

» Гл. ас. д-р Георги Чолаков  
» Базы от данни

**Теория на нормализацията** ➤

# Въведение

В тази част ще разгледаме:

- » Какво е нормализация и каква е нейната роля в процеса на дизайн на базата данни;
- » Кои са нормалните форми;
- » Как се преминава от по-ниска към по-висока нормална форма;
- » Какво е денормализация и кога се прилага.

Т.е.:

- » Дадена съвкупност от данни трябва да бъде представена в базата данни;
- » Как да преценим каква да е логическата структура за тези данни, за да бъдат избегнати излишеството и аномалиите на промените;
- » Какви таблици да създадем и кои атрибути да участват в тях.



# Нормализация

- » Процес на подреждане на данните в таблици с цел минимизиране на излишеството и намаляване на вероятността за поява на аномалии на промените;
- » Базира се на анализ на функционалните зависимости между атрибутите;
- » Нормализацията се извършва на степени, наречени нормални форми – първа, втора, трета... Всяка следваща нормална форма означава по-висока степен на нормализация;



# Нормализация

- » От гледна точка на структурата 2-ра е по-добра от 1-ва, а 3-та е по-добра от втора;
- » В повечето случаи 3-та нормална форма е най-високата, нужна за добър дизайн;
- » Нормализацията трябва да премахне излишеството, но не и за сметка на интегритета;
- » Резултатът е по-добра организираност и по-ефективна работа на базата данни и употреба на физическо пространство;



# Нормализация

- » Не е правилно да се смята, че най-високата степен на нормализация е най-желана;
- » Колкото по-висока е степента на нормализация, толкова повече JOIN операции ще са нужни за извличане на данни, защото ще има повече таблици в схемата;
- » Въпреки че нормализацията обикновено е желателна - възможно е да се стигне до свръхнормализация, която е 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.





# Нормализация – кога?

- » Обикновено в случаите, когато се моделира нова база от данни, базирана на бизнес изисквания от клиентите;
- » След като първоначалният дизайн е готов, дизайнерът може да анализира отново взаимоотношенията между атрибутите и прецени дали може да подобри структурата на базата данни чрез нормализация;
- » Или в случаите, когато трябва да се модифицира съществуваща структура – отново чрез анализ на взаимоотношенията между атрибутите може да се прецени дали чрез нормализация може да се подобри структурата.



# Нормализиращата процедура

- » Включва декомпозиция на дадена входна релация на нови релации;
- » Изисква се тази декомпозиция да бъде обратима - при този процес да не се губи информация (non-loss decomposition);
- » Проблемът за загуба на информация е свързан с несъобразяване с функционалните зависимости.



# Декомпозиция - пример

STUDENTS

FAC_NO	NAME	SUBJECT
1701321059	Иван Колев	STD
1701321063	Елена Петрова	STD

Функционални зависимости:

- ✓  $FAC\_NO \rightarrow NAME$
- ✓  $FAC\_NO \rightarrow SUBJECT$

Декомпозиция 1

FAC_NO	NAME
1701321059	Иван Колев
1701321063	Елена Петрова

FAC_NO	SUBJECT
1701321059	STD
1701321063	STD

Декомпозиционният оператор е PROJECT.  
Композиционният оператор е NATURAL JOIN.

Декомпозиция 2

FAC_NO	SUBJECT
1701321059	STD
1701321063	STD

NAME	SUBJECT
Иван Колев	STD
Елена Петрова	STD





# Обратно композиция

Композиция 1

JOIN

FAC_NO	NAME	FAC_NO	SUBJECT
1701321059	Иван Колев	1701321059	STD
1701321063	Елена Петрова	1701321063	STD

↓

FAC_NO	NAME	SUBJECT
1701321059	Иван Колев	STD
1701321063	Елена Петрова	STD

Функционални зависимости:

- ✓ FAC\_NO → NAME
- ✓ FAC\_NO → SUBJECT

Композиция 2

JOIN

FAC_NO	SUBJECT	NAME	SUBJECT
1701321059	STD	Иван Колев	STD
1701321063	STD	Елена Петрова	STD

↓

FAC_NO	NAME	SUBJECT
1701321059	Иван Колев	STD
1701321063	Иван Колев	STD
1701321059	Елена Петрова	STD
1701321063	Елена Петрова	STD

Изводи:

- ✓ Декомпозиция 1 – без загуба на информация;
- ✓ Декомпозиция 2 – със загуба на информация, защото FAC\_NO → NAME е игнорирана, а не е налице ФЗ между NAME и SUBJECT.

# 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?

Отговор: **функционалните зависимости.**

- » В примера релацията STUDENTS удовлетворява несъкратимото множество от ФЗ:
  - >  $FAC\_NO \rightarrow NAME$
  - >  $FAC\_NO \rightarrow SUBJECT$



# 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  $A = \text{FAC\_NO}$ ,  $B = \text{NAME}$ ,  $C = \text{SUBJECT}$  теоремата потвърждава, че релацията STUDENTS може да бъде декомпозирана без загуба на нейните проекции  $\{\text{FAC\_NO}, \text{NAME}\}$  и  $\{\text{FAC\_NO}, \text{SUBJECT}\}$ .

В същото време виждаме, че STUDENTS не може да бъде декомпозирана без загуба на проекциите  $\{\text{FAC\_NO}, \text{SUBJECT}\}$  и  $\{\text{NAME}, \text{SUBJECT}\}$ , защото губим функционалната зависимост  $\text{FAC\_NO} \rightarrow \text{NAME}$ , а и не е налице ФЗ  $\text{SUBJECT} \rightarrow \text{NAME}$ .



# Демонстрационен пример

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





# Текуща структура на данните...

TOURNAMENT	PLACE	WINNER			
Australian Open	Melbourne, Australia	YEAR	PLAYER	BIRTH_DATE	COUNTRY
		2019	Novak Djokovic	22.05.1987	Serbia
		2018	Roger Federer	08.08.1981	Switzerland
		2017	Roger Federer	08.08.1981	Switzerland
Roland Garros	Paris, France	YEAR	PLAYER	BIRTH_DATE	COUNTRY
		2019	Rafael Nadal	03.06.1986	Spain
		2018	Rafael Nadal	03.06.1986	Spain
		2017	Rafael Nadal	03.06.1986	Spain
Wimbledon	London, England	YEAR	PLAYER	BIRTH_DATE	COUNTRY
		2019	Novak Djokovic	22.05.1987	Serbia
		2018	Novak Djokovic	22.05.1987	Serbia
		2017	Roger Federer	08.08.1981	Switzerland
US Open	New York, USA	YEAR	PLAYER	BIRTH_DATE	COUNTRY
		2019	Rafael Nadal	03.06.1986	Spain
		2018	Novak Djokovic	22.05.1987	Serbia
		2009	J.M. Del Potro	23.09.1988	Argentina





# ... трансформирана в таблица с някои недостатъци

TOURNAMENT	PLACE	YEAR	PLAYER	BIRTH_DATE	COUNTRY
Australian Open	Melbourne, Australia	2019	Novak Djokovic	22.05.1987	Serbia
		2018	Roger Federer	08.08.1981	Switzerland
		2017	Roger Federer	08.08.1981	Switzerland
Belgian Open	Paris, France	2019	Rafael Nadal	03.06.1986	Spain
			Rafael Nadal	03.06.1986	Spain
			Rafael Nadal	03.06.1986	Spain
			Novak Djokovic	22.05.1987	Serbia
			Novak Djokovic	22.05.1987	Serbia
		2017	Roger Federer	08.08.1981	Switzerland
US Open	New York, USA	2019	Rafael Nadal	03.06.1986	Spain
		2018	Novak Djokovic	22.05.1987	Serbia
		2009	J.M. Del Potro	23.09.1988	Argentina

(TOURNAMENT, YEAR) – изглежда, че идентифицира ред от таблицата, т.е. е кандидат ключ, но атрибутът TOURNAMENT има нулеви стойности, което не се допуска за атрибут, част от идентификатор

Държавата може да бъде изписана по различни начини – Swiss, Suisse, Schweiz, Svizzero...



# Аномалии на промените

UPDATE аномалия – когато един факт трябва да бъде променен на няколко места.

TOURNAMENT	PLACE	YEAR	PLAYER	BIRTH_DATE	COUNTRY
Australian Open	Melbourne, Australia	2019	Novak Djokovic	22.05.1987	Serbia
		2018	Roger Federer	08.08.1981	Switzerland
		2017	Roger Federer	08.08.1981	Switzerland
Roland Garros	Paris, France	2019	Rafael Nadal	03.06.1986	Spain
		2018	Rafael Nadal	03.06.1986	Spain
		2017	Rafael Nadal	03.06.1986	Spain
Wimbledon	London, England	2019	Novak Djokovic	22.05.1987	Serbia
		2018	Novak Djokovic	22.05.1987	Serbia
		2017	Roger Federer	08.08.1981	
US C		2019	Rafael Nadal	03.06.1986	
		2018	Novak Djokovic	22.05.1987	
		2009	J.M. Del Potro	23.09.1988	Argentina
?	?	?	Grigor Dimitrov	16.05.1991	Bulgaria

INSERT аномалия – когато за въвеждането на един факт са нужни повече данни от въвежданите.

DELETE аномалия – когато изтривайки един факт изтриваме и други данни.



# Повтарящи се групи от данни

## TOURNAMENTS

TOURNAMENT	PLACE	WINNER			
		YEAR	PLAYER	BIRTH_DATE	COUNTRY
Australian Open	Melbourne, Australia	2019	Novak Djokovic	22.05.1987	Serbia
		2018	Roger Federer	08.08.1981	Switzerland
		2017	Roger Federer	08.08.1981	Switzerland
Roland Garros	Paris, France	2019	Rafael Nadal	03.06.1986	Spain
		2018	Rafael Nadal	03.06.1986	Spain
		2017	Rafael Nadal	03.06.1986	Spain

Повтарящи се групи от данни

## TRADERS

NAME	LOCATIONS
Metro C&C	София, бул. Цариградско шосе - 7; Пловдив, бул. Санкт Петербург – 135; ...
Технополис	Пловдив, бул. Санкт Петербург – 135; Варна, бул. Цар Освободител – 267; ...

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 релационна таблица е в първа нормална форма, защото релациите трябва да съдържат атомарни стойности в атрибутите си.





# Трансформиране до 1NF в стъпки

1. Елиминиране на повтарящите се групи – започва се със представяне на данните в табличен вид, със скаларни стойности в полетата на записите.
2. Идентифициране на първичния ключ – в примера това е комбинацията (TOURNAMENT, YEAR).
3. Идентифициране на всички зависимости – също и тези, в които първичният ключ не участва или участва частично.





# Концептуален вид на таблицата в 1NF

TOURNAMENTS

TOURNAMENT

PLACE

YEAR

PLAYER

BIRTH\_DATE

COUNTRY



# Таблицата в 1NF

<u>TOURNAMENT</u> ↔	PLACE	<u>YEAR</u> ↔	PLAYER	BIRTH_DATE	COUNTRY
Australian Open	Melbourne, Australia	2019	Novak Djokovic	22.05.1987	Serbia
Australian Open	Melbourne, Australia	2018	Roger Federer	08.08.1981	Switzerland
Australian Open	Melbourne, Australia	2017	Roger Federer	08.08.1981	Switzerland
Roland Garros	Paris, France	2019	Rafael Nadal	03.06.1986	Spain
Roland Garros	Paris, France	2018	Rafael Nadal	03.06.1986	Spain
Roland Garros	Paris, France	2017	Rafael Nadal	03.06.1986	Spain
Wimbledon	London, England	2019	Novak Djokovic	22.05.1987	Serbia
Wimbledon	London, England	2018	Novak Djokovic	22.05.1987	Serbia
Wimbledon	London, England	2017	Roger Federer	08.08.1981	Switzerland
US Open	New York, USA	2019	Rafael Nadal	03.06.1986	Spain
US Open	New York, USA	2018	Novak Djokovic	22.05.1987	Serbia
US Open	New York, USA	2009	J.M. Del Potro	23.09.1988	Argentina

Премахнати са повтарящите се групи от стойности.

Първичен ключ – (TOURNAMENT, YEAR)

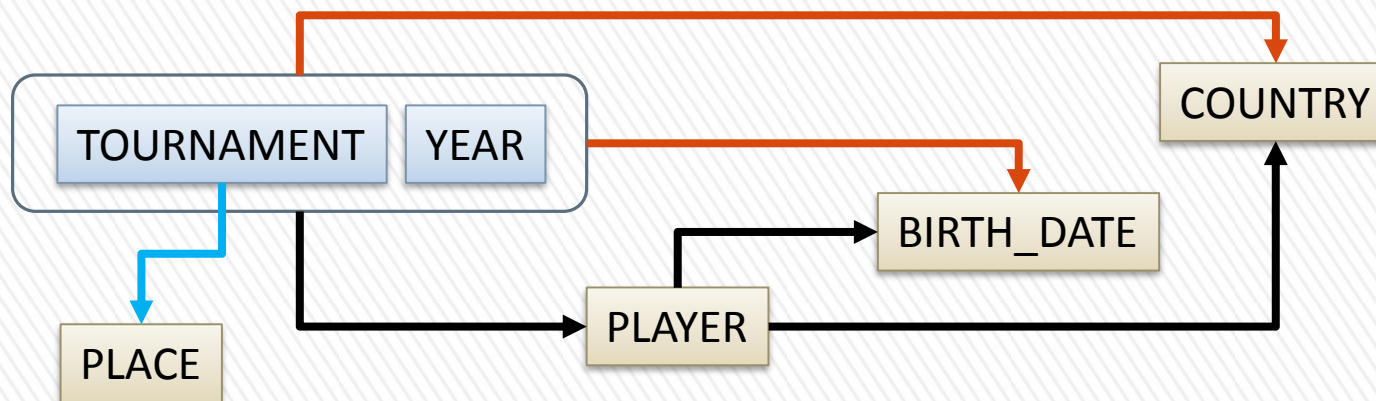


# Функционални зависимости между атрибути

- » **Пълна функционална зависимост** - атрибут е напълно функционално зависим от множество атрибути ако и само ако той е функционално зависим от цялото това множество от атрибути и не е зависим от никое негово подмножество от атрибути;
- » **Частична функционална зависимост** - такава е налице, когато само подмножество от съставен детерминант е достатъчно, за да определи функционално даден атрибут;
- » **Транзитивна функционална зависимост** - транзитивна ФЗ съществува, когато има „преходна“ функционална зависимост. Т.е., ако  $A \rightarrow B$ , а  $B \rightarrow C$ , то  $A \rightarrow C$ .



# Идентифициране на зависимостите - диаграма



» TOURNAMENT → PLACE (частична)

» (TOURNAMENT, YEAR) → PLAYER, PLAYER → BIRTH\_DATE, PLAYER → COUNTRY

Т.е. налице са следните транзитивни зависимости:

» (TOURNAMENT, YEAR) → BIRTH\_DATE

» (TOURNAMENT, YEAR) → COUNTRY



# 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 за релации, които имат съставен ключ. Ако ключът е от един атрибут, то релацията автоматично е във втора нормална форма.



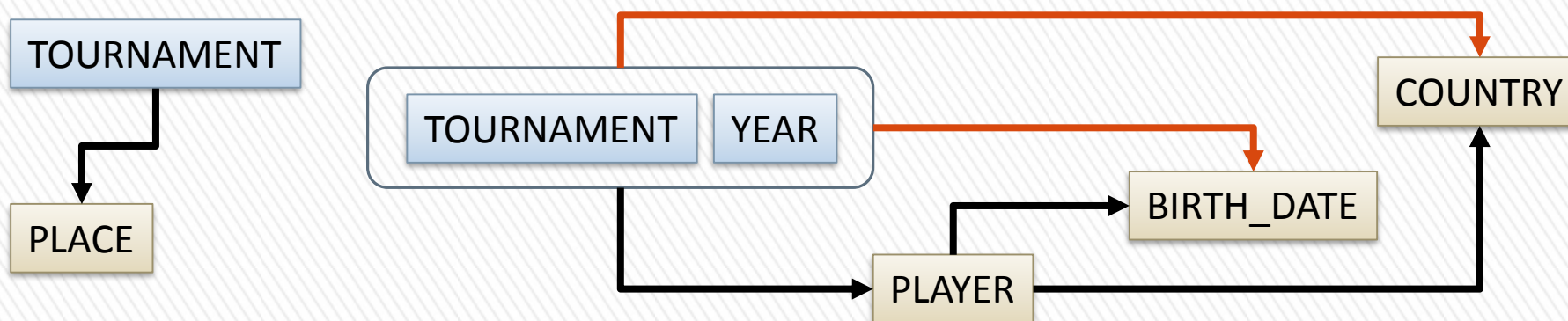


# Трансформиране до 2NF в стъпки

1. За всеки атрибут от първичния ключ, който е детерминант в частична зависимост, се създава нова таблица, където той ще бъде първичен ключ, а зависимите елементи също ще бъдат преместени в нея. В примера: **TOURNAMENT** → **PLACE**
2. В оригиналната таблица е важно да се запазят копия на детерминантите от частичните зависимости, защото там те ще играят ролята на външен ключ, свързващ оригиналната и новата таблица.



# Диаграма на зависимостите в 2NF



»  $TOURNAMENT \rightarrow PLACE$ ,  $(TOURNAMENT, YEAR) \rightarrow PLAYER$ ,  $PLAYER \rightarrow BIRTH\_DATE$ ,  $PLAYER \rightarrow COUNTRY$

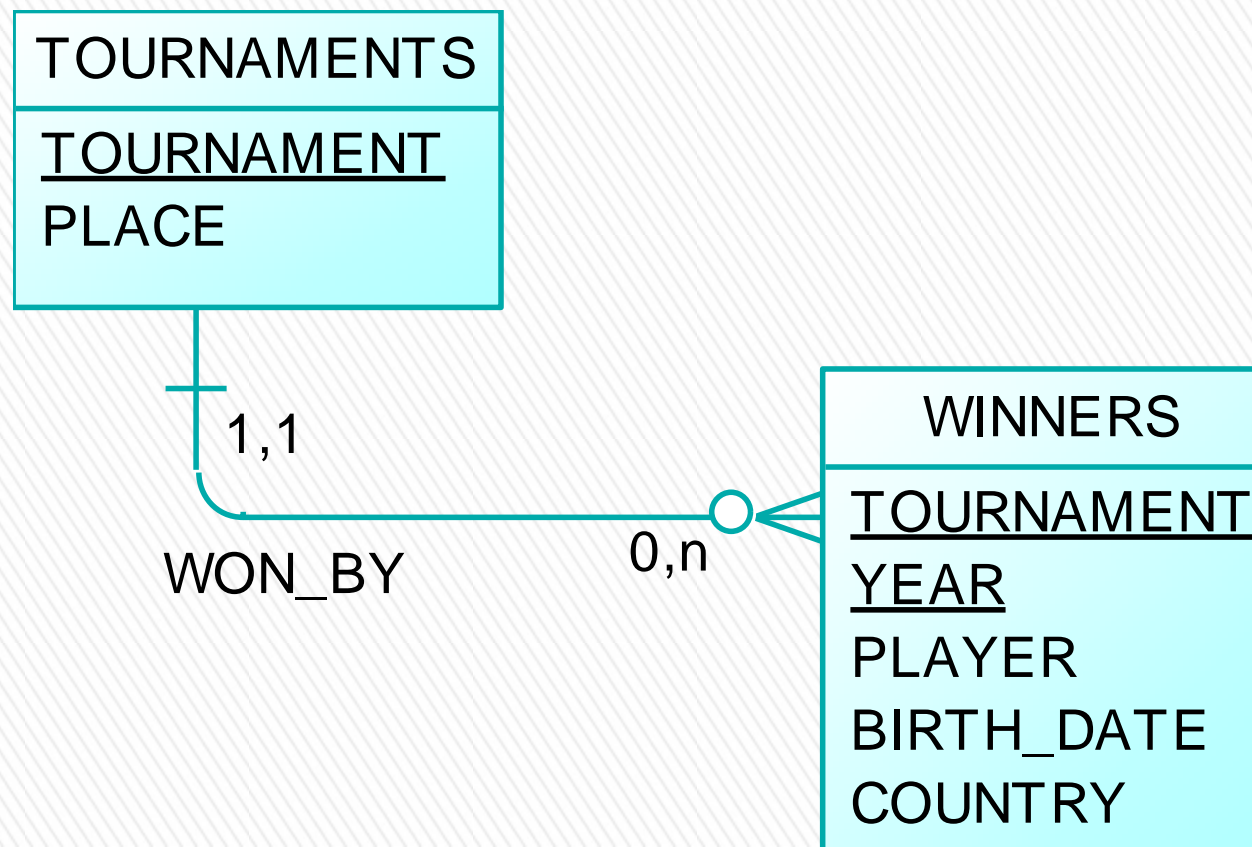
Транзитивни зависимости:

»  $(TOURNAMENT, YEAR) \rightarrow BIRTH\_DATE$


»  $(TOURNAMENT, YEAR) \rightarrow COUNTRY$





# Концептуален вид на таблиците в 2NF



# Таблиците в 2NF

<u>TOURNAMENT</u> 	PLACE
Australian Open	Melbourne, Australia
Roland Garros	Paris, France
Wimbledon	London, England
US Open	New York, USA

<u>TOURNAMENT</u> 	<u>YEAR</u> 	PLAYER	BIRTH_DATE	COUNTRY
Australian Open	2019	Novak Djokovic	22.05.1987	Serbia
Australian Open	2018	Roger Federer	08.08.1981	Switzerland
Australian Open	2017	Roger Federer	08.08.1981	Switzerland
Roland Garros	2019	Rafael Nadal	03.06.1986	Spain
Roland Garros	2018	Rafael Nadal	03.06.1986	Spain
Roland Garros	2017	Rafael Nadal	03.06.1986	Spain
Wimbledon	2019	Novak Djokovic	22.05.1987	Serbia
Wimbledon	2018	Novak Djokovic	22.05.1987	Serbia
Wimbledon	2017	Roger Federer	08.08.1981	Switzerland
US Open	2019	Rafael Nadal	03.06.1986	Spain
US Open	2018	Novak Djokovic	22.05.1987	Serbia
US Open	2009	J.M. Del Potro	23.09.1988	Argentina



# Трета нормална форма

- » Една релация е в трета нормална форма (Third Normal Form - 3NF), ако и само ако:
  - > Тя е във 2NF;
  - > Всеки неключов атрибут е нетранзитивно зависим от първичния ключ.



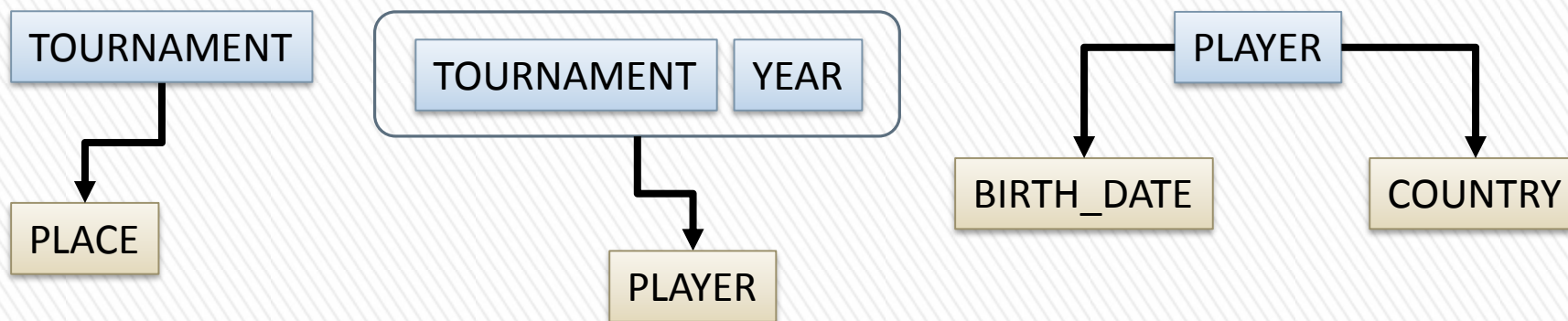


# Трансформиране до 3NF в стъпки

1. Премахване на транзитивните зависимости – чрез декомпозиция на оригиналната таблица на нови, в които са запазени пълните функционални зависимости, а техните детерминанти остават в оригиналната, където ще играят ролята на външни ключове.
2. В новите таблици детерминантите ще бъдат първични ключове.



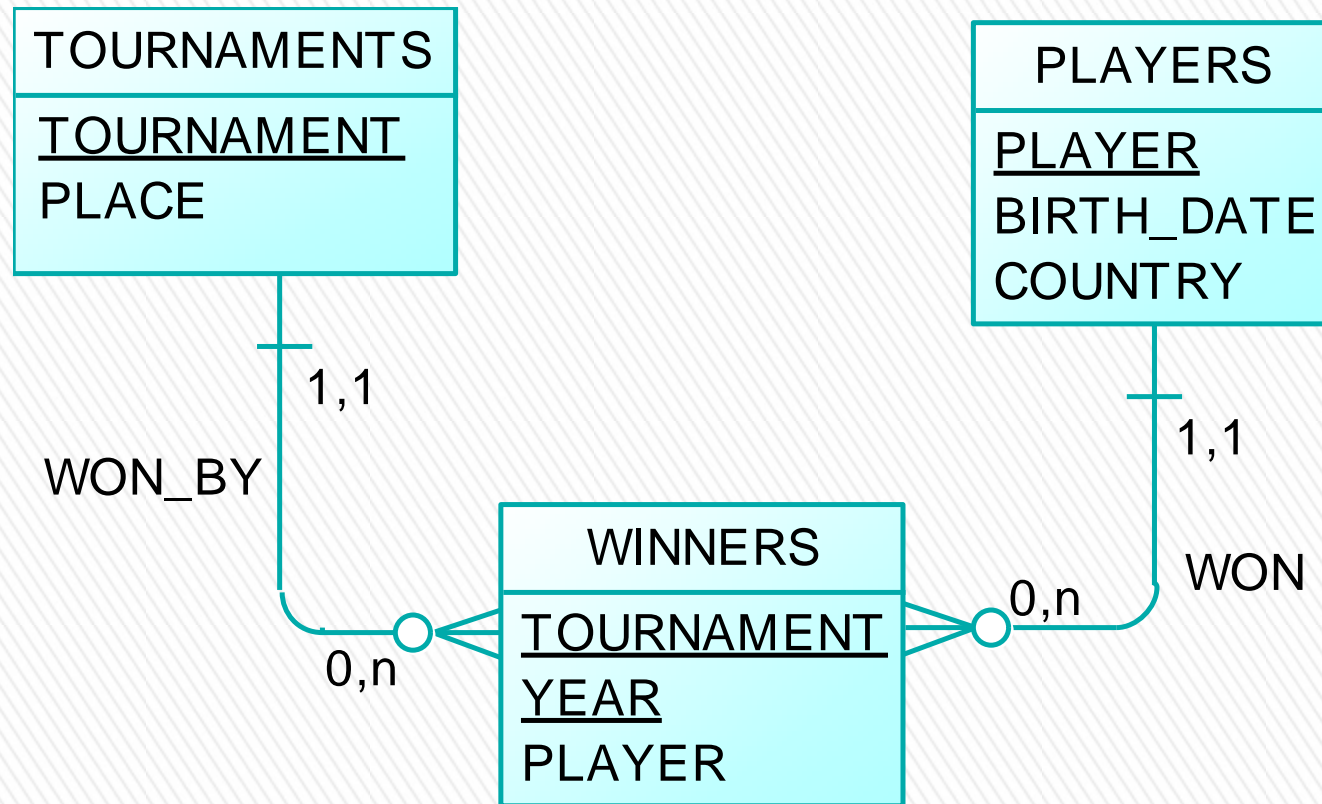
# Диаграма на зависимостите в 3NF



- »  $\text{TOURNAMENT} \rightarrow \text{PLACE}$
- »  $(\text{TOURNAMENT}, \text{YEAR}) \rightarrow \text{PLAYER}$
- »  $\text{PLAYER} \rightarrow \text{BIRTH\_DATE}, \text{PLAYER} \rightarrow \text{COUNTRY}$



# Концептуален вид на таблицата в 3NF



# Таблиците в 3NF

<u>TOURNAMENT</u> 🔑	PLACE
Australian Open	Melbourne, Australia
Roland Garros	Paris, France
Wimbledon	London, England
US Open	New York, USA

<u>PLAYER</u> 🔑	BIRTH_DATE	COUNTRY
Novak Djokovic	22.05.1987	Serbia
Roger Federer	08.08.1981	Switzerland
Rafael Nadal	03.06.1986	Spain
Novak Djokovic	22.05.1987	Serbia
J.M. Del Potro	23.09.1988	Argentina

<u>TOURNAMENT</u> 🔑	<u>YEAR</u> 🔑	PLAYER
Australian Open	2019	Novak Djokovic
Australian Open	2018	Roger Federer
Australian Open	2017	Roger Federer
Roland Garros	2019	Rafael Nadal
...	...	...
US Open	2009	J.M. Del Potro



# Аномалии на промените?

<u>TOURNAMENT</u> 🔑	PLACE
Australian Open	Melbourne, Australia
Roland Garros	Paris, France
Wimbledon	London, England
US Open	New York, USA

<u>PLAYER</u> 🔑	BIRTH_DATE	COUNTRY
Novak Djokovic	22.05.1987	Serbia
Roger Federer	08.08.1981	Switzerland
Rafael Nadal	03.06.1986	Spain
Novak Djokovic	22.05.1987	Serbia
J.M. Del Potro	23.09.1988	Argentina

<u>TOURNAMENT</u> 🔑	<u>YEAR</u> 🔑	PLAYER
Australian Open	2019	Novak Djokovic
Australian Open	2018	Roger Federer
Australian Open	2017	Roger Federer
Roland Garros	2019	Rafael Nadal
...	...	...
US Open	2009	J.M. Del Potro





# INSERT

<u>TOURNAMENT</u> 🔑	PLACE
Australian Open	Melbourne, Australia
Roland Garros	Paris, France
Wimbledon	London, England
US Open	New York, USA
<b>Sofia Open</b>	<b>Sofia, Bulgaria</b>

<u>PLAYER</u> 🔑	BIRTH_DATE	COUNTRY
Novak Djokovic	22.05.1987	Serbia
Roger Federer	08.08.1981	Switzerland
Rafael Nadal	03.06.1986	Spain
Novak Djokovic	22.05.1987	Serbia
J.M. Del Potro	23.09.1988	Argentina
<b>Grigor Dimitrov</b>	<b>16.05.1991</b>	<b>Bulgaria</b>

<u>TOURNAMENT</u> 🔑	<u>YEAR</u> 🔑	PLAYER
Australian Open	2019	Novak Djokovic
Australian Open	2018	Roger Federer
Australian Open	2017	Roger Federer
Roland Garros	2019	Rafael Nadal
...	...	...
US Open	2009	J.M. Del Potro
<b>Sofia Open</b>	<b>2017</b>	<b>Grigor Dimitrov</b>



# UPDATE

<u>TOURNAMENT</u> 🔑	PLACE
Australian Open	Melbourne, Australia
Roland Garros	Paris, France
Wimbledon	London, England
New US Open	New York, USA

<u>PLAYER</u> 🔑	BIRTH_DATE	COUNTRY
Novak Djokovic	22.05.1987	Serbia
Roger Federer	08.08.1981	Swiss
Rafael Nadal	03.06.1986	Spain
Novak Djokovic	22.05.1987	Serbia
J.M. Del Potro	23.09.1988	Argentina

<u>TOURNAMENT</u> 🔑	<u>YEAR</u> 🔑	PLAYER
Australian Open	2019	Novak Djokovic
Australian Open	2018	Roger Federer
Australian Open	2017	Roger Federer
Roland Garros	2019	Rafael Nadal
...	...	...
US Open	2009	J.M. Del Potro



# DELETE

<u>TOURNAMENT</u> 🔑	PLACE
Australian Open	Melbourne, Australia
Roland Garros	Paris, France
Wimbledon	London, England
US Open	New York, USA

<u>PLAYER</u> 🔑	BIRTH_DATE	COUNTRY
Novak Djokovic	22.05.1987	Serbia
Roger Federer	08.08.1981	Switzerland
Rafael Nadal	03.06.1986	Spain
Novak Djokovic	22.05.1987	Serbia
J.M. Del Potro	23.09.1988	Argentina

<u>TOURNAMENT</u> 🔑	<u>YEAR</u> 🔑	PLAYER
Australian Open	2019	Novak Djokovic
Australian Open	2018	Roger Federer
Australian Open	2017	Roger Federer
Roland Garros	2019	Rafael Nadal
...	...	...
<b>US Open</b>	<b>2009</b>	<b>J.M. Del Potro</b>



# По-високи нормални форми

- » Boyce-Codd Normal Form – когато всеки детерминант в таблицата е ключ кандидат. Ако таблицата съдържа само един ключ кандидат, 3NF и BCNF са еквивалентни;
- » Четвърта нормална форма (4NF);
- » Пета нормална форма (5NF - Projection-Join Normal Form);
- » Domain-Key Normal Form (DCNF);
- » Шеста нормална форма (6NF).



# Обобщение на типовете отношения и правилата за конструиране на релации

» Нека имаме релацията  $R(A, B)$ :

	1:1	1:N	M:N
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 $(A, B) \rightarrow C$





# 1:1

- » Атрибутите с такова отношение трябва да присъстват заедно поне в една релация;
- » Един от двата А или В трябва да бъде ключ;
- » Към релацията може да бъде добавен атрибут ако той е функционално зависим от А или В;
- » Ако атрибутът не е функционално зависим от А или В не трябва да бъде добавян в релацията;
- » А и В трябва да се срещат заедно в R, но не трябва да се срещат заедно в друга релация.



# 1:N

- » 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.



# M:N

- » 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.



# Денормализация

- » Оптималният дизайн на релационна база от данни изисква всички таблици да бъдат поне в 3NF – избягвайки излишеството и аномалиите на промените;
- » Нормализирането на таблиците е важна част от дизайна, но не е единствена;
- » Добрият дизайн взема предвид и скоростта на обработка при извличане на данните;
- » Проблемът с нормализацията е, че таблиците са декомпозирани и броят им е нараснал. Затова при извличане на данни комбинирането им изисква множество JOIN операции, което увеличава I/O операциите, влияейки негативно на скоростта за изпълнение;
- » В някои ситуации може да бъде приета по-ниска степен на нормализация за сметка на увеличена ефективност при обработка на данните;
- » Не трябва да се забравя, че повишената ефективност трябва да бъде добре преценена, защото за нея се плаща с аномалии на промените (някои от които могат да бъдат управлявани обикновено без особени затруднения);



# Денормализация

Процес, често обратен на нормализацията, но не задължително – представлява **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 преди това други подходи за това са се провалили - индексирание), 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 интегритета на данните да не бъде нарушен.





# 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.
  
- » Обикновено:
  - > Стартира от “нормализиран” модел;
  - > Добавя излишество към дизайна;
  - > Понижава интегритета на дизайна;
  - > Се нуждае от приложен код, който да компенсира (понякога).



# Техники за денормализация

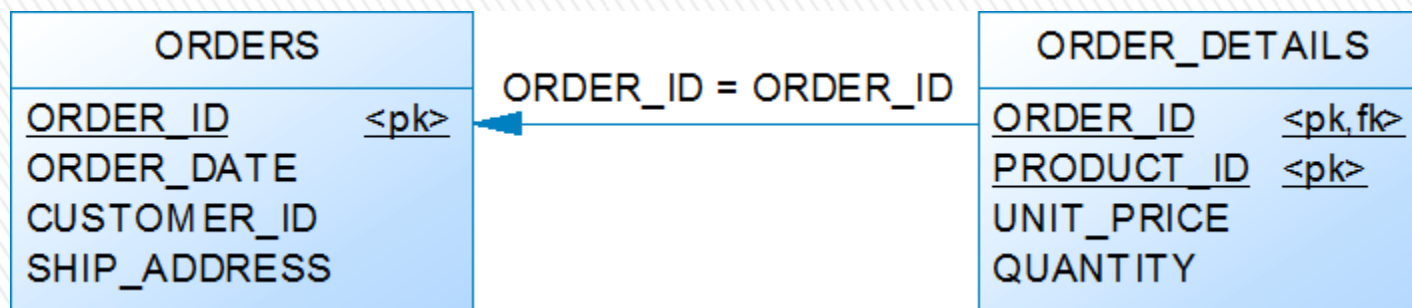
## 1. Съхраняване на стойности, които биха могли да бъдат извлечени.

- » 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 подсигурирм тази стойност да не може да се обновява директно с DML операция. Тази стойност трябва винаги да бъде преизчислявана от системата.

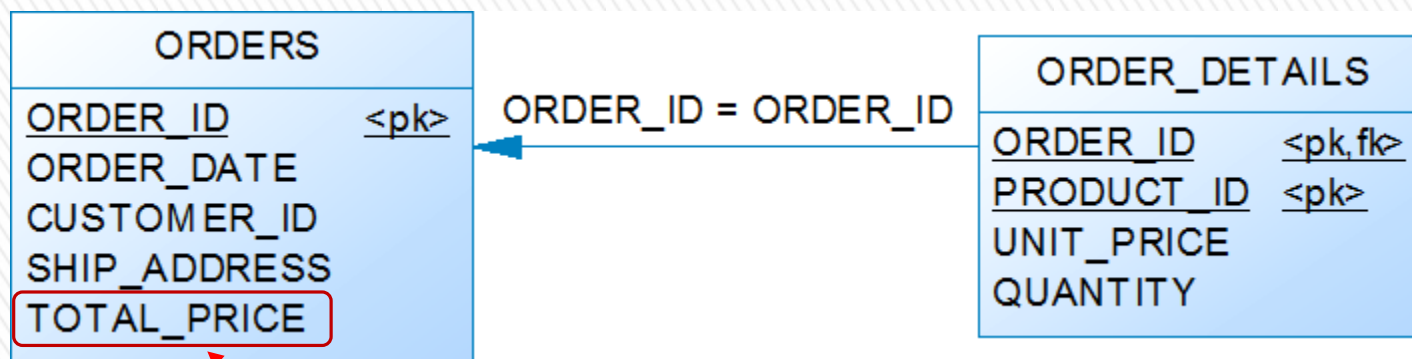


# 1. Съхраняване на стойности, които биха могли да бъдат извлечени

Преди



След



Добавяне на  
изчислена колона



# 1. 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.



## 2. Hard-coded стойности

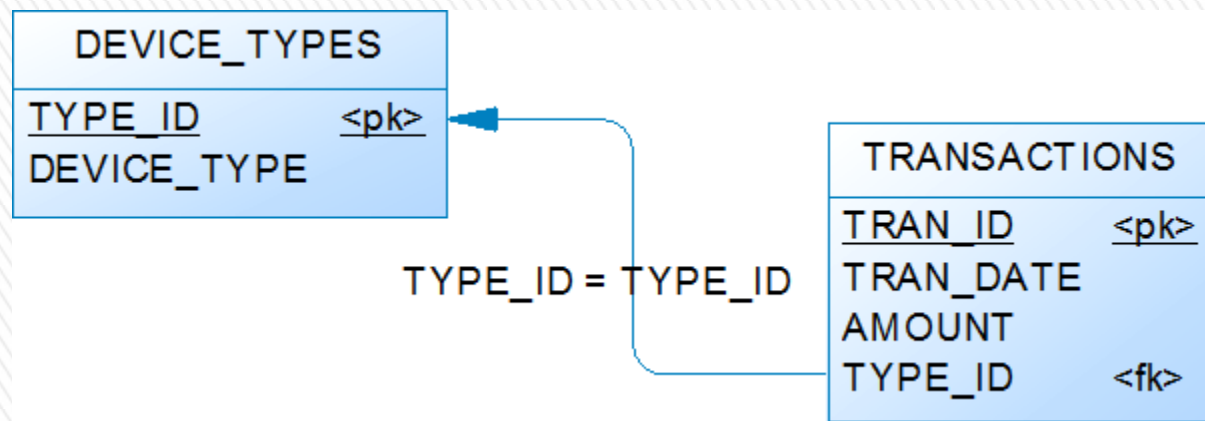
- » Ако референцираната таблица съдържа относително малко записи, които са константни, можем да обмислим “твърдо кодиране” на тия стойности в кода.
- » Това ще означава, че няма да има нужда от съединение на таблици за извличане на референцираните стойности.
- » Допустимо е да се помисли за създаване на check constraint на референциращата таблица, който да проверява за допустими стойности (вместо външен ключ).



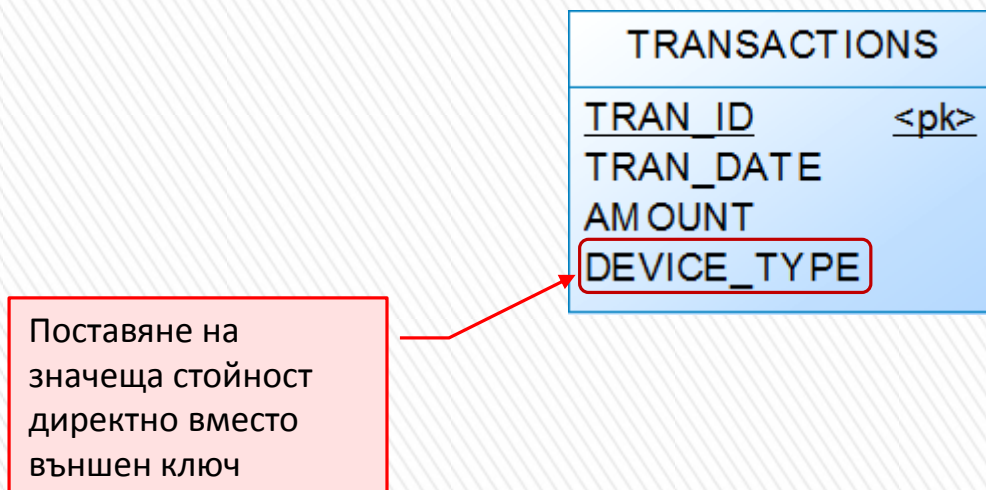


## 2. Hard-coded стойности

Преди



След



## 2. 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.



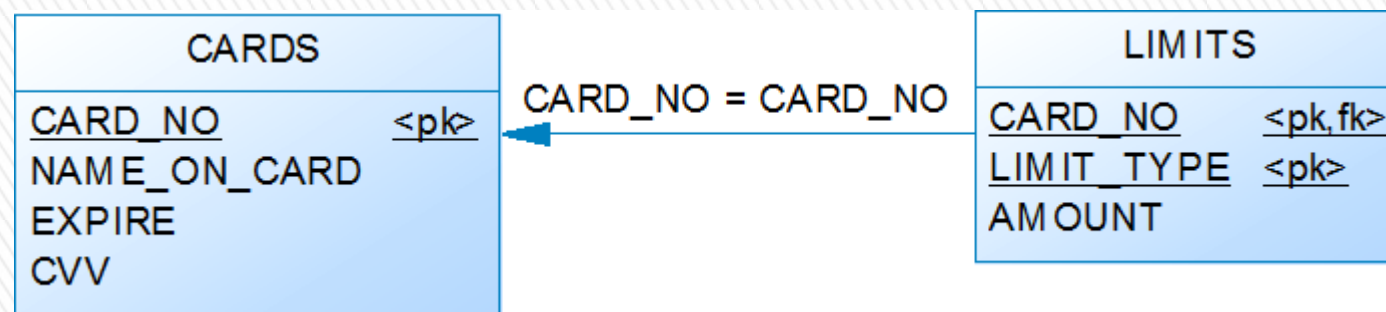
### 3. Запазване на детайлите в основната таблица

- » В ситуация, при която броят на детайлните записи за всеки основен е фиксиран, можем да обмислим добавянето на детайлните колони към основната таблица, спестявайки допълнителната.
- » Този вариант работи най-добре, когато броят на записите в детайлната таблица е малък. По този начин ще редуцираме броя на съединенията в заявката.

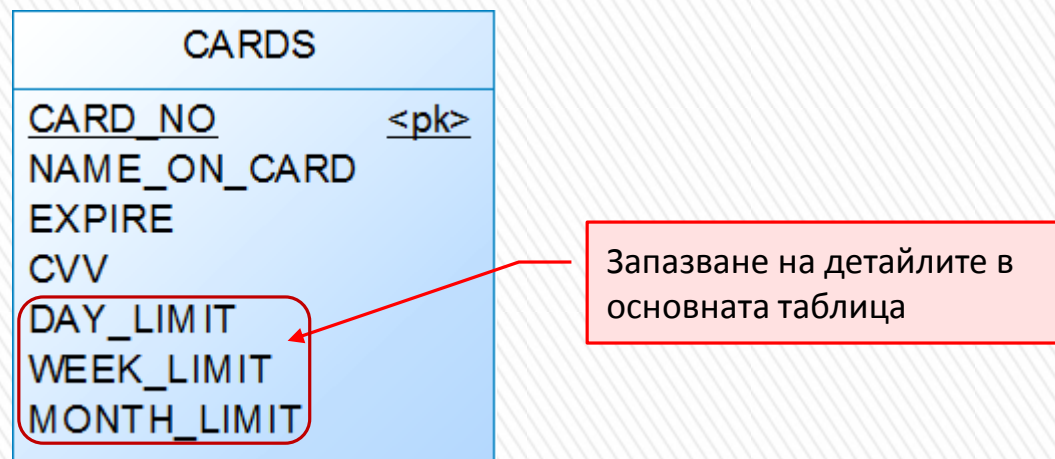


### 3. Запазване на детайлите в основната таблица

Преди



След



### 3. Запазване на детайлите в основната таблица

#### » Подходяща, когато:

- > броят на детайлните записи за основния е фиксиран или статичен;
- > (броят на детайлните записи) \* (броя 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.





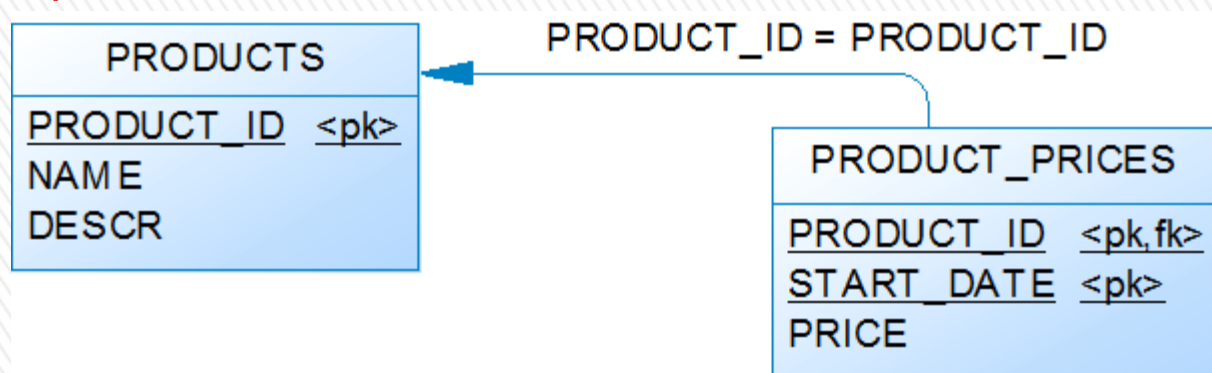
## 4. Повтаряне на единичен детайл в основната

- » 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.

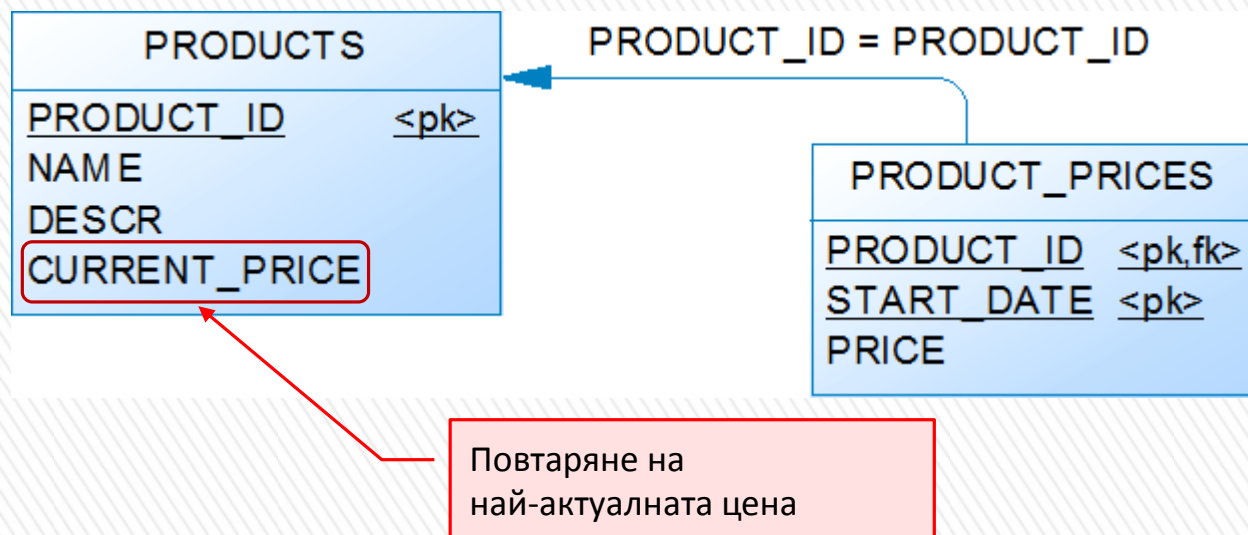


## 4. Повтаряне на единичен детайл в основната

Преди



След



## 4. 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.



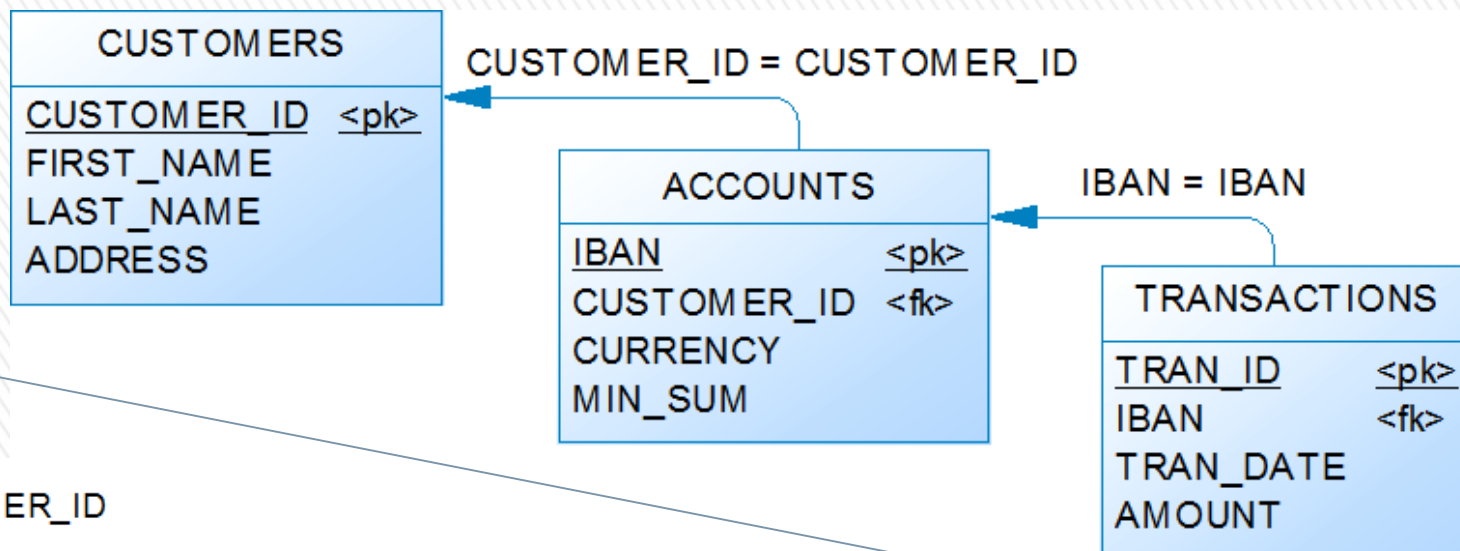
## 5. Short-Circuit ключове

- » 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.

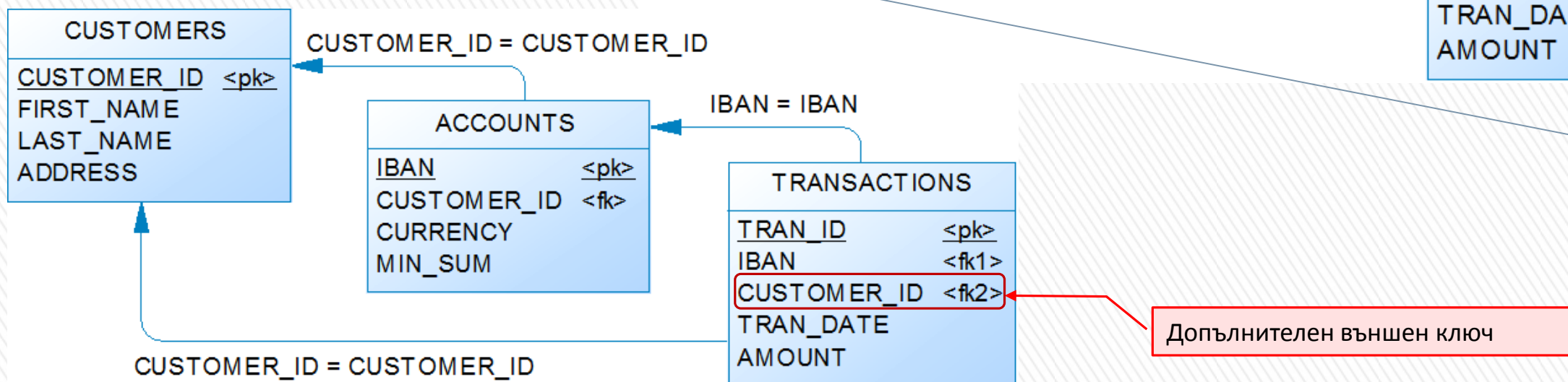


## 5. Short-Circuit ключове

Преди



След





## 5. 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.

