\alpha \rightarrow B$ może być wywnioskowane z F' :

- wylicz α^+ (domknięcie α) nad F'
- jeżeli α^+ zawiera B , to B jest nadmiarowe w β

8.19. [E] Kanoniczne pokrycie zbioru zależności funkcyjnych

Niektóre zbiory zależności mogą zawierać nadmiarowe zależności, które mogą być wywnioskowane z innych.

Intuicyjnie kanoniczne pokrycie F_c zbioru zależności funkcyjnych F to najmniejszy zestaw zależności funkcyjnych równoważny F , w którym brak jest zbędnych zależności bądź części zależności. (czasem któreś z zależności są zbędne, a czasem tylko któreś z ich atrybutów).

Kanoniczne pokrycie dla zbioru F jest zbiorem zależności F_c takich, że:

- F logicznie implikuje wszystkie zależności w F_c
- F_c logicznie implikuje wszystkie zależności w F
-
- żadna z funkcyjnych zależności w F_c nie zawiera nadmiarowego atrybutu
- każda lewa strona zależności funkcyjnych w F_c jest unikalna

Z wykładu:



Kanoniczne pokrycie

- **Kanoniczne pokrycie** dla F jest zbiorem zależności F_c takich że
 - F logicznie implikuje wszystkie zależności w F_c , i
 - F_c logicznie implikuje wszystkie zależności w F , i
 - żadna z funkcjonalnych zależności w F_c nie zawiera nadmiarowego atrybutu, i
 - Każda lewa strona zależności funkcjonalnych w F jest unikalna, czyli nie ma dwóch zależności w F_c : $\alpha_1 \rightarrow \beta_1$ i $\alpha_2 \rightarrow \beta_2$ gdzie $\alpha_1 = \alpha_2$

$$F_c = F$$

repeat

 Use the union rule to replace any dependencies in F_c of the form $\alpha_1 \rightarrow \beta_1$ and $\alpha_1 \rightarrow \beta_2$ with $\alpha_1 \rightarrow \beta_1 \beta_2$

 Find a functional dependency $\alpha \rightarrow \beta$ in F_c with an extraneous attribute either in α or in β

 /* Note: test for extraneous attributes done using F_c , not F */

 If an extraneous attribute is found, delete it from $\alpha \rightarrow \beta$ in F_c

until F_c does not change

8.20. [E2017] Co to jest dekompozycja, cechy dobrej dekompozycji

UWAGA: Często oprócz opisu cech i podania przykładów wymagana jest **dokładna definicja bezstratnej dekompozycji i zachowania zależności**. Dla wielu osób niepodanie tych definicji skończyło się to oblanym egzaminem.

8.20.1. Co to jest dekompozycja

Dekompozycja relacji, to rozbicie jednej relacji na dwie lub więcej relacji.

8.20.2. Cechy dobrej dekompozycji

1. Eliminowanie anomalii
 - a. Redundancja: adres dostawcy powtarza się dla każdego dostarczanego towaru.
 - b. Anomalie przy modyfikacji: uaktualniony adres w jednym wierszu pozostaje niezmieniony w innych.
 - c. Anomalie przy wstawianiu: trudno wstawić dostawcę bez towarów; towar wchodzi w skład klucza - nie może być NULL.
 - d. Anomalie przy usuwaniu: usuwając informacje o wszystkich towarach dostarczanych przez dostawcę (który może zmienić profil produkcji) usuwamy informację o samym dostawcy.
2. Odtwarzalność informacji - [bezstratna dekompozycja](#)
3. [Zachowanie zależności](#)

8.21. [E] Dekompozycja bezstratna + warunki

Dekompozycja jest bezstratna jeżeli nie ma utraty informacji przy zastępowaniu schematu relacji dwoma schematami relacji.

Inaczej: dla wszystkich legalnych instancji bazy danych, relacja r zawiera ten sam zbiór krotek jak wynik zapytania SQL:

```
select * from (select R1 from r) natural join (select R2 from r)
```

Wykorzystanie zależności funkcyjnych do stwierdzenia, kiedy dekompozycje są bezstratne:

Warunek: dekompozycja R do R1 i R2 jest złączeniem bezstratnym, jeżeli przynajmniej jedna z zależności jest w F+ :

- $R1 \cap R2 \rightarrow R1$
- $R1 \cap R2 \rightarrow R2$

Inaczej: – jeżeli $R1 \cap R2$ tworzy superklucz w R1 lub R2 to dekompozycja R jest bezstratna

Z wykładu:



Bezstratna dekompozycja (Lossless Decomposition)

- Niech $r(R)$ – schemat relacji, F – zbiór zależności funkcyjnych w $r(R)$, R_1 i R_2 – tworzą dekompozycję R
- Dekompozycja jest **bezstratna** jeżeli nie ma utraty informacji przy zastępowaniu $r(R)$ dwoma schematami relacji $r_1(R_1)$ i $r_2(R_2)$
- Dokładniej:
 - dla wszystkich legalnych instrancji bazy danych, relacja r zawiera ten sam zbiór krotek jak wynik zapytania SQL:

```
select * from (select R1 from r) natural join (select R2 from r)
```

$$r = \Pi_{R1}(r) \bowtie \Pi_{R2}(r)$$

8.22. [E] Kiedy dekompozycja zachowuje zależność?

Dekompozycja R na (R1, R2 ... Rn) zachowuje zależność jeśli spełnia warunek

$$(F_1 \cup F_2 \cup \dots \cup F_n)^+ = F^+$$

Gdzie F_i to **restrykcja** z F nad R_i . F_i jest podzbiorem takich zależności z F^+ , które zawierają tylko atrybuty z R_i .

Z wykładu - [E2017] Zachowanie zależności funkcyjnych w dekompozycji



Zachowanie zależności

- Niech F będzie zbiorem zależności funkcyjnych w schemacie R , a R_1, R_2, \dots, R_n – dekompozycja R
- **Restrykcja** z F nad R_i jest zbiorem F_i wszystkich zależności funkcyjnych w F^+ które zawierają **tylko** atrybuty z R_i
- Niech F_i będzie zbiorem zależności F^+ takim, że zawiera tylko atrybuty z R_i .
 - Dekompozycja jest **zachowującą zależność**, jeżeli $(F_1 \cup F_2 \cup \dots \cup F_n)^+ = F^+$



Testowanie zachowania zależności – alternatywne podejście

- W celu sprawdzenia czy zależność $\alpha \rightarrow \beta$ jest zachowana w dekompozycji R na R_1, R_2, \dots, R_n
 - $result = \alpha$
 - repeat**
for each R_i in the decomposition
 $t = (result \cap R_i)^+ \cap R_i$
 $result = result \cup t$
 - until** ($result$ does not change)
 - Jeżeli $result$ zawiera wszystkie atrybuty w β , wtedy zależność funkcyjna $\alpha \rightarrow \beta$ jest zachowana.

8.23. [E2017] Zależności wielowartościowe

8.23.1. Zależności wielowartościowe (czwarta postać normalna) na przykładzie

id krotek dla zobrazowania przykładu ze slajdów niżej (dla Nr_stud $\rightarrow\rightarrow$ Przedmiot)	Nr_stud	Przedmiot	Sport
t1	100	Bazy danych	Tenis
t3	100	Bazy danych	Biegi
t4	100	Systemy informacyjne	Tenis
t2	100	Systemy informacyjne	Biegi
	200	Bazy danych	Boks

Schemat relacji jest w postaci normalnej Boyce'a-Codda (bo jedynym kluczem są wszystkie trzy atrybuty) a w tabeli jest redundancja i możliwe są anomalie!

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

$\text{Nr_stud} \rightarrow\!\!\!> \text{Przedmiot}$; $\text{Nr_stud} \rightarrow\!\!\!> \text{Sport}$

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: $\{\text{Nr_stud}, \text{Przedmiot}\}$ i $\{\text{Nr_stud}, \text{Sport}\}$.

Z wykładow - zależności wielowartościowe



Zależności wielowartościowe (Multivalued Dependencies - MVDs)

- Niech $r(R)$ będzie schematem relacji i niech $\alpha \subseteq R$ i $\beta \subseteq R$.
Zależność wielowartościowa

$$\alpha \rightarrow\!\!\!> \beta$$

zachodzi w R , jeżeli w dowolnej legalnej instancji relacji $r(R)$, dla wszystkich par krotek t_1 i t_2 w r takich, że $t_1[\alpha] = t_2[\alpha]$, istnieją krotki t_3 i t_4 w r takie, że:

$$\begin{aligned}t_1[\alpha] &= t_2[\alpha] = t_3[\alpha] = t_4[\alpha] \\t_3[\beta] &= t_1[\beta] \\t_3[R - \beta] &= t_2[R - \beta] \\t_4[\beta] &= t_2[\beta] \\t_4[R - \beta] &= t_1[R - \beta]\end{aligned}$$

equality-generating dependencies - FD

tuple-generating dependencies - MVD

Definicje łatwo zapamiętać patrząc na tabelę - kolorami zaznaczone wartości, które są sobie równe (dokładnie to co w definicji):



Zależności wielowartościowe c.d.

- Tabelaryczna reprezentacja $\alpha \rightarrow\!\!\!> \beta$

	α	β	$R - \alpha - \beta$
t_1	$a_1 \dots a_i$	$a_{i+1} \dots a_j$	$a_{j+1} \dots a_n$
t_2	$a_1 \dots a_i$	$b_{i+1} \dots b_j$	$b_{j+1} \dots b_n$
t_3	$a_1 \dots a_i$	$a_{i+1} \dots a_j$	$b_{j+1} \dots b_n$
t_4	$a_1 \dots a_i$	$b_{i+1} \dots b_j$	$a_{j+1} \dots a_n$

8.23.3. Wykorzystanie wielowartościowych zależności

Do:

1. sprawdzenia relacji w celu ustalenia czy są one legalne dla danego zbioru zależności funkcyjnych i wielowartościowych
2. określenia ograniczeń na zbiorze legalnych relacji; brane będą pod uwagę tylko te relacje, które spełniają dany zbiór zależności funkcyjnych i wielowartościowych.

Jeżeli relacja r nie spełnia danej wielowartościowej zależności, możemy skonstruować relację r' która spełnia tę zależność poprzez dodanie krotek do r .

8.24. [E] Czwarta postać normalna (4NF)

4NF relacja w BCNF, która nie zawiera żadnych nietrywialnych zależności wielowartościowych
Schemat relacji $r(R)$ jest w 4NF w odniesieniu do zbioru D funkcyjnych i wielowartościowych
zależności jeżeli dla wszystkich wielowartościowych zależności w D^+ postaci $\alpha \rightarrow\rightarrow \beta$, gdzie $\alpha \subseteq R$ i $\beta \subseteq R$, zachodzi co najmniej jeden z poniższych warunków:

- $\alpha \rightarrow\rightarrow \beta$ jest trywialną zależnością wielowartościową (tzn. $\beta \subseteq \alpha$ lub $\alpha \cup \beta = R$)
- α jest superkluczem dla R

Jeżeli relacja jest w 4NF to jest w BCNF

8.24.1. [Dodatkowo] Restrykcje wielowartościowych zależności



Restrykcje wielowartościowych zależności

- Dany schemat relacji $r(R)$, niech $r_1(R_1), r_2(R_2), \dots, r_n(R_n)$
dekompozycja $r(R)$
 - aby sprawdzić czy każda r_i jest w 4NF należy znaleźć jakie wielowartościowe zależności zachodzą w każdej r_i
- Restrykcja D (zbior zależności funkcyjnych i wielowartościowych) do R_i jest zbiorem D_i zawierającym:
 - Wszystkie zależności funkcyjne w D^+ które zawierają tylko atrybuty R_i
 - Wszystkie wielowartościowe zależności w postaci:
$$\alpha \rightarrow\rightarrow (\beta \cap R_i)$$
gdzie $\alpha \subseteq R_i$ i $\alpha \rightarrow\rightarrow \beta$ jest w D^+

8.24.2. [Dodatkowo] Algorytm dekompozycji 4NF



Algorytm dekompozycji 4NF

```
result: = {R};  
done := false;  
compute  $D^+$ ; Given schema  $R_i$ , let  $D_i$  denote the restriction  
of  $D^+$  to  $R_i$   
while (not done) do  
  if (there is a schema  $R_i$  in result that is not in 4NF)  
    then begin  
      let  $\alpha \rightarrow\rightarrow \beta$  be a nontrivial multivalued dependency  
      that holds  
      on  $R_i$  such that  $\alpha \rightarrow R_i$  is not in  $D_i$ , and  $\alpha \cap \beta = \emptyset$ ;  
      result := (result -  $R_i$ )  $\cup$  ( $R_i - \beta$ )  $\cup$  ( $\alpha, \beta$ );  
    end  
  else done := true;  
Uwaga: każda  $R_i$  jest w 4NF a dekompozycja jest bezstratna
```



Przykład działania algorytmu

- dana relacja $r_2(ID, dept_name, street, city)$
- $ID \rightarrow\rightarrow dept_name$
 - nietrywialna wielowartościowa zależność i
 - ID nie jest super kluczem dla r_2
- Dekompozycja
 - $r_{21} = (ID, dept_name)$
 - $r_{22} = (ID, street, city)$
- Są w 4NF

8.25. [E2017] Modelowanie danych w czasie

Dane czasowe mają skojarzony przedział czasowy w którym są ważne.

Snapshot (rodzaj zrzutu danych rozumianego jako sposób zapewniania dostępu do starszych wersji danych przechowywanych w pamięci masowej)

Kilka propozycji rozszerzenia modelu ER poprzez dodanie ważnego czasu, ale brak akceptowalnych standardów

Dodanie zależności czasowych skutkuje tym, że zależność funkcyjna np. nie będzie zachodzić, gdyż adres może się zmieniać w czasie

Czasowa zależność funkcyjna $X \text{-t-} Y$ zachodzi na schemacie $r(R)$ jeżeli dla wszystkich legalnych instancji $r(R)$ wszystkie snapshoty r spełniają zależność funkcyjną $X \rightarrow Y$.

W praktyce, można dodać do relacji atrybuty czasu rozpoczęcia i zakończenia

Klucz obcy powinien mieć referencję do aktualnej wersji danych lub do danych w konkretnym punkcie czasowym.

Częste rozwiązanie: relacja history

8.26. [E] Czasowe zależności klucza obcego od głównego

Klucz główny powinien być stały w czasie (w MS SQL da się go zmieniać, ale nie bardzo nie powinno się tego robić) ?

// nie wiem czy to o to chodzi

Klucz obcy powinien mieć referencję do aktualnej wersji danych lub do danych w konkretnym punkcie czasowym (ostatnie 3 slajdy z wykładu o normalizacji) może o to jej chodziło

*książka str. 366 zawiera odpowiedź na to pytanie

9. Wykład - Organizacja plików

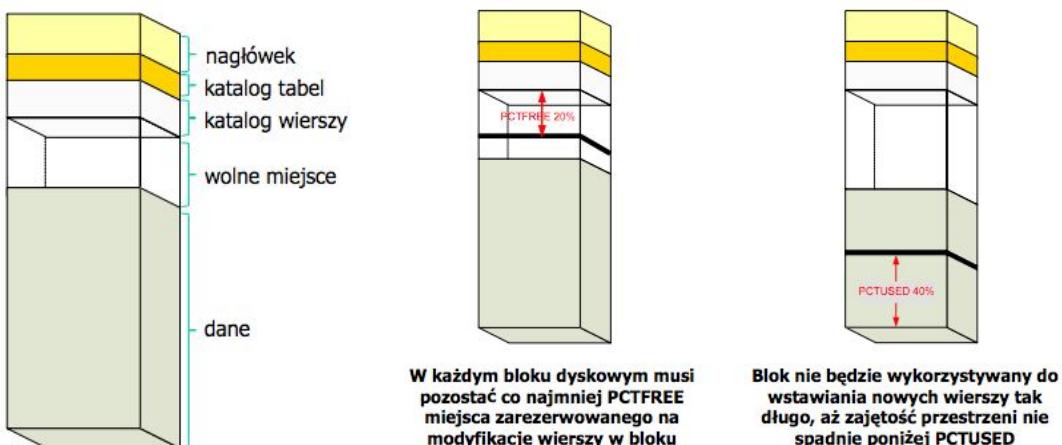
9.1. [E2017] PCTFREE and PCTUSED

Baza danych Oracle przechowuje bloki danych w pamięci. Obowiązkiem administratora jest zapewnienie tego, aby bloki te były jak najgęściej wypełnione danymi, gdyż niedokładnie zapełnione bloki powodują spadek wydajności pracy buforów bazy danych.

To, w jaki sposób zarządzany jest obszar w blokach danych, zależy od ustawień parametrów składowania (storage parameters):

1. PCTFREE liczba – określa, jaka część bloku jest zarezerwowana na rozrost wierszy, które już są w bloku danych (wyrażony w %) – domyślną wartością jest PCTFREE 10
2. PCTUSED liczba – określa rozmiar przestrzeni bloku, poniżej którego będzie dozwolone wstawianie nowego rekordu (tj. określa poziom graniczny, przy którym blok może trafić z powrotem na listę wolnych bloków) (wyrażony w %) – domyślną wartością jest PCTUSED 40

Zarządzanie wolną przestrzenią w bloku



Omówione parametry określa się przy definicji relacji. Jeśli rekordy tabeli nie będą aktualizowane, należy dla takiej tabeli ustawić możliwie najniższą wartość parametru PCTFREE. Natomiast w przypadku aplikacji intensywnie wykorzystujących operacje INSERT i DELETE, warto rozważyć zwiększenie wartości PCTUSED, by zwalniane bloki były jak najszybciej dodawane do listy wolnych bloków.

Zbyt mała wartość parametru PCTFREE może zmusić bazę danych do przeniesienia wiersza do innego bloku w wyniku wykonania na nim operacji UPDATE. To z kolei zmusza bazę do odczytania kilku bloków dla jednego rekordu, co zwiększa liczbę logicznych odczytów potrzebnych do wykonania polecenia. Jeśli rekord nie mieści się w jednym bloku może być przechowywany w wielu blokach

9.2. [E2017] Freelist i w jakiej strukturze

At the beginning of the file, we allocate a certain number of bytes as a file header. The header will contain a variety of information about the file. For now, all we need to store there is the address of the first record whose contents are deleted. We use this first record to store the address of the second available record, and so on. Intuitively, we can think of these stored addresses as pointers, since they point to the location of a record. The deleted records thus form a **linked list**, which is often referred to as a **free list**. Figure 10.7 shows the file of Figure 10.4, with the free list, after records 1, 4, and 6 have been deleted.

header				
record 0	10101	Srinivasan	Comp. Sci.	65000
record 1				
record 2	15151	Mozart	Music	40000
record 3	22222	Einstein	Physics	95000
record 4				
record 5	33456	Gold	Physics	87000
record 6				
record 7	58583	Califieri	History	62000
record 8	76543	Singh	Finance	80000
record 9	76766	Crick	Biology	72000
record 10	83821	Brandt	Comp. Sci.	92000
record 11	98345	Kim	Elec. Eng.	80000

Figure 10.7 File of Figure 10.4, with free list after deletion of records 1, 4, and 6.

On insertion of a new record, we use the record pointed to by the header. We change the header pointer to point to the next available record. If no space is available, we add the new record to the end of the file.

Insertion and deletion for files of fixed-length records are simple to implement, because the space made available by a deleted record is exactly the space needed to insert a record. If we allow records of variable length in a file, this match no longer holds. An inserted record may not fit in the space left free by a deleted record, or it may fill only part of that space.

9.3. Rekordy o stałej długości

9.3.1. Podejście:

- Przechowuj rekord i zaczynając od bajtu $n * (i - 1)$, gdzie n jest rozmiarem każdego rekordu.
- Dostęp do rekordu prosty, ale rekordy mogą zajmować wiele bloków

9.4. [E2017] Rekordy o zmiennej długości

Pytanie z egzaminu: Jak są reprezentowane rekordy zmiennej długości? (+13)

DOKŁADNIEJ: jak są reprezentowane rekordy o atrybutach o stałej i zmiennej długości?

UWAGA: Powyższe dwa to dwa różne pytania, w pierwszym chodzi o [slotted page](#), drugie ma odpowiedź poniżej

OCZEKIWANA ODPOWIEDŹ: najpierw pary(offset, dlugosc zmiennego atrybutu) potem stałej długości, null bitmapa, i potem zmiennej długości.

9.4.1. Reprezentacja rekordów o atrybutach stałej i zmiennej długości

Na początku tablicy w pamięci znajdują się dane dla odpowiednich atrybutów: jeśli atrybut jest stałej długości, jest tam jej wartość, a jeśli zmiennej to para (offset - gdzie zaczyna się dany atrybut, długość) wskazująca na wartość tego atrybutu. Następnie jest null bitmapa, która na każdym bicie opisuje, czy dana wartość jest null(1) czy nie (0). Po niej są wartości kolumn zmiennej długości. Na przykład dla kolumn ID, Imię, Płeć odpowiednio int, varchar, char będzie to wyglądać mniej więcej tak:

23 (32, 6) M 000 Marcin
ID Imię Płeć null bitmapa wartość imienia

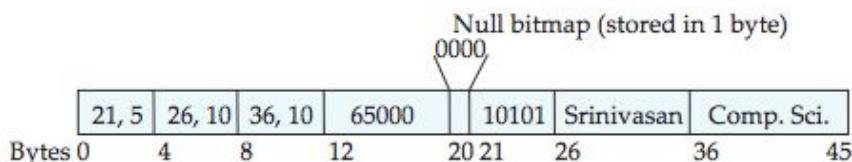


Figure 10.8 Representation of variable-length record.

The figure shows an instructor record, whose first three attributes:

- **ID**(Zaczyna się na **21** bajcie, zajmuje **5** bajtów), **name(26, 10)**, and **dept name(36, 10)** are *variable-length strings*,
 - and whose fourth attribute **salary** is a *fixed-sized number*.

We assume that the offset and length values are stored in two bytes each, for a total of 4 bytes per attribute. The salary attribute is assumed to be stored in 8 bytes, and each string takes as many bytes as it has characters.

The figure also illustrates the use of a **null bitmap**, which indicates which attributes of the record have a null value. In this particular record, if the salary were null, the fourth bit of the bitmap would be set to 1, and the salary value stored in bytes 12 through 19 would be ignored. Since the record has four attributes, the null bitmap for this record fits in 1 byte, although more bytes may be required with more attributes. In some representations, the null bitmap is stored at the beginning of the record, and for attributes that are null, no data (value, or offset/length) are stored at all. Such a representation would save some storage space, at the cost of extra work to extract attributes of the record. This representation is particularly useful for certain applications where records have a large number of fields, most of which are null.

9.5. [E] W rekordzie są 3 pola o zmiennym rozmiarze atrybutów i jeden o stałym, jak to jest fizycznie zorganizowane

Opisane w poprzednim podpunkcie.

9.5.1. Opracowanie z zeszłego roku:

Atrybuty zmiennej długości reprezentowane przez stały rozmiar (offset, długość), z aktualnymi danymi przechowywanymi po wszystkich atrybutach stałej długości (Figure 10.8 z poprzedniego podpunktu, slajd 12 organizacja dysku)

Dodzę się, chodziło jej dokładnie o ten rysunek, co jest na samym dole strony. (Powyższa odpowiedź nic nie dała) Oczekiwana dokładnego opisu, jak on wygląda począwszy od zerowego bajtu. Czyli najpierw lokalizacja i rozmiar każdego atrybutu, później null bitmapa, rekordy stałej długości i na końcu zmiennej.

a null bit mapa nie jest po atrybutach stałej długości? - niech ktoś odpowie :)

według rysunku jest po stałej inaczej te liczby nie miałyby sensu, chociaż kto by wyczuł czy ona się nie pomyliła

9.6. [E] Przechowywanie rekordów o zmiennej długości - struktura slotowa, slotted page

Struktura slotowa Slotted page jest przeważnie używana do przechowywania rekordów o zmiennej długości w bloku

Nagłówek Slotted Page - Block Header (nagłówek na początku każdego bloku) zawiera:

- liczbę rekordów #Entries
- koniec wolnej przestrzeni w bloku End of Free Space
- tablicę z informacją o lokalizacji i rozmiarze każdego rekordu

Po nagłówku jest wolna przestrzeń Free Space

Po wolnej przestrzeni są rekordy Records..

Przy **dodawaniu** nowego rekordu alokowana jest wolna przestrzeń i dodawana jest nowa informacja do tabeli (lokalizacja i rozmiar rekordu).

Przy **usuwaniu** rekordu

- przestrzeń, którą ten rekord zajmował jest zwalniana, its entry is set to deleted (its size is set to -1, for example).
- rekordy znajdujące się przed usuniętym rekordem (wg. grafiki po lewej) są przesuwane (do prawej strony)
- aktualizowany jest wskaźnik na koniec wolnej przestrzeni

Koszt przenoszenia rekordów nie jest zbyt wysoki, bo rozmiar bloku jest ograniczony (zwykle 4-8 KB)

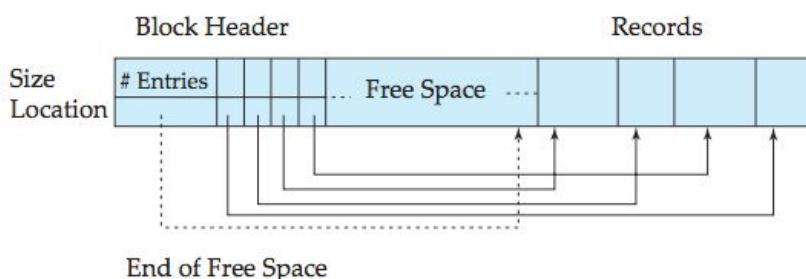


Figure 10.9 Slotted-page structure.

Nie ma konieczności przechowywania wskaźników bezpośrednio do rekordów - zamiast tego wskaźniki wskazują na nagłówek każdego bloku, a nagłówek każdego bloku na konkretne rekordy. Pozwala to na przesuwanie rekordów bez fragmentacji.

9.7. [E2017] Dlaczego istnieją rekordy zmiennej długości?

Using variable-length records might enable you to **save disk space**. When you use fixed-length records, you need to make the record length equal to the length of the longest record. If your application generates many short records with occasional long ones, using fixed-length records wastes a lot of disk space, so variable-length records would be a better choice.

9.8. [Dodatkowo] Kiedy używać rekordów o stałej długości

Because variable-length records require a little more processing and some extra code, it is advisable to use fixed-length records unless variable-length records are actually necessary.

9.9. [E2017] Rodzaje organizacji rekordów w plikach

9.9.1. Nieuporządkowane

Nagłówek pliku zawiera wskaźnik do bloku danych

Blok danych zawiera wskaźnik do bloku następnego i poprzedniego

Rekordy wstawiane na koniec pliku

9.9.2. Uporządkowane (sekwencyjne) - organizacja pliku sekwencyjnego

Records are stored in sequential order, according to the value of a “search key” of each record. To permit fast retrieval of records in search-key order, we

- chain together records by pointers
- the pointer in each record points to the next record in search-key order.

Furthermore, to minimize the number of block accesses in sequential file processing, we store records physically in search-key order, or as close to search-key order as possible.

In that example (Figure10.11), the records are stored in search-key order, using ID as the search key:

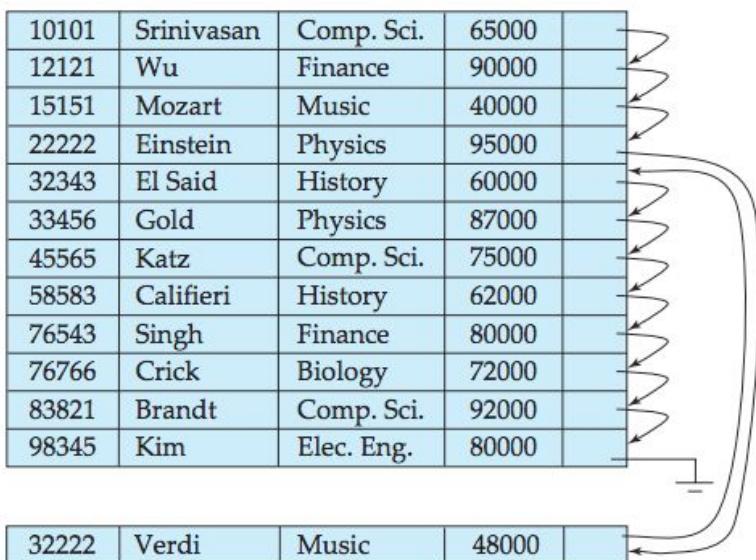


Figure 10.11 Sequential file after an insertion.

Usuwanie z pliku sekwencyjnego – wykorzystanie łańcucha wskaźników

9.9.2.1. [E2017] - Co to jest overflow block, czy to jest korzystne i co trzeba robić?

Wstawianie do pliku sekwencyjnego

- zlokalizuj rekord, który jest przed wstawianym rekordem
- Jeśli jest wolna przestrzeń (pozostała po usunięciu innego rekordu) w tym samym bloku, w którym znajduje się zlokalizowany rekord wstaw tam nowy rekord. W przeciwnym wypadku wstaw nowy rekord do **overflow block**
- Na koniec dostosuj wskaźniki tak, aby rekordy były ustawione w określonym porządku

9.9.2.2. Reorganizacja pliku sekwencyjnego

The structure in Figure 10.11 allows fast insertion of new records, but forces sequential file-processing applications to process records in an order that does not match the physical order of the records.

If relatively few records need to be stored in **overflow blocks**, this approach works well. Eventually, however, the correspondence between search-key order and physical order may be totally lost over a period of time, in which case sequential processing will become much less efficient. At this point, the file should be **reorganized** so that it is once again physically in sequential order. Such reorganizations are costly, and must be done during times when the system load is low. The frequency with which reorganizations are needed depends on the frequency of insertion of new records.

9.9.3. Haszowe

A hash function is computed on some attribute of each record. The result of the hash function specifies in which block of the file the record should be placed.

9.9.4. Klastrowanie wielotablicowe. Multitable clustering file organization

[E2017] Co nam daje przechowywanie różnych tabel w jednym pliku? Jak się nazywa ta struktura?

UWAGA: Chciała ** * DOKŁADNIE ** * * wiedzieć jak są ułożone są rekordy

W skrócie (z wykładu):

- dobre dla zapytań wykorzystujących department instructor i dla zapytań używających 1 wydział i jego instruktorów
- mało efektywne dla zapytań korzystających tylko z department

Bardziej opisowo: Database_system_concepts_6th: Multitable clustering file organization:

A multitable clustering file organization is a file organization, such as that illustrated in Figure 10.14, that stores related records of two or more relations in each block. Such a file organization allows us to read records that would satisfy the join condition by using one block read. Thus, we are able to

Comp. Sci.	Taylor	100000
45564	Katz	75000
10101	Srinivasan	65000
83821	Brandt	92000
Physics	Watson	70000
33456	Gold	87000

process this particular query more efficiently.
Figure 10.14 Multitable clustering file structure.

Our use of clustering of multiple tables into a single file has **enhanced processing of a particular join** (that of department and instructor), but it results in **slowing processing of other types of queries**.

For example,

select * from department;

requires more block accesses than it did in the scheme under which we stored each relation in a separate file, since each block now contains significantly fewer department records. To locate efficiently all tuples of the department relation in the structure of Figure 10.14, we can chain together all the records of that relation using pointers, as in Figure 10.15.

Comp. Sci.	Taylor	100000	
45564	Katz	75000	
10101	Srinivasan	65000	
83821	Brandt	92000	
Physics	Watson	70000	
33456	Gold	87000	

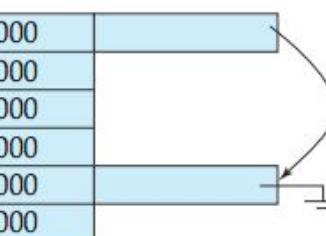


Figure 10.15 Multitable clustering file structure with pointer chains.

When multitable clustering is to be used depends on the types of queries that the database designer believes to be most frequent. Careful use of multitable clustering can produce significant performance gains in query processing.

9.10. [E] Słownik danych co to jest, co przechowuje, jak jest fizycznie zorganizowany

Słownik danych (katalog systemowy) przechowuje metadane; czyli dane o danych, takie jak:

- informacje o relacjach
- informacje o użytkownikach i ich kontach, włączając hasła
- dane statystyczne i opisowe
- informacje o fizycznej organizacji plików
- informacje o indeksach

Używany zarówno przez system zarządzania bazą jak i użytkowników.

Ważny jest przedrostek w nazwie, np. user1_Tables(Tabs) info o tabelach, których user1 jest właścicielem

Fizyczna organizacja słownika danych:

Na ogół jako minibaza (Zbiór tabel) wewnętrznej bazy danych. Żeby zapewnić szybkość dostępu często nieznormalizowana.

9.11. [E] Buffer manager

Plik bazy danych jest podzielony na stałą długości jednostki pamięci zwane blokami. System bazy danych dążą do zminimalizowania liczby transferów bloków między dyskiem, a pamięcią.

Bufor (pula buforów)

9.11.1. [E] Zarządzaniem buforem/Zarządca bufora (buffer manager)

Menedżer bufferu - podsystem odpowiedzialny za alokowanie bufora do pamięci głównej. koniecznie wspomnieć, że on zapisuje z bufora na dysk.

Opis działania z wykładu:

Programy wysyłają żądanie do **zarządcy bufora** gdy potrzebują dany blok z dysku.

1. Jeżeli blok jest już w buforze, zarządcza zwraca adres bloku w pamięci głównej
2. Jeżeli nie, to zarządcza:
 - 2.1. Alokuje przestrzeń w puli buforów dla bloku
 - 2.1.1. Wymiana (wyrzucenie) jakiegoś innego bloku, w razie potrzeby, aby zrobić miejsce na nowy blok.
 - 2.1.2. Zastąpiony blok zapisywany na dysk tylko wtedy gdy został zmodyfikowany od ostatniego czasu gdy został zapisany/pobrany na/z dysku.
 - 2.2. Czyta blok z dysku do bufora i zwraca adres bloku w pamięci głównej żądającemu.

9.12. [E] Buforowanie bloków danych

- odpowiedzialny za to jest Buffer Manager
- chodzi o to, żeby jak najwięcej bloków danych było w pamięci operacyjnej
- minimalizacja ilości przesyłanych danych z plików do bufora

Więcej w poprzednim podpunkcie

9.13. [Dodatkowo] Techniki wykorzystywane przez zarządcę bufora

9.13.1. LRU strategy

Gdy brakuje miejsca w buforze – usuwany blok najdawniej używany.

When there is no room left in the buffer, a block must be removed from the buffer before a new one can be read in. Most operating systems use a least recently used (LRU) scheme, in which the block that was referenced least recently is written back to disk and is removed from the buffer. This simple approach can be improved on for database applications.

9.13.2. Przypięty blok (pinned block)

For the database system to be able to recover from crashes, it is necessary to restrict those times when a block may be written back to disk. For instance, most recovery systems require that a block should not be written to disk while an update on the block is in progress. A block that is not allowed to be written back to disk is said to be pinned.

9.13.3. Wymuszony zapis (forced output)

There are situations in which it is necessary to write back the block to disk, even though the buffer space that it occupies is not needed. This write is called the **forced output** of a block. Reason for **forced output** briefly: main-memory contents and thus buffer contents are lost in a crash, whereas data on disk usually survive a crash.

9.13.4. Strategia natychmiastowego podrzucania (toss-immediate)

Assume that the two relations of this example are stored in separate files. In this example, we can see that, once a tuple of instructor has been processed, that tuple is not needed again. Therefore, once processing of an entire block of instructor tuples is completed, that block is no longer needed in main memory, even though it has been used recently. The buffer manager should be instructed to free the space occupied by an instructor block as soon as the final tuple has been processed. This buffer-management strategy is called the **toss-immediate strategy**.

9.13.5. Strategia ostatniego używanego (most recently used MRU)

Now consider blocks containing department tuples. We need to examine every block of department tuples once for each tuple of the instructor relation. When processing of a department block is completed, we know that that block will not be accessed again until all other department blocks have been processed. Thus, the most recently used department block will be the final block to be re-referenced, and the least recently used department block is the block that will be referenced next.

This assumption set is the exact opposite of the one that forms the basis for the LRU strategy. Indeed, the optimal strategy for block replacement for the above procedure is the **most recently used (MRU)** strategy. If a department block must be removed from the buffer, the MRU strategy chooses the most recently used block (blocks are not eligible for replacement while they are being used).

For the MRU strategy to work correctly for our example, the system must pin the department block currently being processed. After the final department tuple has been processed, the block is unpinned, and it becomes the most recently used block.

Zarządcy bufora mogą wykorzystywać statystyczne informacje dotyczące prawdopodobieństwa, że żądanie będzie dotyczyło konkretnej relacji

10. Wykład - Przetwarzanie transakcyjne

10.1. [E2017] Transakcje - co to jest (dodatekowo - ACID)

Def. z wykładu: **Transakcja** - sekwencja logicznie powiązanych operacji na bazie danych, która przeprowadza bazę danych z jednego stanu spójnego w inny stan spójny.

Wikipedia: **Transakcja** – zbiór operacji na [bazie danych](#), które stanowią w istocie pewną całość i jako takie powinny być wykonane wszystkie lub żadna z nich. Warunki jakie powinny spełniać transakcje bardziej szczegółowo opisują zasady **ACID**:

- **atomicity niepodzielność** - Zbiór operacji wchodzących w skład transakcji jest niepodzielny: albo zostaną wykonane wszystkie operacje transakcji albo żadna.
- **consistency spójność** - Realizacja transakcji w izolacji zachowuje spójność bazy danych. Transakcja nie może naruszać warunków integralnościowych
- **isolation izolacja** - Transakcje są od siebie logicznie odseparowane. Transakcje oddziałują na siebie poprzez dane. Mimo wspólnego wykonywania, transakcje widzą stan bazy danych tak, jak gdyby były wykonywane w sposób sekwencyjny
- **durability trwałość** - Wyniki zatwierdzonych transakcji nie mogą zostać utracone w wyniku wystąpienia awarii systemu. Zatwierdzone dane w bazie danych, w przypadku awarii, muszą być odtwarzalne

Transakcja

- zaczyna się
 - niejawnie
 - jawnie (**begin transaction**)
- Kończy:
 - commit work
 - rollback work

Ale domyślnie w wielu implementacjach SQL: każde wyrażenie SQL jest transakcją i zatwierdza się automatycznie po wykonaniu

Przykładem transakcji może być transakcja bankowa jaką jest przelew. Muszą tu zostać dokonane 2 operacje – zabranie pieniędzy z jednego konta oraz dopisanie ich do drugiego. W przypadku niepowodzenia żadna z tych operacji nie powinna być zatwierdzona, gdyż zajście tylko jednej powodowałoby nieprawidłowości w bazie danych (pojawienie się lub zniknięcie pieniędzy).

Transakcja składa się zawsze z 3 etapów:

- rozpoczęcia
- wykonania
- zamknięcia

W systemach bazodanowych istotne jest, aby transakcja trwała jak najkrócej, ponieważ równolegle może być dokonywanych wiele transakcji i część operacji musi zostać wykonana w pewnej kolejności. Każdy etap transakcji jest **logowany**, dzięki czemu w razie awarii systemu (dzięki zawartości logów) można odtworzyć stan bazy danych sprzed transakcji, która nie została zamknięta.

Część systemów baz danych umożliwia używanie **punktów pośrednich** (ang. *save point*), są to zapamiętane w systemie etapy transakcji, do których w razie wystąpienia błędu można się wycofać, bez konieczności anulowania wszystkich wykonanych działań.

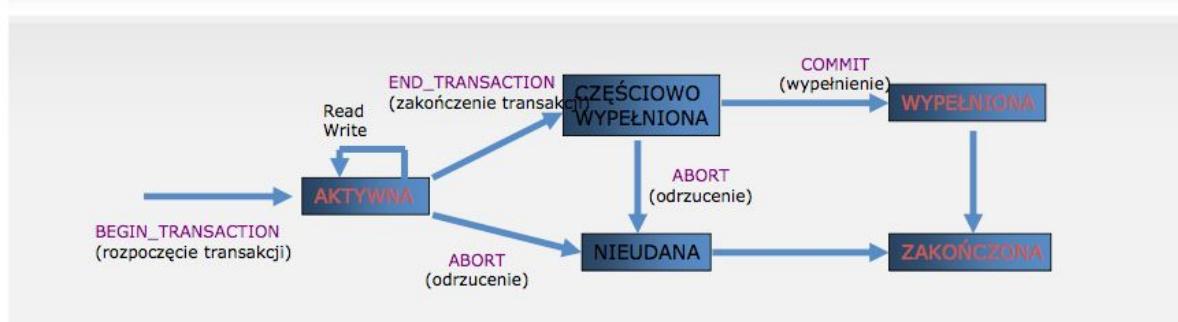
10.2. [E] Diagram stanów transakcji

Każda realizowana transakcja posiada zbiór ścisłe określonych stanów i zbiór ścisłe określonych przejść z jednego stanu do drugiego.

koniecznie NAZWAĆ konkretne stany:

- aktywna > częściowo wypełniona > wypełniona>zakończona
- aktywna > nieudana > zakończona
- aktywna > częściowo wypełniona > nieudana > zakończona

Diagram stanów transakcji



10.3. [E] Wielodostęp, problemy wielodostępu

Wielodostęp - pogodzenie działań wielu transakcji. Instrukcje dwóch transakcji przeplatają się i transakcje są wykonywane jednocześnie. Nawet jeżeli każda z dwóch transakcji jest poprawna, przemieszanie ich operacji może spowodować wystąpienie błędów w bazie.

Wielodostęp utrudnia więc zachowanie integralności i spójności bazy danych. Poniżej przedstawiono **5 problemów wielodostępu**:

10.3.1. Utrata zmian

Wynik poprawnie zakończonej operacji modyfikacji może zostać przesłonięty przez modyfikację innego użytkownika.

Czas	T1	T2	balx
t1		begin_trnsaction	100
t2	begin_transaction	read(balx)	100
t3	read(balx)	balx = balx + 100	100
t4	balx = balx - 10	write(balx)	200
t5	write(balx)	commit	90
t6	commit		90

10.3.2. Niezatwierdzone zależności (“brudne dane”)

Gdy jedna transakcja może widzieć pośrednie wyniki tworzone przez inną transakcję, zanim ta zostanie wypełniona.

Innymi słowami: zmodyfikowana przez t1 wartość zostaje przeczytana przez t2. Kiedy t1 wykonuje rollback, zmodyfikowana (błędna) wartość wciąż jest używana w t2

Czas	T3	T4	balx
t1		begin_trnsaction	100
t2		read(balx)	100
t3		balx = balx + 100	100
t4	begin_transaction	write(balx)	200
t5	read(balx)	...	200
t6	balx = balx - 10	rollback	100
t7	write(balx)		190
t8	commit		190

10.3.3. Analiza niespójności

Poprzednie przykłady: transakcje, które modyfikują dane, i ich wzajemny wpływ na siebie może doprowadzić do zniszczenia bazy danych. Jednak niepoprawne wyniki mogą powstawać również w następstwie transakcji, które tylko odczytują dane jeśli zezwolimy na odczytywanie częściowych wyników niewypełnionych transakcji modyfikujących bazę danych.

Problem analizy niespójności pojawia się, gdy transakcja odczytuje kilka wartości z bazy danych, a inna transakcja w tym czasie modyfikuje niektóre z tych wartości (ten problem może wystąpić nawet w

Czas	T5	T6	balx	baly	balz	sum
t1		begin_transaction	100	50	25	
t2	begin_transaction	sum = 0	100	50	25	0
t3	read(balx)	read(balx)	100	50	25	0
t4	balx = balx - 10	sum = sum + balx	100	50	25	100
t5	write(balx)	read(baly)	90	50	25	100
t6	read(balz)	sum = sum + baly	90	50	25	150
t7	balz = balz + 10		90	50	25	150
t8	write(balz)		90	50	35	150
t9	commit	read(balz)	90	50	35	150
t10		sum = sum + balz	90	50	35	185
t11		commit	90	50	35	185

transakcji, która jedynie odczytuje dane bez zapisywania)

10.3.3.1. Analiza niespójności- Mylące odczyty

Transakcja T ponownie odczytuje wartość danej, którą czytała już wcześniej, lecz w międzyczasie inna transakcja zmodyfikowała tę wartość. Transakcja T otrzymuje dwie różne wartości tej samej danej – zjawisko **niepowtarzalnych (mylących) odczytów**.

10.3.3.2. Analiza niespójności - Odczyty fantomowe

Transakcja T wykonuje zapytanie wyszukujące w relacji zbiór krotek spełniających pewien warunek, a potem powtórnie oblicza wynik tego zapytania, ale okazuje się, że nowy wynik zawiera dodatkowe krotki (**fantomy**), które zostały dopisane w międzyczasie przez inną transakcję – **zjawisko odczytów fantomów**

10.4. [E] Co to jest fantom?

co

Fantom - wiersz (krotka), który zostaje wstawiony do tabeli po tym jak transakcja wykonała operację na tej tabeli a przed jej zatwierdzeniem.

10.5. [E] Szeregowalność

Szeregowalność - pojęcie pozwalające rozpoznać wykonania transakcji, które na pewno nie zaburzą spójności bazy. Zadanie szeregowalności polega na rozpoznaniu harmonogramów niesekwencyjnych, które pozwalają wykonywać transakcje współbieżnie, nie dopuszczając do niepożądanych oddziaływań pomiędzy nimi i doprowadzając bazę do stanu, który mógłby równie dobrze powstać w efekcie wykonania sekwencyjnego.

10.6. [E] Jak mają być ułożone transakcje w harmonogramie?

Z definicji harmonogramu:

Ciąg operacji wykonywanych jednocześnie transakcji, w których zachowany jest wewnętrzny porządek operacji każdej transakcji.

10.7. [E] Harmonogram sekwencyjny i niesekwencyjny

10.6.1. Harmonogram sekwencyjny

Harmonogram sekwencyjny składa się z ciągu akcji transakcji, które się nie przeplatają, tzn. najpierw wykonywana jest cała jedna transakcja, potem druga itd.

10.6.2. Harmonogram niesekwencyjny

Harmonogram niesekwencyjny zawiera przeplatające się elementy obu transakcji, ale kolejność w obrębie transakcji jest zachowana. > mniej więcej tak odpowiedziałam i to była odpowiedź na 5.0

W harmonogramie niesekwencyjnym operacje ze zbioru jednocześnie wykonywanych transakcji są przeplecone. Problemy utraty zmian (1), niezatwierdzonych zależności (2), analizy spójności (3) wynikają ze złej organizacji jednoczesnego wykonania. **Wykonanie sekwencyjne** zapobiega występowaniu takich problemów

Zadanie szeregowalności polega na: rozpoznaniu harmonogramów niesekwencyjnych.

10.6.3. [Dodatkowo] Harmonogram szeregowalny

Jeżeli transakcje z pewnego zbioru są wykonywane jednocześnie, to ich **niesekwencyjny harmonogram uznajemy za poprawny**, jeżeli doprowadza on do takiego samego stanu bazy danych, jak pewne wykonanie sekwencyjne tych transakcji. Taki **harmonogram nazywamy szeregowalnym**.

|

Harmonogramy szeregowalne zachowują spójność bazy danych, przy założeniu, że wszystkie ich transakcje zakończą się powodzeniem.

10.7. [E] Harmonogram odtwarzalny

Definicja harmonogramu odtwarzalnego

Harmonogram odtwarzalny: Jeżeli transakcja A odczytuje wartość zapisaną poprzednio przez transakcję B to operacja wypełnienia transakcji B poprzedza operację wypełnienia transakcji A.

Harmonogram S jest odtwarzalny, jeżeli żadna transakcja T w harmonogramie S nie jest zatwierdzona do momentu, aż wszystkie transakcje T', które zapisały element odczytywany przez transakcji T, zostaną zatwierdzone.

- Transakcja T czyta z transakcji T' w harmonogramie S, jeżeli pewien element X jest najpierw zapisywany przez T', a później odczytywany przez T.
- Ponadto, transakcja T' nie powinna zostać anulowana zanim transakcja T odczyta element X i nie powinny występować żadne transakcje zapisujące X po dokonaniu zapisu przez T', a przed wykonaniem odczytu przez T.

10.8. [E2017] Co to jest graf pierwszeństwa?

10.8.1. Z wykładow:

Graf pierwszeństwa

- Przy założeniu o **zapisie na podstawie poprzedniej wartości** (transakcja modyfikuje wartość danej w oparciu o jej poprzednią wartość, którą najpierw musi odczytać), można stworzyć **graf pierwszeństwa (szeregowalności)**, który pozwoli wykryć naruszenie szeregowalności
- Dla harmonogramu S graf pierwszeństwa to graf skierowany
 - $G = (N, E)$; N -zb. wierzchołków, E -zb. krawędzi skierowanych
- Sposób tworzenia
- Jeżeli w grafie pierwszeństwa harmonogramu S istnieje krawędź $T_i \rightarrow T_j$, to w każdym harmonogramie sekwencyjnym S' równoważnym S , T_i musi występować przed T_j .
- Jeżeli graf pierwszeństwa zawiera **cykl**, to kolizji w harmonogramie nie można uszeregować (czyli harmonogram nie jest szeregowalny)

10.8.2. Bardziej szczegółowo ze strony:

http://math.uni.lodz.pl/~cybula/bd/pbd/wyklad_11_12_podstawy_baz_danych_2009_2010.pdf

Algorytm sprawdzania szeregowalności harmonogramu S z blokadami:

- zapisu (READ_LOCK)
- i
- całkowitymi (WRITE_LOCK),

opiera się na konstrukcji grafu zorientowanego G , w którym wierzchołki odpowiadają transakcjom, a krawędzie wyznacza się korzystając z następujących reguł:

1. Jeśli transakcja T_i z harmonogramu S blokuje zapis elementu A $\{READ_LOCK(A)\}$, a T_j jest następna transakcją (o ile taka istnieje), która chce całkowicie zablokować A $\{WRITE_LOCK(A)\}$, to prowadzimy krawędź $T_i \rightarrow T_j$.



T_i	T_j
...	...
$READ_LOCK(A)$	
...	...
	$WRITE_LOCK(A)$

3. Jeśli T_m jest transakcją blokującą zapis A $\{READ_LOCK(A)\}$, po tym, gdy T_i zdjęła swoją całkowitą blokadę $\{UNLOCK(A)\}$, lecz zanim T_j całkowicie zablokowała A $\{WRITE_LOCK(A)\}$, (jeśli T_j nie istnieje, to T_m jest dowolną transakcją blokującą zapis A po odblokowaniu jej przez T_i), to prowadzimy krawędź $T_i \rightarrow T_m$.



T_i	T_m	T_j
$WRITE_LOCK(A)$		
$UNLOCK(A)$		
	$READ_LOCK(A)$...
		[$WRITE_LOCK(A)$]

Szeregowalność danego harmonogramu rozstrzygamy sprawdzając istnienie cykli w tak skonstruowanym grafie (nazywanym również *grafem pierwszeństwa*).

Jeśli G zawiera cykl, to harmonogram S nie jest szeregowalny.

Twierdzenie. Przedstawiony algorytm sprawdzania szeregowalności w poprawny sposób stwierdza szeregowalność harmonogramu.

Dowód poprawności powyższego algorytmu znajduje się m.in. w książce J.D Ullman „Systemy baz danych”.

10.9. [E] Jak szeregować harmonogram

Przy założeniu o zapisie na podstawie poprzedniej wartości (transakcja modyfikuje wartość danej w oparciu o jej poprzednią wartość, którą najpierw musi odczytać), można stworzyć graf pierwszeństwa (szeregowalności), który pozwoli wykryć naruszenie szeregowalności. Dla harmonogramu S graf pierwszeństwa to graf skierowany

$G = (N, E)$; N zb. wierzchołków, E zb. krawędzi skierowanych

- Jeżeli w grafie pierwszeństwa harmonogramu S istnieje krawędź $T_i > T_j$, to w każdym harmonogramie sekwencyjnym S' równoważnym S, T_i musi występować przed T_j .
- Jeżeli graf pierwszeństwa zawiera cykl, to kolizji w harmonogramie nie można uszeregować (czyli harmonogram nie jest szeregowalny)

Zadanie szeregowalności polega na:

Rozpoznaniu harmonogramów niesekwencyjnych, które pozwalają wykonywać transakcje współbieżnie, nie dopuszczając do niepożądanych oddziaływań pomiędzy nimi i doprowadzając bazę do stanu, który mógłby równie dobrze powstać w efekcie wykonania sekwencyjnego

10.10. [E] Prawo pierwszeństwa w harmonogramach

Dla zapewnienia szeregowalności zapobiegającej niespójności wynikającej z oddziaływań pomiędzy wykonywanymi jednocześnie transakcjami, ważne jest wzajemne uporządkowanie operacji **odczytu** i **zapisu** tych transakcji

- Jeżeli dwie transakcje jedynie czytają ten sam element danych, to ich wzajemna kolejność nie jest istotna
- Jeżeli dwie transakcje czytają lub zapisują różne elementy danych, to nie kolidują ze sobą i ich wzajemna kolejność nie jest istotna.
- Jeżeli jedna transakcja zapisuje element danych, a druga go odczytuje lub zapisuje, to kolejność wykonania tych transakcji ma znaczenie.

Wydaje mi się, że to będzie to (slajd 23 transakcje - graf pierwszeństwa):

Graf pierwszeństwa

- Przy założeniu o **zapisie na podstawie poprzedniej wartości** (transakcja modyfikuje wartość danej w oparciu o jej poprzednią wartość, którą najpierw musi odczytać), można stworzyć **graf pierwszeństwa (szeregowalności)**, który pozwoli wykryć naruszenie szeregowalności
- Dla harmonogramu S graf pierwszeństwa to graf skierowany
 - $G = (N, E)$; N-zb.wierzchołków, E-zb. krawędzi skierowanych
- Sposób tworzenia
- Jeżeli w grafie pierwszeństwa harmonogramu S istnieje krawędź $T_i \rightarrow T_j$, to w każdym harmonogramie sekwencyjnym S' równoważnym S, T_i musi występować przed T_j .
- Jeżeli graf pierwszeństwa zawiera **cykl**, to kolizji w harmonogramie nie można uszeregować (czyli harmonogram nie jest szeregowalny)

A może to coś z grafem pierwszeństwa? (albo nie ma tego w slajdach)

Wydaje mi się, że tu chodzi właśnie o graf pierwszeństwa. To szło mniej więcej tak, że jeśli w grafie istnieje krawędź $T_i > T_j$ to w każdym równoważnym grafie T_i występuje przed T_j , no i T_i musi wykonać się przed T_j (ale tu bym prosił potwierdzenie). Jak graf zawiera cykl to harmonogramu nie da się uszeregować.

10.11. [E] Harmonogramy równoważne

Różnią się kolejnością poszczególnych operacji, ale przekształcające bazę do takiego samego stanu.

Harmonogramy równoważne						
Czas	T7	T8	T7	T8	T7	T8
t1	begin_transaction		begin_transaction		begin_transaction	
t2	read(balx)		read(balx)		read(balx)	
t3	write(balx)		write(balx)		write(balx)	
t4		begin_transaction		begin_transaction		begin_transaction
t5		read(balx)		read(balx)		read(baly)
t6		write(balx)		commit		write(baly)
t7			read(baly)			commit
t8				write(balx)		
t9				commit		
t10					read(baly)	
t11					write(baly)	
t12					commit	

(a)

(b)

(c)

10.12. [E] Techniki kontroli wielodostępu, blokady - podejście pesymistyczne i optymistyczne.

[E2017] - Jakie są metody na zapewnienie szeregowalności? Czym się różnią metody pesymistyczne od optymistycznych?

Szeregowalność można zapewnić na kilka sposobów: Metody optymistyczne i metody pesymistyczne.

10.12.1. Metody pesymistyczne (zachowawcze)

10.12.1.1. Blokady

Blokada – zmienna skojarzona z każdą daną w bazie określająca dostępność danej ze względu na możliwość wykonania na niej określonej operacji.

Metoda oparta na blokadach polega na tym, że transakcja, która odwołuje się do danych blokuje innym transakcjom dostęp do danych.

Rozróżniamy dwa typy blokad:

- Blokada do odczytu R (read locks) – jeżeli transakcji zostanie przyznana blokada read locks jednostki danych, to transakcja może odczytywać wartość tej jednostki, ale nie może jej zmieniać
- Blokada do zapisu W (write locks) - jeżeli transakcji zostanie przyznana blokada write locks jednostki danych, to transakcja może zarówno odczytywać wartość tej jednostki, jak i ją zmieniać

10.12.1.2. Znaczniki czasowe

Do tworzenia harmonogramu wykorzystuje się znaczniki czasowe (timestamps), które oznaczają czas wejścia transakcji w trzy fazy: Start, Validation, Finish.

Jeśli transakcja t1 ma występować przed transakcją t2

$\text{finish}(t1) < \text{start}(t2)$ albo

$\text{start}(t2) < \text{finish}(t1) < \text{validation}(t2)$, przy założeniu, że t2 nie odczytuje danych zapisywanych przez t1.

10.12.2. Metody optymistyczne

Metody optymistyczne:

Walidacja. Wykonanie transakcji podzielone na trzy fazy:

- transakcja jest wykonywana i zapisuje dane w zmiennych lokalnych
- transakcja wykonuje test walidacji, który sprawdza, czy zmienne lokalne mogą być zapisane bez naruszenia integralności BD
- jeśli walidacja przebiegła pomyślnie zmienne lokalne są zapisywane w BD, w przeciwnym wypadku następuje rollback

Algorytmy optymistyczne zarządzania współbieżnością transakcji przeprowadzają transakcje przez trzy stany:

- Faza odczytu: Transakcje czytają dane z bazy danych. Wprowadzane modyfikacje są przechowywane w lokalnych obszarach roboczych transakcji
- Faza walidacji: Wykonywana jest walidacja uszeregowalności transakcji. Transakcje niespełniające kryterium uszeregowalności są wycofywane i restartowane
- Faza zapisu: Jeżeli faza walidacji zakończy się pomyślnie, modyfikacje transakcji są wprowadzane do bazy danych

10.13. [E] Rodzaje blokad

Blokada - zmienna skojarzona z każdą daną w bazie określająca dostępność danej ze względu na możliwość wykonywania na niej określonej operacji

Dwa typy blokad

- **Blokada dzielona** transakcja może odczytywać wartość jednostki, ale nie może jej zmienić. Umożliwia innym transakcjom odczyt, ale nie modyfikację wartości.
- **Blokada wyłączna** transakcja może zarówno odczytywać wartość jednostki, jak i ją zmieniać. Inne transakcje nie mają możliwości odczytu ani modyfikacji.

Z poprzedniego podpunktu: Rozróżniamy dwa typy blokad:

- Blokada do odczytu R (read locks) – jeżeli transakcji zostanie przyznana blokada read locks jednostki danych, to transakcja może odczytywać wartość tej jednostki, ale nie może jej zmieniać
- Blokada do zapisu W (write locks) - jeżeli transakcji zostanie przyznana blokada write locks jednostki danych, to transakcja może zarówno odczytywać wartość tej jednostki, jak i ją zmieniać

Uwaga: blokada wyłączna == write lock, dzielona == read lock

10.14. [E] Co to jest rozszerzenie/redukcja blokady?

Rozszerzanie blokady z dzielonej na wyłączną

Redukcja blokady z wyłącznej na dzieloną

10.15. [E] Kiedy transakcja spełnia warunki blokowania dwufazowego?

Transakcja przestrzega protokołu blokowania dwufazowego (2PL) jeżeli wszystkie operacje zakładania blokady znajdują się w niej przed pierwszą operacją zdjęcia blokady.

Protokół ścisłego blokowania dwufazowego (Strict 2PL):

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

faza wzrostu zakładanie i ew. rozszerzanie blokad

faza zmniejszania zdejmowanie i ew. redukowane blokad

Zastanówmy się czy protokół Strict-2PL generuje wyłącznie plany wolne od anomalii.

- **Niezatwierdzony odczyt** nie może zajść, ponieważ aby zmienić wartość obiektu transakcja musi założyć blokadę X i od tej chwili do momentu zwolnienia blokady na koniec wykonywania transakcji żadna inna transakcja nie ma dostępu do zablokowanego obiektu (nawet w celu jego odczytu).
- **Niepowtarzalny odczyt** też nie może zajść, ponieważ przy pierwszym odczytzie transakcja zakłada blokadę S (lub X) na odczytywany obiekt i zwalnia tę blokadę dopiero na koniec. W międzyczasie żadna inna transakcja nie uzyska blokady X na tym obiekcie aby go zmienić.
- **Nadpisywanie niezatwierdzonych zmian** też nie może mieć miejsca, ponieważ zapisanie obiektu wymaga wcześniejszego założenia blokady X i do czasu zakończenia transakcji żadna inna transakcja nie może ani odczytać ani zapisać tego obiektu.

Podsumujmy wynik naszego rozumowania w postaci twierdzenia:

Protokół Strict-2PL nie dopuszcza do powstania anomalii niezatwierzonego odczytu, niepowtarzalnego odczytu i nadpisywania niezatwierdzonych danych.

Twierdzenie 2

“Jeżeli wszystkie transakcje spełniają protokół dwufazowego blokowania, to wszystkie harmonogramy niesekwencyjne są szeregowalne”

