

13. Теория на нормализацията

Лекционен курс "Бази от данни"

Въведение

Въпросът, на който ще търсим отговор в този раздел:

- Дадена е някаква съвкупност от данни;
- Тя трябва да бъде представена в БД;
- Как ще решим каква е подходящата логическа структура за тези данни?

По-прецизирано:

- Какви релации са необходими?
- Какви атрибути и взаимоотношения ще имат те?

Това е проблем на логическото проектиране на БД.

Нормализация: процес на подреждане на данните в релации.

- Нормализацията се базира на анализ на функционалните зависимости между атрибутите;
- Тя е процесът на декомпозиране на релации с аномалии в по-добре структурирани такива;
- Добре структурираната релация съдържа минимално излишество и позволява добавяне, промяна и изтриване на данни без „странични ефекти“;
- Нормализацията трябва да премахне излишеството, но не и за сметка на интегритета;
- Резултатът е по-добра организираност и по-ефективна употреба на физическо пространство.

Нормализацията не е винаги най-доброто решение. Напр., при **data warehouse** системите е налице напълно различен подход.

Цели на нормализацията

1. Да избягва излишеството като съхранява всеки факт в базата данни само веднъж.
2. Да постави данните във форма, която е по-удачна за акуратната им промяна.
3. Да избягва някои „аномалии на промените“.
4. Да улесни прилагането на ограниченията върху данните.
5. Да избягва ненужно програмиране – би се наложило за управление на ненормализирани данни (чрез тригери и съхранени процедури), а това би понижило ефективността на СУБД.

Нормални форми

Теорията на нормализацията се изгражда около концепцията за нормални форми.

Нормалната форма - една добре дефинирана стандартна мярка за степента на нормализация, която една релация притежава.

Нормализацията се извършва на степени - всяка следваща нормална форма означава по-висока степен на нормализация.

Въпреки че нормализацията обикновено е желателна - възможно е да се стигне до свръхнормализация (нежелателна).

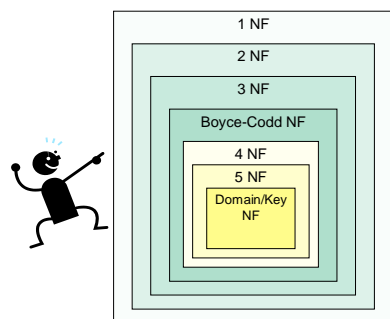
Колкото по-висока е степента на нормализация:

- Толкова повече са таблиците в БД;
- Това прави схемата по-сложна;
- Може да намали ефективността и производителността на базата данни.

В оригинала на своите разработки Codd дефинира:

- Първа нормална форма (1NF)
- Втора нормална форма (2NF)
- Трета нормална форма (3NF)

Класификация на нормалните форми



Всяка от тези нормални форми е създадена да се справи със специфични типове проблеми:

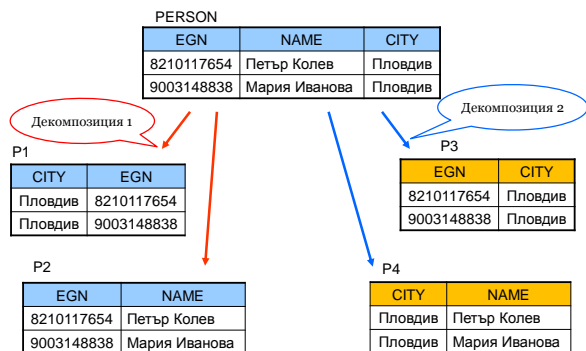
- Първа нормална форма – атомарни данни;
- Втора нормална форма – базирана на функционалните зависимости;
- Трета нормална форма – базирана на функционалните зависимости;
- Boyce/Codd нормална форма – базирана на функционалните зависимости;
- Четвърта нормална форма – базирана на мулти-стойности (multi-valued) зависимости;
- Пета нормална форма – базирана на JOIN зависимости;
- Domain/Key нормална форма – базирана на дефиницията на домейни и ключове.

Декомпозиция без загуба на информация (nonloss decomposition)

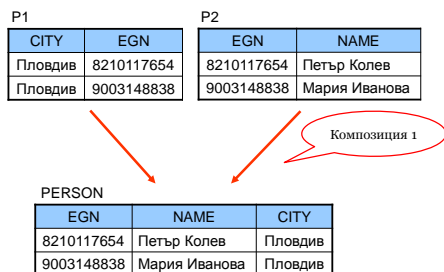
Нормализиращата процедура:

- Включва декомпозиция на дадена входна релация на нови релации;
- Изисква се тя да бъде **обратима** - при този процес да не се губи информация;
- Проблемът за загуба на информация - свързан е с функционалните зависимости.

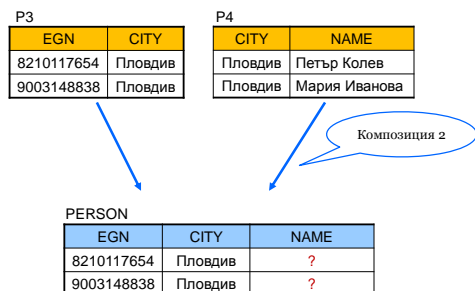
Пример - декомпозиция:



Пример - композиция 1:



Пример - композиция 2:



Кое е това, което прави първата декомпозиция без загуба, а втората със загуба?

- Декомпозиционният оператор е projection – всяка от релациите P1, P2, P3 и P4 са проекции на PERSON, така че нормализационната процедура всъщност е projection;
- В първия случай - ако извършим обратната операция join получаваме оригиналната релация;
- Във втория случай - не получаваме оригиналната релация, следователно губим информация.

Интересен е следният въпрос:

- Нека R_1, R_2 са проекции на една релация R ;
- Нека също R_1 и R_2 съдържат заедно всички атрибути на R ;
- Какви условия трябва да са удовлетворени, за да се гарантира, че обратното свързване (join) на двете проекции ще ни върне оригиналната релация?

Това са функционалните зависимости:

- В примера - релацията PERSON удовлетворява несъкратимото множество от ФЗ:
 $\{ EGN \rightarrow NAME, EGN \rightarrow CITY \}$

Теорема на Хийт (Heath):

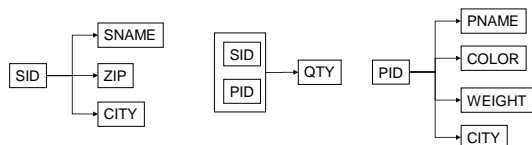
- Нека $R \{A, B, C\}$ е релация, където $\{A, B, C\}$ е множеството на атрибутите;
- Ако R удовлетворява ФЗ $A \rightarrow B$, тогава R е еквивалентна на сливането (join) на нейните проекции $\{A, B\}$ и $\{A, C\}$.

Ако заместим $A = EGN$, $B = NAME$, $C = CITY$ теоремата потвърждава, че релацията PERSON може да бъде декомпозирана без загуба на нейните проекции $\{EGN, NAME\}$ и $\{EGN, CITY\}$.

В същото време виждаме, че PERSON не може да бъде декомпозирана без загуба на проекциите $\{EGN, CITY\}$ и $\{CITY, NAME\}$, защото губим функционалната зависимост $EGN \rightarrow NAME$, а и не е налице ФЗ $CITY \rightarrow NAME$.

Диаграми на ФЗ

Нека R е релация и нека F е някакво несъкратимо множество от ФЗ, които се прилагат в R - удобно е F да се представя с помощта на диаграма на ФЗ.



- Вижда се от диаграмите, че стрелките тръгват от първичния ключ – за всяка стойност на първичния ключ винаги има една стойност от елементите, до които отива стрелката;
- Ако има други стрелки ще възникнат трудности;
- По този начин съвсем информативно нормализиращата процедура може да бъде характеризирана по следния начин: **процедура за елиминиране на стрелките, които не излизат от КК.**

ФЗ са семантични понятия:

- ФЗ са специален вид ограничения за цялостност - като такива, определено, те са семантични понятия;
- Познаването на ФЗ е част от процеса за разбиране на значението на данните.

Напр., фактът, че S удовлетворява ФЗ SID → CITY означава, че всеки доставчик се намира в точно един град – да погледнем на този факт от един друг аспект:

- Това е едно ограничение от реалния свят, което БД трябва да отразява;
- Понеже е част от семантиката на ситуацията – по някакъв начин трябва да се следи от БД;
- По някакъв начин трябва да се специфицира в дефиницията на БД – за да може СУБД да го налага и управлява;
- Спецификациите на ФЗ се задават в декларациите на БД.

Ненормализирани данни

TRADERS

ID	NAME	LOCATIONS
1	Metro C&C	София, Пловдив, Пазарджик
2	Технополис	Пловдив, Варна, Бургас
3	Shell	Пловдив, Асеновград

Повтарящи се групи от данни

EMPLOYEES

ID	NAME	CHILD1	BIRTHDATE1	CHILD2	BIRTHDATE2
101	Иван				
102	Илия				
103	Петър				

Повтарящи се групи от данни

Първа нормална форма

Една релация е в **първа нормална форма** (First Normal Form - 1NF), ако и само ако тя удовлетворява ограничението всички стойности на данните да са **атомарни** (неразложими) – т.е. атрибутите (колоните) да имат единична (скаларна) стойност.

Това означава, че по дефиниция всяка релация от релационния модел е в първа нормална форма - прилежащите домейни могат да се дефинират само върху прости типове, т.е. да съдържат скаларни стойности.

Аномалии на промените

Промяната на данните в някои релации може да доведе до нежелателни последиствия - наречени аномалии на промените.

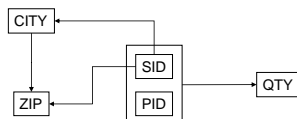
За демонстрация нека предположим, че всички данни за доставчиците и доставките се съдържат в една релация:

FIRST(SID, CITY, ZIP, PID, QTY)
PRIMARY KEY (SID, PID)

За демонстрацията добавяме допълнително ограничение:

CITY → ZIP

Диаграма на функционалните зависимости в релацията FIRST:



Релация FIRST

SID	CITY	ZIP	PID	QTY
S1	Пловдив	4000	P1	300
S1	Пловдив	4000	P2	200
S1	Пловдив	4000	P3	400
S1	Пловдив	4000	P4	200
S1	Пловдив	4000	P5	100
S1	Пловдив	4000	P6	100
S2	София	1000	P1	300
S2	София	1000	P2	400
S3	София	1000	P2	200
S4	Пловдив	4000	P2	200
S4	Пловдив	4000	P4	300
S4	Пловдив	4000	P5	400



Аномалия на вмъкването



SID	CITY	ZIP	PID	QTY
S1	Пловдив	4000	P1	300
S1	Пловдив	4000	P2	200
S1	Пловдив	4000	P3	400
S1	Пловдив	4000	P4	200
S1	Пловдив	4000	P5	100
....
S5	Варна	9000	?	?

- Да се опитае да добавим доставчик;
- За да добавим новия запис доставчикът трябва да е направил поне една доставка, защото иначе няма да имаме стойност за целия първичен ключ;
- Т. е. не можем да представим факта, че доставчикът S5 се намира във Варна.

Аномалия на изтриването



SID	CITY	ZIP	PID	QTY
S1	Пловдив	4000	P1	300
....
S2	София	1000	P1	300
S2	София	1000	P2	400
S3	София	1000	P2	200
S4	Пловдив	4000	P2	200
S4	Пловдив	4000	P4	300
S4	Пловдив	4000	P5	400

- Да предположим, че по някаква причина се анулира доставката на доставчик S3 и трябва да изтрием този запис;
- С това ще изтрием и информацията, че доставчикът S3 се намира в София.

Аномалия на актуализацията

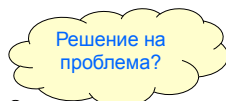


SID	CITY	ZIP	PID	QTY
S1	Пловдив	4000	P1	300
S1	Пловдив	4000	P2	200
S1	Пловдив	4000	P3	400
S1	Пловдив	4000	P4	200
S1	Пловдив	4000	P5	100
S1	Пловдив	4000	P6	100
...
S4	Пловдив	4000	P5	400

- Да предположим, че доставчикът S1 се премести от Пловдив в Ямбол;
- Ще се сблъскаме с проблема да намерим всеки запис, свързващ S1 с Пловдив, и да го променим;
- Това води до риск от възникване на несъответствия – може да пропуснем да отразим промяната в някой запис и данните ще станат неконсистентни – един запис ще свързва S1 с Пловдив (пропуснатият), а останалите ще твърдят, че S1 е в Ямбол.

Забележка:

- Истинският проблем тук е, че релацията FIRST съдържа твърде много информация на едно място;
- Така изтривайки един ред изтриваме *твърде много*, при опит за добавяне данните са ни *твърде малко*, при промяна трябва да променяме на повече от едно място;
- По-точно релацията FIRST съдържа данни, отнасящи се до доставчиците, както и такива, отнасящи се до доставките – и така изтриването на доставката предизвиква изтриване на данните за доставчика;
- Решението на този проблем е да “разпределим” данните – да разположим тези за доставчика в една релация, а доставките – в друга;
- Така можем да характеризираме нормализиращата процедура като “разпределяща” процедура: *поставяне на логически разделени данни в отделни релации.*





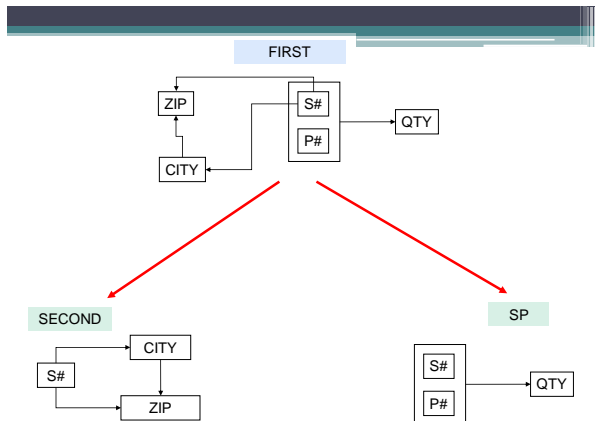
**Декомпозиция на релацията
(без загуба на информация!)**

Заменяме релацията FIRST със следните две релации:

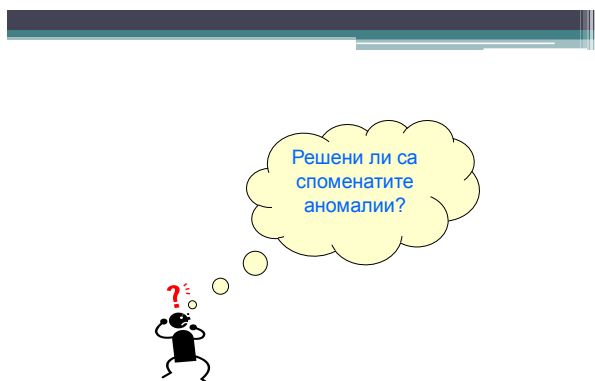
SECOND (SID, CITY, ZIP)

и

SP (SID, PID, QTY)



SECOND			SP		
SID	CITY	ZIP	SID	PID	QTY
S1	Пловдив	4000	S1	P1	300
S2	София	1000	S1	P2	200
S3	София	1000	S1	P3	400
S4	Пловдив	4000	S1	P4	200
S5	Варна	9000	S1	P5	100
			S1	P6	100
			S2	P1	300
			S2	P2	400
			S3	P2	200
			S4	P2	200
			S4	P4	300
			S4	P5	400



Аномалия на вмъкването

SECOND

SID	CITY	ZIP
S1	Пловдив	4000
S2	София	1000
S3	София	1000
S4	Пловдив	4000
S5	Варна	9000



Можем да добавим информацията, че доставчикът S5 се намира във Варна, макар че той в момента все още не е доставял части.

Аномалия на изтриването

SP

SID	PID	QTY
S1	P1	300
S1	P2	200
S1	P3	400
S1	P4	200
S1	P5	100
S1	P6	100
S2	P1	300
S2	P2	400
S3	P2	200
S4	P2	200
S4	P4	300
S4	P5	400



Можем да изтрием доставката на детайл P2 от доставчик S3 без да загубим информацията, че S3 се намира в София.

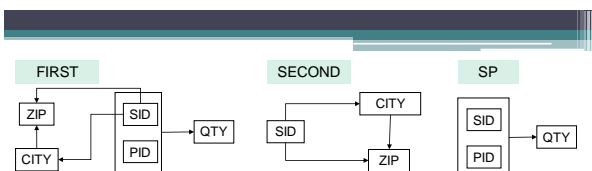
Аномалия на актуализацията

SECOND

SID	CITY	ZIP
S1	Ямбол	8600
S2	София	1000
S3	София	1000
S4	Пловдив	4000
S5	Варна	9000



В новата структура градът на доставчика се появява веднъж, не многократно, защото за всеки доставчик имаме точно по един ред. С други думи, излишеството SID - CITY беше елиминирано. Така сега можем да променим града на S1 от Пловдив на Ямбол, променяйки го само на едно място.



Сравнявайки FIRST със SECOND и SP виждаме, че ефектът от декомпозицията е премахване на зависимостите, които **не са несъкратими** и това реши трудностите.

Интуитивно можем да кажем, че в релацията FIRST атрибутът CITY не описва обект, идентифициран от първичния ключ (SID, PID), наречен *доставка*. Вместо това той описва *доставчик*, участващ в доставката. Същото важи и за ZIP, разбира се. Смесването на двата вида информация в една релация е предизвикало проблемите.

Втора нормална форма

Една релация е във втора нормална форма (Second Normal Form - 2NF), ако и само ако:

- Тя е в 1NF;
- Всеки неключов атрибут е несъкратимо зависим от първичния ключ, т.е. зависи от целия първичен ключ, т.е. няма частични зависимости.

Тази дефиниция се отнася основно за релации, които имат съставен ключ. Ако ключът е от един атрибут, то релацията автоматично е във втора нормална форма.

Релациите SECOND и SP са във втора нормална форма, но FIRST не е. Релация, която е в 1NF и не е в 2NF може винаги да бъде доведена до еквивалентна колекция от релации в 2NF – този процес се състои от замяна на релацията в 1NF с подходящи проекции; оригиналната релация може да бъде възстановена от колекцията от проекции чрез сливане.

Обобщение

Първата стъпка от нормализационната процедура е намирането на проекциите с цел елиминирание на съкратимите ФЗ.
Нека е дадена релация R такава, че:

R (A, B, C, D)
PRIMARY KEY (A, B),
 $A \rightarrow D$

Нормализационната процедура препоръчва замяната на R с две нейни проекции R1 и R2:

R1 (A, D)
PRIMARY KEY (A)
R2 (A, B, C)
PRIMARY KEY (A, B)
FOREIGN KEY (A) REFERENCES R1

Релацията R може да бъде възстановена чрез сливане по съответните общи атрибути (външен-първичен ключ).

Да се върнем отново на примера SECOND-SP, при който все още възникват проблеми.

Релацията SP всъщност е в трета нормална форма. Но SECOND все още страда от зависимости между неключови атрибути.

Зависимостта на ZIP от SID, въпреки че е функционална и несъкратима, е транзитивна – чрез CITY. Всяка стойност на SID определя стойност на CITY, която от своя страна определя стойност на ZIP.

Транзитивните зависимости водят отново до аномалии на промените.

Аномалия на вмъкването

SID	CITY	ZIP
S1	Ямбол	8600
S2	София	1000
S3	София	1000
S4	Пловдив	4000
S5	Варна	9000
?	Бургас	8000



- Да се опитаме регистрираме факта, че конкретен град има определен пощенски код;
- За да добавим новия запис трябва да имаме поне един доставчик от този град;
- Т. е. не можем да представим факта, че Бургас има пощенски код 8000.

Аномалия на изтриването

SID	CITY	ZIP
S1	Ямбол	8600
S2	София	1000
S3	София	1000
S4	Пловдив	4000
S5	Варна	9000



- Ако изтрием доставчика S1 ще изтрием не само факта, че той се намира в Ямбол, но и факта, че на град Ямбол пощенският код е 8600;
- Отново тук релацията съдържа твърде много данни – относно доставчика, но и относно града;
- Отново решението е “разпределяне” на данните – тези за доставчиците в една релация, а тези за градовете – в друга.

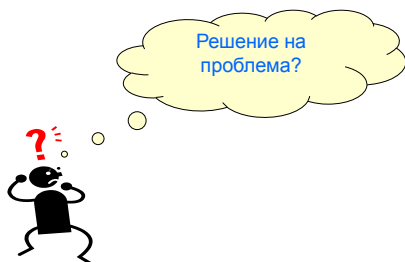
Аномалия на актуализирането

SID	CITY	ZIP
S1	Ямбол	8600
S2	София	1000
S3	София	1000
S4	Пловдив	4000
S5	Варна	9000



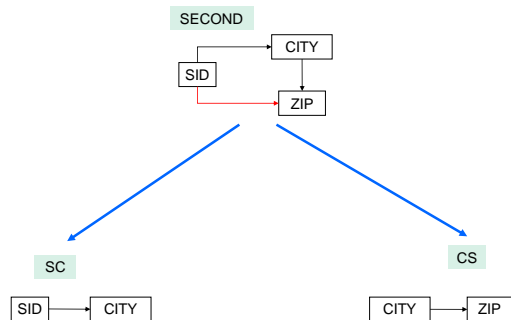
- Пощенският код на даден град се повтаря в SECOND. Така, ако искаме да сменим пощенския код на София, ще трябва да променим на повече от едно място;
- Ще се сблъскаме с проблема да намерим всеки запис, свързващ София с пощенски код 1000;
- Това води до риск от възникване на несъответствия – може да пропуснем да отразим промяната в някой запис и данните ще станат неконсистентни.

Тези аномалии са причинени от
транзитивните зависимости между
неключовите атрибути!



Декомпозиция на релацията
(без загуба на информация)!

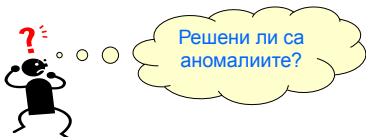




SC		CS	
SID	CITY	CITY	ZIP
S1	Ямбол	Ямбол	8600
S2	София	София	1000
S3	София	Пловдив	4000
S4	Пловдив	Варна	9000
S5	Варна		

Вижда се, че декомпозицията е обратима чрез сливане на релациите върху атрибута CITY.

SC		CS	
SID	CITY	CITY	ZIP
S1	Ямбол	Ямбол	8600
S2	София	София	1000
S3	София	Пловдив	4000
S4	Пловдив	Варна	9000
S5	Варна		



SC

CS

SID	CITY	CITY	ZIP
S1	Ямбол	Ямбол	8600
S2	София	София	1001
S3	София	Пловдив	4000
S4	Пловдив	Варна	9000
S5	Варна	Бургас	8000



Трета нормална форма

Една релация е в трета нормална форма (3NF - Third normal form), ако и само ако:

- Тя е във 2NF;
- Всеки неключов атрибут е нетранзитивно зависим от първичния ключ.

Обобщение

Втората стъпка от нормализационната процедура е намирането на проекциите с цел елиминирание на транзитивните зависимости. Нека е дадена релация R такава, че:

R (A, B, C)
 PRIMARY KEY (A)
 $B \rightarrow C$

Нормализационната процедура препоръчва
замяната на R с две нейни проекции R1 и R2:

R1 (B, C)
PRIMARY KEY (B)

R2 (A, B)
PRIMARY KEY (A)
FOREIGN KEY (B) REFERENCES R1

Релацията R може да бъде възстановена чрез
сливане по съответните външен-първичен ключ.

Нормална форма на Boyce-Codd

Оригиналната дефиниция на Codd за 3NF
страда от някои неадекватности.

По-точно не е удовлетворителна в случаите,
когато една релация:

- Има повече от един ключ-кандидат;
- Някои от ключовете-кандидати са съставни;
- Ключовете-кандидати се препокриват - имат поне един общ атрибут.

Оригиналната дефиниция на третата нормална
форма може да бъде заменена чрез една по-
строга дефиниция - за по-общия случай.

Понеже всъщност става дума за нова нормална
форма, беше въведено и ново наименование за
нея – BCNF.

Ако не са изпълнени горните условия двете
дефиниции са еквивалентни.

FACNUM	SUBJECT	TEACHER
021039	Бази от данни	Стоянов
021039	Алгебра	Желязков
021037	Алгебра	Миховски
021241	Бази от данни	Хаджиколева
021243	Алгебра	Желязков
021244	Алгебра	Миховски

Да разгледаме релацията STUD_TEACHERS. Да предположим, че логиката ѝ е следната:

- всеки студент (FACNUM) може да изучава няколко дисциплини (SUBJECT);
- всяка дисциплина може да се преподава от няколко преподавателя (TEACHER);
- всеки преподавател може да е такъв само по една от дисциплините.

- първичен ключ: (FACNUM, SUBJECT)
- алтернативен ключ: (FACNUM, TEACHER)
- функционални зависимости: TEACHER → SUBJECT

FACNUM	SUBJECT	TEACHER
021039	Бази от данни	Стоянов
021039	Алгебра	Желязков
021037	Алгебра	Миховски
021241	Бази от данни	Хаджиколева
021243	Алгебра	Желязков
021244	Алгебра	Миховски

- ✓ Релацията е в първа нормална форма по дефиниция;
- ✓ Релацията е във втора нормална форма, защото неключовите атрибути зависят от целия първичен ключ;
- ✓ Релацията е в трета нормална форма, защото няма транзитивни зависимости между неключови атрибути.

FACNUM	SUBJECT	TEACHER
021039	Бази от данни	Стоянов
021039	Алгебра	Желязков
021037	Алгебра	Миховски
021241	Бази от данни	Хаджиколева
021243	Алгебра	Желязков
021244	Алгебра	Миховски
?	Софт. технологии	Стоянова

Но въпреки това за тази релация биха се проявили аномалии на промените.

- ✓ Да предположим, че студент с номер 021039 прекъсне обучението си. Ако изтрием неговите записи ще загубим факта, че Стоянов преподава по Бази от данни.
- ✓ Как бихме отразили факта, че Стоянова преподава по Софтуерни технологии? Не можем преди поне един студент да е записал тази дисциплина - аномалия на възмъването.

Подобни ситуации водят до дефиницията на нормалната форма на Boyce-Codd.

Както в предишните примери тази релация може да бъде декомпозирана на две релации, при които аномалиите да не се проявяват.

STUD_TEACH (FACNUM, TEACHER)

Key: (FACNUM, TEACHER)

FACNUM	TEACHER
021039	Стоянов
021039	Желязков
021037	Миховски
021041	Хаджиколева
021043	Желязков
021044	Миховски

SUBJ_TEACH (SUBJECT, TEACHER)

Key: TEACHER

TEACHER	SUBJECT
Стоянов	Бази от данни
Желязков	Алгебра
Миховски	Алгебра
Хаджиколева	Бази от данни
Желязков	Алгебра
Миховски	Алгебра
Стоянова	Софт. технологии

Дефиниция: една релация R е в BCNF, ако и само ако:

- Всяка нетривиална, ляво-несъкратима ФЗ има ключ-кандидат като детерминант
- т.е.
- Всеки детерминант е ключ-кандидат.

Да отбележим – става въпрос за ключове-кандидати, а не точно за първичен ключ.

BCNF-дефиницията е концептуално по-проста в сравнение със старата трета нормална форма - тя няма явна референция към:

- първата и втората нормална форма;
- нито към концепцията на транзитивната зависимост.

Цели на BCNF:

- Да гарантира, че атрибут, който определя стойността на някой или всички не-ключови атрибути от таблицата трябва да бъде кандидат-ключ за тази таблица;
- Да гарантира, че таблицата представя точно един обект.

- BCNF е малко по-строга от трета нормална форма по отношение на ситуацията, когато една таблица има повече от един атрибут, който може да бъде първичен ключ;
- Дефинирана е за точно определяне на всички кандидат-ключове, като по този начин може да бъде гарантирана липсата на транзитивни зависимости - като следствие от това таблицата да не е субект на аномалиите на промените.

Да разгледаме таблицата OrderDetails, съдържаща три детерминанта:

- OrderID, LineNumber
- OrderID, ItemID
- ItemID

За да бъде таблицата в BCNF нормална форма трябва да гарантираме, че всички детерминанти са кандидат-ключове.

OrderDetails

OrderID	LineNumber	ItemID	Item	Qty	Price
1	1	23	Хладилник	2	450
1	2	57	Пералня	3	500
1	3	22	Готварска печка	1	400
2	1	37	Велосипед	10	130
2	2	63	Телевизор	7	300
3	1	85	Бойлер	3	150
4	1	63	Телевизор	1	300
4	2	53	Миксер	12	60

OrderDetails

OrderID	LineNumber	ItemID	Item	Qty	Price
1	1	23	Хладилник	2	450
1	2	57	Пералня	3	500
1	3	22	Готварска печка	1	400
2	1	37	Велосипед	10	130
2	2	63	Телевизор	7	300
3	1	85	Бойлер	3	150
4	1	63	Телевизор	1	300
4	2	53	Миксер	12	60

- ItemID не е ключ-кандидат, но въпреки това определя Item и Price;
- Това означава, че Item и Price са в транзитивна зависимост от двата кандидат-ключа;
- Следователно тези две полета ще трябва да бъдат премахнати от таблицата;
- След като тези полета бъдат премахнати таблицата вече е в BCNF.

OrderDetails

OrderID	LineNumber	ItemID	Qty
1	1	23	2
1	2	57	3
1	3	22	1
2	1	37	10
2	2	63	7
3	1	85	3
4	1	63	1
4	2	53	12

Повечето таблици, които са в BCNF, няма да имат нужда от по-висока степен на нормализация.

Обаче, процесът на нормализация ще трябва да продължи, ако таблицата съдържа мулти-стойностни зависимости.

Четвърта нормална форма

1. Една релация е в четвърта нормална форма (4NF - Fourth normal form), ако и само ако:
 - Тя е във BCNF;
 - Не съдържа мулти-стойностни зависимости.
2. Релация е в 4NF ако и само ако за съществуваща мулти-стойностна зависимост $X \twoheadrightarrow Y$, X е суперключ.

Пример

EmpName	Language	Certificate
Петър Станев	Английски	
Ивайло Колев	Немски	MS SQL Server 2005
Ивайло Колев	Руски	Java
Ивайло Колев	Английски	
Станимира Василева	Английски	Java
Станимира Василева		C++ Builder 7
Никола Петров	Френски	
Никола Петров	Испански	MS SQL Server 2005
Димитър Иванов		

Налице са следните мулти-стойностни зависимости:

- EmpName \twoheadrightarrow Language
- EmpName \twoheadrightarrow Certificate

Пример

Таблица с МСЗ представя повече от един обект. За да постигнем 4NF трябва да декомпозираме таблицата така, че да премахнем мулти-стойностните зависимости.

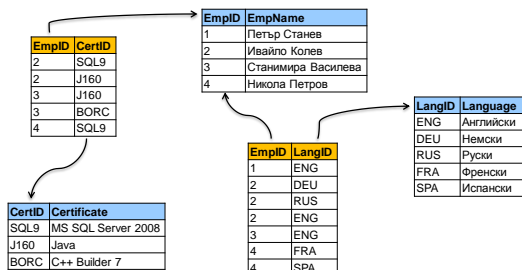
Атрибути, не участващи в мулти-стойностни зависимости, могат да участват в новата релация – напр. дата на придобиване на сертификата.

EmpName	Certificate
Петър Станев	
Ивайло Колев	MS SQL Server 2005
Ивайло Колев	Java
Станимира Василева	Java
Станимира Василева	C++ Builder 7
Никола Петров	MS SQL Server 2005

EmpName	Language
Петър Станев	Английски
Ивайло Колев	Немски
Ивайло Колев	Руски
Ивайло Колев	Английски
Станимира Василева	Английски
Никола Петров	Френски
Никола Петров	Испански

Пример

Но освен това, имаме налице две взаимоотношения много-към-много и ако искаме да бъдем докрай прецизни, схемата ще съдържа повече от тези две релации.



Пета нормална форма

Една релация е в пета нормална форма, ако и само ако всяка нетривиална JOIN зависимост, която е в сила за **R**, се базира на кандидат-ключовете на **R**.

Наричана е още **Projection-Join Normal Form**. Проекцията е нова релация, съдържаща подмножество от атрибутите на оригиналната релация. Когато правилно формираните проекции бъдат съединени (join) трябва да генерират същия набор от данни, който се е съдържал в оригиналната релация.

Пример 1

Релацията съдържа данни за:

- агенти, които са представители на производители;
- производители, които произвеждат продукти;
- агентите, които продават тези продукти.

AGENT	COMPANY	PRODUCT
Smith	Ford	car
Smith	Ford	truck
Smith	GM	car
Smith	GM	truck
Jones	Ford	car
Jones	Ford	truck
Brown	Ford	car
Brown	GM	car
Brown	Toyota	car
Brown	Toyota	bus

Виждат се множеството повторения на данни в тази таблица.

AGENT	COMPANY	PRODUCT
Smith	Ford	car
Smith	Ford	truck
Smith	GM	car
Smith	GM	truck
Jones	Ford	car
Jones	Ford	truck
Brown	Ford	car
Brown	GM	car
Brown	Toyota	car
Brown	Toyota	bus

5NF

AGENT	COMPANY	COMPANY	PRODUCT
Smith	Ford	Ford	car
Smith	GM	Ford	truck
Jones	Ford	GM	car
Brown	Ford	GM	truck
Brown	GM	Toyota	car
Brown	Toyota	Toyota	bus

AGENT	PRODUCT
Smith	car
Smith	truck
Jones	car
Jones	truck
Brown	car
Brown	bus

Разликата е следната: ако добавим нов агент, който продава X продукти за Y компании, където всяка от тези компании произвежда всеки от тези продукти, то добавяме X+Y записи, а в предната схема – X*Y.

Пример 2

Релацията съдържа следните данни:

- проекти, по които работят служителите;
- мениджърите на служителите;
- мениджърите на проектите.

Първичен ключ: (PROJECT, EMPLOYEE, MANAGER)

	PROJECT	EMPLOYEE	MANAGER
1	Analysis	Brad	Joe
2	Analysis	Riffraff	Jim
3	Analysis	Magenta	Jim
4	DW	Janet	Larry
5	DW	Brad	Larry
6	DW	Columbia	Gemima
7	HTML	Columbia	Jackson
8	HTML	Riffraff	Jackson

PROJECT_MANAGER

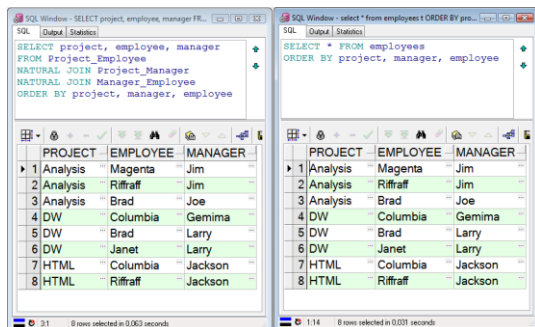
	PROJECT	MANAGER
1	Analysis	Joe
2	Analysis	Jim
3	DW	Larry
4	DW	Gemima
5	HTML	Jackson

PROJECT_EMPLOYEE

	PROJECT	EMPLOYEE
1	Analysis	Brad
2	Analysis	Riffraff
3	Analysis	Magenta
4	DW	Janet
5	DW	Brad
6	DW	Columbia
7	HTML	Columbia
8	HTML	Riffraff

MANAGER_EMPLOYEE

	MANAGER	EMPLOYEE
1	Gemima	Columbia
2	Jackson	Columbia
3	Jackson	Riffraff
4	Jim	Magenta
5	Jim	Riffraff
6	Joe	Brad
7	Larry	Brad
8	Larry	Janet



	PROJECT	EMPLOYEE	MANAGER
1	Analysis	Magenta	Jim
2	Analysis	Riffraff	Jim
3	Analysis	Brad	Joe
4	DW	Columbia	Gemima
5	DW	Brad	Larry
6	DW	Janet	Larry
7	HTML	Columbia	Jackson
8	HTML	Riffraff	Jackson

Domain/Key нормална форма

Една релация **R** е в Domain/Key нормална форма (DKNF), ако и само ако всяко ограничение в **R** е логическо следствие от ограниченията за домейни и ключове, приложени в **R**.

Пример

PERSON	TYPE	NETWORTH
Smith	Ексцентричен милионер	124,543,621
Jones	Алчен милиардер	6,553,228,893
Brown	Ексцентричен милиардер	8,829,462,998
Perry	Алчен милионер	495,565,211

Домейнът TYPE съдържа следните стойности: Ексцентричен милионер, Алчен милиардер, Ексцентричен милиардер, Алчен милионер; NETWORTH: всички цели числа над 1000000.

Налице е ограничение, което свързва TYPE и NETWORTH – то указва, че Ексцентричен милионер и Алчен милионер имат състояние от 1 000 000 до 999 999 999 включително, докато другите два типа – от 1 000 000 000 нагоре. Това ограничение не произлиза нито от ограничения на домейните, нито от такива на ключовете – следователно не можем да разчитаме, че то няма да бъде нарушено с грешна комбинация.

Решението е да се промени домейна TYPE да съдържа само стойностите Ексцентричен и Алчен – дали е милионер или милиардер се определя лесно по състоянието му.

Практически правила за съставяне на релации

В този раздел се опитваме да открием най-добрия начин да групираме атрибутите в релации.

Резултатните релации трябва да удовлетворяват две изисквания:

- Те трябва да съдържат необходимите данни за конструиране на обекта, който представят;
- Не трябва да възникват аномалии на промените при манипулиране на данните им.

Два атрибута **A** и **B** могат да се отнасят един към друг по три основни начина:

- Всеки да определя другия: $A \rightarrow B$ и $B \rightarrow A$ - това е отношение "едно-към-едно";
- Единият определя другия: $A \rightarrow B$, но **B** не определя **A** - това е отношение "много-към-едно";
- Няма функционална зависимост между тях: **A** не определя **B** и **B** не определя **A** - такова отношение се нарича "много-към-много".

Отношение "едно-към-едно"

Ако **A** определя **B** и **B** определя **A**, тогава стойностите на тези атрибути са в отношение едно-към-едно:

- Ето защо като знаем, че **A** определя **B** следва, че отношението на **A** към **B** е много-към-едно;
- Но **B** определя **A** - следователно отношението на **B** към **A** също трябва да е много-към-едно;
- Единственият начин тези две твърдения да са истина едновременно е отношението да е едно-към-едно.

Като пример можем да дадем релация, която има за атрибути ЕГН и номер на осигуровка, и двата уникални за всеки човек.

Всеки от тези атрибути уникално идентифицира човек - следователно ЕГН отговаря на точно един номер на осигуровка и обратно.

Три еквивалентни заключения могат да бъдат направени от този пример:

- Ако два атрибута са функционално зависими един от друг - отношението между стойностите им е едно-към-едно;
- Ако два атрибута уникално идентифицират един и същ обект - отношението между стойностите им е едно-към-едно;
- Ако два атрибута са в отношение едно-към-едно - тогава те функционално се определят един друг.

Изводи:

- Атрибути, които имат отношение едно-към-едно трябва да се срещат заедно поне в една релация;
- Други атрибути, които са функционално зависими от тях, също могат да бъдат в същата релация;
- Атрибути, които не са във функционална зависимост от тях, не трябва да присъстват в същата релация.

Отношение “много-към-едно”

Ако атрибутът **A** определя **B**, но **B** не определя **A**, тогава отношението между стойностите им е много-към-едно;

В релацията с преподавателите:

- В един факултет има много преподаватели, които са негови служители, т.е. факултетът не определя един преподавател;
- Но всеки преподавател може да е служител само на един факултет, т.е. преподавателят определя точно един факултет;
- Това е много-към-едно взаимоотношение (преподавател-факултет).

Lecturers

Name	Faculty	Lecture
Стоян Колев	ФМИ	Алгебра
Владимир Петров	Филология	Методика на обучението
Мария Тодорова	ФМИ	ООП
Илия Желев	Право	Гражданско право

Ако **A** определя **B**, то всички други атрибути, които добавяме в релацията, трябва да зависят само от **A**;

В противен случай ще възникнат транзитивни зависимости и степента на нормализация ще се понижи.

Отношение “много-към-много”

Ако **A** не определя **B** и **B** не определя **A**, тогава отношението между стойностите им е много-към-много.

Например, ако в една релация са представени:

- преподаватели и дисциплини;
- един преподавател може да преподава повече от една дисциплина;
- една дисциплина може да бъде преподавана от повече от един преподавател;

Тогава съществува отношение много-към-много.

- При наличие на такова отношение комбинацията от двата атрибута трябва да бъде ключ;
- При добавяне на нов атрибут трябва да се съобразяваме с изискването той да бъде функционално зависим от целия ключ.

Обобщение на типовете отношения и правилата за конструиране на релации

	1:1	M:1	M:N
Релация	$R(A, B)$	$R(A, B)$	$R(A, B)$
Зависимости	$A \rightarrow B$ $B \rightarrow A$	$A \rightarrow B$, но B не определя A	A не определя B и B не определя A
Ключ	A или B	A	(A, B)
Добавяне на атрибут	Ако $A \rightarrow C$ или $B \rightarrow C$	Ако $A \rightarrow C$	Ако $(A, B) \rightarrow C$

Едно-към-едно

- Атрибутите с такова отношение трябва да присъстват заедно поне в една релация;
- Един от двата A или B трябва да бъде ключ;
- Към релацията може да бъде добавен атрибут ако той е функционално зависим от A или B ;
- Ако атрибутът не е функционално зависим от A или B не трябва да бъде добавян в релацията;
- A и B трябва да се срещат заедно в R , но не трябва да се срещат заедно в друга релация;
- A или B трябва последователно да се използва в други релации, за да представя двойката.

Много-към-едно

- Атрибути с такова отношение могат да се срещат една релация;
- Да приемем, че $A \rightarrow B$ в R , тогава A трябва да е ключ в R ;
- Атрибут може да бъде добавян ако е функционално зависим от A ;
- Атрибут, който не е функционално зависим от A , не трябва да бъде добавян в тази релация.

Много-към-много

- Атрибути с такова взаимоотношение могат да присъстват в една релация;
- Ключ трябва да е (A, B) ;
- Атрибут може да бъде добавян, ако е функционално зависим от (A, B) ;
- Атрибут, който не е функционално зависим от (A, B) , не трябва да бъде добавян;
- Ако добавим нов атрибут, който разширява ключа на (A, B, C) , тогава се променя смисълът на релацията.

Нормализация - пример

- Цел – проектиране на база данни за съхранение на данни за тенис турнири;
- Данни, които трябва да се съхраняват:
 - Име на турнир
 - Година
 - Място на провеждане
 - Име на победител
 - Рождена дата
 - Държава

Досегашен вид на данните

За да постигнем 1NF ще трябва да бъдат премахнати повтарящите се групи от стойности.

TOURNAMENT	PLACE	WINNER			
Roland Garros	Paris, France	YEAR	NAME	BIRTH_DATE	COUNTRY
		2010	Nadal	03.06.1986	Spain
		2009	Federer	08.08.1981	Switzerland
Australian Open	Melbourne, Australia	YEAR	NAME	BIRTH_DATE	COUNTRY
		2010	Federer	08.08.1981	Switzerland
		2009	Nadal	03.06.1986	Spain
US Open	New York, U.S.A	YEAR	NAME	BIRTH_DATE	COUNTRY
		2009	Del Potro	23.09.1988	Argentina
		2008	Federer	03.06.1986	Spain

ONF

Вид на таблицата в 1NF

TOURNAMENTS
TOURNAMENT
YEAR
PLACE
NAME
BIRTH_DATE
COUNTRY

1NF

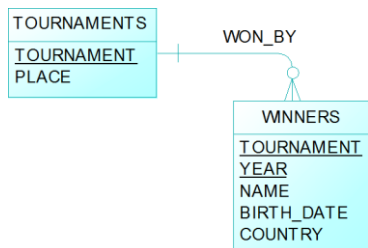
Премахнати са повтарящите се групи от стойности.
Идентификатор: {TOURNAMENT, YEAR}

TOURNAMENT	YEAR	PLACE	NAME	BIRTH_DATE	COUNTRY
Roland Garros	2010	Paris, France	Nadal	03.06.1986	Spain
Roland Garros	2009	Paris, France	Federer	08.08.1981	Switzerland
Australian Open	2010	Melbourne, Australia	Federer	08.08.1981	Switzerland
Australian Open	2009	Melbourne, Australia	Nadal	03.06.1986	Spain
Australian Open	2008	Melbourne, Australia	Djokovic	22.05.1987	Serbia
US Open	2010	New York, U.S.A	Nadal	03.06.1986	Spain
US Open	2009	New York, U.S.A	Del Potro	23.09.1988	Argentina
US Open	2008	New York, U.S.A	Federer	08.08.1981	Spain

За да бъде във втора трябва неключовите атрибути да зависят от целия първичен ключ {TOURNAMENT, YEAR}, но за някои това не е така:
• TOURNAMENT → PLACE

Вид на таблицата в 2NF

За да постигнем втора нормална форма декомпозираме таблицата на две нови, въпреки че от функционалните зависимости.



TOURNAMENT	PLACE
Roland Garros	Paris, France
Australian Open	Melbourne, Australia
US Open	New York, U.S.A

В таблицата с победителите ключ си остава {TOURNAMENT, YEAR}; TOURNAMENT е външен ключ към таблицата с турнирите.

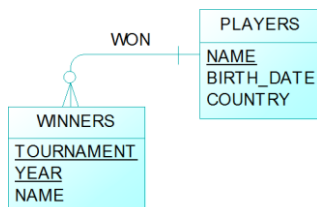
Забеляват се следните транзитивни зависимости:
 • NAME → BIRTH_DATE
 • NAME → COUNTRY

TOURNAMENT	YEAR	NAME	BIRTH_DATE	COUNTRY
Roland Garros	2010	Nadal	03.06.1986	Spain
Roland Garros	2009	Federer	08.08.1981	Switzerland
Australian Open	2010	Federer	08.08.1981	Switzerland
Australian Open	2009	Nadal	03.06.1986	Spain
Australian Open	2008	Djokovic	22.05.1987	Serbia
US Open	2010	Nadal	03.06.1986	Spain
US Open	2009	Del Potro	23.09.1988	Argentina
US Open	2008	Federer	08.08.1981	Switzerland

2NF

Вид на таблицата в 3NF

За да постигнем трета нормална форма декомпозираме таблицата на две нови, за да се отървем от споменатите транзитивни зависимости.



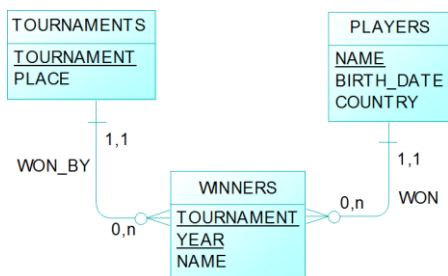
Декомпозираме таблицата, за да отстраним споменатите транзитивни зависимости NAME→BIRTH_DATE и NAME→COUNTRY

NAME	BIRTH_DATE	COUNTRY
Nadal	03.06.1986	Spain
Federer	08.08.1981	Switzerland
Djokovic	22.05.1987	Serbia
Del Potro	23.09.1988	Argentina

3NF

TOURNAMENT	YEAR	NAME
Roland Garros	2010	Nadal
Roland Garros	2009	Federer
Australian Open	2010	Federer
Australian Open	2009	Nadal
Australian Open	2008	Djokovic
US Open	2010	Nadal
US Open	2009	Del Potro
US Open	2008	Federer

Краен вид на схемата



Краен вид на схемата

TOURNAMENT	PLACE
Roland Garros	Paris, France
Australian Open	Melbourne, Australia
US Open	New York, U.S.A

NAME	BIRTH_DATE	COUNTRY
Nadal	03.06.1986	Spain
Federer	08.08.1981	Switzerland
Djokovic	22.05.1987	Serbia
Del Potro	23.09.1988	Argentina

TOURNAMENT	YEAR	NAME
Roland Garros	2010	Nadal
Roland Garros	2009	Federer
Australian Open	2010	Federer
Australian Open	2009	Nadal
Australian Open	2008	Djokovic
US Open	2010	Nadal
US Open	2009	Del Potro
US Open	2008	Federer

Денормализация

Процес, често обратен на нормализацията, но не задължително – понижаване степента на нормализация на релациите – чрез добавяне на дублираща се информация, групиране на данни и др.

Причини:

- ✓ повишаване на ефективността на достъп (след като други подходи за това са се провалили - индексиране), която би могла да е намалена заради прекалено висока степен на нормализация. Прекалената гранулираност по време на разработване на модела на базата може освен да реши проблеми да създаде и нови;
- ✓ създаване на data warehouse или reporting таблици.

Денормализираният модел на схемата не е същият като този, преди тя да бъде нормализирана – той трябва да съдържа правила, които да следят целостността на данните да не бъде нарушена.

Денормализацията

- ✓ Би могла драматично да подобри достъпа до данните, но без гаранция за сигурен успех, а винаги се плаща цена за това;
 - ✓ Усложнява обработката и дава възможност за поява на проблеми относно интегритета на данните, затова обикновено се изисква допълнително програмиране, за да се поддържат данните в денормализираната схема;
 - ✓ Прави схемата по-неточна и води до забавяне на DML операциите.
- Стартира от "нормализиран" модел;
 - Добавя излишество към дизайна;
 - Понижава интегритета на дизайна;
 - Се нуждае от приложен код, който да компенсира (понякога).

Техники на денормализация

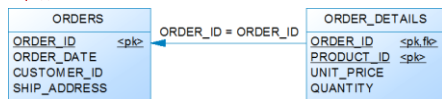
1. Съхраняване на стойности, които биха могли да бъдат извлечени

Когато има чести изчисления в заявките би могло да си струва да се съхранява резултатът от изчисленията. Ако изчислението включва детайлни записи може да се съхранява изчислената стойност в основната (master) таблица.

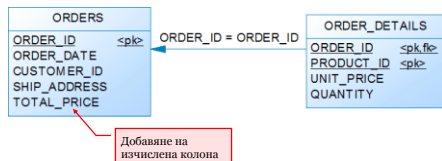
Трябва, обаче, да се добави код, който да преизчислява съхранената стойност всеки път при промяна на детайлните записи.

Във всички ситуации на съхранение на изчислена стойност е добре да се подсигурир тази стойност да не може да се обновява директно с DML операция. Тази стойност трябва винаги да бъде преизчислявана от системата.

Преди



След



✓ Съхраняване на стойности, които биха могли да бъдат извлечени

Подходяща, когато:

- стойностите-източници са в множество от записи или таблици;
- изчислените стойности са често нужни, а стойностите-източници не;
- стойностите-източници не се променят често.

Предимства:

- стойностите-източници не е нужно да бъдат намирани всеки път, когато е нужна изчислената стойност;
- изчислението не е нужно да се прави по време на заявка.

Недостатъци:

- DML операция, засягаща стойностите-източници, ще изисква преизчисление на изчислената стойност;
- дублирането на данни внася възможността от неконсистентност на данните.

Техники на денормализация

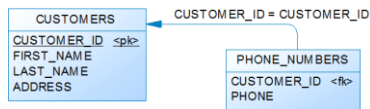
2. Pre-Joining на таблици

Постига се чрез включване на неключова колона в таблица, когато стойността на първичния (съответно на външния) ключ няма "смислено" значение.

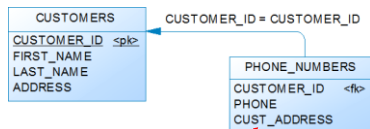
По този начин, включвайки неключова колона със "смислено" значение, може да бъде избегнато съединението на таблици, ускорявайки заявката.

Трябва да се добави код, който да обновява "денормализираната" колона всеки път, когато "master" колоната промени стойността си в референтния запис.

Преди



След



Добавяне на
ключова колона

✓ Pre-Joining на таблици

Подходяща, когато:

- се използват чести заявки върху няколко таблици;
- “леко неактуални” данни са приемливи.

Предимства:

- времеотнемачи съединения могат да бъдат избягнати;
- обновяванията могат да бъдат отложени, когато е приемлива не толкова актуална информация.

Недостатъци:

- допълнителни DML операции са нужни за обновяване на “денормализираната” колона;
- допълнителната колона изисква допълнителна памет.

Техники на денормализация

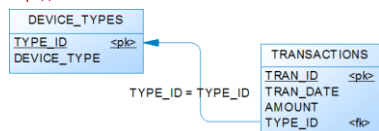
3. Hard-Coded стойности

Ако референцирана таблица съдържа записи, които са константни, можем да обмислим “твърдо кодиране” на тия стойности в кода.

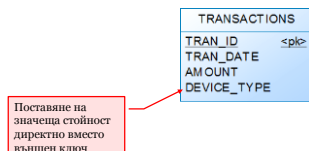
Това ще означава, че няма да има нужда от съединение на таблици за извличане на референцираните стойности.

Допустимо е да се помисли за създаване на check constraint на референциращата таблица, който да проверява за допустими стойности (вместо външен ключ).

Преди



След



✓ Hard-Coded стойности

Подходяща, когато:

- множество от позволени стойности могат да се считат за статични в системата;
- множеството от допустимите стойности е относително малко, напр. < 30.

Предимства:

- избягва се реализацията на таблица със статичните стойности;
- избягва се съединението с тази таблица.

Недостатъци:

- промяната на "статичните" стойности изисква прекодиране и повторно тестване.

Техники на денормализация

4. Запазване на детайлите в основната таблица

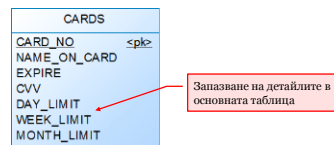
В ситуация, при която броят на детайлните записи за всеки основен е фиксиран, можем да обмислим добавянето на детайлните колони към основната таблица.

Този вариант работи най-добре, когато броят на записите в детайлната таблица е малък. По този начин ще редуцираме броя на съединенията в заявката.

Преди



След



✓ Запазване на детайлите в основната таблица

Подходяща, когато:

- броят на детайлните записи за основния е фиксиран или статичен;
- (броят на детайлните записи) * (брой на колоните) < 30.

Предимства:

- не е нужно съединение;
- икономия на памет от спестените записвания на ключовете.

Недостатъци:

- увеличава сложността на DML операциите;
- проверките (ако има) на AMOUNT колоната трябва да се сложат за всяка XXX_LIMIT колона;
- името на таблицата вече може да не отговаря на съдържанието ѝ.

Техники на денормализация

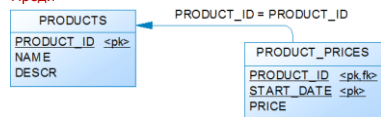
5. Повтаряне на единичен детайл в основната

Често когато съхранение на история на промените е нужна, много заявки се нуждаят само от най-актуалния запис.

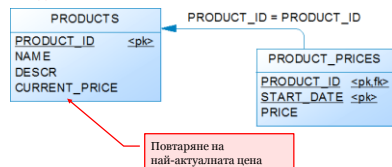
Можем да добавим колона, която да съхранява този единствен детайл при основните данни.

Трябва да не се забравя добавянето на код, който да актуализира денормализираната колона всеки път, когато се добавя нов запис в таблицата с историята.

Преди



След



✓ Повтаряне на единичен детайл в основната

Подходяща, когато:

- детайлните записи имат атрибут, който определя кой от тях е актуален, а останалите са история;
- заявките често се нуждаят от този единичен детайл, а рядко от останалите.

Предимства:

- не е нужно съединение за заявките, които изискват този детайл.

Недостатъци:

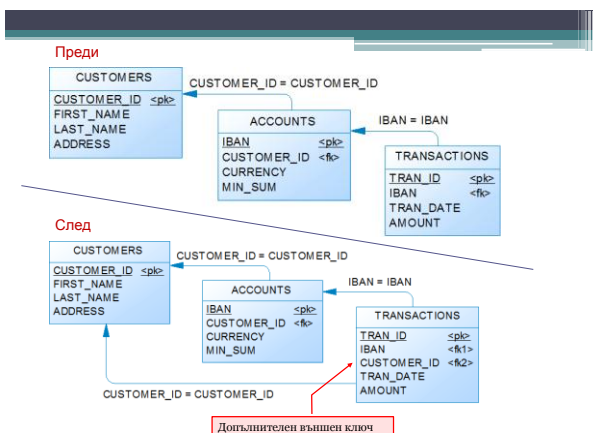
- детайлната стойност трябва да се повтаря – възможност за неконсистентност;
- допълнителен код за актуализация на денормализираната колона.

Техники на денормализация

6. Short-Circuit ключове

За схеми, съдържащи 3 или повече нива на master-detail таблици и при нужда да се извличат данни от първата и последната таблици в референциалния път, може да се направи допълнителен външен ключ, свързващ директно последната с първата таблица.

В резултат на това заявките могат да съдържат по-малко таблици, участващи в съединенията.



✓ Short-Circuit ключове

Подходяща, когато:

- заявките често изискват данни от първата и последната, но не и от междинните таблици.

Предимства:

- по-малко таблици в съединенията.

Недостатъци:

- допълнителен външен ключ;
- допълнителен код, подsigуряващ че стойността на CUSTOMER_ID за транзакциите ще е същата, каквато е в таблицата със сметките, за които се отнасят.