Основи проектування баз даних

Створення бази даних та таблиць

Якісне проектування бази даних може спростити роботу з нею. З добре спроектованою базою даних легше працювати, легше писати запити. І в цьому посібнику ми розглянемо основні засади проектування баз даних.

Для якісного проектування бази даних існують різні методики, різні послідовності кроків чи етапів, які багато в чому схожі. І загалом ми можемо виділити такі етапи:

1. Виділення сутностей та його атрибутів, які зберігатимуться у базі даних, і формування з них таблиць. Атомізація складних атрибутів більш прості.
2. Визначення унікальних ідентифікаторів (первинних ключів) об'єктів, що зберігаються у рядках таблиці
3. Визначення відносин між таблицями за допомогою зовнішніх ключів
4. Нормалізація бази даних

У першому етапі відбувається виділення сутностей. **Сутність** (entity) є типом об'єктів, які повинні зберігатися в базі даних. Кожна таблиця у базі даних має представляти одну сутність. Як правило, сутності відповідають об'єктам із реального світу.

Кожна сутність визначає набір атрибутів. **Атрибут** є властивістю, яка описує деяку характеристику об'єкта.

Кожен стовпець повинен зберігати один атрибут сутності. А кожен рядок представляє окремий об'єкт чи екземпляр сутності.

Висхідний та низхідний підходи

При проектуванні бази даних на етапі виділення сутностей та їх атрибутів ми можемо використовувати два підходи: висхідний та низхідний.

Висхідний підхід передбачає виділення необхідних атрибутів, які треба зберегти у бд. Потім виділені атрибути групуються по суті, котрим згодом

створюється таблиці. Такий підхід найбільше підходить для проектування невеликих баз даних з невеликою кількістю атрибутів.

Наприклад, нам дано таку інформацію:

```
Том відвідує курс математики, який викладає професор Сміт.
Сем відвідує курс математики, які викладає професор Сміт.
Том відвідує курс з мови JavaScript, який викладає помічник Адамс.
Боб відвідує курс з алгоритмів, який викладає помічник Адамс.
Сем має такі електронну адресу sam@gmail.com та телефон +1235768789.
```

Які дані ми можемо зберегти: ім'я студента, назва курсу, навчальна посада викладача, ім'я викладача, електронна адреса студента.

Потім ми можемо виконати угруповання по суті, до яких належать ці дані:

Студент	Викладач	Курс
Ім'я студента	Ім'я викладача	Ім'я студента
Назва курсу	Посада викладача	Ім'я викладача
Дата народження студента	Назва курсу	Назва курсу
Електронна адреса студента		
Телефон студента		

Так, дані, які є дозволяють виділити три сутності: студент, викладач і курс. При цьому ми цілком можемо додавати якісь відсутні дані. Також слід зазначити, що якісь дані можуть стосуватися різних сутностей. Наприклад, курс зберігає інформацію про студента, який його відвідує. А студент зберігає інформацію про відвідуваний курс. Подібна надмірність даних вирішується на наступних кроках проектування у процесі нормалізації бази даних.

Але подібних атрибутів може бути дуже багато: сотні і навіть тисячі. І в цьому випадку оптимальнішим буде низхідний підхід. Цей підхід передбачає виявлення сутностей. Потім відбувається аналіз сутностей, виявляються зв'язок між ними, та був і атрибути сутностей.

Тобто в даному випадку ми могли б одразу визначити, що нам треба зберігати дані щодо студентів, курсів та викладачів. Потім у межах кожної сутності виявити атрибути

Наприклад, у сутності "Студент" ми могли б виділити такі атрибути, як ім'я студента, його адресу, телефон, зростання, вагу, рік народження. У той же час нам треба враховувати не взагалі всі властивості, які в принципі можуть бути

у сутності "Студент", а лише ті, які мають значення в рамках системи, що описується. Навряд чи в цьому випадку відіграють роль такі властивості, як зростання або вага студента, тому ми можемо їх викреслити зі списку атрибутів при проектуванні таблиці.

Іноді підходи комбінуються. Для опису різних частин системи можна використовувати різні підходи. А потім їх результати поєднуються.

Атомізація атрибутів

При визначенні атрибутів відбувається поділ складних комплексних елементів більш прості. Так, у випадку з ім'ям студента ми можемо його розбити на власне ім'я та прізвище. Це дозволить згодом виконувати операції з цими поделементами окремо, наприклад, сортувати студентів лише на прізвище.

Те саме стосується адреси - ми можемо зберегти всю адресу цілком, а можемо розбити її на частини - будинок, вулицю, місто тощо.

У той самий час можливість поділу одного елемента на поделементи який завжди може бути затребуваною. У низці завдань це може бути просто не потрібно. Виділяти необхідно лише ті елементи, які справді потрібні.

Відповідно до цього аспекту ми можемо виділити у сутності "Студент" наступні атрибути: ім'я студента, прізвище студента, рік народження, місто, вулиця, будинок, телефон.

Домен

Кожен атрибут має **домен** . Домен представляє набір допустимих значень одного чи кількох атрибутів. По суті, домен визначає зміст і джерело значень, які можуть мати атрибути.

Домени можуть відрізнятися для різних атрибутів, але кілька атрибутів можуть мати один домен.

Наприклад, вище було визначено атрибути сутності студента. Визначимо використовувані домени:

- **Ім'я** . Домен представляє всі можливі імена, які можна використовувати. Кожне ім'я становить рядок довжиною максимум 20

символів (малоймовірно, що нам можуть зустрітися імена понад 20 символів).

- **Прізвище** . Домен представляє всі можливі прізвища, які можна використовувати. Кожне прізвище представляє рядок довжиною максимум 20 символів.
- **Рік народження** . Домен представляє всі роки народження. Щороку є числовим значенням від 1950 до 2017 року.
- **Місто** . Домен представляє усі міста поточної країни. Кожне місто представляє рядок завдовжки максимум 50 символів.
- **Вулиця** . Домен представляє всі вулиці поточної країни. Кожна вулиця становить рядок довжиною максимум 50 символів.
- **Будинок** . Домен представляє всі можливі номери будинків. Кожен номер будинку є числом від 1 до, скажімо, 10 000.
- **Телефон** . Домен представляє всі телефонні номери. Кожен номер є рядком довжиною 11 символів.

Визначаючи домен, ми відразу бачимо, які дані та яких типів зберігатимуть атрибути. Якесь інше значення, яке відповідає домену, атрибут мати не може. У прикладі вище, кожен атрибут має свій домен. Але домени можуть збігатися. Наприклад, якби сутність містила б такі два атрибути: місто народження та місто проживання, то домен би збігався і був би одним і тим самим для обох атрибутів.

Визначник NULL

При визначенні атрибутів та їхнього домену необхідно проаналізувати, а чи може у атрибута бути відсутнім значення. Визначник NULL дозволяє встановити відсутність значення. Наприклад, у прикладі вище у студента обов'язково має бути якесь ім'я, тому неприпустима ситуація, коли атрибут, який представляє ім'я, не має значення.

У той же час студент може не мати номер телефону або в рамках системи телефон не обов'язковий. Тому на етапі проектування таблиці можна вказати, що цей атрибут дозволяє значення NULL.

Як правило, більшість сучасних реляційних СУБД підтримують визначник NULL і дозволяють встановити його допустимість для стовпця таблиці.

Ключі

Ключі є способом ідентифікації рядків у таблиці. За допомогою ключів ми також можемо зв'язувати рядки між різними таблицями у відносинах.

Суперключ

Superkey (суперключ) - комбінація атрибутів (стовпців), які унікально ідентифікують кожен рядок таблиці. Це можуть бути і всі стовпці, і кілька, і один. При цьому рядки, які містять значення цих атрибутів, не повинні повторюватися.

Наприклад, у нас є сутність Student, яка подає дані про користувачів і яка має такі атрибути:

- FirstName (ім'я)
- LastName (прізвище)
- Year (рік народження)
- Phone (номер телефону)

Які атрибути в даному випадку можуть складати суперключ:

- `{FirstName, LastName, Year, Phone}`
- `{FirstName, Year, Phone}`
- `{LastName, Year, Phone}`
- `{FirstName, Phone}`
- `{LastName, Phone}`
- `{Year, Phone}`
- `{Phone}`

Кожен студент унікально може ідентифікувати телефонний номер, тому будь-які набори, в яких зустрічається атрибут Phone, представляють суперключ.

А ось, наприклад, набір {FirstName, LastName, Year}не є суперключом, тому що у нас теоретично можуть бути як мінімум два студенти з однаковим ім'ям, прізвищем та роком народження.

Потенційний ключ

Candidate key (потенційний ключ) - є мінімальним суперключом відношення (таблиці), тобто набір атрибутів, який задовольняє ряду умов:

- **Непривідність** : він не може бути скорочений, він містить мінімально можливий набір атрибутів
- **Унікальність** : він повинен мати унікальні значення незалежно від зміни рядка
- **Наявність значення** : він повинен мати значення NULL, тобто він обов'язково повинен мати значення.

Візьмемо раніше виділені суперключі та знайдемо серед них candidate key. Перший п'ять суперключів не відповідають першій умові, тому що всі їх можна скоротити до суперключа.

- `{FirstName, LastName, Year, Phone}`
- `{FirstName, Year, Phone}`
- `{LastName, Year, Phone}`
- `{FirstName, Phone}`
- `{LastName, Phone}`
- `{Year, Phone}`

Суперключ {Phone} відповідає першій та другій умові, оскільки він має унікальне значення (у даному випадку всі користувачі можуть мати тільки унікальні телефонні номери). Але чи відповідає він третій умові? Загалом немає, тому що теоретично студент може і не мати телефону. У цьому випадку атрибут Phone буде мати значення NULL, тобто значення не буде.

Водночас це може залежати від ситуації. Якщо в якійсь системі номер телефону є невід'ємним атрибутом, наприклад, використовується для реєстрації та входу до системи, його можна вважати потенційним ключем. Але в цьому випадку ми розглядаємо загальну ситуацію. І розуміння потенційного ключа необхідно відштовхуватися від конкретної системи, яку визначає база даних.

І в такому випадку суперключі таблиці не містять потенційного ключа.

Первинний ключ

Первинний ключ (primary key) безпосередньо застосовується для ідентифікації рядків таблиці. Він повинен відповідати наступним обмеженням:

- Первинний ключ має бути унікальним весь час
- Він повинен бути присутнім у таблиці і мати значення
- Він повинен часто змінювати своє значення. В ідеалі він взагалі не повинен змінювати значення.

Як правило, первинний ключ представляє один стовпець таблиці, але може бути складеним і складатися з декількох стовпців.

Якщо таблиці можна виділити потенційний ключ, його можна використовувати як первинного ключа.

Якщо ж потенційні ключі відсутні, то для первинного ключа можна додати до сутності спеціальний атрибут, який, як правило, називається Id або має форму *[Ім'я_сутності]Id* (наприклад, StudentId), або може мати іншу назву. І зазвичай цей атрибут набуває цілого значення, починаючи з 1.

Якщо ж у нас є кілька потенційних ключів, то потенційні ключі, які не становлять первинний ключ, є **альтернативними ключами** (alternative key).

Наприклад, візьмемо подання користувачів на сайтах з двофакторною авторизацією, де нам обов'язково мати електронну адресу, яка нерідко виступає як логін, і якийсь номер телефону. У цьому випадку таблицю користувачів ми можемо задати за допомогою таких атрибутів:

- Name (ім'я користувача)
- Email (електронна адреса)
- Password (пароль)
- Phone (телефонний номер)

В даному випадку атрибути Email і Phone є потенційними ключами, вони обов'язкові в рамках системи, що розглядається, і в принципі унікальні. І теоретично, ми можемо використовувати один із цих атрибутів як первинний ключ, тоді другий буде альтернативним ключем. Однак знову ж таки оскільки

теоретично значення обох атрибутів можуть змінюватися, то краще все ж таки визначити додатковий атрибут спеціально під первинний ключ.

Зовнішні ключі та зв'язки

Бази даних можуть містити таблиці, пов'язані між собою різними зв'язками. Зв'язок (relationship) є асоціацією між сутностями різних типів.

При виділенні зв'язку виділяють головну або батьківську таблицю (primary key table/master table) та залежну, дочірню таблицю (foreign key table/child table). Дочірня таблиця залежить від батьківської.

Для організації зв'язку застосовуються зовнішні ключі. Зовнішній ключ являє собою один або кілька стовпців з однієї таблиці, який одночасно є потенційним ключем з іншої таблиці. Зовнішній ключ необов'язково має відповідати первинному ключу з головної таблиці. Хоча, як правило, зовнішній ключ із залежної таблиці вказує на первинний ключ із головної таблиці.

Зв'язки між таблицями бувають таких типів:

- **Один до одного** (One to one)
- **Один до багатьох** (One to many)
- **Багато до багатьох** (Many to many)

Зв'язок один до одного

Цей тип зв'язків зустрічає не часто. І тут об'єкту однієї сутності можна зіставити лише одне об'єкт інший сутності. Наприклад, на деяких сайтах користувач може мати лише один блог. Тобто виникає відношення один користувач – один блог.

Нерідко цей тип зв'язків передбачає розбиття однієї великої таблиці кілька маленьких. Основна батьківська таблиця в цьому випадку продовжує містити дані, що часто використовуються, а дочірня залежна таблиця зазвичай зберігає дані, які використовуються рідше.

Щодо цього первинний ключ залежної таблиці в той же час є зовнішнім ключем, який посилається на первинний ключ із головної таблиці.

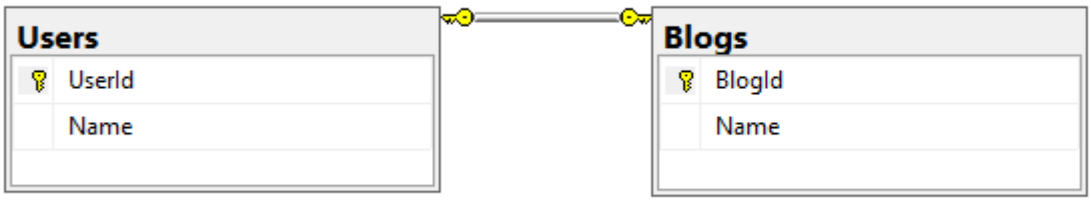
Наприклад, таблиця «Користувачі» представляє користувачів і має такі стовпці:

- UserId (ідентифікатор, первинний ключ)
- Name (ім'я користувача)

І таблиця Blogs представляє блоги користувачів і має такі стовпці:

- BlogId (ідентифікатор, первинний та зовнішній ключ)
- Name (назва блогу)

У цьому випадку стовпець BlogId зберігатиме значення зі стовпця UserId з таблиці користувачів. Тобто стовпець BlogId виступатиме одночасно первинним та зовнішнім ключем.



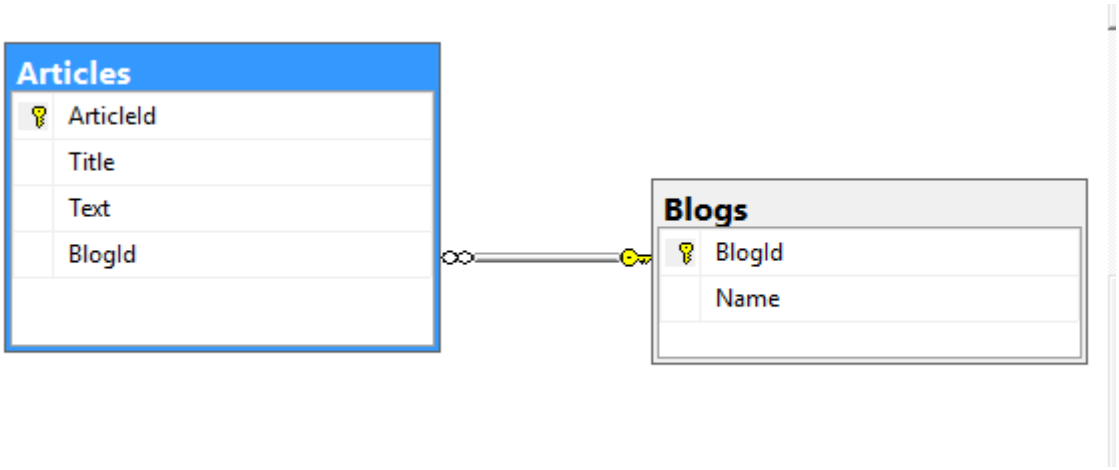
Зв'язок один до багатьох

Це найпоширеніший тип зв'язків. У цьому типі зв'язків кілька рядків із дочірньої таблиці залежать від одного рядка батьківської таблиці. Наприклад, в одному блозі може бути кілька статей. У цьому випадку таблиця блогів є батьківською, а таблиця статей – дочірньою. Тобто один блог – багато статей. Або інший приклад, у футбольній команді може грати кілька футболістів. І водночас один футболіст одночасно може грати лише в одній команді. Тобто одна команда – багато футболістів.

Наприклад, нехай буде таблиця Articles, яка представляє статті блогу і яка має такі стовпці:

- ArticleId (ідентифікатор, первинний ключ)
- BlogId (зовнішній ключ)
- Title (назва статті)
- Text (текст статті)

У цьому випадку стовпець BlogId з таблиці статей зберігатиме значення зі стовпця BlogId з таблиці блогів.



Зв'язок багато до багатьох

При цьому типі зв'язків один рядок з таблиці A може бути пов'язаний з безліччю рядків з таблиці B. У свою чергу один рядок з таблиці A може бути пов'язаний з безліччю рядків з таблиці A. Типовий приклад - студенти та курси: один студент може відвідувати кілька курсів і відповідно на один курс можуть записатися кілька студентів.

Інший приклад - статті та теги: для однієї статті можна визначити кілька тегів, а один тег може бути визначений для кількох статей.

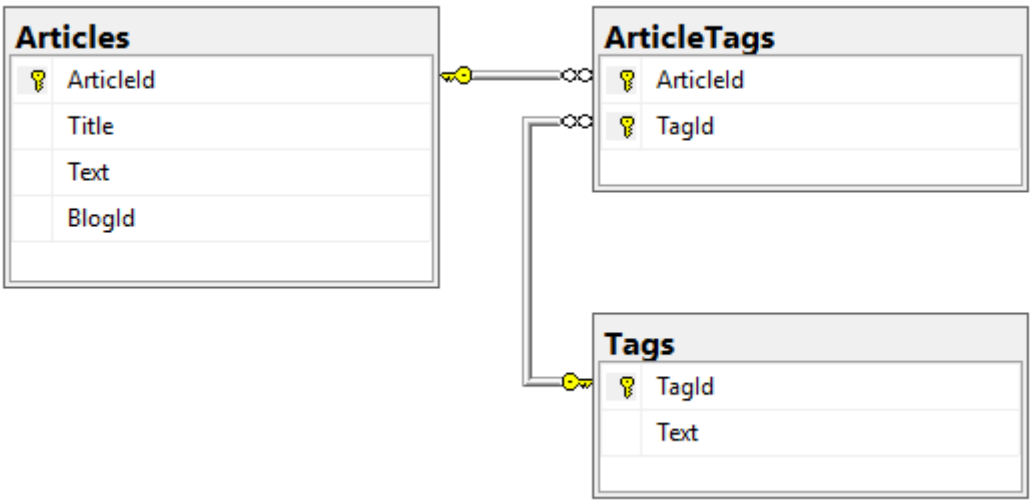
Але в SQL Server на рівні бази даних ми не можемо встановити прямий зв'язок до багатьох між двома таблицями. Це робиться у вигляді допоміжної проміжної таблиці. Іноді дані цієї проміжної таблиці представляють окрему сутність.

Наприклад, у випадку зі статтями та тегами нехай буде таблиця Tags, яка має два стовпці:

- TagId (ідентифікатор, первинний ключ)
- Text (текст тега)

Також нехай буде проміжна таблиця ArticleTags з наступними полями:

- TagId (ідентифікатор, первинний та зовнішній ключ)
- ArticleIdId (ідентифікатор, первинний та зовнішній ключ)



Технічно ми отримаємо два зв'язки одним-багатьом. Стовпець TagId із таблиці ArticleTags буде посилатися на стовпець TagId із таблиці Tags. А стовпець ArticleId із таблиці ArticleTags буде посилатися на стовпець ArticleId із таблиці Articles. Тобто стовпці TagId і ArticleId у таблиці ArticleTags являють собою складовий первинний ключ і одночасно є зовнішніми ключами для зв'язку з таблицями Articles і Tags.

Посилальна цілісність даних

При зміні первинних і зовнішніх ключів слід дотримуватися такого аспекту, як **посилальна цілісність даних** (referential integrity). Її основна ідея у тому, щоб дві таблиці у базі даних, які зберігають одні й самі дані, підтримували їх узгодженість. Цілісність даних представляє правильно збудовані відносини між таблицями з коректною установкою посилань між ними. У яких випадках цілісність даних може порушуватися:

- **Аномалія видалення** (deletion anomaly). Виникає у разі видалення рядка з головної таблиці. У цьому випадку зовнішній ключ із залежної таблиці продовжує посилатися на віддалений рядок із головної таблиці
- **Аномалія вставки** (insertion anomaly). Виникає при вставці рядка у залежну таблицю. У цьому випадку зовнішній ключ із залежної таблиці не відповідає первинному ключу жодного з рядків із головної таблиці.
- **Аномалії оновлення** (update anomaly). При подібній аномалії кілька рядків однієї таблиці можуть містити дані, які належать одному й тому

об'єкту. При зміні даних в одному рядку вони можуть суперечити даними з іншого рядка.

Аномалія видалення

Для вирішення аномалії видалення зовнішнього ключа слід встановлювати одне з двох обмежень:

- Якщо рядок із залежної таблиці обов'язково вимагає наявності рядка із головної таблиці, то для зовнішнього ключа встановлюється каскадне видалення. Тобто при видаленні рядка із головної таблиці відбувається видалення зв'язаного рядка (рядків) із залежної таблиці.
- Якщо рядок із залежної таблиці допускає відсутність зв'язку з рядком із головної таблиці (тобто такий зв'язок необов'язковий), то для зовнішнього ключа при видаленні зв'язаного рядка із головної таблиці задається встановлення значення NULL. У цьому стовпець зовнішнього ключа повинен допускати значення NULL.

Аномалія вставки

Для вирішення аномалії вставки при додаванні до залежної таблиці даних стовпець, який представляє зовнішній ключ, повинен допускати значення NULL. І таким чином, якщо об'єкт, що додається, не має зв'язку з головною таблицею, то в стовпці зовнішнього ключа стоятиме значення NULL.

Аномалії оновлення

Для вирішення проблеми аномалії поновлення застосовується нормалізація, яка буде розглянута далі.

Нормалізація

Нормалізація представляє процес поділу даних за окремими таблицями. Нормалізація усуває надмірність даних (data redundancy) і цим уникнути порушення цілісності даних за її зміні, тобто уникнути аномалій зміни (update anomaly).

Як правило, нормалізація переважно застосовується при висхідному підході проектуванні бази даних, тобто коли ми всі атрибути, які треба зберегти в бд,

групуємо по суті, для яких потім створюються таблиці. Однак при низхідному підході, коли спочатку виявляються сутності, а потім їх атрибути та зв'язки між ними, нормалізація також може застосовуватися, наприклад, для перевірки коректності спроектованих таблиць.

У ненормалізованій формі таблиця може зберігати інформацію про дві і більше сутності. Також вона може містити стовпці, що повторюються. Також стовпці можуть зберігати значення, що повторюються. У нормалізованій формі кожна таблиця зберігає інформацію лише про одну сутність.

Нормалізація передбачає застосування нормальних форм структури даних. Існує 7 нормальних форм. Кожна нормальна форма (за винятком першої) має на увазі, що до даних вже було застосовано попередню нормальну форму. Наприклад, перш ніж застосувати третю нормальну форму даних повинна бути застосована друга нормальна форма. І, строго кажучи, база даних вважається нормалізованою, якщо до неї застосовується третя нормальна форма і вище.

Перша нормальна форма (1NF) передбачає, що дані, що зберігаються на перетині рядків і стовпців повинні представляти скалярне значення, а таблиці не повинні містити рядків, що повторюються.

Друга нормальна форма (2NF) передбачає, кожен стовпець, який є ключем, повинен залежати від первинного ключа.

Третя нормальна форма (3NF) передбачає, кожен стовпець, який є ключем, повинен залежати тільки від первинного ключа.

Нормальна форма Бойса-Кодда (BCNF) є трохи суворішою версією третьої нормальної форми.

Четверта нормальна форма (4NF) застосовується для усунення багатозначних залежностей (multivalued dependencies) - таких залежностей, де стовпець з первинним ключем має зв'язок один-багатьом зі стовпцем, який не є ключем. Ця нормальна форма усуває некоректні відносини багато-багатьом.

П'ята нормальна форма (5NF) поділяє таблиці більш малі таблиці усунення надмірності даних. Розбиття йде доти, доки не можна буде відтворити оригінальну таблицю шляхом об'єднання малих таблиць.

Шоста нормальна форма (domain key normal form/6NF). Кожне обмеження зв'язків між таблицями повинно залежати тільки від обмежень ключа та обмежень домену, де домен представляє набір допустимих значень для стовпця. Ця форма запобігає додаванню неприпустимих даних шляхом встановлення обмеження лише на рівні відносин між таблицями, але з рівні таблиць чи стовпців. Ця форма, зазвичай, не застосовна лише на рівні СУБД, зокрема й у SQL Server.

Функціональна залежність

Ключовим поняттям нормалізації є **функціональна залежність** . Функціональна залежність визначає зв'язок між атрибутами відносини. Наприклад, якщо атрибут B функціонально залежить від атрибуту A (A → B), кожне значення атрибуту A пов'язане лише з одним значенням атрибуту B. Причому атрибути A і можуть складатися з одного або декількох атрибутів. Тобто, якщо два рядки мають одне й те саме значення атрибуту A, то вони обов'язково мають одне й те саме значення атрибуту B. При цьому для одного значення атрибуту можуть існувати кілька різних значень атрибуту A. Атрибут A в цій залежності ще називається детермінантом .

Наприклад, візьмемо наступну таблицю, яка представляє університетські курси:

Course	Teacher	Position
Математика	Сміт	Професор
Алгорити	Адамс	Помічник
JavaScript	Адамс	Помічник

Тут атрибут Teacher функціонально **залежить** від атрибута Course (Course → Teacher). Тобто знаючи назву курсу, ми можемо визначити його викладача. І в цьому випадку можна говорити, що між атрибутами Course та Teacher є зв'язок

1:1, а між Teacher та Course зв'язок 1:N, оскільки є кілька курсів, які може вести один викладач. При цьому атрибут Course **не залежить** від атрибута Teacher.

Крім того, тут можна простежити ще дві функціональні залежності. Зокрема, атрибут Position залежить від атрибута Teacher (Teacher → Position). Знаючи ім'я викладача, ми можемо визначити його посаду.

А також атрибут Position функціонально залежить від атрибуту Course - знаючи назву курсу, ми можемо сказати посаду викладача.

У таблиці у нормалізованій базі даних єдиним детермінантом має бути атрибут, який є первинним ключем. А решта атрибутів повинна функціонально залежати від первинного ключа.

Наприклад, в даному випадку ми можемо взяти як первинний ключ назву курсу з урахуванням, що курси можуть мати тільки унікальні назви. Однак посада викладача в даному випадку залежатиме одразу від двох атрибутів – від Course та Teacher. І подібні залежності можуть свідчити, що база даних і саме таблиця курсів має недоліки у проектуванні і може бути джерелом аномалій оновлення.

Перша нормальна форма

Перша нормальна форма передбачає, що таблиця не повинна містити стовпців, що повторюються, або таких стовпців, які містять набори значень. Ненормалізована таблиця в цьому випадку може містити одну або кілька груп даних, що повторюються. Група, що повторюється, - це група з одного або декількох атрибутів таблиці, в якій можлива наявність декількох значень для ключового атрибута таблиці.

Підсумком застосування першої форми має стати наявність для одного атрибуту сутності лише одного стовпця в таблиці, який повинен містити скалярне значення.

Існують два походи до переходу до ненормалізованої таблиці до першої нормальної форми. Перший спосіб називається вирівнюванням або flattaning. Він передбачає декомпозицію рядка з групами даних, що

повторюються, при якому для кожної групи, що повторюється, створюється свій рядок. Отримана в результаті таблиця міститиме атомарні значення кожного з атрибутів. Хоча водночас цей підхід збільшить надмірність даних.

Другий підхід передбачає, що один атрибут або група атрибутів призначаються ключем ненормалізованої таблиці, а потім групи, що повторюються, видаляються з таблиці і поміщаються в окрему таблицю разом з копіями ключа з вихідної таблиці.

Розглянемо застосування нормалізації з прикладу. Нехай у нас є система, яка описується такою інформацією:

```
1   Том відвідує курс математики, який викладає Сміт. Дата запису
    11/06/2017.
2   Сем відвідує курс з алгоритмів, які викладає Адамс. Дата запису
    12/06/2017.
3   Боб відвідує курс математики, який викладає Сміт. Дата запису
    13/06/2017.
4   Том відвідує курс JavaScript, який викладає Адамс. Дата запису
    14/06/2017.
5   Сем має дві електронні адреси: sam@gmail.com та sam@hotmail.com.
6   В університеті може бути лише один курс із певним ім'ям. Один викладач
    може викладати кілька курсів.
```

Спочатку визначимо ненормалізовану таблицю StudentCourses, яка містить усю цю інформацію:

StudentId	Name	Emails	Course1	Date1	TeacherId1	Teacher1	Course2	Date2	TeacherId2	Teacher2
1	Tom		Mathematics	11/06/2017	1	Smith	JavaScript	14/06/2017	2	Adams
2	Sam	sam@gmail.com sam@hotmail.com	Algorithms	12/06/2017	2	Adams				
3	Bob		Mathematics	13/06/2017	1	Smith				

Для кожного студента визначено унікальний ідентифікатор StudentId, а також атрибут Name (ім'я), Emails (всі електронні адреси), Course1/Course2(курс), Date1/Date2 (дата вступу), Teacher1/Teacher2 (викладач). Також щоб розрізняти викладачів (оскільки теоретично можуть бути викладачі з одним і тим же прізвищем), доданий атрибут TeacherId1/TeacherId2. Для курсів такий ідентифікатор не потрібний, тому що в нашому випадку назва курсу є унікальною.

Оскільки Том записаний одразу на два курси, то кілька атрибутів довелося дублювати. Але що буде, коли Том у прагненні здобути нікому не потрібні сертифікати запишеться ще на десяток курсів?

Ця таблиця є чудовим прикладом відхилення від першої нормальної форми. В першу чергу ми бачимо групу атрибутів, що повторюються, які представляють дані по одному курсу: Course, Date, TeacherId, Teacher. Ці атрибути представляють групу, що повторюється, яку можна умовно назвати StudentCourse.

```
StudentCourse = (Course, Date, TeacherId, Teacher)
```

Друга проблема – атрибут Emails містить набір електронних адрес. Фактично цей атрибут також утворює групу, що повторюється.

Для позбавлення від першої групи атрибутів, що повторюється, застосуємо перший підхід: створимо для кожної групи, що повторюється, окремий рядок.

StudentId	Name	Emails	CourseId	Course	Date	TeacherId	Teacher
1	Том		1	Математика	11/06/2017	1	Сміт
1	Том		2	JavaScript	14/06/2017	2	Адамс
2	Сем	sam@gmail.com sam@hotmail.com	3	Алгоритми	12/06/2017	2	Адамс
3	Боб		1	Математика	13/06/2017	1	Сміт

У разі збільшилася надмірність даних, проте ми позбулися повторюваної групи. Також слід зазначити, що тепер атрибут StudentId не може використовуватись як первинний ключ. І в даному випадку проглядається тільки один потенційний ключ, який і використовуватиметься як первинний - це відразу два стовпці StudentId і Course. Але назва курсу – не найкращий ключ, якщо враховувати, що ця назва може редагуватися та змінюватися. Тому для кожного курсу доданий ще один атрибут – CourseId – унікальний номер курсу, який разом із StudentId складає первинний ключ. Хоча в принципі могло б залишити в якості частини первинного ключа ім'я курсу з урахуванням, що воно унікальне.

Для позбавлення від другої групи, що повторюється - атрибуту Emails застосуємо другий підхід: винесення цієї групи з копією ключа в окрему таблицю. Для цього визначимо таблицю Emails:

Email	StudentId
sam@gmail.com	2
sam@hotmail.com	2

Оскільки електронна адреса в принципі унікальна, то її можна зробити первинним ключем.

Таким чином, таблиці Emails з таблицею StudentCourses буде пов'язана один до багатьох зв'язків (один студент - багато електронних адрес). І в цьому випадку таблиця StudentCourses скоротиться так:

StudentId	Name	CourseId	Course	Date	TeacherId	Teacher
1	Том	1	Математика	11/06/2017	1	Сміт
1	Том	2	JavaScript	14/06/2017	2	Адамс
2	Сем	3	Алгоритми	12/06/2017	2	Адамс
3	Боб	1	Математика	13/06/2017	1	Сміт

Тепер у нас немає стовпців, що повторюються, але збільшилася надмірність даних, так як для студента Том визначено вже два рядки в таблиці, і відповідно Id повторюється. Проте 1-а нормальна форма застосована.

У принципі можна відзначити, що якщо групи, що повторюються, містять унікальні значення для кожного рядка таблиці (як у випадку з електронними адресами), то ми маємо справу з потенційним зв'язком один до багатьох. Якщо ж групи, що повторюються, містять неунікальні значення, які можуть мати різні рядки таблиці (як у випадку з атрибутами курсів), то це ховається потенційний зв'язок багато хто до багатьох.

Друга нормальна форма

У другій нормальній формі кожен стовпець у таблиці, який є ключем, **повинен залежати від ключа** .

Ключовий момент другої нормальної форми – повна функціональна залежність. Вона передбачає, що атрибут B повністю функціонально залежить

від атрибуту A, якщо атрибут B функціонально залежить від повного значення атрибуту A, а не від будь-якого підмножини значень з атрибуту A. Тобто якщо атрибут A складають кілька значень, скажімо, A1 і A2, то атрибут повністю функціонально залежить від A, якщо він залежить і від A1 і від A2 (A1, A2 → B).

Якщо атрибут залежить тільки від будь-якого підмножини з атрибуту A, наприклад, тільки від A1, то має місце часткова функціональна залежність.

Ця форма застосовується до таблиць, які мають складовий первинний ключ, тобто де первинний ключ складається з декількох атрибутів. Якщо таблиці **несоставной** первинний ключ, то цьому випадку вважається, що й інші атрибути автоматично перебувають у повної функціональної залежності від первинного ключа.

Друга нормальна форма застосовується лише до таблицям, які у першій нормальній формі. Після застосування другої форми, всі стовпці таблиці залежать від первинного ключа.

Візьмемо сформовану в минулій темі таблицю StudentCourses після застосування першої нормальної форми:

StudentId	Name	CourseId	Course	Date	TeacherId	Teacher
1	Том	1	Математика	11/06/2017	1	Сміт
1	Том	2	JavaScript	14/06/2017	2	Адамс
2	Сем	3	Алгоритми	12/06/2017	2	Адамс
3	Боб	1	Математика	13/06/2017	1	Сміт

На даний момент ця таблиця має складовий первинний ключ StudentId+CourseId. Які функціональні залежності від ключових атрибутів тут можна виділити:

```
StudentId, CourseId → Date
StudentId → Name
CourseId → Course, TeacherId, Teacher
```

Від обох частин складеного ключа StudentId+CourseId залежить лише арібут Date - дата, у яку студент із ідентифікатором StudentId вступив на курс із ідентифкатором CourseId.

Атрибут Name залежить тільки від частини складового ключа - від атрибуту StudentId, оскільки знаючи ідентифікатор студента, можна сказати, яке ім'я. У разі має факт часткової залежності.

Атрибути Course, TeacherId, Teacher, Position залежить від іншої частини ключа – від атрибуту CourseId. Знаючи значення CourseId, можна сказати, як називається курс, який у курсу викладач, яку посаду він обіймає. Знову ж таки тут часткова залежність.

Наявність часткових залежностей свідчить, що таблиця немає у другій нормальній формі. І для переходу до цієї форми необхідно перемістити атрибути, які не входять до первинного ключа, у нову таблицю разом з копією частини первинного ключа, від якої вони функціонально залежать.

У нашому випадку із однієї таблиці вийдуть три. Таблиця Students:

StudentId	Name
1	Том
2	Сем
3	Боб

Таблиця Courses:

CourseId	Course	TeacherId	Teacher
1	Математика	1	Сміт
2	JavaScript	2	Адамс
3	Алгоритми	2	Адамс

І таблиця StudentCourses:

StudentId	CourseId	Date
1	1	11/06/2017
1	2	14/06/2017
2	3	12/06/2017
3	1	13/06/2017

Підсумком стало утворення зв'язку багато до багатьох (багато студентів - багато курсів) між таблицями Students і Courses через таблицю StudentCourses. Таким чином, база даних перейшла до другої нормальної форми.

Третя нормальна форма

Третя нормальна форма передбачає, що кожен стовпець, що не є ключем, повинен залежати **тільки** від стовпця, який є ключем, тобто має бути відсутня **транзитивна функціональна залежність** (transitive functional dependency)

Транзитивна функціональна залежність виражається так: A → B і B → C. Тобто атрибут C транзитивно залежить від атрибуту A, якщо атрибут залежить від атрибуту B, а атрибут залежить від атрибуту A (за умови, що атрибут A функціонально не залежить ні від атрибуту, ні від атрибуту C).

Якщо стовпець залежить не тільки від первинного ключа, то цей стовпець знаходиться не в тій таблиці, в якій він повинен знаходитися, або є похідним від інших стовпців.

Для нормалізації з вихідної таблиці атрибути, які у транзитивної залежності від ключа, виносяться в окрему таблицю з копією того атрибута, від якого вони безпосередньо залежать.

При застосуванні третьої нормальної форми таблиця повинна бути у другій нормальній формі. 3NF дозволяє значно знизити надмірність даних.

Для прикладу візьмемо сформовану в минулій темі таблицю Courses, яка містить інформацію про курси і яка знаходиться у другій нормальній формі:

CourseId	Course	TeacherId	Teacher
1	Математика	1	Сміт
2	JavaScript	2	Адамс
3	Алгоритми	2	Адамс

Які функціональні залежності тут можна виділити:

```
CourseId → Course, TeacherId, Teacher
Course → CourseId, TeacherId, Teacher
TeacherId → Teacher
```

Друга залежність фактично аналогічна першій і свідчить, що атрибут Course є потенційним ключем.

Третя залежність говорить про те, що, знаючи ідентифікатор викладача, ми можемо дізнатися про його прізвище та посаду. Тобто через атрибут TeacherId атрибут Teacher залежить від CourseId (CourseId → TeacherId та TeacherId →

Teacher). І в даному випадку ми можемо говорити про транзитивну залежність Teacher, Position від CourseId.

Для нормалізації необхідно винести окрему таблицю атрибути TeacherId і Teacher. Для цього нехай буде окрема таблиця Teachers:

TeacherId	Teacher
1	Сміт
2	Адамс

А таблиця Courses скоротиться так:

CourseId	Course	TeacherId
1	Математика	1
2	JavaScript	2
3	Алгоритми	2