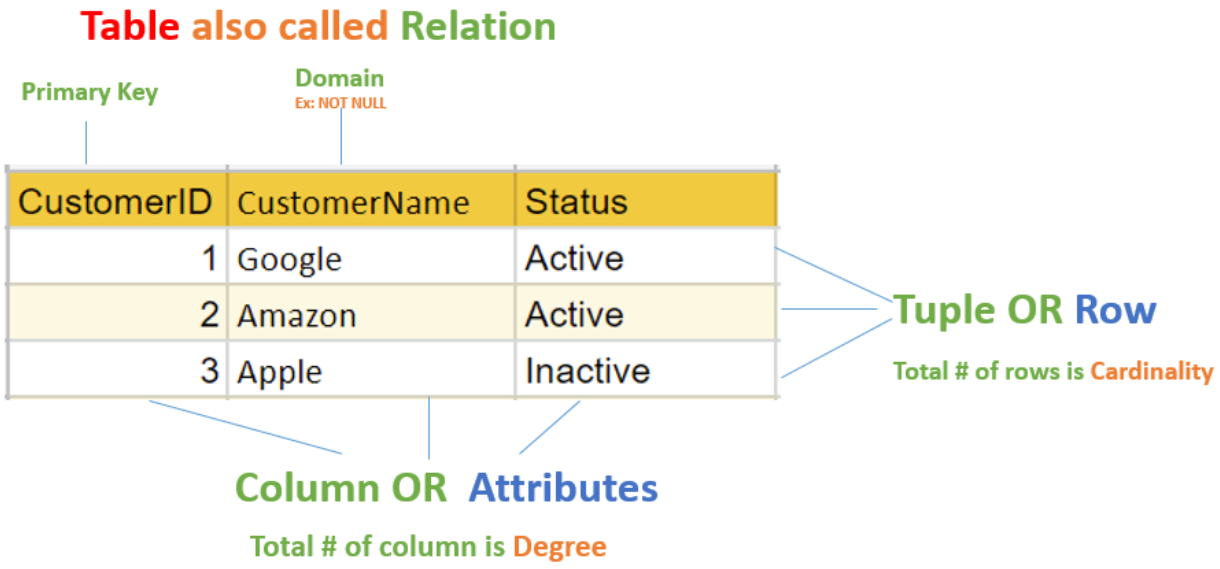
Релационни бази данни

I. Основна терминология

Релационните бази данни съществуват от 70-те години на миналия век. Терминът "релационна база данни" е въведен от E. F. Codd през 1970 г. в своя научно-изследователския труд "Релационен модел на данните за големи споделени банки от данни." Според Microsoft релационната база данни е вид база данни, която съхранява и осигурява достъп до данни, които са свързани помежду си. Релационните бази данни използват **релационен модел** за описание на данните. Този модел дефинира начин за представяне на данни в таблици и връзките между тези таблици. При релационния модел логическите структури (таблици) са отделени от физическите структури за съхранение на данните. Това разделяне означава, че управлението на физическото съхранение на данните не се отразява на достъпа до тези данни като логическа структура. Например, преименуването на файл на база данни не води до преименуване на таблиците, съхранявани в него. Основният начин за изпращане на заявки към релационните бази данни е чрез езика за структурирани заявки **SQL**. Той е основан на релационната алгебра и предоставя вътрешно съгласуван математически език, който улеснява подобряването на производителността на всички заявки към бази данни. Предназначен е за работа със структурирани данни. Той е декларативен език, защото описва какъв да бъде желания резултат. На Фиг. 1 са показани основните термини, използвани при релационен модел за описание на данните.



Фиг. 1 Основни термини в релационните бази данни

Таблица (Table): В релационния модел връзките се записват във формат на таблица. Тя се съхранява заедно с нейните същности. Една таблица има две свойства - редове и колони. Редовете представляват *записи*, а колоните - *атрибути*.

Атрибут (Attribute): Всяка колона в една таблица. Атрибутите са свойствата, които определят дадена връзка, например “customerName” и “Status”.

Област на атрибута (Attribute domain): Всеки атрибут има някаква предварително определена стойност и обхват, които са известни като домейн на атрибута.

Кортеж (Tuple): Един ред от таблица, който съдържа един запис.

Степен (Degree): Общият брой на атрибутите, които се съдържат в таблицата.

Кардиналност (Cardinality): Общият брой на редовете в таблицата.

Колона: Колоната представлява набор от стойности за определен атрибут.

Схема на релацията (Relation Schema): Схемата на релацията представя името на релацията с нейните атрибути.

Ключ на релацията (Relation key): Всеки ред има един, два или няколко атрибута, които се наричат ключ на релацията.

Инстанция на връзката (Relation instance): Инстанцията на релацията е крайно множество от кортежи в системите за управление на бази данни (СУБД). Релационните инстанции никога нямат дублиращи се кортежи.

Релационният модел организира данните в една или повече **таблици** (релации), всяка съставена от колони и редове. Редовете се наричат още **записи** или **кортежи**, а колоните – **полета** или **атрибути**. **Записът** е смислен и последователен начин за обединяване на информацията за нещо. Полето е отделен информационен елемент – тип елемент, който присъства във всеки запис. Например, в таблица “Студенти” всеки ред (запис) ще съдържа информацията за един студент. **Полетата** съдържат някакъв вид информация за студента, например неговото име. Релационният модел е добър за поддържане на последователност на данните в приложенията и копията на базата данни (наречени инстанции). Например, когато клиентът тегли пари чрез банкомат и след това преглежда баланса по сметката на мобилния си телефон, той очаква тегленето да бъде отразено незабавно в актуализиран баланс по сметката. Релационните бази данни са отлични в този вид **съгласуваност на данните**, като гарантират, че многобройните екземпляри на базата данни имат едни и същи данни през цялото време. Повечето нерелационни бази данни (NoSQL) осигуряват само "евентуална съгласуваност" на данните. Евентуалната съгласуваност е приемлива за много услуги, но не и за критичните бизнес операции, като транзакции при онлайн пазаруване и банкиране.

Четири ключови свойства определят транзакциите в релационните бази данни: атомарност, съгласуваност, изолираност и трайност - **ACID** бази данни:

- **Atomicy** (атомарност) – всяко действие (транзакция) с базата данни или се изпълнява изцяло или не се изпълнява (връща се грешка). Ако една част от транзакция се провали, се анулира цялата транзакция.
- **Consistency** (съгласуваност) – всяка транзакция променя базата данни от едно състояние с консистентна информация в друго консистентно състояние.
- **Isolation** (изолираност) – няма възможност за промяна на данни, които участват в все още незавършила операция. Това се гарантира с различни видове блокировки. Трябва да се има предвид, че тези блокировки има опасност да доведат до мъртва хватка и затова някои бази данни не гарантират 100% изолация.
- **Durability** (трайност) – промените в данните стават постоянни, само след като транзакцията бъде завършена.

В релационната база данни всеки ред в таблицата е запис с уникален идентификатор, наречен ключ. **Ключът** е стойност, която се използва за уникално идентифициране на записи в таблица. Ключът в SQL е единична колона или комбинация от няколко колони, използвани за уникално идентифициране на редове или кортежи в таблицата. SQL ключът се използва за идентифициране на дублираща се информация, а също така помага за установяване на връзка между няколко таблици в базата данни. Колоните в таблицата, които не се използват за уникално идентифициране на запис, се наричат неключови колони.

Кандидат ключ:

- Атрибут, който еднозначно идентифицира ред в релация.
- Може да бъде комбинация от (нередундантни) атрибути.
- Всяко неключово поле е функционално зависимо от всеки кандидат ключ.

Първичен ключ (primary key) на една релация R се нарича множеството от атрибути A, така избрани, че произволни два кортежа от таблицата не могат да имат еднакви стойности на атрибутите от множеството A. Първичният ключ трябва да отговаря на следните изисквания:

- Първичният ключ не може да бъде NULL.
- Стойността на първичния ключ трябва да е уникална.
- Стойностите на първичния ключ трябва рядко да се променят.
- Първичният ключ трябва да получи стойност, когато се вмъква нов запис.

Съставен ключ (composite key) е първичен ключ, съставен от няколко колони, използван за уникално идентифициране на запис. Например в една база данни може да има колона, която описва потребители и друга колона, която описва адресът на тези потребители. В тази база данни първичния ключ е съставен – той включва както името, така и адреса на потребителите. Причината за това е, че може да има няколко потребители с едно и също име, които обаче живеят на различни адреси.

Външният ключ (foreign Key) препраща към първичния ключ на друга таблица. Той помага да свържете таблиците си. Външният ключ може да има различно име от първичния ключ. Той гарантира, че редовете в една таблица имат съответни редове в друга таблица. За разлика от първичния ключ, външните ключове не е задължително да бъдат уникални. Външните ключове могат да бъдат нулеви, въпреки че първичните ключове не могат.

Функционални зависимости

Функционална зависимост (ФЗ) описва връзката между колоните в дадена таблица. Стойността на един атрибут (детерминанта) определя стойността на друг атрибут. A → B означава: "Атрибут B е функционално зависим от атрибут A". Ако два реда имат еднаква стойност на A, те задължително имат еднаква стойност и на B. Ако знаем стойността на A, във всички записи ще намерим само една стойност на B, която има тази стойност на A, във всеки един момент от време. Функционалните зависимости се определят от семантиката: не можете да кажете, че съществува функционална зависимост, само като погледнете данните. Но може да се каже дали тя не съществува, като се гледат данните.

Определение на ФЗ:

- X → Y
- X (функционално) определя Y или Y е функционално зависим от X.
- X - лява страна (LHS) или детерминанта, Y – дясна страна (RHS).
- За всяка стойност на X има най-много една стойност на Y.

Частична функционална зависимост - има атрибути, които зависят пряко от повече от един първичен ключ.

Преходна функционална зависимост - промяната на неключова колона може да доведе до промяна на някоя от другите неключови колони.

Пълна функционална зависимост - ако A и B са колони на таблица, B е напълно зависима от A, ако B е функционално зависима от A, но не и от някое подмножество на A.

Идентифициране на функционалните зависимости

Проектантите на бази данни трябва да могат да идентифицират ФЗ, когато събират изискванията към базата данни:

- Някои ФЗ могат да бъдат идентифицирани чрез твърдения за уникалност. Например всеки студент има уникален факултетен номер. Следователно зависимостта е следната: StudenID → Student.

- В повечето случаи ФЗ се откриват чрез търсене на всички 1-М връзки. ФЗ трябва да са в посока дете-родител, а не в посока родител-дете.

  Например твърдението "В един факултет се преподават много дисциплини, но дадена дисциплина се преподава само в конкретен факултет" определя ФЗ от уникална колона на дисциплините към уникална колона на факултета, например SubjectNo → FacultyNo. Функционалната зависимост FacultyNo → SubjectNo е неправилна, защото ФЗ трябва да позволява най-много една асоциирана стойност, а не колекция от стойности (дисциплини).
- ФЗ, в които LHS не е първичен ключ или кандидат ключ, също могат да бъдат трудни за идентифициране. Трябва внимателно да търсите ФЗ, в които LHS не е кандидат ключ или първичен ключ.
- Трябва също така да разгледате ФЗ в таблици със съставен първичен ключ или кандидат ключ, в които LHS е част от ключа, но не е целият ключ. Нормализацията с използване на подобни ФЗ могат да доведат до аномалии при модификация.
- Друго важно съображение при изборът на ФЗ е минимализмът на LHS. Важно е да се разграничи кога само една колона е определяща спрямо комбинация от колони. ФЗ, в който LHS съдържа повече от една колона, обикновено представлява М-Н връзка. Например твърдението "При формиране на количеството на закупените стоки на една поръчката се отчита количеството на всеки продукт, закупен в поръчката". Като ФЗ това се записва по следния начин: OrdNo, ProdNo → OrdQty – количеството зависи както от номера на поръчката, така и продуктите в поръчката, а не само от един от тези атрибути.
- Част от объркването относно минимализма на LHS се дължи на значението на колоните в лявата спрямо дясната страна на функционалната зависимост. Минимална детерминанта означава, че детерминантата (колоната(ите), която(ито) се появява(т) в LHS на ФЗ не трябва да съдържа(т) допълнителни колони и това изискване за минимализъм е подобно на изискването за минимализъм на кандидат-ключовете. Например, за да запишете, че социално осигурителният номер на ученика определя града и класа, можете да напишете:

  `StdSSN → StdCity, StdClass`

  Ако приемете, че имейл адресът също е уникален за всеки ученик, тогава можете да напишете:

  Email → StdCity, StdClass

  Следната ФЗ обаче не е коректна:

  StdSSN, Email → StdCity, StdClass

  Това е така, защото тези ФЗ предполагат, че комбинацията от StdSSN и Email е определяща. Следователно трябва да напишете ФЗ, така че LHS да не съдържа ненужни колони.
- Забраната за ненужни колони за детерминанти е същата като забраната за ненужни колони в кандидат-ключове. Както детерминантите, така и кандидат-ключовете трябва да са минимални.
- Съществуват няколко програмни инструмента за проектиране на бази данни, които автоматизират процеса на премахване на зависимостите чрез анализ на примерни редове. В крайна сметка проектантът на бази данни трябва да вземе окончателното решение за ФЗ, които съществуват в дадена таблица.

II. Проектиране на релационни бази данни

При проектиране на базата данни трябва да се следи за:

- Намаляване на излишните данни в базата чрез разделяне на изходната информация на необходимия брой таблици.
- Правилен подбор на първичните и външни ключове.
- Отговаря ли базата данни на изискванията на клиента от гледна точка на надеждност, сигурност, бързодействие, мащабируемост и изготвяне на справки и отчети?

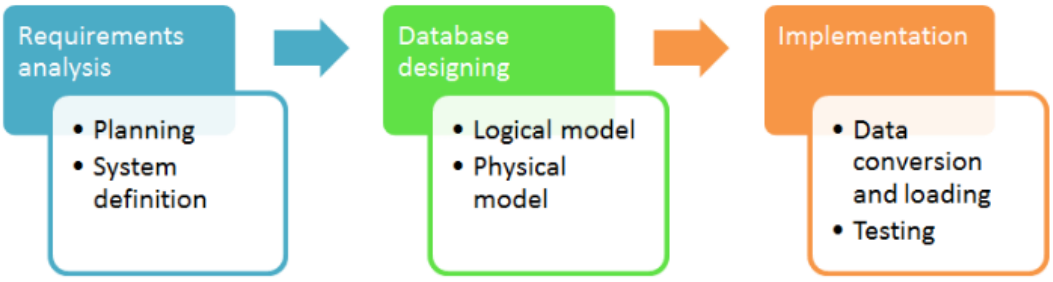
Проектирането на базата данни преминава през следните по-важни стъпки:

- Определяне на предназначението на базата данни.
- Намиране и организиране на необходимата информация.
- Разделяне на информацията в таблици.
- Превръщане на информационните елементи в колони.
- Задаване на първични ключове.
- Дефиниране на релациите между таблиците.
- Прилагане на правила за нормализиране на базата данни.
- Населяване на базата с тестови данни. Провеждане на необходимите тестове.

Проектирането на бази данни е съвкупност от процеси, които улесняват разработването, внедряването и поддръжката на корпоративни системи за управление на данни. Правилно проектираните бази данни са лесни за поддръжка и са рентабилни по отношение на дисковото пространство за съхранение. Дизайнерът на бази данни решава как елементите на данните се съотнасят и какви данни трябва да се съхраняват.

Основните цели на проектирането на бази данни в СУБД са да се създадат логически и физически модели на проектите на предложената система от бази данни. Логическият модел се концентрира върху изискванията към данните и данните, които трябва да се съхраняват, независимо от физическите съображения. Той не се занимава с това как ще се съхраняват данните или къде ще се съхраняват физически. Моделът на физическия дизайн на данните включва пренасяне на логическия дизайн на базата данни върху физически носители с помощта на хардуерни ресурси и софтуерни системи, като например системи за управление на бази данни (СУБД). Процесът на проектиране на бази данни в СУБД е от решаващо значение за високопроизводителната система от бази данни.

Жизненият цикъл на разработване на бази данни има няколко етапа, които се следват при разработването на бази данни. Етапите в жизнения цикъл на разработката не е задължително да се прилагат стриктно. При малките системи процесът на проектиране на бази данни обикновено е много прост и не включва много стъпки. Нека да разгледаме отделните компоненти, изброени във всяка стъпка от жизнения цикъл на проектиране на бази данни.



Фиг. 2. Жизнен цикъл при проектиране на бази данни

1. Анализ на изискванията

- Планиране - този етап от концепциите за проектиране на бази данни се отнася до планирането на целия жизнен цикъл на разработване на бази данни. При него се взема предвид стратегията на информационните системи на организацията.
- Определяне на системата (System definition) - на този етап се определят обхватът и границите на предлаганата система за бази данни.

2. Проектиране на бази данни

Логически модел - Този етап се занимава с разработването на модел на базата данни въз основа на изискванията. Цялото проектиране е на хартия, без физически реализации или специфични съображения за СУБД.

Физически модел - На този етап се реализира логическият модел на базата данни, като се вземат предвид факторите на СУБД и физическата реализация.

3. Реализация

Конвертиране и зареждане на данни - този етап от проектирането на релационни бази данни е свързан с импортирането и конвертирането на данни от старата система в новата база данни.

Тестване - този етап се занимава с идентифицирането на грешки в ново приложената система. При него базата данни се проверява спрямо спецификациите на изискванията.

Има два вида техники за работа с бази данни

- Нормализация.
- ER моделиране.

Какво представлява ER моделирането?

Моделът на взаимоотношенията на същностите (ER Modeling) е графичен подход към проектирането на бази данни. Това е модел на данни от високо ниво, който определя елементите на данните и техните взаимоотношения за определена софтуерна система.

III. Нормализация на базите данни

Нормализацията е формален процес на декомпозиране на релации с аномалии с цел получаване на по-малки, добре структурирани и стабилни релации. Нормализацията е инструмент за валидиране и подобряване на логическия дизайн, така че той да отговаря на определени ограничения, които избягват ненужното дублиране на данни. Нормализацията е част от процеса на оптимизация на базата данни. След нормализация базата данни трябва да има структура, която минимизира дублирането на данни, без наличие на аномалии в базата данни.

Добре структурирани релации:

Релация, която съдържа минимално количество излишни данни и позволява на потребителите да вмъкват, изтриват и актуализират редове, без да предизвикват несъответствия в данните. Целта е да се избегнат (сведат до минимум) аномалии с данните:

- Аномалия при въвеждане - добавянето на нови редове принуждава потребителя да създаде дублирани данни.
- Аномалия при изтриване - изтриването на ред може да доведе до загуба на данни други данни, представляващи напълно различни факти.
- Аномалия при модифициране - промяната на данни в ред принуждава промени в други редове поради дублиране.

Правилата за **нормализация** се прилагат последователно, така че на всяка стъпка да се уверите, че вашият проект отговаря на едно от така наречените "правила за нормализация". Има няколко широко разпространени правила за нормализация:

- Първа нормална форма - First Normal Form (1NF).
- Втора нормална форма - Second Normal Form (2NF).
- Трета нормална форма - Third Normal Form (3NF).
- Бойс-Код нормална форма - Boyce/Codd Normal Form (3.5NF, BCNF).
- Четвърта нормална форма - Fourth Normal Form (4NF).
- Пета нормална форма - Fifth Normal Form (5NF).
- Шеста нормална форма - Sixth Normal Form (6NF).

В повечето практически случаи е достатъчно да достигнете до трета нормална форма. В Табл. 1 е показано кога една същност е в една от първите три нормални форми и какво трябва да се направи, за да се премине до трета нормална форма.

Табл. 1. Нормализация до трета нормална форма

0NF	
Имат ли някои атрибути няколко стойности за една инстанция на същност?	**Да:** Премахнете повтарящите се атрибути и повтарящите се групи. Създайте същност, която описва тези атрибутите. Обикновено ще трябва да добавите връзка, за да свържете старите и новите същности.
	Не: Таблицата е в 1NF
1NF	
Състои ли се ключа от повече от един атрибут? Ако е така, зависи ли някоя от стойностите на атрибут само от част от ключа?	**Да:** Премахнете частичната зависимост. Преместете атрибутите към същност, в която стойностите им зависят от целия ключ. Обикновено трябва да създадете нова същност и да добавите връзка, за да свържете старата и новата същност.
	Не: Таблицата е в 2NF
2NF	
Зависи ли някоя от стойностите на атрибут от друг атрибут, който не е ключ на същността?	**Да:** Премахнете преходната зависимост или производния атрибут. Преместете атрибутите в същност, в която стойностите им зависят изцяло от ключа. Обикновено ще трябва да създадете нова същност и да добавите връзка, за да свържете старата и новата същност.
	Не: Таблицата е в 3NF

Първа нормална форма (1NF)

- Само атомарни (прости, с една стойност) атрибути.
- Идентифициран е първичен ключ.
- Всеки запис трябва да бъде уникален

До 1NF се достига чрез идентифициране и премахване на атрибутите с множество стойности чрез вмъкване на нови кортежи.

Втора нормална форма (2NF)

- 2NF се прилага само за таблици със съставни първични ключове (повече от един първичен ключ).
- Релацията е в 1NF, всеки неключов атрибут е напълно функционално зависим от целия първичен ключ.
- Няма частични функционални зависимости. Нарушението на 2NF възниква при функционална зависимост, при която част от ключа (вместо целите ключове) определя неключов атрибут.

До 2NF се достига чрез премахване на частичните функционални зависимости.

Трета нормална форма (3NF)

- Релацията е в 2NF. Една таблица е в 3NF, ако нито една колона, която не е първичен ключ, не е транзитивно зависима от първичния ключ.
- Ако съществува транзитивна зависимост върху първичния ключ, таблицата не е в 3NF. Следователно, трябва да няма преходни функционални зависимости.

До 3NF се достига чрез премахване на преходните зависимости. Нарича се транзитивна зависимост, защото първичният ключ е определящ за друг атрибут, който на свой ред е определящ за трети атрибут. Преходна зависимост е тази, при която неключов атрибут зависи от друг неключов атрибут.

Бойс-Код нормална форма (3.5NF)

- За да бъде в 3NF, но не и в BCNF, са необходими два съставни кандидат-ключа, като един атрибут на единия ключ зависи от един атрибут на другия.
- Ако дадена таблица съдържа само един кандидат ключ, 3NF и BCNF са еквивалентни.
- Една релация е в BCNF, ако и само ако всеки детерминант е кандидат ключ.
- Разликата между 3NF и BCNF се състои в това, че за функционална зависимост A→B при 3NF позволява тази зависимост в дадена релация, ако B е атрибут на първичен ключ, а A не е кандидат ключ, докато при BCNF, за да остане тази зависимост в дадена релация, A трябва да е кандидат ключ.

До 3.5NF се достига чрез премахване на останалите аномалии, произтичащи от множество кандидат-ключове.

Четвърта нормална форма (4NF)

- Трябва да имате многозначни зависимости, обобщение на функционалните зависимости.

Ако нито един екземпляр на таблица от базата данни не съдържа две или повече независими и многозначни данни, описващи съответната същност, тогава тя е в 4-та нормална форма.

Многозначна зависимост е налице, когато:

- Има поне 3 атрибута A, B, C в дадена релация и
- За всяка стойност на A има добре дефинирано множество от стойности за B и добре дефинирано множество от стойности за C, но множеството от стойности за B е независимо от множеството от стойности за C.

Пета нормална форма (5NF)
Една таблица е в пета нормална форма само ако е в 4NF и не може да бъде разложена на произволен брой по-малки таблици без загуба на данни.

Шеста нормална форма (6NF)
Шестата нормална форма все още не е стандартизирана.

IV. Примери за нормализация на бази данни

Пример 1:

Да се създаде база данни за преподавателите от дадена катедра. Базата данни да съдържа информация за: имената на преподавателите; дисциплините, които преподават; специалностите на студентите на които преподават и името на базовото звено в което са назначени на работа (катедра). Заявките към базата данни са следните:

- Какви дисциплини води даден преподавател?
- Какви дисциплини води даден преподавател от избрана специалност?
- Кои преподаватели водят учебен процес на избрана специалност?
- Кои преподаватели преподават избрана дисциплина?

Първичната таблица, която описва данните от бъдещата база данни, е показана в Табл. 2. Таблицата има пет колони. Чрез колона 1 се описва идентификатора на всеки един от преподавателите. Това е необходимо, тъй като всички заявки са свързани по някакъв начин с преподавателите. Ако няма специален начин за формиране на този идентификатор в съответния Университет, може да използвате представянето му с цяло неповтарящо се число, което се инкрементира при добавяне на нов преподавател. Останалите колони описват: името на преподавателя; катедрата в която работи; специалностите на които преподава и дисциплините, които преподава.

Табл. 2. Таблично представяне на данните – 0NF

id	name	department	specialty	subject
1	Росен Иванов	КСТ	КСТ, СКИ	ОК, ПИС, НБД
2	Делян Генков	КСТ	КСТ, СКИ	БД, КМ, КМС, ВОТ
3	Валентина Кукенска	КСТ	КСТ, СКИ	ОС, ИС, ППИ
4	Мирослав Славов	КСТ	КСТ, СКИ	КМ, КМС

Тази таблица не е нормализирана (0NF), защото има атрибути (колони) с няколко възможни стойности: “specialty” и “subject”. За да преминем до 1NF е необходимо да се премахнат повтарящите се атрибути. Това се реализира чрез добавяне на нови кортежи в таблицата (редове), както е показано в Табл. 3.

Табл. 3. Таблично представяне на данните – 1NF

id	name	department	specialty	subject
1	Росен Иванов	КСТ	КСТ	ОК
1	Росен Иванов	КСТ	КСТ	ПИС
1	Росен Иванов	КСТ	СКИ	ОК
1	Росен Иванов	КСТ	СКИ	ПИС
1	Росен Иванов	КСТ	СКИ	НБД
2	Делян Генков	КСТ	КСТ	БД
2	Делян Генков	КСТ	КСТ	КМ
2	Делян Генков	КСТ	СКИ	БД

2	Делян Генков	КСТ	СКИ	КМ
2	Делян Генков	КСТ	СКИ	КМС
2	Делян Генков	КСТ	СКИ	ВОТ
3	Валентина Кукенска	КСТ	КСТ	ОС
3	Валентина Кукенска	КСТ	КСТ	ИС
3	Валентина Кукенска	КСТ	СКИ	ОС
3	Валентина Кукенска	КСТ	СКИ	ИС
3	Валентина Кукенска	КСТ	СКИ	ППИ
4	Мирослав Славов	КСТ	КСТ	КМ
4	Мирослав Славов	КСТ	СКИ	КМ
4	Мирослав Славов	КСТ	СКИ	КМС

Ясно се вижда, че така се получава дублиране на част от информацията, което ще доведе до безсмислено нарастване на размера на базата данни с времето. Следователно, трябва да преминем към 2NF.

Една база данни е във 2NF, ако тя вече е в 1NF и всеки атрибут от нея, който не е част от ключа, зависи изцяло от първичния ключ (не зависи от кое да е подмножество атрибути на първичния ключ). Следователно, за да достигнем до 2NF е необходимо да дефинираме първичния ключ за таблицата, която е в 1NF.

Съгласно дефиницията за първичен ключ, при конкретния пример този ключ е съставен: {id, specialty, subject}. При този ключ има две функционални зависимости, които нарушават изискванията за 2NF:

- id → name (ФЗ-1);
- specialty → department (ФЗ-2).

Съгласно Табл. 1 трябва да премахнем частичните зависимости. За целта ще преместим атрибутите, които зависят частично от ключа, към нова таблица, в която стойностите им зависят от целия ключ. Първата функционална зависимост ФЗ-1 води до разделяне на първоначалната таблица на следните две таблици:

- R1 (id, name).
- R2 (id, subject, specialty, department).

Втората функционална зависимост ФЗ-2 разделя релация R2 на следните релации:

- R3 (specialty, department).
- R4 (id, subject, specialty).

Следователно, при 2NF данните ще бъдат описани с три релации: R1, R3 и R4 (Табл. 4).

Табл. 4. Таблично представяне на данните – 2NF

Релация R1 (Teacher) – ключ “id”

id	name
1	Росен Иванов
2	Делян Генков
3	Валентина Кукенска
4	Мирослав Славов

Релация R3 (Specialty-Department) – ключ “specialty”

specialty	department
КСТ	КСТ
СКИ	КСТ

Релация R4 (Subject-Specialty) – ключ “teacherId”

teacherId	subject	specialty
1	ОК	КСТ
1	ОК	СКИ
1	ПИС	КСТ
1	ПИС	СКИ
1	НБД	СКИ
2	БД	КСТ
2	БД	СКИ
2	КМ	КСТ
2	КМ	СКИ
2	КМС	СКИ
2	ВОТ	СКИ
3	ОС	КСТ
3	ОС	СКИ
3	ИС	КСТ
3	ИС	СКИ
3	ППИ	СКИ
4	КСТ	КМ
4	СКИ	КМ
4	СКИ	КМС

Нека да проверим дали таблицата не е и в 3NF. За целта трябва да проверим дали някоя от стойностите на атрибут зависи от друг атрибут, който не е ключ на таблицата. Единствената релация, която има повече от две колони е R4. Атрибути “subject” и “specialty” не зависят един от друг и следователно базата данни е и в 3NF.

Тестване на базата данни

Създайте необходимите таблици и въведете тестови данни:

```
-----------------------------------------------------------------------------------------------------------
create table Teacher(id integer, name varchar(100));
insert into Teacher(id, name) values
      (1, 'Росен Иванов'),
      (2, 'Делян Генков'),
      (3, 'Валентина Кукенска'),
      (4, 'Мирослав Славов');

-----------------------------------------------------------------------------------------------------------
create table SpecialtyDepartment(specialty varchar(10), department
varchar(10));
insert into SpecialtyDepartment(specialty, department) values
      ('КСТ', 'КСТ'),
      ('СКИ', 'КСТ');

-----------------------------------------------------------------------------------------------------------
create table SubjectSpecialty(teacherId integer, subject varchar(10),
specialty varchar(10));
insert into SubjectSpecialty(teacherId, subject, specialty) values
      (1, 'ОК', 'КСТ'),
      (1, 'ОК', 'СКИ'),
      (1, 'ПИС', 'КСТ'),
      (1, 'ПИС', 'СКИ'),
      (1, 'НБД', 'СКИ'),
      (2, 'БД', 'КСТ'),
      (2, 'БД', 'СКИ'),
      (2, 'КМ', 'КСТ'),
      (2, 'КМ', 'СКИ'),
      (2, 'КМС', 'СКИ'),
      (2, 'ВОТ', 'СКИ'),
      (3, 'ОС', 'КСТ'),
      (3, 'ОС', 'СКИ'),
      (3, 'ИС', 'КСТ'),
      (3, 'ИС', 'СКИ'),
      (3, 'ППИ', 'СКИ'),
      (4, 'КМ', 'КСТ'),
      (4, 'КМ', 'СКИ'),
      (4, 'КМС', 'СКИ');

-----------------------------------------------------------------------------------------------------------
```

Задачи за изпълнение:

Задача 1: Какви дисциплини води даден преподавател?

```
SELECT DISTINCT SubjectSpecialty.subject
FROM SubjectSpecialty
INNER JOIN Teacher
ON SubjectSpecialty.teacherId = Teacher.id
WHERE Teacher.name = 'Росен Иванов'
```

Задача 2: Какви дисциплини води даден преподавател в дадена специалност?

```
SELECT SubjectSpecialty.subject
FROM SubjectSpecialty
INNER JOIN Teacher
ON SubjectSpecialty.teacherId = Teacher.id
WHERE SubjectSpecialty.specialty = 'СКИ' AND Teacher.name = 'Росен Иванов'
```

Задача 3: Какви дисциплини (в азбучен ред) се водят в дадена специалност?

```
SELECT SubjectSpecialty.subject
FROM SubjectSpecialty
INNER JOIN Teacher
ON SubjectSpecialty.teacherId = Teacher.id
WHERE SubjectSpecialty.specialty = 'СКИ'
ORDER BY SubjectSpecialty.subject ASC
```

Задача 4: Кой (кои) преподавател (и) води (ят) зададена дисциплина?

```
SELECT DISTINCT Teacher.name
FROM Teacher
INNER JOIN SubjectSpecialty
ON Teacher.id = SubjectSpecialty.teacherId
WHERE SubjectSpecialty.subject = 'КМ'
```

Задача 5: Кои преподаватели водят зададена дисциплина?

```
SELECT Teacher.name
FROM Teacher
JOIN SubjectSpecialty
ON Teacher.id = SubjectSpecialty.teacherId
WHERE SubjectSpecialty.subject = 'КМ'
GROUP BY Teacher.name
ORDER BY Teacher.name ASC
```

Задача 6: Колко на брой дисциплини води даден преподавател?

```
SELECT COUNT(DISTINCT SubjectSpecialty.subject) as 'Брой дисциплини:'
FROM SubjectSpecialty
INNER JOIN Teacher
ON SubjectSpecialty.teacherId = Teacher.id
WHERE Teacher.name = 'Валентина Кукенска'
```

Задача 7: Кои преподаватели водят учебен процес на избрана специалност?

```
SELECT DISTINCT Teacher.name
FROM Teacher
JOIN SubjectSpecialty
ON Teacher.id = SubjectSpecialty.teacherId
WHERE SubjectSpecialty.specialty = 'КСТ'
ORDER BY Teacher.name ASC
```