

1

Клієнт-серверна архітектура СУБД DBMS client-server architecture Клиент-серверная архитектура СУБД

Тема 1
Topic 1

2

GENERIC SYSTEM ARCHITECTURES

3

Поняття клієнт-серверної архітектури та переваги цього типу архітектури для СУБД

The meaning of the client-server architecture and the advantages of this type of architecture for a DBMS

Понятие клиент-серверной архитектуры и преимущества этого типа архитектуры для СУБД

Різниця між двохрівневою, трирівневою та n-рівневою архітектурою клієнт-сервер

The difference between two-tier, three-tier and n-tier client-server architectures

Разница между двухуровневой, трехуровневой и многоуровневой клиент-серверной архитектурой

4

Хмарні обчислення та дані як послуга (DaaS) та база даних як послуга (DBaaS)

Cloud computing and data as a service (DaaS) and database as a service (DBaaS)

Облачные вычисления и данные как услуга (DaaS) и база данных как услуга (DBaaS)

Програмні компоненти СУБД

Software components of a DBMS

Программные компоненты СУБД

Призначення веб-сервісів та технологічні стандарти, що використовуються для розробки веб-сервісів

The purpose of a Web service and the technological standards used to develop a Web service

Назначение веб-сервисов и технологические стандарты, используемые для разработки веб-сервиса

5

Поняття сервіс-орієнтованої архітектури (SOA)
The meaning of service-oriented architecture (SOA)
Понятие сервис-ориентированной архитектуры (SOA)

Різниця між розподіленими СУБД та розподіленою обробкою

The difference between distributed DBMSs, and distributed processing

Разница между распределенными СУБД и распределенной обработкой

Архітектура сховища даних
The architecture of a data warehouse
Архитектура хранилища данных

6

Хмарні обчислення та хмарні бази даних
About cloud computing and cloud databases
Облачные вычисления и облачные базы данных

Програмні компоненти СУБД
The software components of a DBMS
Программные компоненты СУБД

The common architectures that are used to implement multi-user database management systems:

- File-server
- Client-server

Загальні архітектури, які використовуються для реалізації багатокористувачьких систем управління базами даних:

- Файловий сервер
- Клієнт-сервер

Общие архитектуры, которые используются для реализации многопользовательских систем управления базами данных:

- Файловый сервер
- Клиент-сервер

File-Server

File-server is connected to several workstations across a network. Database resides on file-server. DBMS and applications run on each workstation.

Файловий сервер підключений до кількох робочих станцій у мережі. База даних знаходиться на файловому сервері. СУБД і програми працюють на кожній робочій станції.

Файловый сервер подключен к нескольким рабочим станциям в сети. База данных находится на файловом сервере. СУБД и приложения работают на каждой рабочей станции.

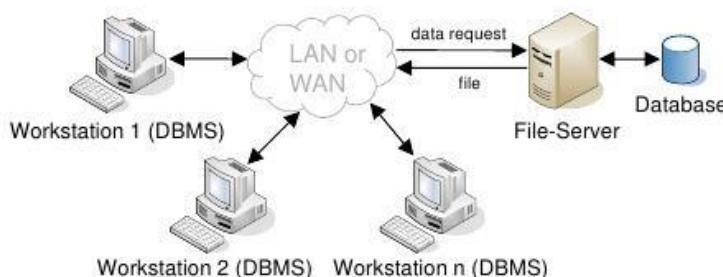
Significant network traffic.
 Copy of DBMS on each workstation.
 Concurrency, recovery and integrity control more complex.

Значний мережевий трафік.
 Копія СУБД на кожній робочій станції.
 Паралелізм, відновлення та контроль цілісності більш складні.

Значительный сетевой трафик.
 Копия СУБД на каждой рабочей станции.
 Более сложный контроль параллелизма, восстановления и целостности.

In a file-server environment, the processing is distributed about the network, typically a local area network (LAN).
 У середовищі файлового сервера обробка розподіляється по мережі, як правило, це локальна мережа (LAN).

В среде файлового сервера обработка распределяется по сети, обычно по локальной сети (LAN).



Traditional Two-Tier Client-Server

Client (tier 1) manages user interface and runs applications.

Server (tier 2) holds database and DBMS. Advantages include:

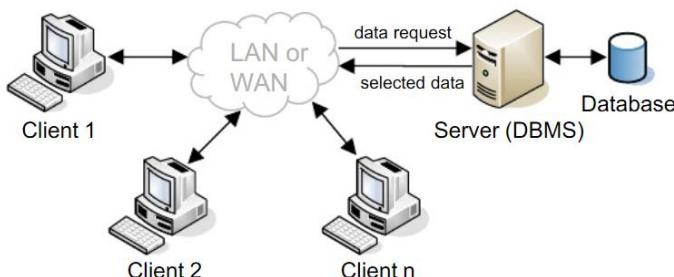
- wider access to existing databases;
- increased performance;
- possible reduction in hardware costs;
- reduction in communication costs;
- increased consistency.

Клієнт (рівень 1) управляє інтерфейсом користувача і запускає програми. Сервер (рівень 2) містить базу даних та СУБД. Переваги включають:

- ширший доступ до існуючих баз даних;
- підвищена продуктивність;
- можливе зменшення апаратних витрат;
- зменшення витрат на зв'язок;
- підвищена консистенція.

Клиент (уровень 1) управляет пользовательским интерфейсом и запускает приложения. Сервер (уровень 2) содержит базу данных и СУБД. Преимущества включают:

- более широкий доступ к существующим базам данных;
- повышенная производительность;
- возможное снижение затрат на оборудование;
- снижение затрат на связь;
- повышенная консистенция.



13

CLIENT	SERVER
Manages the user interface	Accepts and processes database requests from clients
Accepts and checks syntax of user input	Checks authorization
Processes application logic	Ensures integrity constraints not violated
Generates database requests and transmits to server	Performs query/update processing and transmits response to client
Passes response back to user	Maintains system catalog Provides concurrent database access Provides recovery control

14

Three-Tier Client-Server

The need for enterprise scalability challenged the traditional two-tier client-server model.

Потреба в масштабованості підприємств кинула виклик традиційній дворівневій моделі клієнт-сервер.

Потребность в масштабируемости предприятия поставила под сомнение традиционную двухуровневую модель клиент-сервер.

Client side presented two problems preventing true scalability:

- ‘Fat’ client, requiring considerable resources on client’s computer to run effectively.
- Significant client side administration overhead.

Клієнтська сторона представила дві проблеми, що перешкоджають справжній масштабованості:

- «Товстий» клієнт, який вимагає значних ресурсів на комп’ютері клієнта для ефективної роботи.
- Значні адміністративні витрати на стороні клієнта.

15

На стороне клиента возникли две проблемы, препятствующие настоящей масштабируемости:

- «Толстый» клиент, требующий значительных ресурсов на компьютере клиента для эффективной работы.
- Значительные накладные расходы на администрирование на стороне клиента.

Advantages:

- ‘Thin’ client, requiring less expensive hardware.
- Application maintenance centralized.
- Easier to modify or replace one tier without affecting others.
- Separating business logic from database functions makes it easier to implement load balancing.
- Maps quite naturally to Web environment.

16

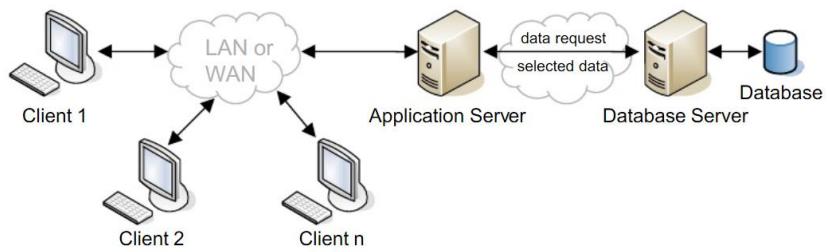
Переваги:

- «Тонкий» клієнт, що вимагає менш дорогого обладнання.
- Технічне обслуговування додатків централізоване.
- Простіше змінити або замінити один рівень, не впливаючи на інші.
- Відокремлення бізнес-логіки від функцій бази даних полегшує впровадження балансування навантаження.
- Цілком природно відображається у веб-середовищі.

Преимущества:

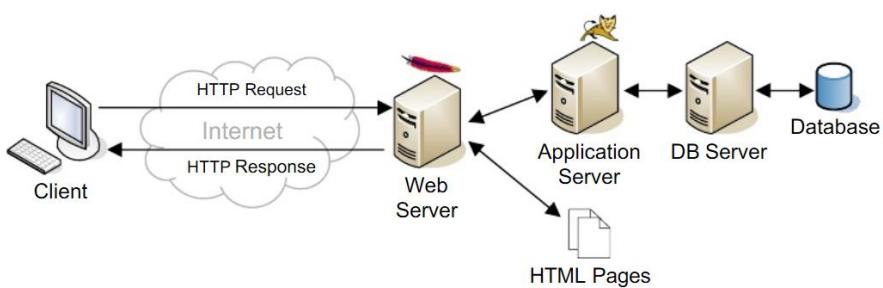
- «Тонкий» клиент, требующий менее дорого оборудования.
- Централизованное обслуживание приложений.
- Легче изменить или заменить один уровень, не затрагивая другие.
- Отделение бизнес-логики от функций базы данных упрощает реализацию балансировки нагрузки.
- Вполне естественно отображается в веб-среду.

17



18

n-Tier Client-Server



19

The three-tier architecture can be expanded to n tiers, with additional tiers providing more flexibility and scalability. Applications servers host API to expose business logic and business processes for use by other applications.

Тривневу архітектуру можна розширити до n рівнів, додаткові рівні забезпечують більшу гнучкість та масштабованість.

Сервери програм розміщують API для надання бізнес-логіки та бізнес-процесів для використання іншими програмами.

Трехуровневая архитектура может быть расширена до n уровней, при этом дополнительные уровни обеспечивают большую гибкость и масштабируемость.

Серверы приложений размещают API для предоставления бизнес-логики и бизнес-процессов для использования другими приложениями.

20

Middleware

Middleware is a generic term used to describe software that mediates with other software and allows for communication between disparate applications in a heterogeneous system. The need for middleware arises when distributed systems become too complex to manage efficiently without a common interface.

Проміжне програмне забезпечення – це загальний термін, що використовується для опису програмного забезпечення, яке здійснює посередницьку діяльність з іншим програмним забезпеченням і дозволяє здійснювати зв'язок між різними програмами в неоднорідній системі. Потреба в проміжних програмах виникає, коли розподілені системи стають занадто складними для ефективного управління без спільногого інтерфейсу.

21

Промежуточное ПО – это общий термин, используемый для описания программного обеспечения, которое является посредником с другим программным обеспечением и обеспечивает связь между разнородными приложениями в гетерогенной системе. Потребность в промежуточном программном обеспечении возникает, когда распределенные системы становятся слишком сложными для эффективного управления без общего интерфейса.

22

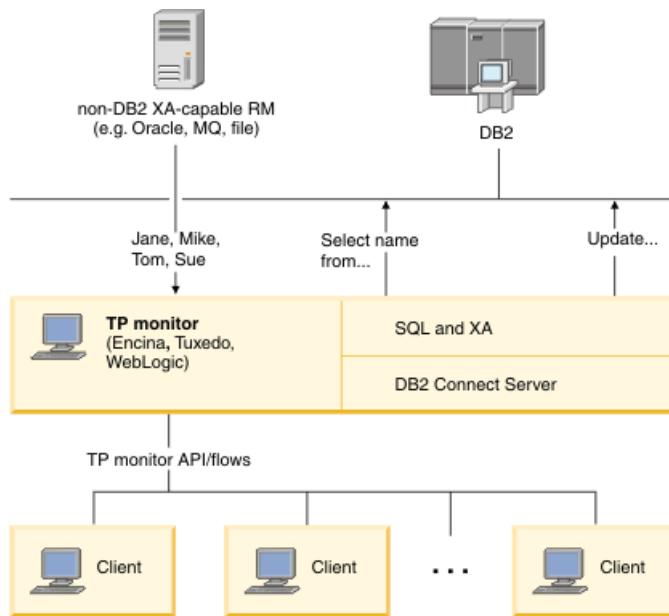
Transaction Processing Monitors

TP monitor is a program that controls data transfer between clients and servers in order to provide a consistent environment, particularly for online transaction processing (OLTP).

TP monitor – це програма, яка контролює передачу даних між клієнтами та серверами, щоб забезпечити стабільне середовище, особливо для обробки онлайн-транзакцій (OLTP).

TP monitor – это программа, которая контролирует передачу данных между клиентами и серверами, чтобы обеспечить согласованную среду, особенно для онлайн-обработки транзакций (OLTP).

23



24

Web Services and Service-Oriented Architectures

Web service is a software system designed to support interoperable machine-to-web service machine interaction over a network. Web services share business logic, data, and processes through a programmatic interface across a network. Developers can add the Web service to a Web page (or an executable program) to offer specific functionality to users.

Веб-служба – це програмна система, розроблена для підтримки взаємодії машин та веб-сервісів через мережу. Веб-служби обмінюються бізнес-логікою, даними та процесами через програмний інтерфейс у мережі. Розробники можуть додати веб-службу на веб-сторінку (або виконувану програму), щоб запропонувати користувачам певні функції.

25

Веб-служба – это программная система, предназначенная для поддержки взаимодействия машин и компьютеров веб-сервисов по сети. Веб-сервисы совместно используют бизнес-логику, данные и процессы через программный интерфейс в сети. Разработчики могут добавить веб-службу на веб-страницу (или исполняемую программу), чтобы предложить пользователям определенные функции.

Web services approach uses accepted technologies and standards, such as:

- XML (extensible Markup Language).
- SOAP (Simple Object Access Protocol) is a communication protocol for exchanging structured information over the Internet and uses a message format based on XML. It is both platform- and language-independent.
- WSDL (Web Services Description Language) protocol, again based on XML, is used to describe and locate a Web service.
- UDDI (Universal Discovery, Description, and Integration) protocol is a platform independent, XML-based registry for businesses to list themselves on the Internet.

26

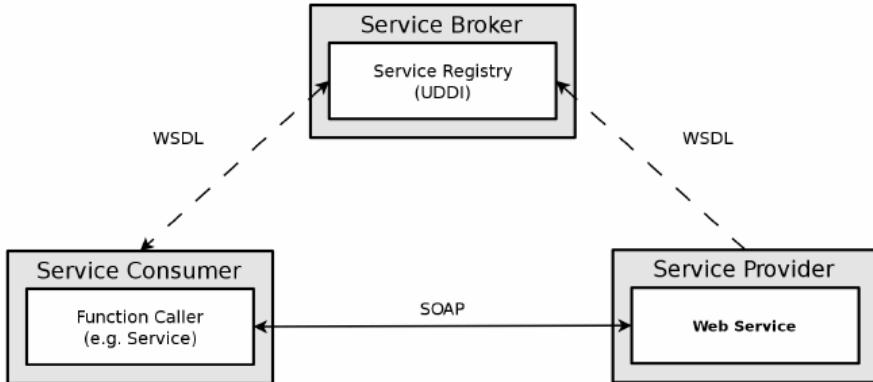
Підхід до веб-сервісів використовує прийняті технології та стандарти, такі як:

- XML (розширювана мова розмітки).
- SOAP (Simple Object Access Protocol) – це комунікаційний протокол для обміну структурованою інформацією через Інтернет, що використовує формат повідомлень на основі XML. Він не залежить як від платформи, так і від мови.
- Протокол WSDL (Мова опису веб-служб), також заснований на XML, використовується для опису та пошуку веб-служби.
- Протокол UDDI (Universal Discovery, Description and Integration) – це незалежний від платформи реєстр, заснований на XML, що дозволяє підприємствам розмістити свої сервіси в Інтернеті.

Подход веб-сервисов использует принятые технологии и стандарты, такие как:

- XML (расширяемый язык разметки).
- SOAP (простой протокол доступа к объектам) – это протокол связи для обмена структурированной информацией через Интернет, использующий формат сообщений на основе XML. Он не зависит от платформы и языка.
- Протокол WSDL (язык описания веб-служб), снова основанный на XML, используется для описания и определения местоположения веб-службы.
- Протокол UDDI (универсальное обнаружение, описание и интеграция) – это независимый от платформы реестр на основе XML, позволяющий предприятиям размещать свои сервисы в Интернете.

27



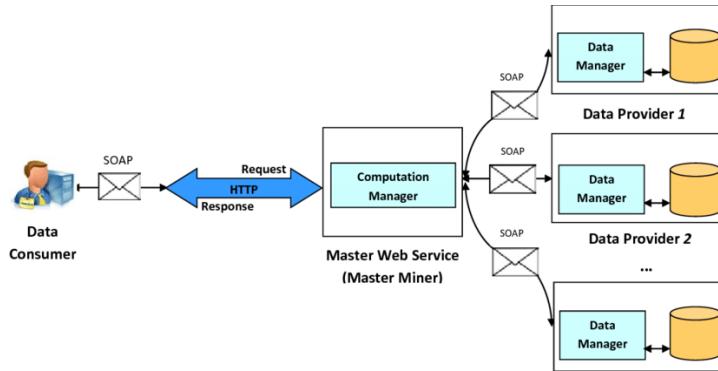
28

Service-Oriented Architectures (SOA)

A business-centric software architecture for building applications that implement business processes as sets of services published at a granularity relevant to the service consumer. Services can be invoked, published, and discovered, and are abstracted away from the implementation using a single standards-based form of interface.

Орієнтована на бізнес архітектура програмного забезпечення для побудови програм, що реалізують бізнес-процеси як набори послуг, опубліковані із деталізацією, що відповідає споживачеві послуги. Послуги можуть бути викликані, опубліковані, виявлені та абстраговані окремо від реалізації за допомогою єдиної стандартної форми інтерфейсу.

Архітектура програмного обслуговування, орієнтована на бізнес, для створення застосунків, реалізуючих бізнес-процеси в формі наборів послуг, опублікованих з деталізацією, відповідаючою споживачеві службі. Служби можна викликати, публікувати та виявляти, а також вони абстраговані від реалізації з використанням єдиного стандартної форми інтерфейса.



Distributed DBMSs

A distributed database is a logically interrelated collection of shared data (and a description of this data), physically distributed over a computer network. A distributed DBMS is the software system that permits the management of the distributed database and makes the distribution transparent to users.

Розподілена база даних – це логічно взаємопов’язаний набір спільних даних (та опис цих даних), що фізично розподіляється через комп’ютерну мережу. Розподілена СУБД – це програмна система, яка дозволяє керувати розподіленою базою даних і робить розподіл прозорим для користувачів.

31

Распределенная база данных – это логически взаимосвязанный набор общих данных (и описание этих данных), физически распределенный по компьютерной сети. Распределенная СУБД – это программная система, которая позволяет управлять распределенной базой данных и делает распространение прозрачным для пользователей.

A DDBMS consists of a single logical database split into a number of fragments.

DDBMS складається з єдиної логічної бази даних, розділеної на декілька фрагментів.

DDBMS состоит из единой логической базы данных, разделенной на несколько фрагментов.

32

Each fragment is stored on one or more computers (replicas) under the control of a separate DBMS, with the computers connected by a network.

Кожен фрагмент зберігається на одному або декількох комп'ютерах (репліках) під контролем окремої СУБД, причому комп'ютери з'єднані мережею.

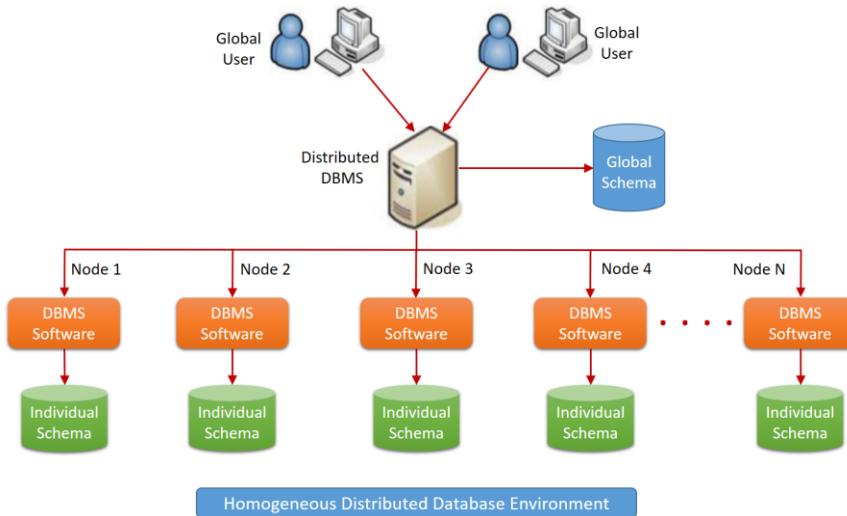
Каждый фрагмент хранится на одном или нескольких компьютерах (репликах) под управлением отдельной СУБД, причем компьютеры соединены сетью.

Each site is capable of independently processing user requests that require access to local data (that is, each site has some degree of local autonomy) and is also capable of processing data stored on other computers in the network.

Кожен вузол може самостійно обробляти запити користувачів, які потребують доступу до локальних даних (тобто кожен вузол має певний ступінь локальної автономії), а також здатний обробляти дані, що зберігаються на інших комп'ютерах мережі.

Каждый узел способен независимо обрабатывать пользовательские запросы, требующие доступа к локальным данным (то есть каждый узел имеет некоторую степень локальной автономии), а также может обрабатывать данные, хранящиеся на других компьютерах в сети.

33



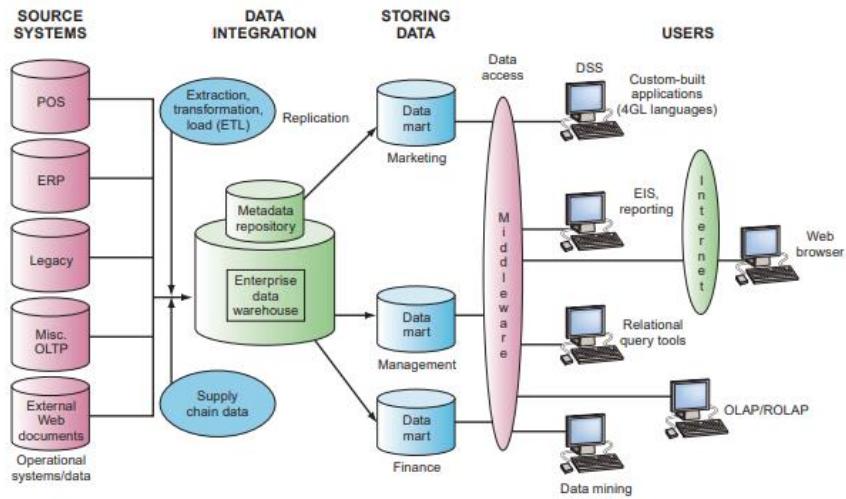
34

Data Warehousing

A data warehouse was deemed the solution to meet the requirements of a system capable of supporting decision making, receiving data from multiple operational data sources.

Сховище даних було визнано рішенням для задоволення вимог системи, здатної підтримувати прийняття рішень, отримуючи дані з декількох оперативних джерел даних.

Хранилище данных признано решением, отвечающим требованиям системы, способной поддерживать принятие решений, получая данные из нескольких источников операционных данных.



CLOUD COMPUTING

37

Cloud Computing

A model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g. networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction.

Модель для забезпечення повсюдного, зручного мережевого доступу на вимогу до спільногопулу конфігурованих обчислювальних ресурсів (наприклад, мереж, серверів, сховищ, додатків та послуг), які можна швидко надати та випустити за мінімальних зусиль управління або взаємодії з постачальником послуг.

38

Модель для обеспечения повсеместного, удобного сетевого доступа по требованию к общему пулу конфигурируемых вычислительных ресурсов (например, сетей, серверов, хранилищ, приложений и служб), которые могут быть быстро предоставлены и выпущены с минимальными усилиями по управлению или взаимодействию с поставщиком услуг.

Cloud Computing – Key Characteristics

On-demand self-service. Consumers can obtain, configure and deploy cloud services without help from provider.

Самообслуговування на замовлення. Споживачі можуть отримувати, налаштовувати та розгорнати хмарні служби без допомоги постачальника.

Самообслуживание по запросу. Потребители могут получать, настраивать и развертывать облачные сервисы без помощи поставщика.

Broad network access. Accessible from anywhere, from any standardized platform (e.g. desktop computers, laptops, mobile devices).

Широкий доступ до мережі. Доступний з будь-якого місця з будь-якої стандартизованої платформи (наприклад, настільних комп'ютерів, ноутбуків, мобільних пристрій).

Широкий доступ к сети. Доступен из любого места, с любой стандартизированной платформы (например, настольных компьютеров, ноутбуков, мобильных устройств).

Resource pooling. Provider's computing resources are pooled to serve multiple consumers, with different physical and virtual resources dynamically assigned and reassigned according to consumer demand. Examples of resources include storage, processing, memory, and network bandwidth.

Об'єднання ресурсів. Обчислювальні ресурси постачальника об'єднані для обслуговування кількох споживачів, при цьому різні фізичні та віртуальні ресурси динамічно призначаються та перепризначаються відповідно до споживчого попиту.
Приклади ресурсів включають зберігання, обробку, пам'ять та пропускну здатність мережі.

Объединение ресурсов. Вычислительные ресурсы провайдера объединяются для обслуживания нескольких потребителей, при этом различные физические и виртуальные ресурсы динамически назначаются и переназначаются в соответствии с потребительским спросом. Примеры ресурсов включают хранилище, обработку, память и пропускную способность сети.

41

Rapid elasticity. Provider's capacity caters for customer's spikes in demand and reduces risk of outages and service interruptions. Capacity can be automated to scale rapidly based on demand.

Швидка еластичність. Потужність постачальника забезпечує задоволення попиту споживачів та зменшує ризик перебоїв у роботі та перебоїв у обслуговуванні. Потужність може бути автоматизована для швидкого масштабування залежно від попиту.

Быстрая эластичность. Возможности провайдера удовлетворяют резкий рост спроса со стороны клиентов и снижают риск простоев и перерывов в обслуживании. Емкость можно автоматизировать для быстрого масштабирования в зависимости от спроса.

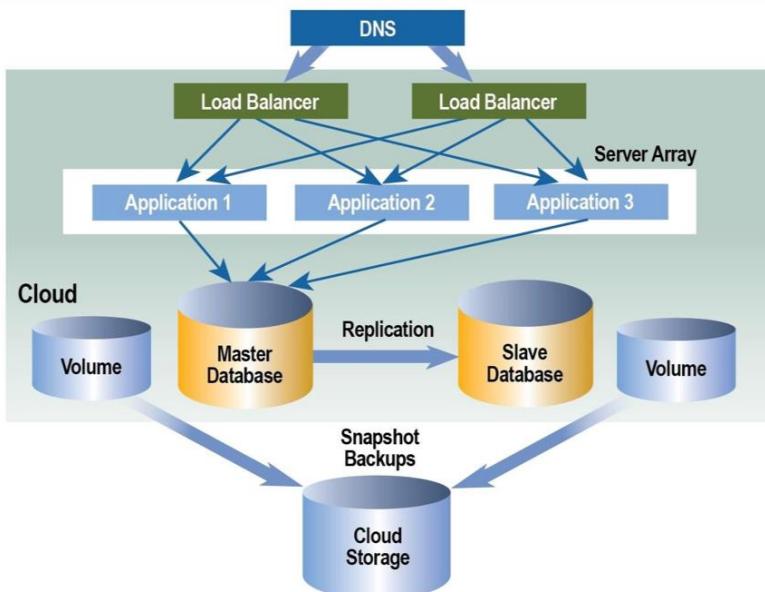
42

Measured service. Provider uses a metering capability to measure usage of service (e.g. storage, processing, bandwidth, and active user accounts).

Вимірювання обслуговування. Постачальник використовує цю можливість для вимірювання використання послуги (наприклад, зберігання, обробка, пропускна здатність та активні облікові записи користувачів).

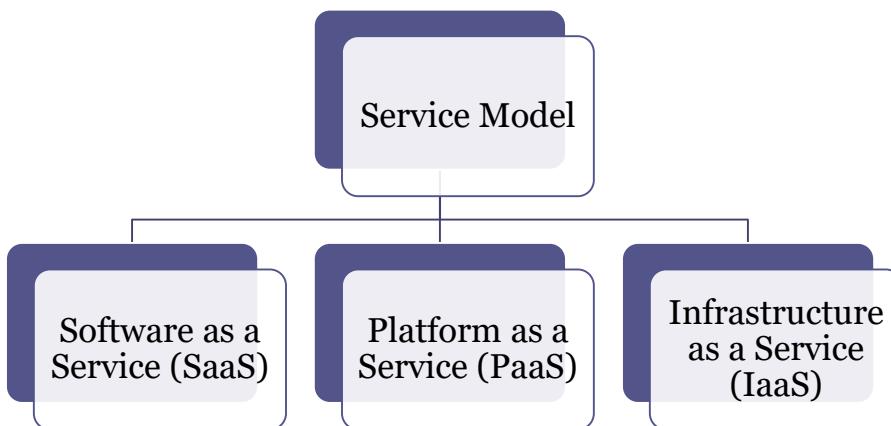
Измеряемый сервис. Провайдер использует эту возможность для измерения использования услуги (например, хранилище, обработка, пропускная способность и активные учетные записи пользователей).

43



44

Cloud Computing – Service Models



45

Software as a Service (SaaS)

Software and data hosted on cloud. Accessed through using thin client interface (e.g. web browser). Consumer may be offered limited user specific application configuration settings.

Програмне забезпечення та дані, розміщені на хмарі.

Доступ через тонкий клієнтський інтерфейс (наприклад, веб-браузер). Споживачеві можуть бути запропоновані обмежені параметри конфігурації додатка для конкретного користувача.

Програмное обеспечение и данные, размещенные в облаке. Доступ через интерфейс тонкого клиента (например, веб-браузер). Потребителю могут быть предложены ограниченные пользовательские настройки конфигурации приложения.

46

Platform as a Service (PaaS)

Allows creation of web applications without buying/maintaining the software and underlying infrastructure. Provider manages the infrastructure including network, servers, OS and storage, while customer controls deployment of applications and possibly configuration.

Дозволяє створювати веб-додатки без придбання / обслуговування програмного забезпечення та базової інфраструктури. Постачальник управляє інфраструктурою, включаючи мережу, сервери, ОС та сховище, а замовник контролює розгортання програм та, можливо, конфігурацію.

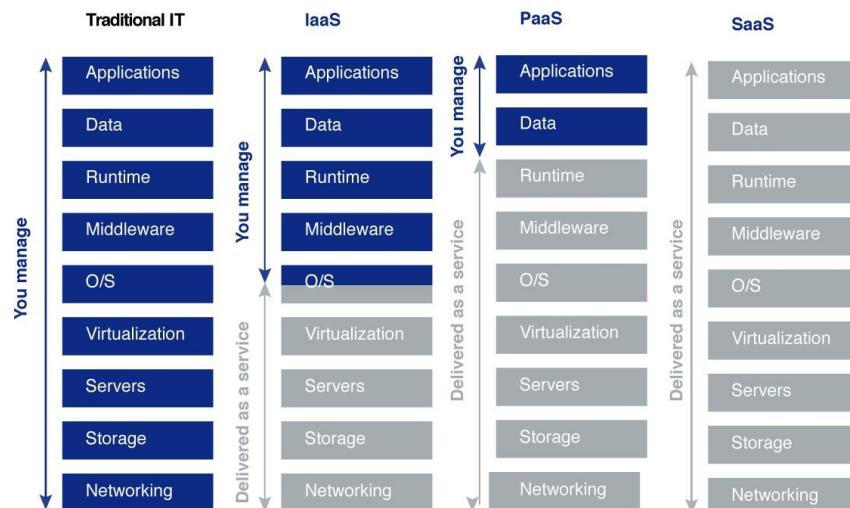
Позволяет создавать веб-приложения без покупки / обслуживания программного обеспечения и базовой инфраструктуры. Провайдер управляет инфраструктурой, включая сеть, серверы, ОС и хранилище, а заказчик контролирует развертывание приложений и, возможно, конфигурацию.

Infrastructure as a Service (IaaS)

Provider's offer servers, storage, network and operating systems – typically a platform virtualization environment – to consumers as an on-demand service, in a single bundle and billed according to usage. A popular use of IaaS is in hosting websites.

Постачальник пропонує споживачам сервери, сховище, мережу та операційні системи – як правило, середовище віртуалізації платформи – як послугу на вимогу, в одному пакеті та з оплатою в залежності від використання. Популярне використання IaaS – це розміщення веб-сайтів.

Поставщик предлагает потребителям серверы, хранилище, сеть и операционные системы – обычно среди виртуализации платформы – в качестве услуги по требованию, в едином пакете и оплачиваемой в соответствии с использованием. IaaS широко используется для хостинга веб-сайтов.



Source: Microsoft

Benefits of Cloud Computing

Cost-Reduction: Avoid up-front capital expenditure.

Зниження витрат: уникнення попередніх капітальних витрат.

Снижение затрат: избежание первоначальных капитальных затрат.

Scalability/Agility: Organizations set up resources on an as-needs basis.

Масштабованість / гнучкість: організації створюють ресурси за потребою.

Масштабируемость / гибкость: организации создают ресурсы по мере необходимости.

Improved Security: Providers can devote expertise and resources to security.

Покращена безпека: постачальники можуть приділяти знання та ресурси безпеці.

Повышенная безопасность: поставщики могут вкладывать знания и ресурсы в обеспечение безопасности.

Improved Reliability: Providers can devote expertise and resources on reliability of systems.

Покращена надійність: постачальники можуть приділяти знання та ресурси для забезпечення надійності систем.

Повышенная надежность: поставщики могут использовать свой опыт и ресурсы для обеспечения надежности систем.

Access to new technologies: Through use of provider's systems, customers may access latest technology.

Доступ до нових технологій: завдяки використанню систем провайдера клієнти можуть отримати доступ до новітніх технологій.

Доступ к новым технологиям: используя системы провайдера, клиенты могут получить доступ к новейшим технологиям.

Faster development: Provider's platforms can provide many of the core services to accelerate development cycle.

Швидший розвиток: платформи постачальників можуть надавати багато основних послуг для прискорення циклу розробки.

Ускоренная разработка: платформы поставщиков могут предоставлять многие из основных услуг для ускорения цикла разработки.

51

Large scale prototyping/load testing: Providers have the resources to enable this.

Широкомасштабне прототипування / тестування навантаження: постачальники мають ресурси для цього.

Крупномасштабное прототипирование / нагрузочное тестирование: у провайдеров есть ресурсы для этого.

More flexible working practices: Staff can access files using mobile devices.

Більш гнучкі практики роботи: персонал може отримувати доступ до файлів за допомогою мобільних пристройів.

Более гибкие методы работы: сотрудники могут получать доступ к файлам с мобильных устройств.

Increased competitiveness: Allows organizations to focus on their core competencies rather than their IT infrastructures.

Підвищена конкурентоспроможність: дозволяє організаціям зосередитися на своїх основних компетенціях, а не на ІТ-інфраструктурі.

Повышенная конкурентоспособность: позволяет организациям сосредоточиться на своих основных областях деятельности, а не на своей ИТ-инфраструктуре.

52

Risks of Cloud Computing

Network Dependency: Power outages, bandwidth issues and service interruptions.

Залежність від мережі: відключення електроенергії, проблеми з пропускною здатністю та перебої з обслуговуванням

Сетевая зависимость: перебои в подаче электроэнергии, проблемы с пропускной способностью и перебои в обслуживании.

System Dependency: Customer's dependency on availability and reliability of provider's systems.

Залежність системи: залежність замовника від доступності та надійності систем провайдера.

Системная зависимость: зависимость клиента от доступности и надежности систем поставщика.

Cloud Provider Dependency: Provider could became insolvent or acquired by competitor, resulting in the service suddenly terminating.

Залежність хмарного провайдера: постачальник може стати неплатоспроможним або придбаним конкурентом, в результаті чого послуга раптово припиняється.

Зависимость от облачного провайдера: провайдер может стать неплатежеспособным или быть приобретенным конкурентом, в результате чего услуга внезапно прекратится.

Lack of control: Customers unable to deploy technical or organizational measures to safeguard the data. May result in reduced availability, integrity, confidentiality, intervenability and isolation.

Відсутність контролю: клієнти не можуть застосувати технічні або організаційні заходи для захисту даних. Це може привести до зниження доступності, цілісності, конфіденційності, можливості втручання та ізоляції.

Отсутствие контроля: клиенты не могут применять технические или организационные меры для защиты данных. Может привести к снижению доступности, целостности, конфиденциальности, возможности вмешательства и изоляции.

Lack of information on processing transparency

Відсутність інформації щодо прозорості обробки

Отсутствие информации о прозрачности обработки

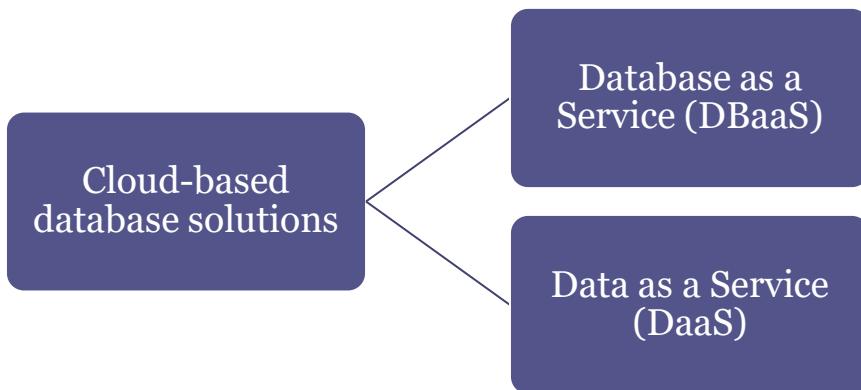
Cloud-based database solutions

As a type of Software as a Service (SaaS), cloud-based database solutions fall into two basic categories. Key difference between the two options is mainly how the data is managed.

Як тип програмного забезпечення як послуги (SaaS), хмарні рішення баз даних діляться на дві основні категорії. Ключова різниця між цими двома варіантами полягає головним чином у керуванні даними.

Как тип «Програмное обеспечение как услуга» (SaaS) облачные решения для баз данных делятся на две основные категории. Ключевое различие между двумя вариантами заключается в том, как управлять данными.

55



56

DBaaS

Offers full database functionality to application developers. Provides a management layer that provides continuous monitoring and configuring of the database to optimized scaling, high availability, multi-tenancy (that is, serving multiple client organizations), and effective resource allocation in the cloud, thereby sparing the developer from ongoing database administration tasks.

Пропонує розробникам додатків повну функціональність бази даних. Забезпечує рівень управління, який забезпечує постійний моніторинг та налаштування бази даних для оптимізованого масштабування, високої доступності, багатокористувачького використання (тобто обслуговування кількох клієнтських організацій) та ефективного розподілу ресурсів у хмарі, тим самим позбавляючи розробника від поточних завдань адміністрування баз даних.

Предлагает разработчикам приложений полную функциональность базы данных. Предоставляет уровень управления, который обеспечивает непрерывный мониторинг и настройку базы данных для оптимального масштабирования, высокой доступности, многопользовательского режима (то есть обслуживания нескольких клиентских организаций) и эффективного распределения ресурсов в облаке, тем самым избавляя разработчика от текущих задач администрирования базы данных.

DaaS

Services enables data definition in the cloud and subsequently querying. Does not implement typical DBMS interfaces (e.g. SQL) but instead data is accessed via common APIs. Enables organization with valuable data to offer access to others.

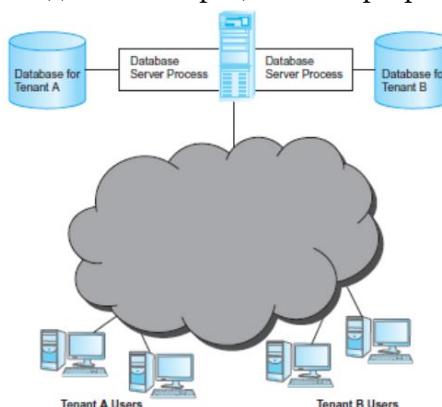
Послуги дозволяють визначення даних у хмарі та виконання подальших запитів. Не реалізує типові інтерфейси СУБД (наприклад, SQL), але натомість доступ до даних здійснюється через загальні API. Дозволяє організації з цінними даними пропонувати доступ іншим.

Сервисы позволяют определять данные в облаке и впоследствии выполнять запросы. Не реализует типичные интерфейсы СУБД (например, SQL), вместо этого доступ к данным осуществляется через общие API. Позволяет организациям с ценными данными предлагать доступ другим.

Multi-tenant cloud database – shared server, separate database server process architecture.

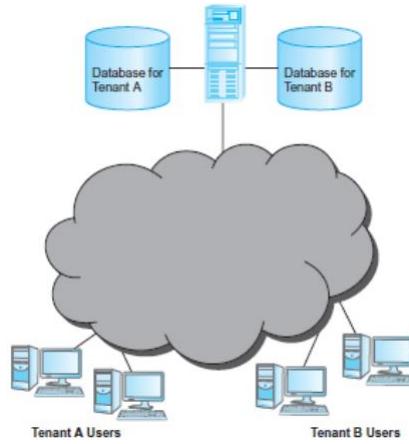
Багатокористувальська хмарна база даних – спільний сервер, архітектура з окремими процесами сервера баз даних.

Многопользовательская облачная база данных – общий сервер, архитектура с отдельными процессами сервера баз данных.



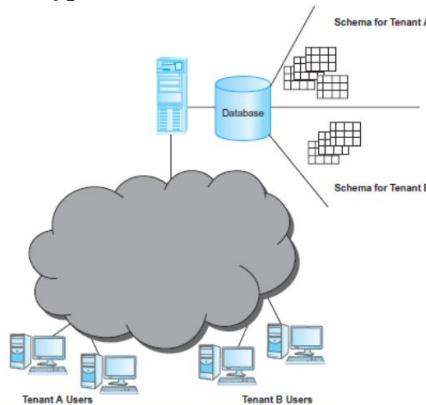
59

- Multi-tenant cloud database – shared DBMS server, separate databases.
 Багатокористувальська хмарна база даних – спільний сервер СУБД, окремі бази даних.
 Многопользовательская облачная база данных – общий сервер СУБД, отдельные базы данных.



60

- Multi-tenant cloud database – shared database, separate schema architecture.
 Багатокористувальська хмарна база даних – спільна база даних, архітектура з окремими схемами.
 Многопользовательская облачная база данных – общая база данных, архитектура с отдельными схемами.



Components of a DBMS

A DBMS is partitioned into several software components (or modules), each of which is assigned a specific operation. As stated previously, some of the functions of the DBMS are supported by the underlying operating system.

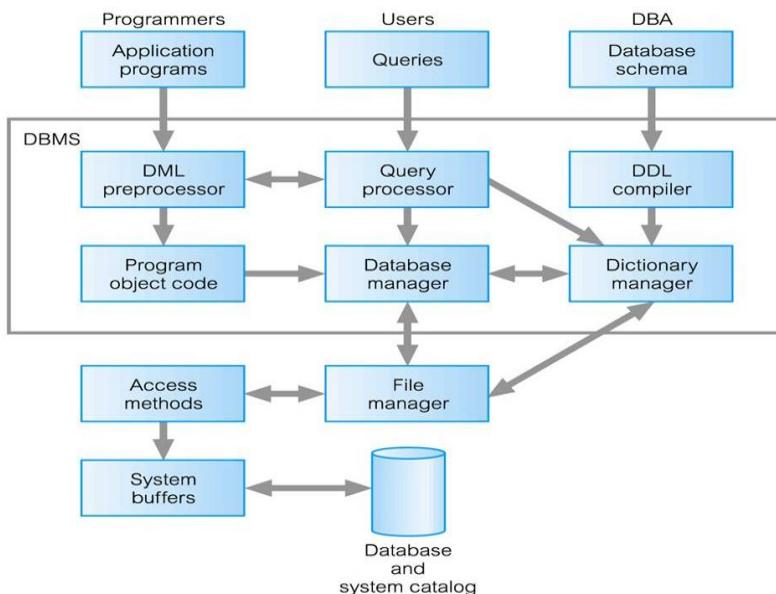
СУБД розділена на кілька програмних компонентів (або модулів), кожному з яких призначена певна операція. Як зазначалося раніше, деякі функції СУБД підтримуються базовою операційною системою.

СУБД разделена на несколько программных компонентов (или модулей), каждому из которых назначена определенная операция. Как указывалось ранее, некоторые функции СУБД поддерживаются базовой операционной системой.

The DBMS interfaces with other software components, such as user queries and access methods (file management techniques for storing and retrieving data records).

СУБД взаємодіє з іншими програмними компонентами, такими як запити користувачів та методи доступу (методи управління файлами для зберігання та отримання записів даних).

СУБД взаимодействует с другими программными компонентами, такими как пользовательские запросы и методы доступа (методы управления файлами для хранения и извлечения записей данных).



Query processor is a major DBMS component that transforms queries into a series of low-level instructions directed to the database manager.

Процесор запитів – це основний компонент СУБД, який перетворює запити в ряд інструкцій низького рівня, спрямованих до менеджера баз даних.

Оброботчик запитів – це основний компонент СУБД, який преобразує запити в серію низкоуровневих інструкцій, направляемих менеджеру бази даних.

Database manager (DM) interfaces with user-submitted application programs and queries. The DM examines the external and conceptual schemas to determine what conceptual records are required to satisfy the request. The DM then places a call to the file manager to perform the request.

Менеджер баз даних (DM) взаємодіє з поданими користувачем прикладними програмами та запитами. DM вивчає зовнішні та концептуальні схеми, щоб визначити, які концептуальні записи необхідні для задоволення запиту. Потім DM здійснює виклик диспетчера файлів для виконання запиту.

Менеджер баз данных (DM) взаимодействует с пользовательскими прикладными программами и запросами. DM исследует внешние и концептуальные схемы, чтобы определить, какие концептуальные записи необходимы для удовлетворения запроса. Затем DM обращается к файловому менеджеру, чтобы выполнить запрос.

File manager manipulates the underlying storage files and manages the allocation of storage space on disk. It establishes and maintains the list of structures and indexes defined in the internal schema.

Файловий менеджер маніпулює базовими файлами сховища та керує розподілом місця на диску. Він встановлює та підтримує перелік структур та індексів, визначених у внутрішній схемі.

Файловый менеджер управляет лежащими в основе файлами хранилища и распределяет дисковое пространство. Он устанавливает и поддерживает список структур и индексов, определенных во внутренней схеме.

DML preprocessor converts DML statements embedded in an application program into standard function calls in the host language. The DML preprocessor must interact with the query processor to generate the appropriate code.

Препроцесор DML перетворює вбудовані в прикладну програму оператори DML у стандартні виклики функцій основною мовою. Препроцесор DML повинен взаємодіяти з процесором запитів, щоб сформувати відповідний код.

Препроцессор DML преобразует операторы DML, встроенные в прикладную программу, в стандартные вызовы функций на основном языке. Препроцессор DML должен взаимодействовать с обработчиком запросов для генерации соответствующего кода.

65

DDL compiler converts DDL statements into a set of tables containing metadata. These tables are then stored in the system catalog while control information is stored in data file headers.

Компілятор DDL перетворює оператори DDL у набір таблиць, що містять метадані. Потім ці таблиці зберігаються в системному каталогі, тоді як інформація управління зберігається в заголовках файлів даних.

Компилятор DDL преобразует операторы DDL в набор таблиц, содержащих метаданные. Эти таблицы затем сохраняются в системном каталоге, а управляющая информация сохраняется в заголовках файлов данных.

Catalog manager manages access to and maintains the system catalog. The system catalog is accessed by most DBMS components.

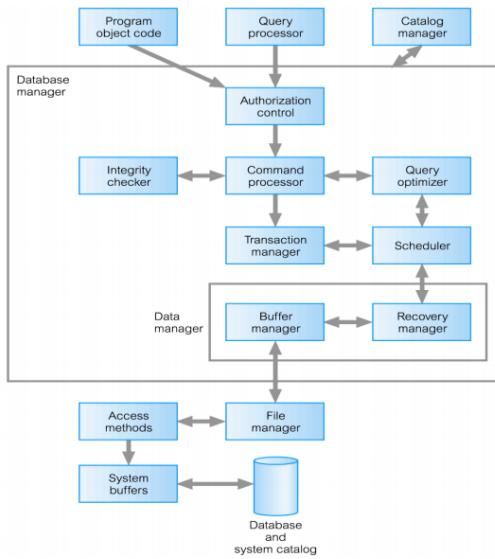
Менеджер каталогів управляет доступом до системного каталога и поддерживает его. До системного каталога обращаются большинство компонентов СУБД.

Диспетчер каталогов управляет доступом к системному каталогу и поддерживает его. К системному каталогу обращаются большинство компонентов СУБД.

66

DBMS ARCHITECTURES

Components of Database Manager (DM)



Authorization control – to confirm whether the user has the necessary permission to carry out the required operation.

Контроль авторизації – для підтвердження того, чи має користувач необхідний дозвіл на виконання необхідної операції.

Контроль авторизации – для подтверждения того, есть ли у пользователя необходимое разрешение для выполнения требуемой операции.

Command processor – on confirmation of user authority, control is passed to the command processor.

Командний процесор – при підтвердженні повноважень користувача, управління передається командному процесору.

Командный процессор – при подтверждении полномочий пользователя, управление передается командному процессору.

Integrity checker – ensures that requested operation satisfies all necessary integrity constraints (e.g. key constraints) for an operation that changes the database.

Засіб перевірки цілісності – гарантує, що запитана операція задовільняє всім необхідним обмеженням цілісності (наприклад, обмеженням ключів) для операції, яка змінює базу даних.

Средство проверки целостности – гарантирует, что запрошенная операция удовлетворяет всем необходимым ограничениям целостности (например, ключевым ограничениям) для операции, которая изменяет базу данных.

Query optimizer – determines an optimal strategy for the query execution.
 Оптимізатор запитів – визначає оптимальну стратегію для виконання запиту.
 Оптимизатор запросов – определяет оптимальную стратегию выполнения запроса.

Transaction manager – performs the required processing of operations that it receives from transactions.
 Менеджер транзакцій – виконує необхідну обробку операцій, які він отримує від транзакцій.
 Менеджер транзакций – выполняет необходимую обработку операций, которые он получает от транзакций.

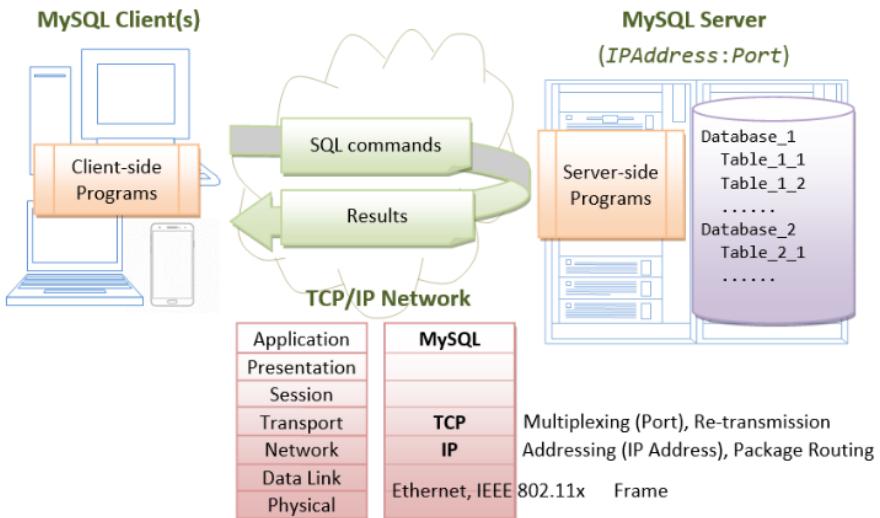
Scheduler – ensures that concurrent operations on the database proceed without conflicting with one another. It controls the relative order in which transaction operations are executed.
 Планувальник – забезпечує одночасні операції з базою даних, що не конфліктують між собою. Він контролює відносний порядок, в якому виконуються транзакційні операції.
 Планировщик – гарантирует, что параллельные операции с базой данных продолжаются без конфликта друг с другом. Он контролирует относительный порядок, в котором выполняются операции транзакции.

Recovery manager – ensures that the database remains in a consistent state in the presence of failures. It is responsible for transaction commit and abort.
 Менеджер відновлення – гарантує, що база даних залишається в стабільному стані при наявності збоїв. Він відповідає за здійснення та припинення транзакцій.
 Менеджер восстановления – гарантирует, что база данных останется в согласованном состоянии даже при сбоях. Он отвечает за фиксацию и прерывание транзакции.

Buffer manager – responsible for the transfer of data between main memory and secondary storage, such as disk and tape.
 Менеджер буферів – відповідальний за передачу даних між основною пам'яттю та вторинним ховищем.
 Диспетчер буферов – отвечающий за передачу данных между основной памятью и вторичным хранилищем.

The recovery manager and the buffer manager are also known as the data manager. The buffer manager also known as the cache manager.
 Менеджер відновлення та менеджер буферів, також відомі як менеджер даних.
 Менеджер буферов, также известный как кеш-менеджер.
 Диспетчер восстановления и диспетчер буферов, также известные как диспетчер данных. Диспетчер буферов, также известный как диспетчер кэша.

Client-Server Database Setup



MySQL

MySQL is a Relational Database Management System (“RDBMS”). It is used by most modern websites and web-based services as a convenient and fast-access storage and retrieval solution for large volumes of data. A simple example of items which might be stored in a MySQL database would be a site-registered user’s name with associated password (encrypted for security), the user registration date, and number of times visited, etc.

MySQL – це реляційна система управління базами даних (“СУБД”). Вона використовується більшістю сучасних веб-сайтів та веб-сервісів як зручне та швидке рішення для зберігання та пошуку великих обсягів даних. Простим прикладом елементів, які можуть зберігатися в базі даних MySQL, може бути ім’я користувача, зареєстрованого на сайті, із пов’язаним паролем (зшифрованим для безпеки), датою реєстрації користувача та кількістю відвідувань тощо.

MySQL – это система управления реляционными базами данных («СУБД»). Она используется большинством современных веб-сайтов и веб-служб в качестве удобного и быстрого решения для хранения и поиска больших объемов данных. Простым примером элементов, которые могут храниться в базе данных MySQL, могут быть имя пользователя, зарегистрированного на сайте, с соответствующим паролем (зашифрованным для безопасности), дата регистрации пользователя, количество посещений и т. д.

Why MySQL?

- Free database management system
 - Client-server architecture
 - Open-Source (MariaDB)
 - Cross-platform system
 - Application Programming Interfaces (APIs)
 - Multithreading
 - Concurrent access
 - Speed
 - Scalability
 - Security and access management based on the privileges system
 - Relational database management system (DBMS)
-
- Безкоштовна система управління базами даних
 - Клієнт-серверна архітектура
 - З відкритим кодом (MariaDB)
 - Кроссплатформна система
 - Інтерфейси прикладного програмування (API)
 - Багатопотоковість
 - Одночасний доступ
 - Швидкість
 - Масштабованість
 - Безпека та управління доступом на основі системи привілеїв
 - Реляційна система управління базами даних (СУБД)

- Бесплатная система управления базами данных
- Клиент-серверная архитектура
- Открытый исходный код (MariaDB)
- Кросс-платформенная система
- Интерфейсы прикладного программирования (API)
- Многопоточность
- Одновременный доступ
- Скорость
- Масштабируемость
- Управление безопасностью и доступом на основе системы привилегий
- Система управления реляционными базами данных (СУБД)

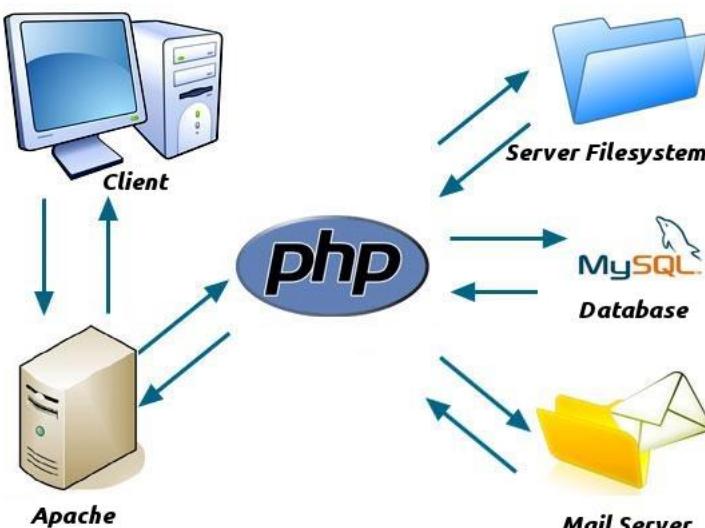
75

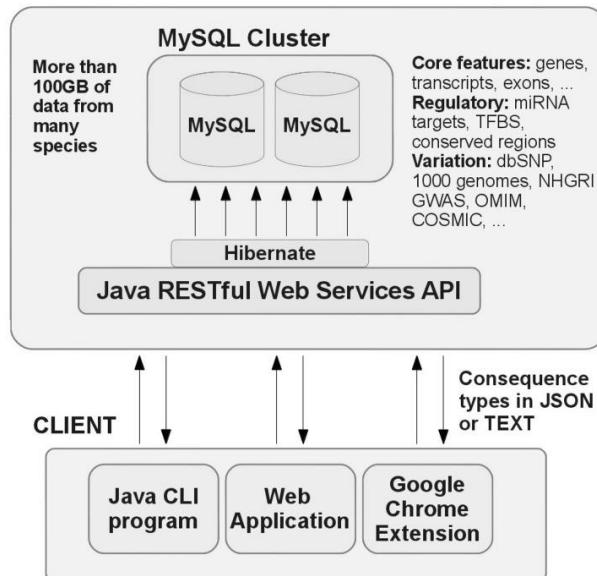
MySQL can also be accessed using many tools. It can be easily communicated with via PHP (PHP Hypertext Preprocessor), a scripting language whose primary focus is to manipulate HTML for a webpage on the server before it is delivered to a client's machine. A user can submit queries to a database via PHP, allowing insertion, retrieval and manipulation of information into/from the database.

Доступ до MySQL також можна отримати за допомогою багатьох інструментів. З ним можна легко спілкуватися за допомогою PHP (PHP Hypertext Preprocessor), мови сценаріїв, основною метою якої є маніпулювання HTML-кодом веб-сторінки на сервері перед її доставкою на машину клієнта. Користувач може надсилати запити до бази даних за допомогою PHP, дозволяючи вставляти, отримувати та маніпулювати інформацією в / з бази даних.

Доступ к MySQL также можно получить с помощью многих инструментов. С ним можно легко взаимодействовать через PHP (PHP Hypertext Preprocessor), языке сценариев, основной задачей которого является управление HTML для веб-страницы на сервере до того, как он будет доставлен на клиентский компьютер. Пользователь может отправлять запросы к базе данных через PHP, что позволяет вставлять, извлекать и изменять информацию в / из базы данных.

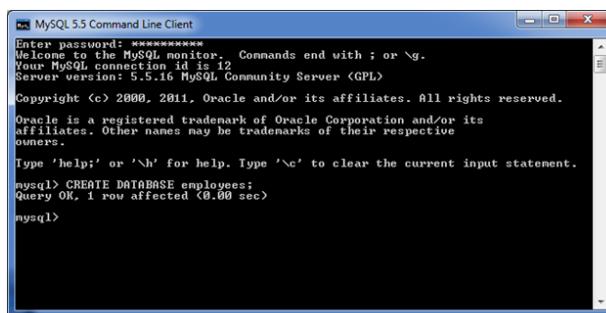
76



SERVER

There are many MySQL clients available including remote devices
Доступно багато клієнтів MySQL, включаючи віддалені пристрії
Доступно множество клиентов MySQL, включая удаленные устройства

The standard MySQL client is the command-line application
Стандартним клієнтом MySQL є програма командного рядка
Стандартный клиент MySQL – это приложение командной строки



MySQL Installation

System-wide installation using a graphical installation package

Загальносистемна установка за допомогою графічного інсталяційного пакету

Общесистемная установка с использованием графического установочного пакета

Local installation using a “no-install” package

Локальна установка за допомогою пакета “no-install”

Локальная установка с использованием пакета «без установки»

System-wide installation using the XAMPP integrated package

Загальносистемна установка за допомогою інтегрованого пакету XAMPP

Общесистемная установка с использованием интегрированного пакета XAMPP

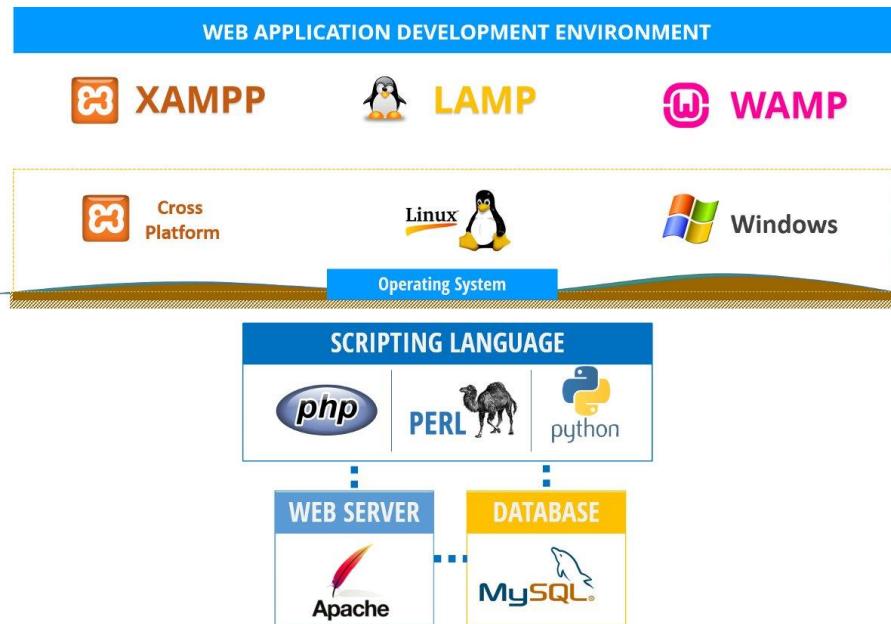
XAMPP

XAMPP is a freely available software package which integrates distributions for Apache web server, MySQL, PHP and Perl/Python into one easy installation. If you wish to set up a web server on your home computer, this is the recommended route.

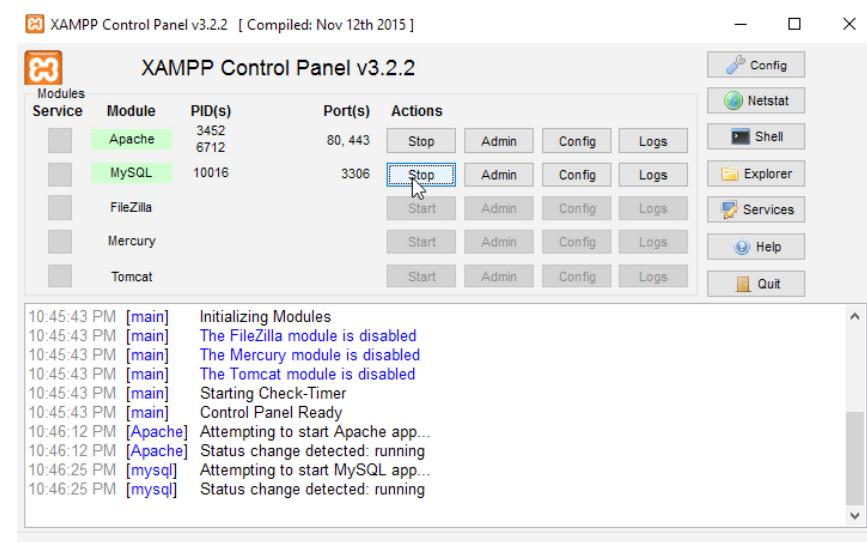
XAMPP – це вільно доступний програмний пакет, який інтегрує дистрибутиви для веб-сервера Apache, MySQL, PHP та Perl/Python в одну просту установку. Якщо ви хочете встановити веб-сервер на домашньому комп’ютері, це рекомендований шлях.

XAMPP – это свободно доступный программный пакет, который объединяет дистрибутивы для веб-сервера Apache, MySQL, PHP и Perl/Python в одну простую установку. Если вы хотите настроить веб-сервер на своем домашнем компьютере, это рекомендуемый путь.

81



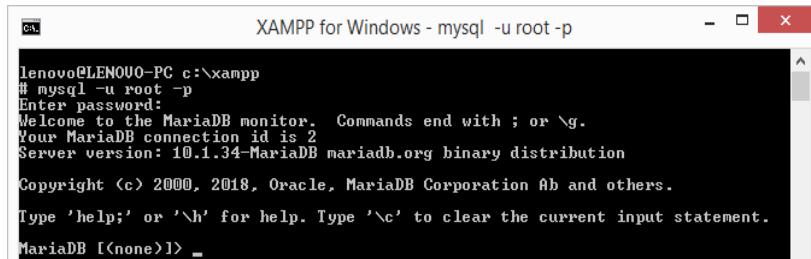
82



83

MySQL Command Line Client (2-Tier Architecture)

C:\xampp\mysql\bin > mysql -u root -p
 XAMPP Control Panel > Shell > mysql -u root -p



```
lenovo@LENOVO-PC c:\xampp
# mysql -u root -p
Enter password:
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 2
Server version: 10.1.34-MariaDB mariadb.org binary distribution

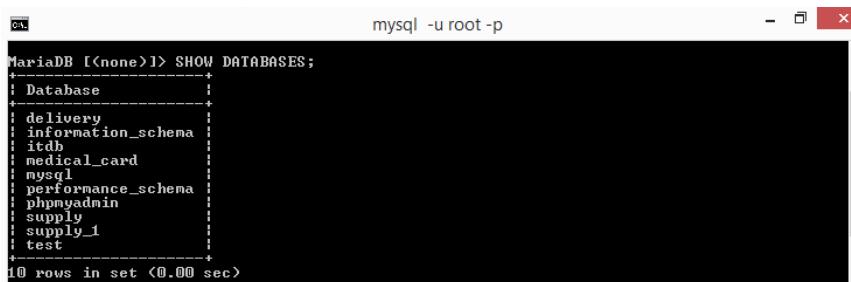
Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MariaDB [(none)]> _
```

84

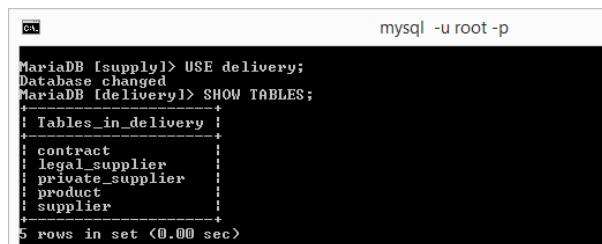
SHOW DATABASES;



```
MariaDB [(none)]> SHOW DATABASES;
+-----+
| Database |
+-----+
| delivery |
| information_schema |
| itdb |
| medical_card |
| mysql |
| performance_schema |
| phpmyadmin |
| supply |
| supply_1 |
| test |
+-----+
10 rows in set (0.00 sec)
```

USE <table_name>;

SHOW TABLES;



```
MariaDB [supply]> USE delivery;
Database changed
MariaDB [delivery]> SHOW TABLES;
+-----+
| Tables_in_delivery |
+-----+
| contract |
| legal_supplier |
| private_supplier |
| product |
| supplier |
+-----+
5 rows in set (0.00 sec)
```

Dumping a database as SQL statements

Вивантаження бази даних як операторів SQL

Выгрузка базы данных в виде операторов SQL

```
mysqldump
--user=<user_name>
--password=<user_password>
--result-file=<file_name>
<db_name>
```

```
mysqldump --user=root --result-file=supply.sql
supply
```

```
19  -- Table structure for table `contract`
20  --
21
22  DROP TABLE IF EXISTS `contract`;
23  /*!40101 SET @saved_cs_client      = @@character_set_client */;
24  /*!40101 SET character_set_client = utf8 */;
25  CREATE TABLE `contract` (
26    `contract_number` int(11) NOT NULL AUTO_INCREMENT,
27    `contract_date` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
28    `supplier_id` int(11) NOT NULL,
29    `contract_note` varchar(100) DEFAULT NULL,
30    PRIMARY KEY (`contract_number`),
31    KEY `contract_ibfk_1` (`supplier_id`),
32    CONSTRAINT `contract_ibfk_1` FOREIGN KEY (`supplier_id`) REFERENCES `supplier` (`supplier_id`)
33  ) ENGINE=InnoDB AUTO_INCREMENT=14 DEFAULT CHARSET=latin1;
34  /*!40101 SET character_set_client = @saved_cs_client */;
35
36  --
37  -- Dumping data for table `contract`
38  --
39
40  LOCK TABLES `contract` WRITE;
41  /*!40000 ALTER TABLE `contract` DISABLE KEYS */;
42  INSERT INTO `contract` VALUES (1,'2018-08-31 21:00:00',1,'Order 34 on 30.08.2018'),(2,'2018-09-09
43  /*!40000 ALTER TABLE `contract` ENABLE KEYS */;
44  UNLOCK TABLES;
```

87

Backup file does not contain CREATE DATABASE and
USE statements

Файл резервної копії не містить операторів CREATE
DATABASE та USE

Файл резервной копии не содержит операторов
CREATE DATABASE и USE.

MUST HAVE BEFORE RECOVERY:

```
DROP DATABASE <existing_db>;
CREATE DATABASE <recovered_db>;
USE <recovered_db>;
```

88

```
MariaDB [(none)]> CREATE DATABASE rec_supply;
Query OK, 1 row affected (0.00 sec)

MariaDB [(none)]> USE rec_supply;
Database changed
MariaDB [rec_supply]> SOURCE supply.sql;
Query OK, 0 rows affected (0.00 sec)

Query OK, 0 rows affected (0.00 sec)
Query OK, 0 rows affected (0.00 sec)
Query OK, 0 rows affected (0.00 sec)
Query OK, 0 rows affected (0.00 sec)
Query OK, 0 rows affected (0.00 sec)
Query OK, 0 rows affected (0.00 sec)
Query OK, 0 rows affected (0.00 sec)
Query OK, 0 rows affected (0.00 sec)
```

89

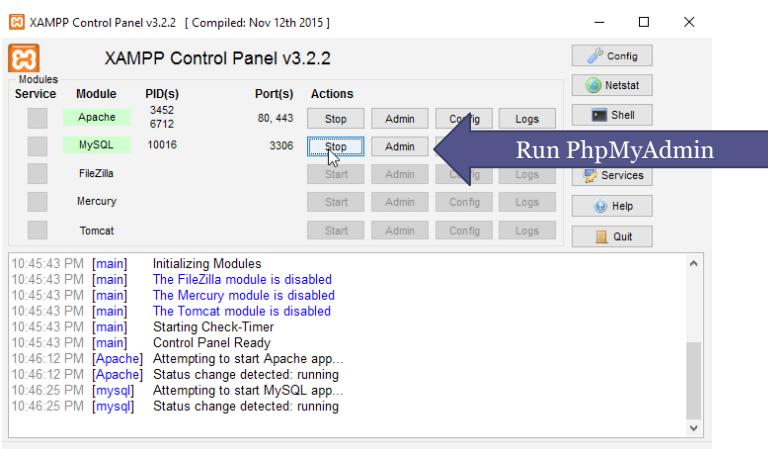
```
MariaDB [rec_supply]> SHOW TABLES;
+-----+
| Tables_in_rec_supply |
+-----+
| contract
| contract_supplier
| supplied
| supplier
| supplier_info
| supplier_org
| supplier_person
+-----+
7 rows in set (0.00 sec)

MariaDB [rec_supply]> SELECT * FROM supplier_org;
+-----+-----+
| supplier_id | supplier_org_name |
+-----+-----+
| 2 | Interfruit Ltd. |
| 4 | Transservice LLC |
+-----+-----+
2 rows in set (0.00 sec)
```

90

PhpMyAdmin Client (3/N-Tier Architecture)

XAMPP Control Panel > MySQL > Admin



Also included within XAMPP is phpMyAdmin, a web-based frontend (“graphical interface”) for MySQL, allowing queries to be submitted via mouse clicks in a web browser or by writing these queries in the SQL box inside phpMyAdmin.

Також до складу XAMPP входить phpMyAdmin, веб-інтерфейс («графічний інтерфейс») для MySQL, що дозволяє надсилати запити за допомогою клапання миші у веб-браузері або шляхом запису цих запитів у поле SQL усередині phpMyAdmin.

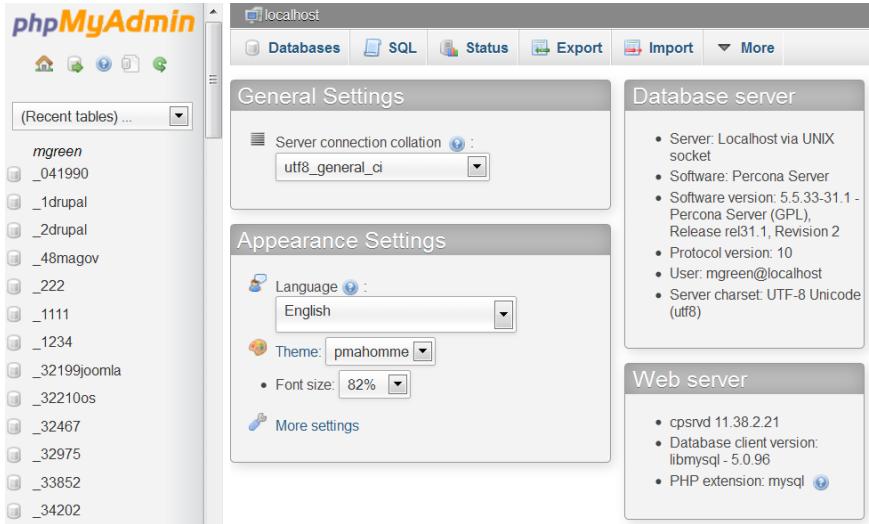
В XAMPP також включен phpMyAdmin, веб-інтерфейс («графический интерфейс») для MySQL, позволяющий отправлять запросы с помощью щелчков мыши в веб-браузере или путем написания этих запросов в поле SQL внутри phpMyAdmin.

PhpMyAdmin allows the user to write SQL command from the SQL tab. It also provides a mechanism to import SQL command from a file. However, its interface provides other ways to perform tasks using a graphical user interface (GUI). For instance, you can write a command to create a table in your database. You can also achieve this from phpMyAdmin GUI.

PhpMyAdmin дозволяє користувачеві писати команду SQL із вкладки SQL. Він також надає механізм імпорту команди SQL з файлу. Однак його інтерфейс забезпечує інші способи виконання завдань за допомогою графічного інтерфейсу користувача (GUI). Наприклад, ви можете написати команду для створення таблиці у вашій базі даних. Ви також можете досягти цього за допомогою графічного інтерфейсу phpMyAdmin.

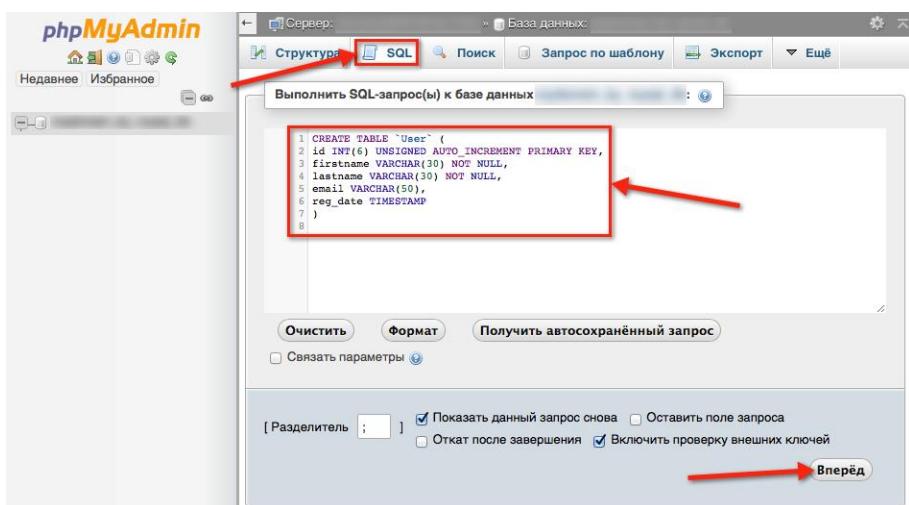
PhpMyAdmin позволяет пользователю писать команду SQL из вкладки SQL. Он также предоставляет механизм для импорта команды SQL из файла. Однако его интерфейс предоставляет другие способы выполнения задач с использованием графического пользовательского интерфейса (GUI). Например, вы можете написать команду для создания таблицы в своей базе данных. Вы также можете добиться этого из графического интерфейса phpMyAdmin.

93



94

Execute SQL in PhpMyAdmin



95

Observe Database Schema

The screenshot shows the phpMyAdmin interface for a MySQL database named 'world'. On the left, the database structure is listed with tables: Usuarios, Usuarios1, VSet, VsetAdmin, Xss, uam, uam2, ube_db, victoria_base, vseti, and world (which contains City, New, Country, and CountryLanguage). The 'City' table is selected, displaying its columns: ID (int(11)), Name (char(35)), CountryCode (char(3)), District (char(20)), and Population (int(11)). A green arrow points from the 'Country' table to the 'CountryLanguage' table, which is also selected. The 'CountryLanguage' table has columns: CountryCode (char(3)), Language (char(30)), IsOfficial (enum('T', 'F')), and Percentage (float(4,1)). The 'Country' table is shown on the right with columns: Code (char(3)), Name (char(52)), Continent (enum('Asia', 'Europe', 'North America', 'Africa', 'Oceania', 'Antarctica', 'South America')), Region (char(26)), SurfaceArea (float(10,2)), IndepYear (smallint(6)), Population (int(11)), LifeExpectancy (float(3,1)), GNP (float(10,2)), GNPOld (float(10,2)), LocalName (char(45)), GovernmentForm (char(45)), HeadOfState (char(60)), Capital (int(11)), and Code2 (char(2)).

96

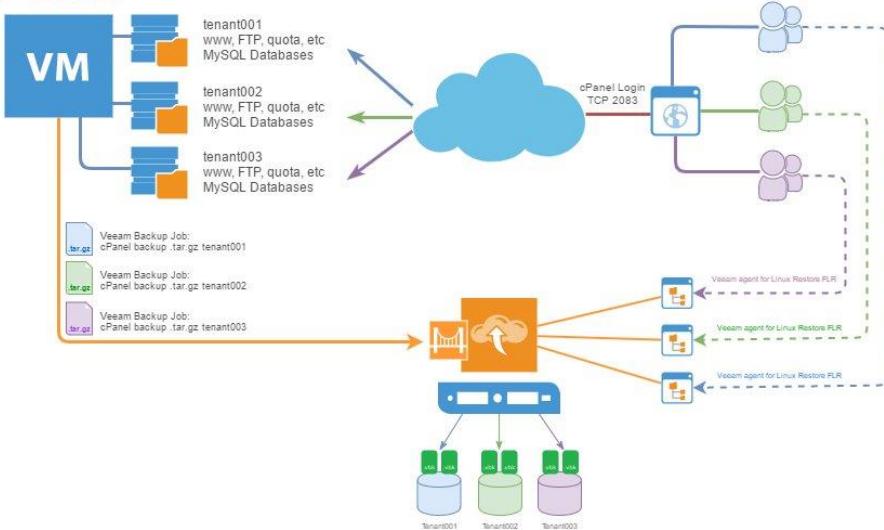
cPanel

Most web hosting companies provide a web hosting control panel called cPanel. cPanel provides a graphical interface and automation tools to simplify web hosting for customers. cPanel has phpMyAdmin integrated to its system. cPanel has loads of features like website builders, easy transfer of websites, email setup, remote access, etc.

Більшість компаній, що займаються веб-хостингом, пропонують панель управління веб-хостингом під назвою cPanel. cPanel надає графічний інтерфейс та засоби автоматизації для спрощення веб-хостингу для клієнтів. cPanel має інтегровану в систему систему phpMyAdmin. cPanel має безліч функцій, таких як конструктори веб-сайтів, легке переміщення веб-сайтів, налаштування електронної пошти, віддалений доступ тощо.

Большинство хостинговых компаний предоставляют панель управления веб-хостингом под названием cPanel. cPanel предоставляет графический интерфейс и инструменты автоматизации для упрощения веб-хостинга для клиентов. В cPanel интегрирован phpMyAdmin. cPanel имеет множество функций, таких как конструкторы веб-сайтов, простой перенос веб-сайтов, настройка электронной почты, удаленный доступ и т. д.

97



98



The screenshot shows the cPanel control panel interface:

- FILES:** Includes links for File Manager, Images, Directory Privacy, Disk Usage, Backup, and Backup Wizard.
- DATABASES:** Includes links for phpMyAdmin, MySQL® Databases, MySQL® Database Wizard, and Remote MySQL®.
- DOMAINS:** Includes links for Site Publisher, .com Addon Domains, Subdomains, and Aliases.

99

SUMMARY

Client – a software used to represent data and implement business logic

Клієнт – програмне забезпечення, що використовується для представлення даних та реалізації бізнес-логіки

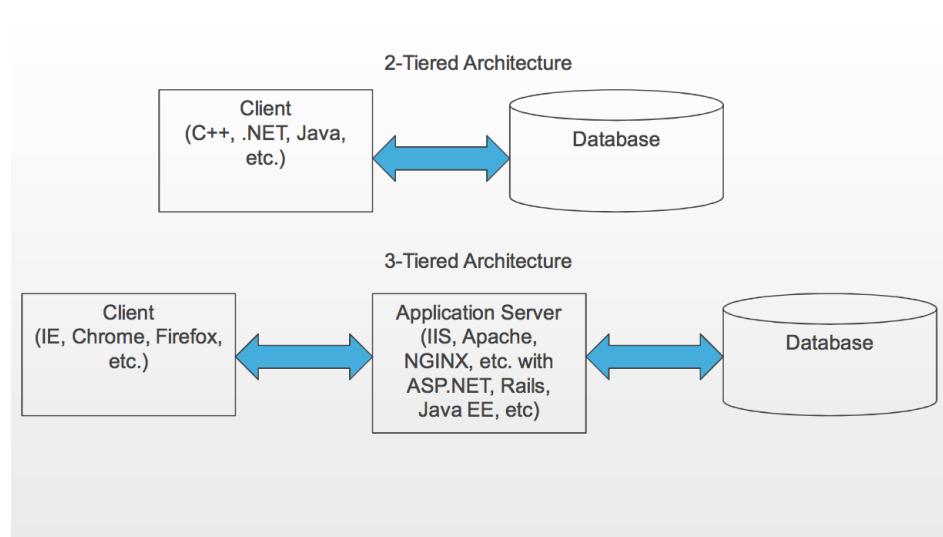
Клиент – программное обеспечение, используемое для представления данных и реализации бизнес-логики

Server – a software used to manage data and data itself

Сервер – програмне забезпечення, що використовується для управління даними та самі дани

Сервер – программное обеспечение, используемое для управления данными и сами данные

100



101

3-Tier architecture

Presentation layer – data representation

Application layer – business logic

Database layer – data management and data

Презентаційний рівень – представлення даних

Прикладний рівень – бізнес-логіка

Рівень бази даних – управління даними та даними

Уровень представления – представление данных

Уровень приложения – бизнес-логика

Уровень базы данных – управление данными и данные

102

Presentation Layer

```
<div class="col-4">
    <ul class="list-group">
        <li class="list-group-item active">Contracts</li>
        <?php foreach ($service->getAllContracts() as $contract) { ?>
            <li class="list-group-item">
                <a href="contracts.php?details=<?= $contract->getNumber() ?>">
                    #<?= $contract->getNumber() ?, <?= $contract->getAgreed() ?,>
                </a>
            </li>
        <?php } ?>
    </ul>
</div>
```

103

Application Layer

```
class ContractService
{
    private $repository;

    public function __construct(ContractRepositoryInterface $repository)
    {
        $this->repository = $repository;
    }

    public function getAllContracts()
    {
        return $this->repository->getContractList();
    }
}
```

104

Database Layer

```
public function getContractList()
{
    $conn = MySQLConnectionUtil::getConnection();
    $contracts = array();

    $query = 'SELECT number, agreed, supplier.name, title, note
              FROM contract INNER JOIN supplier ON contract.supplier = supplier.id';
    $result = mysqli_query($conn, $query);
```

number	agreed	supplier	title	note
1	1999-09-01	2	Contract 1	Invoice 34from 8/30/99
2	1999-09-10	2	Contract 2	Invoice 08-78 from 8/28/99
3	1999-09-10	4	Contract 3	Invoice 08-78 from8/28/99
4	1999-09-23	4	Contract 4	Order 56from 8/28/99
5	1999-09-24	3	Contract 5	Invoice 74from 9/11/99
6	1999-10-01	2	Contract 6	Invoice 9-12from 9/28/99
7	1999-10-02	3	Contract 7	Invoice 85from 9/21/99

105

QUESTIONS

106

Контрольні питання

1. Файловий сервер
2. Традиційний дворівневий клієнт-сервер
3. Трирівневий клієнт-сервер
4. n-рівневий клієнт-сервер
5. Проміжне програмне забезпечення
6. Монітори обробки транзакцій
7. Веб-сервіси та сервісно-орієнтовані архітектури
8. Сервісно-орієнтовані архітектури (SOA)
9. Розподілені СУБД
10. Зберігання даних
11. Хмарні обчислення
12. Рішення хмарних баз даних
13. Компоненти СУБД
14. Компоненти диспетчера баз даних (DM)

Assessment questions

1. File-Server
2. Traditional Two-Tier Client-Server
3. Three-Tier Client-Server
4. n-Tier Client-Server
5. Middleware
6. Transaction Processing Monitors
7. Web Services and Service-Oriented Architectures
8. Service-Oriented Architectures (SOA)
9. Distributed DBMSs
10. Data Warehousing
11. Cloud Computing
12. Cloud-based database solutions
13. Components of a DBMS
14. Components of Database Manager (DM)

Контрольные вопросы

1. Файловый сервер
2. Традиционный двухуровневый клиент-сервер
3. Трехуровневый клиент-сервер
4. n-уровневый клиент-сервер
5. ПО промежуточного слоя
6. Мониторы обработки транзакций
7. Веб-службы и сервис-ориентированные архитектуры
8. Сервисно-ориентированные архитектуры (SOA)
9. Распределенные СУБД
10. Хранилище данных
11. Облачные вычисления
12. Облачные решения для баз данных
13. Компоненты СУБД
14. Компоненты менеджера баз данных (DM)

109

Проектування та реалізація БД DB design and implementations Проектирование и реализация БД

Тема 2

Topic 2

110

BUSINESS PROCESSES

111

Data Flow Diagrams (DFD)

Data-flow diagrams (DFDs) model a perspective of the system that is most readily understood by users – the flow of information through the system and the activities that process this information.

Діаграми потоків даних (DFD) моделюють перспективу системи, яка найбільш легко зрозуміла користувачам – потік інформації через систему та види діяльності, що обробляють цю інформацію.

Диаграммы потоков данных (DFD) моделируют перспективу системы, которая наиболее понятна пользователям – поток информации через систему и действия, которые обрабатывают эту информацию.

112

Data-flow diagrams provide a graphical representation of the system that aims to be accessible to computer specialist and non-specialist users alike. The models enable software engineers, customers and users to work together effectively during the analysis and specification of requirements. Although this means that our customers are required to understand the modeling techniques and constructs, in data-flow modeling only a limited set of constructs are used, and the rules applied are designed to be simple and easy to follow. These same rules and constructs apply to all data-flow diagrams (i.e., for each of the different software process activities in which DFDs can be used).

Діаграми потоків даних забезпечують графічне представлення системи, яка прагне бути доступною як для спеціалістів, так і для неспеціалістів. Моделі дозволяють інженерам програмного забезпечення, замовникам та користувачам ефективно працювати разом під час аналізу та уточнення вимог. Хоча це означає, що від наших клієнтів вимагається розуміння методів моделювання та конструкцій, у моделюванні потоків даних використовується лише обмежений набір конструкцій, а застосувані правила спроектовані таким чином, щоб їх було просто і легко виконувати. Ці самі правила та конструкції застосовуються до всіх діаграм потоку даних (тобто для кожної з різних програмних операцій, в яких можна використовувати DFD).

113

Диаграммы потоков данных обеспечивают графическое представление системы, которая должна быть доступна как специалистам, так и неспециалистам. Модели позволяют разработчикам программного обеспечения, заказчикам и пользователям эффективно работать вместе во время анализа и спецификации требований. Хотя это означает, что наши клиенты должны понимать методы и конструкции моделирования, в моделировании потока данных используется только ограниченный набор конструкций, а применяемые правила разработаны так, чтобы быть простыми и легкими для выполнения. Эти же правила и конструкции применяются ко всем диаграммам потоков данных (т. е. для каждой из различных операций процесса программного обеспечения, в которых могут использоваться DFD).

114

DFD notation consists of only four main symbols:

1. Processes — the activities carried out by the system which use and transform information. Processes are notated as rectangles with three parts.
2. Data-flows — the data inputs to and outputs from these activities. Data-flows are notated as named arrows.
3. External entities — the sources from which information flows into the system and the recipients of information leaving the system. External entities are notated as ovals.
4. Data stores — where information is stored within the system. Data stores are notated as rectangles with two parts.

The diagrams are supplemented by supporting documentation including a data dictionary, describing the contents of data-flows and data stores; and process definitions, which provide detailed descriptions of the processes identified in the data-flow diagram.

115

Нотація DFD складається лише з чотирьох основних символів:

1. Процеси - діяльність, здійснювана системою, яка використовує та перетворює інформацію. Процеси позначені як прямокутники з трьох частин.
2. Потоки даних - дані, що надходять і виводяться з цих видів діяльності. Потоки даних позначаються як іменовані стрілки.
3. Зовнішні сутності - джерела, з яких інформація надходить у систему, а одержувачі інформації виходять із системи. Зовнішні об'єкти позначаються як овали.
4. Накопичувачі даних - де інформація зберігається в системі. Накопичувачі даних позначені як прямокутники з двох частин.

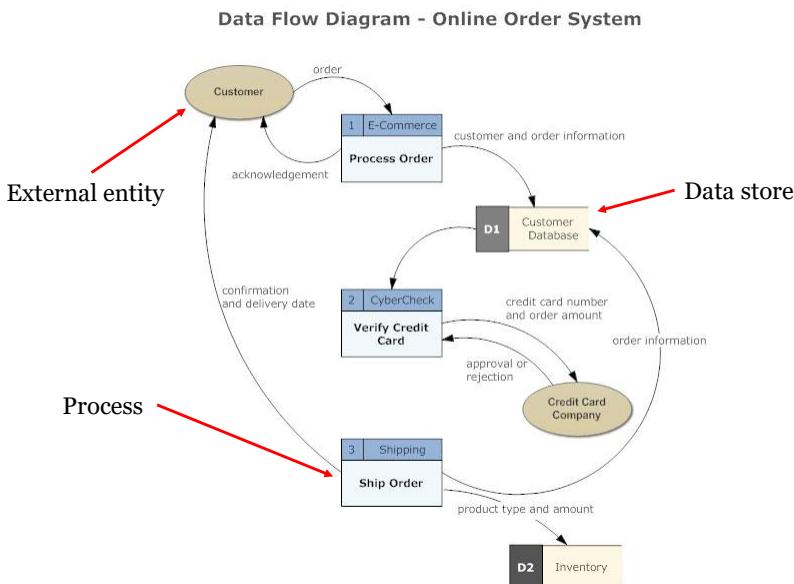
Діаграми доповнені допоміжною документацією, включаючи словник даних, що описує вміст потоків даних та сховищ даних; та визначення процесів, які надають детальний опис процесів, визначених на схемі потоку даних.

116

Обозначение DFD состоит всего из четырех основных символов:

1. Процессы - действия, выполняемые системой, которые используют и преобразуют информацию. Процессы обозначаются прямоугольниками, состоящими из трех частей.
2. Потоки данных - входы данных и выходы этих действий. Потоки данных обозначены как именованные стрелки.
3. Внешние сущности - источники, из которых информация поступает в систему, и получатели информации, покидающие систему. Внешние объекты обозначены овалами.
4. Накопители данных - где информация хранится в системе. Накопители данных обозначены прямоугольниками, состоящими из двух частей.

Диаграммы дополняются сопроводительной документацией, включая словарь данных, описывающий содержимое потоков данных и хранилищ данных; и определения процессов, которые представляют подробные описания процессов, идентифицированных на диаграмме потоков данных.



Data-flow diagrams provide a very important tool for software engineering, for a number of reasons:

- The system scope and boundaries are clearly indicated on the diagrams.
- The technique of decomposition of high level data-flow diagrams to a set of more detailed diagrams, provides an overall view of the complete system, as well as a more detailed breakdown and description of individual activities, where this is appropriate, for clarification and understanding.

Діаграми потоків даних забезпечують дуже важливий інструмент для інженерії програмного забезпечення з ряду причин:

- Обсяг системи та межі чітко вказані на діаграмах.
- Методика декомпозиції діаграм потоків даних високого рівня на набір більш детальних діаграм забезпечує загальний огляд всієї системи, а також більш детальну розбивку та опис окремих видів діяльності, де це доцільно, для роз'яснення та розуміння.

119

Диаграммы потоков данных предоставляют очень важный инструмент для разработки программного обеспечения по ряду причин:

- Объем и границы системы четко обозначены на схемах.
- Техника декомпозиции диаграмм потоков данных высокого уровня на набор более подробных диаграмм обеспечивает общий вид всей системы, а также более подробную разбивку и описание отдельных действий, где это уместно, для разъяснения и понимания.

120

Although all data-flow diagrams are composed of the same types of symbols, and the validation rules are the same for all DFDs, there are **three main types of data-flow diagram**:

1. Context diagrams — context diagram DFDs are diagrams that present an overview of the system and its interaction with the rest of the “world”.
2. Level 1 data-flow diagrams — Level 1 DFDs present a more detailed view of the system than context diagrams, by showing the main sub-processes and stores of data that make up the system as a whole.
3. Level 2 (and lower) data-flow diagrams — a major advantage of the data-flow modelling technique is that, through a technique called “levelling”, the detailed complexity of real world systems can be managed and modeled in a hierarchy of abstractions. Certain elements of any dataflow diagram can be decomposed (“exploded”) into a more detailed model a level lower in the hierarchy.

Хоча всі діаграми потоків даних складаються з однакових типів символів, а правила перевірки одинакові для всіх DFD, існує **три основних типи діаграм потоків даних**:

1. Контекстні діаграми - контекстна діаграма DFD - це діаграми, що представляють огляд системи та її взаємодії з рештою "світу".
2. Діаграми потоків даних рівня 1 - DFD рівня 1 представляють більш детальне уявлення про систему, ніж контекстні діаграми, показуючи основні підпроцеси та сховища даних, що складають систему в цілому.
3. Діаграми потоків даних рівня 2 (і нижчих) - головною перевагою техніки моделювання потоку даних є те, що завдяки техніці, яка називається «нівелювання», детальна складність систем реального світу може управлятися та моделюватися в ієрархії абстракцій. Деякі елементи будь-якої діаграми потоку даних можуть бути розкладені ("розібрани") у більш детальну модель на рівень нижче в ієрархії.

Хотя все диаграммы потоков данных состоят из символов одного и того же типа, а правила проверки одинаковы для всех DFD, существует **три основных типа диаграмм потоков данных**:

1. Контекстные диаграммы - контекстная диаграмма DFD - это диаграммы, которые представляют собой обзор системы и ее взаимодействия с остальным «миром».
2. Диаграммы потоков данных уровня 1 - DFD уровня 1 представляют более подробное представление системы, чем контекстные диаграммы, показывая основные подпроцессы и хранилища данных, которые составляют систему в целом.
3. Диаграммы потоков данных уровня 2 (и ниже) - главное преимущество метода моделирования потоков данных заключается в том, что с помощью метода, называемого «выравнивание», можно управлять подробной сложностью реальных систем и моделировать их в иерархии абстракций. Некоторые элементы любой диаграммы потока данных могут быть разложены («разнесены») в более подробную модель на уровень ниже в иерархии.

123

Elements of data-flow diagrams. Four basic elements are used to construct data-flow diagrams:

- processes
- data-flows
- data stores
- external entities

Елементи діаграм потоків даних. Для побудови діаграм потоків даних використовуються чотири основні елементи:

- процесів
- потоки даних
- накопичувачі даних
- зовнішні сутності

Элементы диаграмм потоков данных. Для построения диаграмм потоков данных используются четыре основных элемента:

- процессы
- потоки данных
- накопители данных
- внешние организации

124

Processes

Processes are the essential activities, carried out within the system boundary, that use information. A process is represented in the model only where the information which provides the input into the activity is manipulated or transformed in some way, so that the data-flowing out of the process is changed compared to that which flowed in.

Процеси - це основні дії, що виконуються в межах системи та використовують інформацію. Процес представляється у моделі лише там, де інформація, яка поступає на вход, якимось чином обробляється або трансформується таким чином, що інформація на виході відрізняється від інформації на вході.

Процессы - это важные действия, выполняемые в рамках системы, при которых используется информация. Процесс представляется в модели только там, где информация, поступающая на вход, каким-либо образом обрабатывается или преобразуется таким образом, что информация на выходе отличается от информации на входе.

125

The activity may involve capturing information about something that the organization is interested in, such as a customer or a customer's maintenance call. It may be concerned with recording changes to this information, a change in a customer's address for example. It may require calculations to be carried out, such as the quantity left in stock following the allocation of stock items to a customer's job; or it may involve validating information, such as checking that faulty equipment is covered by a maintenance contract.

Діяльність може включати збір інформації про те, що зацікавило організацію, наприклад, про клієнта або дзвінок технічного обслуговування клієнта. Це може стосуватися запису змін до цієї інформації, наприклад, зміни адреси клієнта. Це може вимагати проведення розрахунків, таких як кількість, що залишилася на складі після розподілу предметів запасу на роботу замовника; або це може включати перевірку інформації, таку як перевірка того, чи несправне обладнання охоплюється договором на технічне обслуговування.

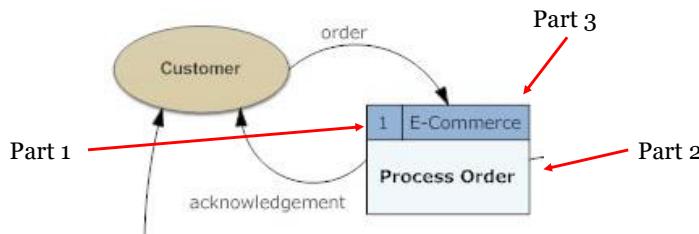
126

Эта деятельность может включать сбор информации о чем-то, в чем заинтересована организация, например о клиенте или обращении клиента в службу поддержки. Это может быть связано с записью изменений в этой информации, например, изменения адреса клиента. Может потребоваться выполнение расчетов, например количества, остающегося на складе после распределения складских единиц для работы клиента; или это может включать проверку информации, например, проверку того, что неисправное оборудование покрывается контрактом на техническое обслуживание.

127

Processes are depicted with a box, divided into three parts.

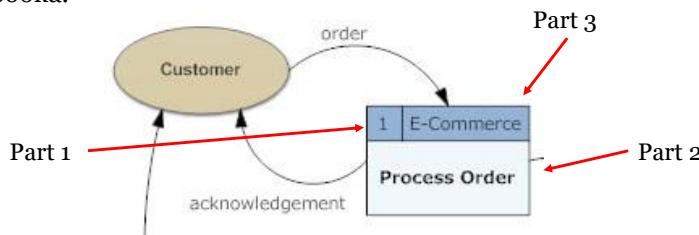
1. The top left-hand box contains the process number. This is simply for identification and reference purposes, and does not in any way imply priority and sequence.
2. The main part of the box is used to describe the process itself, giving the processing performed on the data it receives.
3. The smaller rectangular box at the top of the process is used to indicate the location where the processing takes place.



128

Процеси зображені прямокутниками, розділеними на три частини.

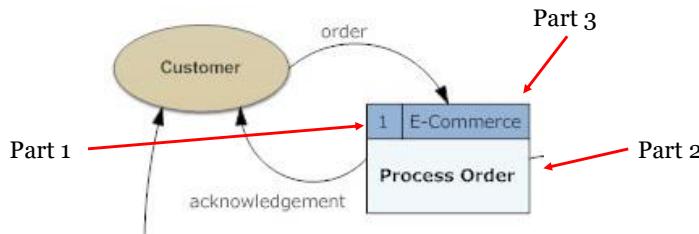
1. У верхньому лівому прямокутнику є номер процесу для ідентифікації та довідкових цілей і жодним чином не означає пріоритету та послідовності.
2. Основна частина прямокутника використовується для опису самого процесу, надаючи обробку, виконану за отриманими даними.
3. Менший прямокутник у верхній частині процесу використовується для позначення місця, де відбувається обробка.



129

Процессы изображены прямоугольником, разделенным на три части.

1. Верхнее левое поле содержит номер процесса для идентификации и справочных целей и никоим образом не подразумевает приоритета и последовательности.
2. Основная часть блока используется для описания самого процесса, показывая обработку, выполняемую над данными, которые он получает.
3. Меньшее прямоугольное поле в верхней части процесса используется для указания места, где происходит обработка.



130

The **rules for processes** are:

- Process names should be an imperative verb specific to the activity in question, followed by a pithy and meaningful description of the object of the activity. Create Contract, or Schedule Jobs, as opposed to using very general or non-specific verbs, such as Update Customer Details or Process Customer Call.
- Processes may not act as data sources or sinks. Data flowing into a process must have some corresponding output, which is directly related to it. Similarly, data-flowing out of a process must have some corresponding input to which it is directly related.
- Normally only processes that transform system data are shown on data-flow diagrams. Only where an enquiry is central to the system is it included.
- Where a process is changing data from a data store, only the changed information flow to the data store (and not the initial retrieval from the data store) is shown on the diagram.
- Where a process is passing information from a data store to an external entity or another process, only the flow from the data store to the process is shown on the diagram.

131

Правилами процесів є:

- Назви процесів повинні бути імперативним дієсловом, специфічним для даної діяльності, за яким слідує суттєвий та змістовний опис об'єкта діяльності. «Створити контракт» або «запланувати завдання», на відміну від використання дуже загальних або неспецифічних дієслів, таких як «оновлення даних про клієнта» або «обробка дзвінків клієнтів».
- Процеси не можуть виконувати роль джерел даних або поглиначів. Дані, що надходять у процес, повинні мати певний відповідний вихід, який безпосередньо з ним пов'язаний. Analogічно, дані, що витикають із процесу, повинні мати певний відповідний вхід, до якого він безпосередньо пов'язаний.
- Зазвичай на діаграмах потоків даних відображаються лише процеси, які перетворюють системні дані. Лише там, де запит є центральним для системи, він включається.
- Якщо процес змінює дані із сховища даних, на схемі відображається лише змінений потік інформації до сховища даних (а не початковий пошук із сховища даних).
- Якщо процес передає інформацію із сховища даних зовнішньому об'єкту або іншому процесу, на схемі відображається лише потік із сховища даних до процесу.

132

Правила для процесов:

- Имена процессов должны быть императивным глаголом, специфичным для рассматриваемой деятельности, за которым следует содержательное и содержательное описание объекта действия. «Создать контракт» или «запланировать задания», в отличие от использования очень общих или неспецифических глаголов, таких как «Обновить сведения о клиенте» или «Обработка звонка от клиента».
- Процессы не могут действовать как источники или приемники данных. Данные, поступающие в процесс, должны иметь какой-либо соответствующий вывод, который непосредственно связан с ним. Точно так же поток данных из процесса должен иметь соответствующие входные данные, с которыми он непосредственно связан.
- Обычно на диаграммах потоков данных отображаются только процессы, преобразующие системные данные. Он включается только в том случае, если запрос является центральным для системы.
- Если процесс изменяет данные из хранилища данных, на диаграмме отображается только измененный поток информации в хранилище данных (а не первоначальное извлечение из хранилища данных).
- Если процесс передает информацию из хранилища данных внешнему объекту или другому процессу, на диаграмме показан только поток из хранилища данных в процесс.

133

Data-flows

A data-flow represents a package of information flowing between two objects in the data-flow diagram. Data-flows are used to model the flow of information into the system, out of the system, and between elements within the system.

Потік даних являє собою пакет інформації, що протікає між двома об'єктами на схемі потоку даних. Потоки даних використовуються для моделювання потоку інформації в систему, поза системою та між елементами всередині системи.

Поток данных представляет собой пакет информации, перемещающийся между двумя объектами на диаграмме потока данных. Потоки данных используются для моделирования потока информации в систему, из системы и между элементами внутри системы.

134

Occasionally, a data-flow is used to illustrate information flows between two external entities, which is, strictly speaking, outside of the system boundaries. However, knowledge of the transfer of information between external entities can sometimes aid understanding of the system under investigation, in which case it should be depicted on the diagram.

Іноді потік даних використовується для ілюстрування інформаційних потоків між двома зовнішніми сущностями, який, строго кажучи, знаходиться поза межами системи. Однак знання про передачу інформації між зовнішніми сущностями іноді може допомогти зрозуміти систему, що досліджується, і в цьому випадку вона повинна бути зображена на діаграмі.

Иногда поток данных используется для иллюстрации информационных потоков между двумя внешними сущностями, которые, строго говоря, находятся за пределами системы. Однако знание передачи информации между внешними сущностями иногда может помочь понять исследуемую систему, и в этом случае ее следует отобразить на диаграмме.

135

A data-flow is depicted on the diagram as a directed line drawn between the source and recipient of the data-flow, with the arrow depicting the direction of flow.

The directed line is labelled with the data-flow name, which briefly describes the information contained in the flow.

Data-flows between external entities are depicted by dashed, rather than unbroken, lines.

Потік даних зображеній на діаграмі у вигляді спрямованої лінії, проведеної між джерелом і одержувачем потоку даних, зі стрілкою, що відображає напрямок потоку.

Напрямлена лінія позначена назвою потоку даних, яка коротко описує інформацію, що міститься в потоці.

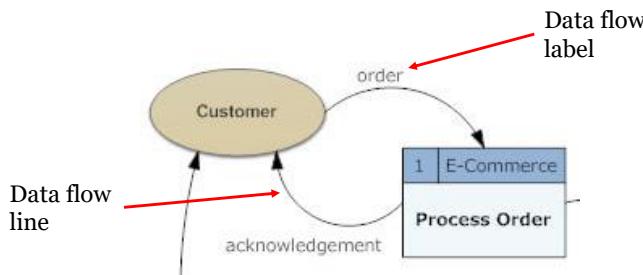
Потоки даних між зовнішніми сущностями зображуються пунктирними, а не безперервними лініями.

136

Поток данных изображен на схеме как направленная линия, проведенная между источником и получателем потока данных, со стрелкой, показывающей направление потока.

Направленная линия помечена именем потока данных, которое кратко описывает информацию, содержащуюся в потоке.

Потоки данных между внешними сущностями изображаются пунктирными, а не непрерывными линиями.



The **rules for drawing data-flows** are:

- Information always flows to or from a process; the other end of the flow may be an external entity, a data store or another process. An occasional exception to this rule is a data-flow between two external entities.
- Data stores may not be directly linked by data-flows; information is transformed from one stored state to another via a process.
- Information may not flow directly from a data store to an external entity, nor may it flow from an external entity directly to a data store. This communication and receipt of information stored in the system always takes place via a process.
- The sources (where data of interest to the system is generated without any corresponding input) and sinks (where data is swallowed up without any corresponding output) of data-flows are always represented by external entities.
- When something significant happens to a data-flow, as a result of a process acting on it, the label of the resulting data-flow should reflect its transformed status. For example, “Telephoned Service Call” becomes “Service Call Form” once it has been logged.

Правилами зображення потоків даних є:

- Інформація завжди надходить до або з процесу; інший кінець потоку може бути зовнішньою сутністю, накопичувачем даних або іншим процесом. Випадковим винятком із цього правила є потік даних між двома зовнішніми сутностями.
- Накопичувачі даних не можуть бути безпосередньо пов’язані потоками даних; інформація перетворюється з одного збереженого стану в інший за допомогою процесу.
- Інформація може не надходити безпосередньо із накопичувача даних до зовнішньої сутності, а також не може надходити від зовнішньої сутності безпосередньо до накопичувача даних. Це сплкування та отримання інформації, що зберігається в системі, завжди відбувається за допомогою процесу.
- Джерела (де дані, що цікавлять систему, генеруються без будь-якого відповідного входу) та поглиначі (де дані заглиблюються без будь-якого відповідного виводу) потоків даних завжди представлені зовнішніми сутностями.
- Коли з потоком даних відбувається щось суттєве, як результат процесу, що діє на нього, мітка результуючого потоку даних повинна відображати його трансформований статус. Наприклад, “Телефонний дзвінок у службу” стає “Формою дзвінка в службу” після реєстрації.

Правила изображения потоков данных:

- Информация всегда поступает в процесс или из него; другой конец потока может быть внешней сущностью, накопителем данных или другим процессом. Случайным исключением из этого правила является поток данных между двумя внешними сущностями.
- Накопители данных не могут быть напрямую связаны потоками данных; информация преобразуется из одного сохраненного состояния в другое посредством процесса.
- Информация не может поступать напрямую из накопителя данных во внешний объект, а также не может передаваться из внешней сущности напрямую в накопитель данных. Эта связь и получение информации, хранящейся в системе, всегда происходит через процесс.
- Источники (где данные, представляющие интерес для системы, генерируются без какого-либо соответствующего ввода) и приемники (где данные поглощаются без какого-либо соответствующего вывода) потоков данных всегда представлены внешними сущностями.
- Когда что-то существенное происходит с потоком данных в результате воздействия на него процесса, метка результирующего потока данных должна отражать его преобразованный статус. Например, «Телефонный вызов службы» становится «Формой вызова службы» после регистрации.

Data stores

A data store is a place where data is stored and retrieved within the system. This may be a file, a catalogue or reference list, a log book such as the Job Book, and so on.

Накопичувач даних - це місце, де дані зберігаються та отримуються в системі. Це може бути файл, каталог або список посилань, журнал тощо.

Накопитель данных - это место, где данные хранятся и извлекаются в системе. Это может быть файл, каталог или список литературы, журнал, и т. д.

141

A data store is represented in the data-flow diagram by a long rectangle, containing two locations.

The small left-hand box is used for the identifier, which comprises a numerical reference prefixed by a letter.

The main area of the rectangle is labelled with the name of the data store. Brief names are chosen to reflect the content of the data store.

Накопичувач даних представлений на діаграмі потоків даних довгим прямокутником, що містить два місця.

Маленький лівий блок використовується для ідентифікатора, який містить цифрове посилання, що має префікс-літеру.

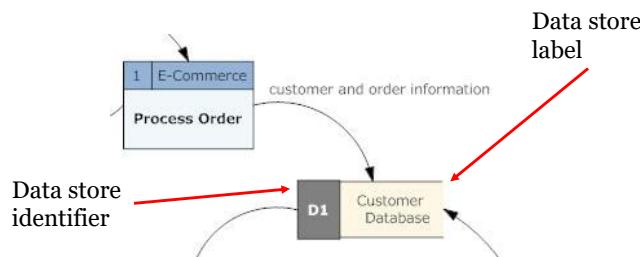
Основна область прямокутника позначена назвою накопичувача даних. Короткі назви вибираються для відображення змісту накопичувача даних.

142

Накопитель данных представлен на диаграмме потока данных длинным прямоугольником, содержащим два местоположения.

Маленькое левое поле используется для идентификатора, который содержит числовую ссылку с буквенным префиксом.

Основная область прямоугольника помечена именем накопителя данных. Краткие имена выбраны для отражения содержимого накопителя данных.



143

The **rules for representing data stores** are:

- One convention that could be used is to determine the letter identifying a data store by the store's nature.
- “M” is used where a manual data store is being depicted.
- “D” is used where it is a computer based data store.
- “T” is used where a temporary data store is being represented.
- Data stores may not act as data sources or sinks. Data flowing into a data store must have some corresponding output, and vice versa.
- Because of their function in the storage and retrieval of data, data stores often provide input dataflows to receive output flows from a number of processes. For the sake of clarity and to avoid crisscrossing of data-flows in the data-flow diagram, a single data store may be included in the diagram at more than one point. Where the depiction of a data store is repeated in this way, this is signified by drawing a second vertical line along the left-hand edge of the rectangle for each occurrence of the data store.

144

Правилами представлення накопичувачів даних є:

- Однією з домовленостей, яка може бути використана, є визначення букви, що ідентифікує накопичувач даних, за характером сховища.
- “M” використовується там, де зображене накопичувач даних, який здійснюється вручну.
- “D” використовується там, де це комп'ютерний накопичувач даних.
- “T” використовується там, де представлено тимчасовий накопичувач даних.
- Накопичувачі даних не можуть виступати в якості джерел даних або приймачів. Дані, що надходять у накопичувач даних, повинні мати певний відповідний вихід, і навпаки.
- Через свою функцію зберігання та отримання даних накопичувачі даних часто забезпечують вхідні потоки даних для отримання вихідних потоків з ряду процесів. Задля ясності та уникнення перехрещення потоків даних на діаграмі потоків даних, один накопичувач даних може бути включено в діаграму більш ніж в одній точці. Якщо зображення накопичувача даних повторюється таким чином, це позначається накресленням другої вертикальної лінії вздовж лівого краю прямокутника для кожного входження накопичувача даних.

145

Правила представления накопителей данных:

- Одно из соглашений, которое может быть использовано, - определить букву, идентифицирующую накопитель данных, по его характеру.
- «М» используется, когда отображается ручной накопитель данных.
- «Д» используется там, где это компьютерный накопитель данных.
- «Т» используется там, где представлено временный накопитель данных.
- Накопители данных не могут выступать в качестве источников или приемников данных. Данные, поступающие в накопитель данных, должны иметь соответствующий вывод, и наоборот.
- Из-за своей функции в хранении и извлечении данных накопители данных часто предоставляют входные потоки данных для приема выходных потоков от ряда процессов. Для ясности и во избежание пересечения потоков данных в диаграмме потоков данных один накопитель данных может быть включен в диаграмму более чем в одной точке. Если изображение накопителя данных повторяется таким образом, это обозначается проведением второй вертикальной линии вдоль левого края прямоугольника для каждого экземпляра накопителя данных.

146

External entities

External entities are entities outside of the system boundary which interact with the system, in that they send information into the system or receive information from it. External entities may be external to the whole organization or just external to the application area where users' activities are not directly supported by the system under investigation. External entities are often referred to as sources and sinks. All information represented within the system is sourced initially from an external entity. Data can leave the system only via an external entity.

Зовнішні сутності - це сутності поза межами системи, які взаємодіють із системою, оскільки надсилають інформацію в систему або отримують від неї інформацію. Зовнішні сутності можуть бути зовнішніми для всієї організації або лише зовнішніми для сфери застосування, де діяльність користувачів не підтримується безпосередньо системою, що досліджується. Зовнішні сутності часто називають джерелами та поглиначами. Вся інформація, представлена в системі, спочатку надходить від зовнішньої сутності. Дані можуть залишати систему лише через зовнішню сутність.

147

Внешние сущности - это объекты за пределами системы, которые взаимодействуют с системой в том смысле, что они отправляют информацию в систему или получают информацию от нее. Внешние сущности могут быть внешними для всей организации или просто внешними по отношению к области приложения, где действия пользователей не поддерживаются непосредственно исследуемой системой. Внешние сущности часто называют источниками и стоками. Вся информация, представленная в системе, изначально поступает от внешней сущности. Данные могут покинуть систему только через внешнюю сущность.

148

External entities are represented on the diagram as ovals drawn outside of the system boundary, containing the entity name and an identifier.

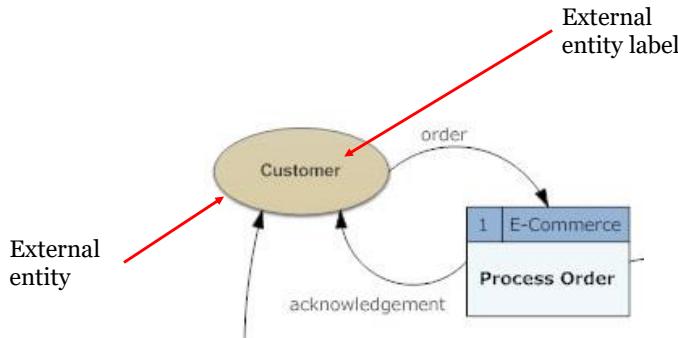
Names consist of a singular noun describing the role of the entity. Above the label, a lower case letter is used as the identifier for reference purposes.

Зовнішні сутності представлені на діаграмі як овали, виведені за межі системи, що містять ім'я сутності та ідентифікатор. Назви складаються з іменника однини, що описує роль сутності. Над ярликом, як ідентифікатор для довідкових цілей використовується мала літера.

Внешние сущности представлены на схеме в виде овалов, нарисованных за пределами границы системы, содержащих имя объекта и идентификатор.

Имена состоят из существительного в единственном числе, описывающего роль сущности. Над этикеткой в справочных целях используется строчная буква в качестве идентификатора.

149



150

The **rules associated with external entities** are:

- Each external entity must communicate with the system in some way, thus there is always a dataflow between an external entity and a process within the system.
- External entities may provide and receive data from a number of processes. It may be appropriate, for the sake of clarity and to avoid crisscrossing of data flows, to depict the same external entity at a number of points on the diagram. Where this is the case, a line is drawn across the left corner of the ellipse, for each occurrence of the external entity on the diagram.

Правила, пов'язані із зовнішніми сутностями:

- Кожна зовнішня сутність повинна якимось чином взаємодіяти із системою, отже, завжди існує потік даних між зовнішньою сутністю та процесом у системі.
- Зовнішні сутності можуть надавати та отримувати дані з ряду процесів. Для наочності та уникнення перехресних потоків даних може бути доцільним зобразити одну і ту ж зовнішню сутність у кількох точках на діаграмі. Там, де це так, лінія проводиться через лівий кут еліпса для кожного входження зовнішньої сутності на діаграмі.

151

Правила, связанные с внешними сущностями:

- Каждая внешняя сущность должна каким-то образом взаимодействовать с системой, поэтому всегда существует поток данных между внешней сущностью и процессом внутри системы.
- Внешние сущности могут предоставлять и получать данные от ряда процессов. Для ясности и во избежание пересечения потоков данных может оказаться целесообразным изобразить одну и ту же внешнюю сущность в нескольких точках диаграммы. В этом случае линия проходит через левый угол эллипса для каждого появления внешней сущности на диаграмме.

152

Context diagrams

The context diagram is used to establish the context and boundaries of the system to be modelled: which things are inside and outside of the system being modelled, and what is the relationship of the system with these external entities.

A context diagram, sometimes called a level 0 data-flow diagram, is drawn in order to define and clarify the boundaries of the software system. It identifies the flows of information between the system and external entities. The entire software system is shown as a single process.

The process of establishing the analysis framework by drawing and reviewing the context diagram inevitably involves some initial discussions with users regarding problems with the existing system and the specific requirements for the new system.

Having agreed on the framework, the detailed investigation of the current system must be planned. This involves identifying how each of the areas included within the scope will be investigated. This could be by interviewing users, providing questionnaires to users or clients, studying existing system documentation and procedures, observation and so on. Key users are identified and their specific roles in the investigation are agreed upon.

Контекстні діаграми

Контекстна діаграма використовується для встановлення контексту та меж системи, що підлягає моделюванню: які речі знаходяться всередині та за межами системи, що моделюється, та який взаємозв'язок системи з цими зовнішніми сутностями.

Контекстну діаграму, яку іноді називають діаграмою потоку даних рівня 0, складають для того, щоб визначити та уточнити межі програмної системи. Вона визначає потоки інформації між системою та зовнішніми сутностями. Вся система програмного забезпечення показана як єдиний процес.

Процес створення основи для аналізу шляхом складання та перегляду контекстної діаграми неминуче передбачає деякі початкові обговорення з користувачами щодо проблем із існуючою системою та конкретних вимог до нової системи.

Погодивши основи, слід запланувати детальне дослідження поточної системи. Це передбачає визначення того, як буде досліджуватися кожна із сфер, що входять до сфери застосування. Це може бути шляхом опитування користувачів, надання анкет користувачам або клієнтам, вивчення існуючої системної документації та процедур, спостереження тощо. Визначаються ключові користувачі та узгоджується їх конкретна роль у дослідженні.

Контекстная диаграмма

Контекстная диаграмма используется для установления контекста и границ моделируемой системы: какие объекты находятся внутри и вне моделируемой системы, и каковы отношения системы с этими внешними объектами.

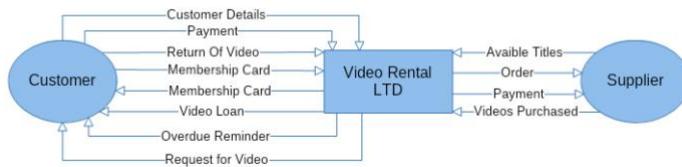
Контекстная диаграмма, иногда называемая диаграммой потоков данных уровня 0, рисуется для определения и уточнения границ программной системы. Она определяет потоки информации между системой и внешними сущностями. Вся программная система показана как единый процесс.

Процесс создания структуры анализа путем рисования и просмотра контекстной диаграммы неизбежно включает в себя некоторые начальные обсуждения с пользователями проблем с существующей системой и конкретных требований к новой системе.

После согласования структуры необходимо запланировать детальное исследование существующей системы. Это включает определение того, как будет исследоваться каждая из областей, включенных в объем. Это может быть опрос пользователей, предоставление анкет для пользователей или клиентов, изучение существующей системной документации и процедур, наблюдение и т. д. Выявляются ключевые пользователи и согласовываются их конкретные роли в расследовании.

155

Video Rental Data Flow Diagram (Context Diagram)



156

In order to produce the context diagram and agree on system scope, the following must be identified:

- external entities
- data-flows

You may find the following steps useful:

1. Identify data-flows by listing the major documents and information flows associated with the system, including forms, documents, reference material, and other structured and unstructured information (emails, telephone conversations, information from external systems, etc.).
2. Identify external entities by identifying sources and recipients of the data-flows, which lie outside of the system under investigation. The actors in any use case models you have created may often be external entities.
3. Draw and label a process box representing the entire system.
4. Draw and label the external entities around the outside of the process box.
5. Add the data-flows between the external entities and the system box. Where documents and other packets of information flow entirely within the system, these should be ignored from the point of view of the context diagram – at this stage they are hidden within the process box.

This system boundary and details depicted in the context diagram should then be discussed (and updated if necessary) in consultation with your customers until an agreement is reached.

Having defined the system boundary and scope, the areas for investigation will be determined, and appropriate techniques for investigating each area will need to be decided.

Для того, щоб скласти контекстну діаграму та узгодити сферу застосування системи, слід визначити наступне:

- зовнішні сутності
- потоки даних

Можуть виявитися корисними такі кроки:

1. Визначте потоки даних, перерахувавши основні документи та інформаційні потоки, пов'язані із системою, включаючи форми, документи, довідковий матеріал та іншу структуровану та неструктуровану інформацію (електронні листи, телефонні розмови, інформація із зовнішніх систем тощо).
2. Ідентифікуйте зовнішні сутності шляхом ідентифікації джерел та одержувачів потоків даних, які лежать поза системою, що досліджується. Дійові особи в будь-яких створених вами моделях варіантів використання часто можуть бути зовнішніми сутностями.
3. Намалюйте та позначте блок процесу, що представляє всю систему.
4. Намалюйте та позначте зовнішні сутності навколо блока процесу.
5. Додайте потоки даних між зовнішніми сутностями та блоком системи. Там, де документи та інші пакети інформації повністю протікають усередині системи, їх слід ігнорувати з точки зору контекстної діаграми - на цьому етапі вони приховані у блоці процесу.

Потім ці межі системи та деталі, зображені на контекстній діаграмі, слід обговорити (та, за необхідності, оновити) у консультації зі своїми клієнтами досягнення згоди.

Визначивши межі та сферу застосування системи, необхідно буде визначити зони для дослідження та відповідні методи дослідження кожної області.

Чтобы создать контекстную диаграмму и согласовать объем системы, необходимо определить следующее:

- внешние сущности
- потоки данных

Могут быть полезны следующие шаги:

- Идентифицируйте потоки данных, перечисляя основные документы и информационные потоки, связанные с системой, включая формы, документы, справочные материалы и другую структурированную и неструктурную информацию (электронные письма, телефонные разговоры, информацию из внешних систем и т. Д.).
- Идентифицируйте внешние сущности путем определения источников и получателей потоков данных, которые находятся за пределами исследуемой системы. Действующие лица в любых созданных вами моделях вариантов использования часто могут быть внешними объектами.
- Нарисуйте и пометьте блок процесса, представляющий всю систему.
- Нарисуйте и пометьте внешние сущности вокруг блока процесса.
- Добавьте потоки данных между внешними сущностями и блоком системы. Если документы и другие пакеты информации полностью перемещаются внутри системы, их следует игнорировать с точки зрения контекстной диаграммы - на этом этапе они скрыты в блоке процесса.

Эти границы системы и детали, изображенные на контекстной диаграмме, должны затем обсуждаться (и при необходимости обновляться) в консультации с вашими клиентами, пока не будет достигнуто соглашение.

После определения границ системы и области действия необходимо будет определить области исследования и соответствующие методы исследования каждой области.

159

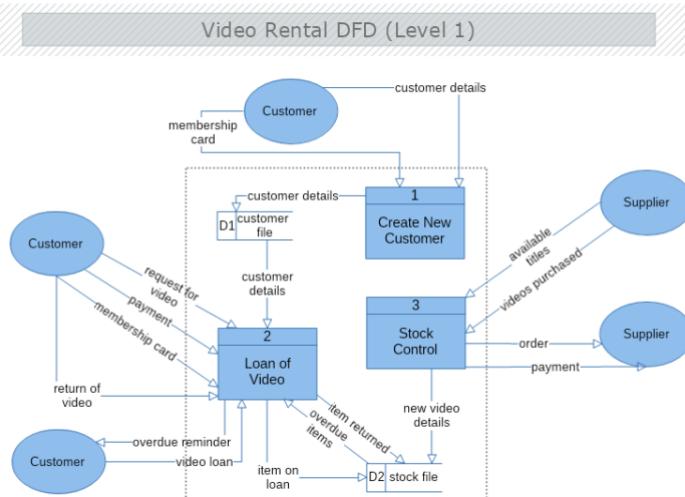
Level 1 data-flow diagrams

As described previously, context diagrams (level 0 DFDs) are diagrams where the whole system is represented as a single process. A level 1 DFD notates each of the main sub-processes that together form the complete system. We can think of a level 1 DFD as an “exploded view” of the context diagram.

Як було описано раніше, контекстними діаграмами (DFD рівня 0) є діаграми, де вся система представлена як єдиний процес. DFD рівня 1 містить кожен з основних підпроцесів, які разом утворюють цілісну систему. Ми можемо розглядати DFD рівня 1 як «розгорнутий вигляд» контекстної діаграми.

Как описано ранее, контекстные диаграммы (DFD уровня 0) представляют собой диаграммы, на которых вся система представлена как единий процесс. DFD уровня 1 включает каждый из основных подпроцессов, которые вместе образуют полную систему. Мы можем думать о DFD уровня 1 как о «развернутом виде» контекстной диаграммы.

160



161

Notice that the external entities have been included on this diagram, but outside of the rectangle that represents the boundary of this diagram (i.e., the system boundary). It is not necessary to always show the external entities on level 1 (or lower) DFDs, however you may wish to do so to aid clarity and understanding. We can see that on this level 1 DFD there are a number of data stores, and data-flows between processes and the data stores.

It is important to notice that the same data-flows to and from the external entities appear on this level 1 diagram and the level 0 context diagram. Each time a process is expanded to a lower level, the lower level diagram must show all the same data-flows into, and out of the higher level process it expands.

Зверніть увагу, що зовнішні сущності були включені на цю діаграму, але поза прямокутником, який представляє межу цієї діаграми (тобто межу системи). Не обов'язково завжди показувати зовнішні сущності на рівнях 1 (або нижчих) DFD, однак ви можете зробити це, щоб досягти ясності та розуміння.

Ми бачимо, що на цьому рівні 1 DFD існує ряд накопичувачів даних та потоків даних між процесами та сховищами даних.

Важливо зауважити, що однакові потоки даних до і від зовнішніх сущностей з'являються на цій діаграмі рівня 1 та на контекстній діаграмі рівня 0.

Кожного разу, коли процес розширяється до нижчого рівня, діаграма нижчого рівня повинна відображати всі ті самі потоки даних з та до процесу вищого рівня, який вона розширяє.

162

Обратите внимание, что внешние сущности были включены в эту диаграмму, но за пределами прямоугольника, который представляет границу этой диаграммы (то есть границу системы). Необязательно всегда показывать внешние сущности на уровне 1 (или ниже) DFD, однако это можно сделать для большей ясности и понимания.

Мы видим, что на этом уровне 1 DFD есть несколько накопителей данных и потоки данных между процессами и хранилищами данных.

Важно отметить, что одни и те же потоки данных к внешним сущностям и от них появляются на этой диаграмме уровня 1 и контекстной диаграмме уровня 0. Каждый раз, когда процесс расширяется до более низкого уровня, диаграмма более низкого уровня должна показывать все те же потоки данных в процесс более высокого уровня и из процесса, который она расширяет.

The following steps are suggested to aid the construction of Level 1 DFD:

1. Identify processes. Each data-flow into the system must be received by a process. For each dataflow into the system examine the documentation about the system and talk to the users to establish a plausible process of the system that receives the data-flow. Each process must have at least one output data-flow. Each output data-flow of the system must have been sent by a process; identify the processes that sends each system output.
2. Draw the data-flows between the external entities and processes.
3. Identify data stores by establishing where documents / data needs to be held within the system. Add the data stores to the diagram, labelling them with their local name or description.
4. Add data-flows flowing between processes and data stores within the system. Each data store must have at least one input data-flow and one output data-flow (otherwise data may be stored, and never used, or a store of data must have come from nowhere). Ensure every data store has input and output data-flows to system processes. Most processes are normally associated with at least one data store.
5. Check diagram. Each process should have an input and an output. Each data store should have an input and an output. Check the system details so see if any process appears to be happening for no reason (i.e., some “trigger” data-flow is missing, that would make the process happen).

Для сприяння побудові DFD рівня 1 пропонуються наступні кроки:

1. Визначте процеси. Кожен потік даних у систему повинен прийматися процесом. Для кожного потоку даних у систему вивчіть документацію про систему та поговоріть з користувачами, щоб встановити правдоподібний процес системи, яка приймає потік даних. Кожен процес повинен мати принаймні один вихідний потік даних. Кожен вихідний потік даних системи повинен бути надісланий процесом; визначте процеси, які надсилає кожен вихід системи.
2. Намалюйте потоки даних між зовнішніми сутностями та процесами.
3. Визначте накопичувачі даних, встановивши, де документи / дані потрібно зберігати в системі. Додайте накопичувачі даних на діаграму, позначивши їх короткою назвою або описом.
4. Додайте потоки даних, що протікають між процесами та накопичувачами даних у системі. Кожен накопичувач даних повинен мати принаймні один вихідний потік даних і один вихідний потік даних (інакше дані можуть зберігатися і ніколи не використовуватися, або накопичувач даних повинен надходити з нізвідки). Переконайтесь, що кожен накопичувач даних має вхідні та вихідні потоки даних для процесів системи. Більшість процесів, як правило, пов'язані принаймні з одним накопичувачем даних.
5. Перевірте схему. Кожен процес повинен мати вхідні та вихідні дані. Кожне складове даних повинно мати вхідні та вихідні дані. Перевірте системні деталі, щоб побачити, чи не існує якийсь процес без причини для виконання (тобто відсутній деякий „тригерний“ потік даних, який би змусив процес відбутися).

Предлагаются следующие шаги, которые помогут построить DFD уровня 1:

1. Определите процессы. Каждый поток данных в систему должен быть получен процессом. Для каждого потока данных в систему изучите документацию о системе и поговорите с пользователями, чтобы установить правдоподобный процесс в системе, которая получает поток данных. У каждого процесса должен быть хотя бы один выходной поток данных. Каждый выходной поток данных системы должен быть отправлен процессом; определите процессы, которые отправляют каждый системный вывод.
2. Нарисуйте потоки данных между внешними сущностями и процессами.
3. Идентифицируйте накопители данных, установив, где в системе должны храниться документы / данные. Добавьте накопители данных на диаграмму, пометив их кратким именем или описанием.
4. Добавьте потоки данных, текущие между процессами и накопителями данных в системе. Каждый накопитель данных должен иметь по крайней мере один входной поток данных и один выходной поток данных (в противном случае данные могут храниться и никогда не использоваться, или накопитель данных может быть получен из ниоткуда). Убедитесь, что каждый накопитель данных имеет потоки входных и выходных данных для процессов системы. Большинство процессов обычно связаны как минимум с одним накопителем данных.
5. Проверить схему. У каждого процесса должны быть вход и выход. Каждый накопитель данных должен иметь вход и выход. Проверьте сведения о системе, чтобы увидеть, не происходит ли какой-либо процесс без причины (т.е. отсутствует какой-то «третичный» поток данных, который заставил бы процесс произойти).

Constructing level 2 DFDs

We have already seen how a level o context diagram can be decomposed (exploded) into a level 1 DFD. In DFD modeling terms we talk of the context diagram as the “parent” and the level 1 diagram as the “child”.

This same process can be applied to each process appearing within a level 1 DFD. A DFD that represents a decomposed level 1 DFD process is called a level 2 DFD. There can be a level 2 DFD for each process that appears in the level 1 DFD.

Ми вже бачили, як контекстну діаграму рівня 0 можна декомпозувати (розкладти) до DFD рівня 1. У термінах моделювання DFD ми говоримо про контекстну діаграму як про «батьківську», а діаграму рівня 1 як про «дочірню».

Цей самий процес можна застосувати до кожного процесу, що з'являється в DFD рівня 1. DFD, який представляє розкладений процес DFD рівня 1, називається DFD рівня 2. Для кожного процесу, який відображається в DFD рівня 1, може бути DFD рівня 2.

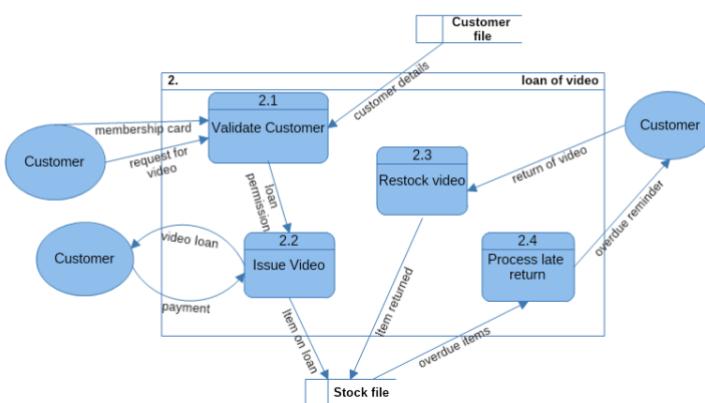
167

Мы уже видели, как контекстная диаграмма уровня О может быть декомпозирована (разложена) до DFD уровня 1. В терминах моделирования DFD мы говорим о контекстной диаграмме как о «родительской», а о диаграмме уровня 1 как о «дочерней».

Этот же процесс можно применить к каждому процессу, появляющемуся в DFD уровня 1. DFD, представляющий разложенный процесс DFD уровня 1, называется DFD уровня 2. Для каждого процесса, который появляется в DFD уровня 1, может быть DFD уровня 2.

168

Video Rental DFD (Level 2 - Loan of Video)



169

Note, that every data-flow into and out of the parent process must appear as part of the child DFD. The numbering of processes in the child DFD is derived from the number of the parent process – so all processes in the child DFD of process 2, will be called 2.X (where X is the arbitrary number of the process on the level 2 DFD). Also there are no new data-flows into or out of this diagram – this kind of data-flow validation is called balancing.

Зверніть увагу, що кожен потік даних у батьківський процес і з нього повинен відображатися як частина дочірнього DFD. Нумерація процесів у дочірньому DFD виводиться з числа батьківського процесу - тому всі процеси в дочірньому DFD процесу 2 будуть називатися 2.X (де X - довільне число процесу на рівні 2 DFD). Крім того, немає нових потоків даних, що входять у цю діаграму або виходять з неї - такий тип перевірки потоку даних називається балансуванням.

Обратите внимание, что каждый поток данных в родительский процесс и из него должен появляться как часть дочернего DFD. Нумерация процессов в дочернем DFD определяется номером родительского процесса, поэтому все процессы в дочернем DFD процесса 2 будут называться 2.X (где X - произвольный номер процесса на уровне 2 DFD.). Также нет никаких новых потоков данных в эту диаграмму или из нее - этот вид проверки потока данных называется балансировкой.

170

The level 1 data-flow diagram provides an overview of the system. As the software engineers' understanding of the system increases it becomes necessary to expand most of the level 1 processes to a second or even third level in order to depict the detail within it. Decomposition is applied to each process on the level 1 diagram for which there is enough detail hidden within the process. Each process on the level 2 diagrams also needs to be checked for possible decomposition, and so on.

Діаграма потоку даних рівня 1 забезпечує огляд системи. Зі збільшенням розуміння інженерами-програмістами системи стає необхідним розширити більшість процесів рівня 1 до другого або навіть третього рівня, щоб зобразити деталі всередині неї. Розкладання застосовується до кожного процесу на діаграмі рівня 1, для якого в процесі є достатня кількість деталей. Кожен процес на діаграмах рівня 2 також потрібно перевіряти на можливе розкладання тощо.

Диаграмма потока данных уровня 1 дает обзор системы. По мере того, как инженеры-программисты понимают систему, возникает необходимость расширить большинство процессов уровня 1 до второго или даже третьего уровня, чтобы отобразить детали внутри него. Декомпозиция применяется к каждому процессу на диаграмме уровня 1, для которого достаточно деталей, скрытых внутри процесса. Каждый процесс на диаграммах уровня 2 также необходимо проверить на предмет возможной декомпозиции и т. д.

171

A process box that cannot be decomposed further is marked with an asterisk in the bottom right hand corner. A brief narrative description of each bottom-level process should be provided with the dataflow diagrams to complete the documentation of the data-flow model. These make up part of the process definitions which should be supplied with the DFD.

Поле процесу, яке не можна розкласти далі, позначено зірочкою в правому нижньому куті. Короткий опис кожного процесу нижчого рівня повинен бути забезпечений для діаграм потоків даних для завершення документації моделі потоку даних. Вони складають частину визначень процесу, які повинні надаватися разом з DFD.

Блок процесса, который не может быть подвергнут дальнейшей декомпозиции, отмечен звездочкой в правом нижнем углу. Краткое описание каждого процесса нижнего уровня должно сопровождать диаграммы потоков данных для завершения документации модели потока данных. Они составляют часть определений процессов, которые должны поставляться с DFD.

172

Each process on the level 1 diagram is investigated in more detail, to give a greater understanding of the activities and data-flows. Normally processes are decomposed where:

- There are more than six data-flows around the process.
- The process name is complex or very general which indicates that it incorporates a number of activities.

Кожен процес на діаграмі рівня 1 досліджується більш детально, щоб дати глибше розуміння діяльності та потоків даних. Зазвичай процеси розкладаються там, де:

- Навколо процесу існує більше шести потоків даних.
- Назва процесу є складною або дуже загальною, що вказує на те, що вона включає ряд видів діяльності.

Каждый процесс на диаграмме уровня 1 исследуется более подробно, чтобы лучше понять действия и потоки данных. Обычно процессы раскладываются в случае, если:

- Вокруг процесса более шести потоков данных.
- Название процесса является сложным или очень общим, что указывает на то, что он включает в себя ряд действий.

173

BUSINESS RULES

174

Every organization operates according to an extensive set of policies, laws, and industry standards. Industries such as banking, aviation, and medical device manufacture must comply with volumes of government regulations. Such controlling principles are known collectively as **business rules** or business logic. Business rules often are enforced through manual implementation of policies and procedures. In many cases, though, software applications also need to enforce these rules.

Кожна організація працює відповідно до широкого набору політик, законів та галузевих стандартів. Такі галузі, як банківська справа, авіація та виробництво медичних виробів, повинні відповідати обсягам державних нормативних актів. Такі принципи контролю відомі в сукупності як **бізнес-правила** або бізнес-логіка. Бізнес-правила часто дотримуються шляхом ручного впровадження політик та процедур. У багатьох випадках, однак, програмним застосункам також потрібно дотримуватися цих правил.

Каждая организация работает в соответствии с обширным набором политик, законов и отраслевых стандартов. Такие отрасли, как банковское дело, авиация и производство медицинского оборудования, должны соответствовать объемам государственных постановлений. Такие принципы контроля в совокупности известны как **бизнес-правила** или бизнес-логика. Бизнес-правила часто применяются путем ручного внедрения политик и процедур. Однако во многих случаях программные приложения также должны обеспечивать соблюдение этих правил.

175

Most business rules originate outside the context of any specific software application. The corporate policy requiring annual training in handling hazardous chemicals applies even if all chemical purchasing and dispensing is done manually. Standard accounting practices were in use long before the digital computer was invented. Because business rules are a property of the business, they are not in themselves software requirements. However, business rules are a rich source of requirements because they dictate properties the system must possess to conform to the rules.

Більшість бізнес-правил походять поза контекстом будь-якого конкретного програмного забезпечення. Корпоративна політика, що вимагає щорічного навчання з поводження з небезпечними хімічними речовинами, застосовується, навіть якщо всі придбання та видача хімічних речовин здійснюються вручну. Використовувались стандартні методи бухгалтерського обліку задовго до винаходення цифрового комп'ютера. Оскільки бізнес-правила є власністю бізнесу, вони самі по собі не є вимогами до програмного забезпечення. Однак бізнес-правила є багатим джерелом вимог, оскільки вони диктують властивості, якими повинна володіти система, щоб відповідати правилам.

Большинство бизнес-правил возникает вне контекста какого-либо конкретного программного приложения. Корпоративная политика, требующая ежегодного обучения обращению с опасными химическими веществами, применяется, даже если все закупки и дозировки химикатов осуществляются вручную. Использовались стандартные методы бухгалтерского учета задолго до изобретения цифрового компьютера. Поскольку бизнес-правила являются собственностью бизнеса, они сами по себе не являются требованиями к программному обеспечению. Однако бизнес-правила – богатый источник требований, потому что они диктуют свойства, которыми должна обладать система, чтобы соответствовать правилам.

176

Requirement type	Business rules influence	Example
Business requirement	Government regulations can lead to necessary business objectives for a project	The Chemical Tracking System must enable compliance with all state chemical usage and disposal reporting regulations within five months
User requirement	Privacy policies dictate which users can and cannot perform certain tasks with the system	Only laboratory managers are allowed to generate chemical exposure reports for anyone other than themselves
Functional requirement	Company policy is that all vendors must be registered and approved before an invoice will be paid	If an invoice is received from an unregistered vendor, the Supplier System shall email the vendor editable PDF versions of the supplier intake form and the W-9 form
Quality attribute	Regulations from government agencies, such as OSHA and EPA, can dictate safety requirements, which must be enforced through system functionality	The system must maintain safety training records, which it must check to ensure that users are properly trained before they can request a hazardous chemical

177

Тип вимоги	Вплив бізнес-правил	Приклад
Бізнес-вимога	Державні норми можуть привести до необхідних бізнес-цілей проекту	Система відстеження хімічних речовин повинна забезпечити дотримання всіх норм використання та складання звітності щодо утилізації протягом п'яти місяців
Користувацька вимога	Політика конфіденційності визначає, які користувачі можуть і не можуть виконувати певні завдання з системою	Тільки керівникам лабораторій дозволяється створювати звіти про вплив хімічних речовин для будь-кого, крім них самих
Функціональна вимога	Політика компанії полягає в тому, що всі продавці повинні бути зареєстровані та затверджені перед оплатою рахунку-фактури	Якщо рахунок-фактура надійшов від незареєстрованого постачальника, система постачальника надсилає продавцю електронну версію PDF-форми прийому постачальника та форми W-9
Атрибут якості	Нормативні акти державних установ, такі як OSHA та EPA, можуть диктувати вимоги безпеки, які повинні застосовуватися через функціональність системи	Система повинна вести записи про навчання з техніки безпеки, які вона повинна перевіряти, щоб забезпечити належну підготовку користувачів, перш ніж вони зможуть запитати небезпечний хімікат

178

Тип требования	Влияние бизнес-правил	Пример
Бизнес-требование	Правительственные постановления могут привести к достижению необходимых бизнес-целей для проекта	Система отслеживания химикатов должна обеспечивать соблюдение всех государственных правил отчетности по использованию и утилизации химических веществ в течение пяти месяцев
Пользовательское требование	Политики конфиденциальности определяют, какие пользователи могут и не могут выполнять определенные задачи с системой	Только руководители лабораторий могут создавать отчеты о химическом воздействии для кого-либо, кроме себя
Функциональное требование	Политика компании заключается в том, что все поставщики должны быть зарегистрированы и утверждены, прежде чем счет будет оплачен	Если счет-фактура получен от незарегистрированного поставщика, система поставок отправляет электронное письмо с редактируемыми поставщиками PDF-версиями формы приема заявок поставщика и формы W-9
Атрибут качества	Нормативные акты государственных органов, таких как OSHA и EPA, могут диктовать требования безопасности, которые должны выполняться через функциональность системы	Система должна вести записи по обучению технике безопасности, которые она должна проверять, чтобы гарантировать, что пользователи прошли надлежащее обучение, прежде чем они смогут запросить опасное химическое вещество

179

Business requirement states a desirable outcome or a high-level objective of the organization that builds or procures a software solution. A business process describes a series of activities that transform inputs into outputs to achieve a specific result. Information systems frequently automate business processes, which could lead to efficiencies and other benefits that achieve stated business requirements. Business rules influence business processes by establishing vocabulary, imposing restrictions, triggering actions, and governing how computations are carried out. The same business rule could apply to multiple manual or automated processes, which is one reason why it's best to treat business rules as a separate set of information.

Бізнес-вимоги визначають бажаний результат або мету високого рівня організації, яка розробляє або закуповує програмне рішення. Бізнес-процес описує ряд заходів, які перетворюють вхідні дані на вихідні для досягнення конкретного результату. Інформаційні системи часто автоматизують бізнес-процеси, що може привести до підвищення ефективності та інших переваг для досягнення заявлених бізнес-вимог. Бізнес-правила впливають на бізнес-процеси шляхом встановлення глосарію, введення обмежень, активізації дій та регулювання способу обчислень. Ті самі бізнес-правила можуть застосовуватися до декількох ручних або автоматизованих процесів, що є однією з причин, чому найкраще розглядати бізнес-правила як окремий набір інформації.

Бизнес-требование устанавливает желаемый результат или высокоуровневую цель организации, которая создает или закупает программное решение. Бизнес-процесс описывает серию действий, которые преобразуют входы в выходы для достижения определенного результата. Информационные системы часто автоматизируют бизнес-процессы, что может привести к повышению эффективности и другим преимуществам, которые соответствуют заявленным бизнес-требованиям. Бизнес-правила влияют на бизнес-процессы, устанавливая глоссарий, накладывая ограничения, инициируя действия и управляя тем, как выполняются вычисления. Одно и то же бизнес-правило может применяться к нескольким ручным или автоматизированным процессам, что является одной из причин, по которой бизнес-правила лучше рассматривать как отдельный набор информации.

180

Not all companies treat their essential business rules as the valuable enterprise asset they are. Certain departments might document their local rules, but many companies lack a unified effort to document business rules in a common repository accessible to the IT organization. Treating this vital information as corporate folklore leads to numerous problems. If business rules are not properly documented and managed, they exist only in the heads of select individuals.

Не всі компанії розглядають свої основні бізнес-правила як цінний актив підприємства, яким вони є. Деякі департаменти можуть задокументувати свої місцеві правила, але багатьом компаніям бракує єдиних зусиль для документування бізнес-правил у загальному репозиторії, доступному для ІТ-організації. Ставлення до цієї життєво важливої інформації як до корпоративного фольклору призводить до численних проблем. Якщо бізнес-правила не задокументовані належним чином і не управляються, вони існують лише у головах окремих осіб.

Не все компании рассматривают свои основные бизнес-правила как ценный актив предприятия, которым они являются. Некоторые отделы могут документировать свои местные правила, но многим компаниям не хватает единных усилий по документированию бизнес-правил в общем репозитории, доступном для ИТ-организации. Отношение к этой важной информации как к корпоративному фольклору приводит к многочисленным проблемам. Если бизнес-правила не документируются и не управляются должным образом, они существуют только в головах отдельных лиц.

181

Individuals can have conflicting understandings of the rules, which can lead to different software applications enforcing the same business rule inconsistently or overlooking it entirely. Having a master repository of business rules makes it easier for all projects that are affected by certain rules to learn about them and implement them in a consistent fashion.

Особи можуть мати суперечливі розуміння правил, що може привести до того, що різні програмні застосунки виконують одне і те ж правило бізнесу непослідовно або повністю ігнорують його. Наявність головного репозиторію бізнес-правил полегшує всім проектам, на які впливають певні правила, можливість дізнатися про них та послідовно їх реалізовувати.

Люди могут иметь противоречивое понимание правил, что может привести к тому, что разные программные приложения непоследовательно применяют одно и то же бизнес-правило или полностью его игнорируют. Наличие главного репозитория бизнес-правил упрощает всем проектам, на которые влияют определенные правила, их изучение и последовательное их внедрение.

182

As an example, your organization likely has security policies that control access to information systems. Such policies might state the minimum and maximum length and the allowed characters in passwords, dictate the frequency of required password changes, state how many failed login attempts a user gets before his account is locked, and the like. Applications that the organization develops should apply these policies—these business rules—consistently. Tracing each rule into the code that implements it makes it easier to update systems to comply with changes in the rules, such as altering the required frequency of password changes. It also facilitates code reuse across projects.

Наприклад, у вашій організації, ймовірно, є політики безпеки, які контролюють доступ до інформаційних систем. Такі політики можуть містити мінімальну та максимальну довжину та дозволені символи в паролях, диктувати частоту необхідних змін пароля, стверджувати, скільки невдалих спроб входу в систему користувач отримує до того, як його обліковий запис заблоковано тощо. Програми, які розробляє організація, повинні послідовно застосовувати цю політику – ці бізнес-правила. Відстеження кожного правила в коді, що його реалізує, полегшує оновлення систем, щоб вони відповідали змінам правил, таким як зміна необхідної частоти змін пароля. Це також полегшує повторне використання коду в різних проектах.

Например, ваша организация, скорее всего, имеет политики безопасности, которые контролируют доступ к информационным системам. Такие политики могут указывать минимальную и максимальную длину и разрешенные символы в паролях, определять частоту требуемых изменений пароля, указывать, сколько неудачных попыток входа в систему делает пользователь, прежде чем его учетная запись будет заблокирована, и т.п. Приложения, которые разрабатывает организация, должны применять эти политики - эти бизнес-правила - последовательно. Отслеживание каждого правила в коде, который его реализует, упрощает обновление систем в соответствии с изменениями в правилах, такими как изменение необходимой частоты смены пароля. Это также облегчает повторное использование кода в проектах.

183

Business rules are

From the business perspective: "A business rule is guidance that there is an obligation concerning conduct, action, practice, or procedure within a particular activity or sphere." (There ought to be an explicit motivation for the rule, as well as enforcement methods and an understanding of what the consequences would be if the rule were broken.)

З точки зору бізнесу: "Бізнес-правило – це вказівка, згідно з якою існує обов'язок щодо поведінки, дій, практики чи процедури в рамках певної діяльності чи сфери". (Повинна бути чітка мотивація правила, а також методи застосування та розуміння того, якими будуть наслідки, якщо правило буде порушене.)

С точки зрения бизнеса: «Бизнес-правило – это указание на то, что существует обязательство в отношении поведения, действия, практики или процедуры в рамках определенного вида деятельности или сферы». (Должна быть явная мотивация для правила, а также методы принуждения и понимание того, какие будут последствия, если правило будет нарушено.)

184

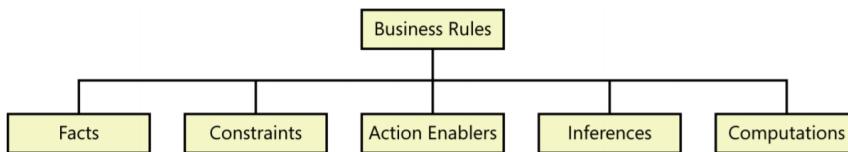
Business rules are

From the information system perspective: "A business rule is a statement that defines or constrains some aspect of the business. It is intended to assert business structure or to control or influence the behavior of the business."

З точки зору інформаційної системи: «Бізнес-правило – це твердження, яке визначає або обмежує якийсь аспект бізнесу. Воно призначено для затвердження структури бізнесу або для контролю чи впливу на поведінку бізнесу».

С точки зрения информационной системы: «Бизнес-правило – это утверждение, которое определяет или ограничивает какой-то аспект бизнеса. Оно предназначено для утверждения структуры бизнеса или для контроля или влияния на поведение бизнеса».

185



Бізнес-правила:

- факти
- обмеження
- активатори операцій
- висновки
- обчислення

Бизнес-правила:

- факты
- ограничения
- активаторы операций
- выводы
- вычисления

186

Recording the business rules in a consistent way is more important than having heated arguments about precisely how to classify each one. However, a taxonomy is helpful to identify business rules you might not have thought of otherwise. Classifying the rules also gives you an idea of how you might apply them in a software application. For instance, constraints often lead to system functionality that enforces the restrictions, and action enablers lead to functionality to make something happen under certain conditions.

Послідовний запис бізнес-правил є більш важливим, ніж загострені суперечки щодо того, як саме класифікувати їх. Однак таксономія корисна для визначення бізнес-правил, про які ви, можливо, і не думали. Класифікація правил також дає уявлення про те, як можна застосувати їх у програмному застосунку. Наприклад, обмеження часто призводять до функціональності системи, яка застосовує обмеження, а активатори операцій призводять до функціональності, щоб змусити щось статися за певних умов.

Последовательная запись бизнес-правил важнее, чем горячие споры о том, как именно классифицировать каждое из них. Однако таксономия полезна для определения бизнес-правил, о которых вы иначе могли бы и не подумать. Классификация правил также дает вам представление о том, как вы можете применить их в программном приложении. Например, ограничения часто приводят к функциональным возможностям системы, которые обеспечивают соблюдение ограничений, а активаторы операций приводят к функциональным возможностям, позволяющим что-то произойти при определенных условиях.

187

Facts are simply statements that are true about the business at a specified point in time. A fact describes associations or relationships between important business terms. Facts about data entities that are important to the system might appear in data models.

Факти – це просто твердження, які відповідають дійсності бізнесу в певний момент часу. Факт описує асоціації або взаємозв'язки між важливими термінами бізнесу. Факти про сутності даних, які є важливими для системи, можуть з'являтися в моделях даних.

Факты – это просто правдивые утверждения о бизнесе в определенный момент времени. Факт описывает ассоциации или отношения между важными бизнес-терминами. Факты об объектах данных, которые важны для системы, могут появляться в моделях данных.

188

Examples of facts:

Each product is characterized by a code and name.
 A code is used to identify each product.
 Each product belongs to a specific product group.

Приклади фактів:

Кожний товар характеризується кодом і назвою.
 Для ідентифікації кожного товару використовується код.
 Кожний товар належить до конкретної товарної групи.

Примеры фактов:

Каждый товар характеризуется кодом и названием.
 Для идентификации каждого товара используется код.
 Каждый товар относится к конкретной товарной группе.

189

Of course, there are countless facts floating around about businesses. Collecting irrelevant facts can bog down business analysis. Even if they're true, it might not be obvious how the development team is to use the information. Focus on facts that are in scope for the project, rather than trying to amass a complete collection of business knowledge. Try to connect each fact to the context diagram's inputs and outputs, to system events, to known data objects, or to specific user requirements.

Звичайно, є незліченна кількість фактів про бізнес. Збір недоречних фактів може заглибити бізнес-аналіз. Навіть якщо вони відповідають дійсності, можливо, не очевидно, як команда розробників використовує інформацію. Зосередьтеся на фактах, які охоплюють проект, а не на спробі накопичити повну колекцію бізнес-знань. Спробуйте прив'язати кожен факт до входів і виходів контекстної діаграми, до системних подій, до відомих об'єктів даних або до конкретних вимог користувача.

Конечно, существует бесчисленное множество фактов о бизнесе. Сбор не относящихся к делу фактов может затруднить бизнес-анализ. Даже если они верны, может быть неочевидно, как команда разработчиков использует информацию. Сосредоточьтесь на фактах, которые входят в объем проекта, вместо того, чтобы пытаться собрать полную коллекцию бизнес-знаний. Попробуйте связать каждый факт с входами и выходами контекстной диаграммы, с системными событиями, с известными объектами данных или с конкретными требованиями пользователя.

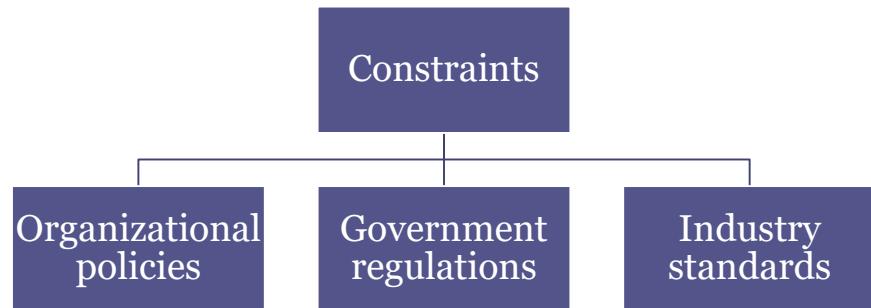
190

A **constraint** is a statement that restricts the actions that the system or its users are allowed to perform. Someone describing a constraining business rule might say that certain actions must or must not or may not be performed, or that only certain people or roles can perform particular actions.

Обмеження – це твердження, яке обмежує дії, які дозволяється виконувати системі або її користувачам. Хтось, хто описує обмежувальне бізнес-правило, може сказати, що певні дії повинні або не повинні виконуватися, або не можуть виконуватися, або що лише певні люди або ролі можуть виконувати певні дії.

Ограничение – это утверждение, которое ограничивает действия, которые система или ее пользователи могут выполнять. Кто-то, описывающий ограничивающее бизнес-правило, может сказать, что определенные действия должны или не должны или не могут выполняться, или что только определенные люди или роли могут выполнять определенные действия.

191



192

Organizational policies

Each supply contract is concluded with a specific supplier. Lack of supplier information is not allowed

The date of concluding the contract must be indicated. If no date is specified, the current date must be used.

Організаційна політика

Кожний договір постачання укладається з конкретним постачальником. Відсутність даних про постачальника не допускається

Дата укладання договору вказується обов'язково. У тому випадку, якщо дата не зазначена, повинна бути використана поточна дата.

Организационная политика

Каждый договор поставки заключается с конкретным поставщиком. Отсутствие данных о поставщике не допускается

Дата заключения договора указывается обязательно. В том случае, если дата не указана, должна быть использована текущая дата.

193

Government regulations

The supplier as a business entity can be either a legal entity or an individual.

For suppliers – legal entities, the VAT payer's certificate number and individual tax number cannot be repeated.

Урядові розпорядження

Постачальник як суб'єкт підприємницької діяльності може бути або юридичною, або фізичною особою.

Для постачальників – юридичних осіб номер свідчення платника ПДВ і індивідуальний податковий номер повторюватися не можуть.

Постановления правительства

Поставщик как субъект предпринимательской деятельности может быть либо юридическим или физическим лицом.

Для поставщиков – юридических лиц номер свидетельства плательщика НДС и индивидуальный налоговый номер повторяться не могут.

194

Industry standards

The quantity of delivered products is always indicated. It cannot be zero or negative.

The price of the delivered production is always specified. It cannot be zero or negative.

Галузеві стандарти

Кількість поставленої продукції завжди вказується. При цьому вона не може бути нульовою або негативною.

Ціна поставленої продукції завжди вказується. При цьому вона не може бути нульовою або негативною.

Отраслевые стандарты

Количество поставленной продукции всегда указывается. При этом она не может быть нулевой или отрицательной.

Цена поставленной продукции всегда указывается. При этом она не может быть нулевой или отрицательной.

195

Constraining business rules can convey implications for software development even if they don't translate directly into functionality. Consider a retail store's policy that only supervisors and managers are allowed to issue cash refunds larger than \$50. If you're developing a point-of-sale application for use by store employees, this rule implies that each user must have a privilege level. The software must check to see if the current user is of sufficiently high privilege level to perform certain actions, such as opening the cash register drawer so a cashier can issue a refund to a customer.

Обмежуючі бізнес-правила можуть мати наслідки для розробки програмного забезпечення, навіть якщо вони не перетворюються безпосередньо на функціональні можливості. Розглянемо політику роздрібного магазину, згідно з якою лише керівники та менеджери можуть повернати гроші, що перевищують 50 доларів США. Якщо ви розробляєте додаток до торгової точки для використання працівниками магазину, це правило передбачає, що кожен користувач повинен мати рівень привілеїв. Програмне забезпечення повинно перевірити, чи має поточний користувач достатньо високий рівень привілеїв для виконання певних дій, наприклад, відкриття ящика каси, щоб касир міг повернути кошти клієнту.

196

Ограничение бизнес-правил может иметь последствия для разработки программного обеспечения, даже если они не отражаются непосредственно в функциональности. Рассмотрим политику розничного магазина, согласно которой только инспекторам и менеджерам разрешается возвращать денежные средства на сумму более 50 долларов. Если вы разрабатываете приложение для торговых точек для использования сотрудниками магазина, это правило подразумевает, что каждый пользователь должен иметь определенный уровень привилегий. Программное обеспечение должно проверять, имеет ли текущий пользователь достаточно высокий уровень привилегий для выполнения определенных действий, таких как открытие кассового аппарата, чтобы кассир мог вернуть деньги клиенту.

197

Because many constraint-type business rules deal with which types of users can perform which functions, a concise way to document such rules is with a roles and permissions matrix. The roles have been separated into employees and non-employees. The system functions are grouped into system operations, operations dealing with patron records, and operations involving individual library items. An X in a cell indicates that the role named in the column has permission to perform the operation shown in the row.

Оскільки багато бізнес-правил типу «обмеження» вирішують, які типи користувачів можуть виконувати певні функції, стислий спосіб документування таких правил полягає в матриці ролей та дозволів. Ролі були розділені на працівників та не працівників. Системні функції згруповані в системні операції, операції, що стосуються записів клієнтів, та операції, що стосуються окремих бібліотечних елементів. Х у комірці означає, що роль, зазначена у стовпці, має дозвіл на виконання операції, показаної в рядку.

198

Поскольку многие бизнес-правила ограничительного типа определяют, какие типы пользователей могут выполнять какие функции, краткий способ документировать такие правила - использовать матрицу ролей и разрешений. Роли были разделены на сотрудников и не сотрудников. Системные функции сгруппированы в системные операции, операции, связанные с записями клиентов, и операции, связанные с отдельными элементами библиотеки. Х в ячейке указывает, что роль, указанная в столбце, имеет разрешение на выполнение операции, показанной в строке.

Roles and Permissions Matrix		Employee	Administrator	Circulation Staff	Library Aide	Non-Employee	Volunteer	Patron
System Operations								
Log in to library system		X	X	X				
Set up new staff members		X						
Print hold pick list		X	X	X				
Patron Records								
View a patron record		X	X					
Edit a patron record		X	X					
View your own patron record		X	X	X		X	X	
Issue a library card		X	X					
Accept a fine payment		X	X					
Item Operations								
Search the library catalog		X	X	X		X	X	
Check out an item		X	X					
Check in an item		X	X	X		X		
Route an item to another branch		X	X	X		X		
Put an item on hold		X	X	X		X	X	

A rule that triggers some activity if specific conditions are true is an **action enabler**. A person could perform the activity in a manual process. Alternatively, the rule might lead to specifying software functionality that makes an application exhibit the correct behavior when the system detects the triggering event. The conditions that lead to the action could be a complex combination of true and false values for multiple individual conditions.

Правило, яке запускає певну активність, якщо конкретні умови відповідають дійсності, - це **активатор операції**. Людина могла б виконувати цю діяльність в ручному процесі. Однак правило може привести до визначення функціональності програмного забезпечення, яке змушує програму демонструвати правильну поведінку, коли система виявляє ініціюючу подію. Умовами, що призводять до дії, можуть бути складні комбінації істинних та хибних значень для багатьох окремих умов.

Правило, которое запускает некоторую активность при выполнении определенных условий, является **активатором операции**. Человек мог бы выполнять действие в ручном режиме. Однако правило может привести к определению программных функций, которые заставляют приложение проявлять правильное поведение, когда система обнаруживает инициирующее событие. Условия, которые приводят к действию, могут быть сложной комбинацией истинных и ложных значений для нескольких отдельных условий.

201

Examples of action enablers:

When adding data about the supplier – legal entity, you need to check whether the data about him as an individual have already been entered. If so, adding data is prohibited.

When adding data about the supplier – an individual, you need to check whether the data about him as a legal entity have already been entered. If so, adding data is prohibited.

Приклади активаторів операцій:

При додаванні даних про постачальника – юридичну особу потрібно перевіряти, чи не введені вже дані про нього як про фізичну особу. Якщо це так, то додавання даних забороняється.

При додаванні даних про постачальника – фізичну особу потрібно перевіряти, чи не введені вже дані про нього як про юридичну особу. Якщо це так, то додавання даних забороняється.

Примеры активаторов операций:

При добавлении данных о поставщике – юридическом лице нужно проверять, не введены уже данные о нем как о физическом лице. Если это так, то добавление данных запрещается.

При добавлении данных о поставщике – физическом лице нужно проверять, не введены уже данные о нем как о юридическом лице. Если это так, то добавление данных запрещается.

202

An **inference** creates a new fact from other facts. Inferences are often written in the “if/then” pattern also found in action-enabling business rules, but the “then” clause of an inference simply provides a piece of knowledge, not an action to be taken.

Висновок створює новий факт із інших фактів. Висновки часто пишуться за зразком «якщо / тоді», який також міститься в активаторах операцій, але пункт «тоді» висновку просто надає частину знань, а не дію, яку потрібно вжити.

Выход создает новый факт из других фактов. Выводы часто записываются в шаблоне «если / тогда», который также присутствует в активаторах операций, но предложение «тогда» вывода просто предоставляет часть знания, а не действие, которое необходимо предпринять.

Examples of inferences:

If the payment is not received within 30 calendar days from the date of sending the invoice, the invoice is considered overdue.

If the supplier cannot deliver the ordered goods within five days of receiving the order, the order is considered unfulfilled.

Приклади висновків:

Якщо платіж не надійшов протягом 30 календарних днів з моменту відправлення рахунку, рахунок уважається просроченим.

Якщо постачальник не може поставити замовлений товар протягом п'яти днів з моменту одержання замовлення, замовлення вважається невиконаним.

Примеры выводов:

Если платеж не поступил в течение 30 календарных дней с момента отправки счета, счет считается просроченным.

Если поставщик не может поставить заказанный товар в течение пяти дней с момента получения заказа, заказ считается невыполненным.

The fifth class of business rules defines **computations** that transform existing data into new data by using specific mathematical formulas or algorithms. Many computations follow rules that are external to the enterprise, such as income tax withholding formulas.

П'ятий клас бізнес-правил визначає **обчислення**, які перетворюють наявні дані в нові дані за допомогою конкретних математичних формул або алгоритмів. Багато обчислень дотримуються зовнішніх для підприємства правил, таких, наприклад, як формули утримання податку на прибуток.

Пятый класс бизнес-правил определяет **вычисления**, которые преобразуют существующие данные в новые с помощью определенных математических формул или алгоритмов. Многие вычисления выполняются по правилам, которые не относятся к предприятию, такие, например, как формулы удержания подоходного налога.

205

Representing the details of computations in natural language like this can be wordy and confusing. As an alternative, you could represent these in some symbolic form, such as a mathematical expression or in a table of rules that is clearer and easier to maintain.

Подання подібних деталей обчислень природною мовою може бути багатослівним і запутаним. Як альтернативу, ви можете подати їх у якісь символічній формі, наприклад, у математичному виразі або у таблиці правил, яка є зрозумілішою та простішою в обслуговуванні.

Такое представление деталей вычислений на естественном языке может быть многословным и запутанным. В качестве альтернативы вы можете представить их в некоторой символьической форме, такой как математическое выражение или в таблице правил, которая будет более ясной и простой в обслуживании.

206

ID	Number of units purchased	Percent discount
DISC-1	1 through 5	0
DISC-2	6 through 10	10
DISC-3	11 through 20	20
DISC-4	More than 20	30

Atomic business rules

Composite business rules can be hard to understand and maintain. It's also hard to confirm that all possible conditions are covered. If several functionality segments trace back to this complex rule, it can be time-consuming to find and modify the appropriate code when just one part of the rule changes in the future.

Складені бізнес-правила важко зрозуміти та підтримувати. Також важко підтвердити, що всі можливі умови охоплені. Якщо кілька сегментів функціональності відстежуються до цього складного правила, пошук та модифікація відповідного коду може зайняти багато часу, коли в майбутньому зміниться лише одна частина правила.

Составные бизнес-правила бывает сложно понять и поддерживать. Также трудно подтвердить, что все возможные условия соблюdenы. Если несколько функциональных сегментов связаны с этим сложным правилом, то поиск и изменение соответствующего кода может занять много времени, если в будущем изменится только одна часть правила.

A better strategy is to write your business rules at the atomic level, rather than combining multiple details into a single rule. This keeps your rules short and simple. It also facilitates reusing the rules, modifying them, and combining them in various ways. These business rules are called atomic because they can't be decomposed further. You will likely end up with many atomic business rules, and your functional requirements will depend on various combinations of them.

Кращою стратегією є написання своїх бізнес-правил на атомарному рівні, а не поєднання кількох деталей в одне правило. Це робить ваші правила короткими та простими. Це також сприяє повторному використанню правил, їх модифікації та їх різному поєднанню. Ці бізнес-правила називаються атомарними, оскільки їх не можна розкладати далі. Швидше за все, ви отримаєте багато атомарних бізнес-правил, і ваши функціональні вимоги залежатимуть від різних їх комбінацій.

Лучшая стратегия – написать бизнес-правила на атомарном уровне, а не объединять несколько деталей в одно правило. Благодаря этому ваши правила будут короткими и простыми. Это также облегчает повторное использование правил, их изменение и комбинирование различными способами. Эти бизнес-правила называются атомарными, потому что их нельзя подвергнуть дальнейшей декомпозиции. Скорее всего, вы получите множество элементарных бизнес-правил, и ваши функциональные требования будут зависеть от их различных комбинаций.

209

«You can check out a DVD or Blu-ray Disc for one week, and you may renew it up to two times for three days each, but only if another patron hasn't placed a hold on it.»



ID	Rule
Video.Media.Types	DVD discs and Blu-ray Discs are video items.
Video.Checkout.Duration	Video items may be checked out for one week at a time.
Renewal.Video.Times	Video items may be renewed up to two times.
Renewal.Video.Duration	Renewing a checked-out video item extends the due date by three days.
Renewal.HeldItem	A patron may not renew an item that another patron has on hold.

210

Documenting business rules

Because business rules can influence multiple applications, organizations should manage their rules as enterprise-level assets. A simple business rules catalog will suffice initially. If you're using a requirements management tool, you can store business rules as a requirement type, provided they are accessible to all of your software projects. Large organizations or those whose operations and information systems are heavily business-rule driven should establish a database of business rules. Commercial rule-management tools become valuable if your rules catalog outgrows a solution using a word processor, spreadsheet, Wiki, or other collaboration tool. Some business-rule management systems contain rules engines, which can automate the implementation of the rules in your applications.

Оскільки бізнес-правила можуть впливати на кілька додатків, організації повинні керувати своїми правилами як активами підприємства. На початку буде достатньо простого каталогу бізнес-правил. Якщо ви використовуєте інструмент управління вимогами, ви можете зберігати бізнес-правила як тип вимоги, за умови, що вони доступні для всіх ваших програмних проектів. Великі організації або ті, чия діяльність та інформаційні системи значною мірою керуються бізнес-правилами, повинні створити базу даних бізнес-правил. Інструменти управління комерційними правилами стають цінними, якщо ваш каталог правил переростає рішення за допомогою текстового процесора, електронної таблиці, Wiki або іншого інструменту для співпраці. Деякі системи управління бізнес-правилами містять механізми правил, які можуть автоматизувати впровадження правил у ваших додатках.

211

Поскольку бизнес-правила могут влиять на несколько приложений, организациям следует управлять своими правилами как активами корпоративного уровня. Сначала будет достаточно простого каталога бизнес-правил. Если вы используете инструмент управления требованиями, вы можете сохранить бизнес-правила как тип требований при условии, что они доступны для всех ваших программных проектов. Крупные организации или те, чьи операции и информационные системы в значительной степени управляются бизнес-правилами, должны создать базу данных бизнес-правил. Коммерческие инструменты управления правилами становятся цennыми, если ваш каталог правил превосходит решение, использующее текстовый процессор, электронную таблицу, Wiki или другой инструмент для совместной работы. Некоторые системы управления бизнес-правилами содержат механизмы правил, которые могут автоматизировать выполнение правил в ваших приложениях.

212

As you gain experience with identifying and documenting business rules, you can apply structured templates for defining rules of different types. These templates describe patterns of keywords and clauses that structure the rules in a consistent fashion. They also facilitate storing the rules in a database, a commercial business-rule management tool, or a business rules engine. Sets of related rules can also be represented by using tools such as decision trees and decision tables (particularly when complex logic is involved) and roles and permissions matrices.

Набуваючи досвіду з ідентифікації та документування бізнес-правил, ви можете застосовувати структуровані шаблони для визначення правил різних типів. Ці шаблони описують схеми ключових слів і речень, які послідовно структурують правила. Вони також полегшують зберігання правил у базі даних, комерційному інструменті управління бізнес-правилами або механізмі бізнес-правил. Набори пов'язаних правил також можуть бути представлені за допомогою таких інструментів, як дерева рішень, таблиці рішень (особливо, коли задіяна складна логіка) та матриць ролей та дозволів.

213

По мере того, как вы приобретаете опыт определения и документирования бизнес-правил, вы можете применять структурированные шаблоны для определения правил разных типов. Эти шаблоны описывают шаблоны ключевых слов и предложений, которые единообразно структурируют правила. Они также облегчают хранение правил в базе данных, коммерческом инструменте управления бизнес-правилами или механизме бизнес-правил. Наборы связанных правил также могут быть представлены с помощью таких инструментов, как деревья решений и таблицы решений (особенно когда задействована сложная логика), а также матрицы ролей и разрешений.

214

ID	Rule definition	Type of rule	Static or dynamic	Source
ORDER-5	If the customer ordered a book by an author who has written multiple books, then offer the author's other books to the customer before completing the order.	Action enabler	Static	Marketing policy XX
ACCESS-8	All website images must include alternative text to be used by electronic reading devices to meet accessibility requirements for visually impaired users.	Constraint	Static	ADA Standards for Accessible Design
DISCOUNT-13	A discount is calculated based on the size of the current order, as defined in Table BR-060.	Computation	Dynamic	Corporate pricing policy XX

Giving each business rule a unique identifier lets you link requirements back to a specific rule. For instance, some templates for use cases contain a field for business rules that influence the use case. Instead of including the rule definition in the use case description, simply enter the identifiers for the relevant rules. Each ID serves as a pointer to the master instance of the business rule. This way you don't have to worry about the use case specification becoming obsolete if the rule changes.

Надання кожному бізнес-правилу унікального ідентифікатора дозволяє зв'язати вимоги з певним правилом. Наприклад, деякі шаблони для випадків використання містять поле для бізнес-правил, які впливають на варіант використання. Замість того, щоб включати визначення правила в опис випадку використання, просто введіть ідентифікатори відповідних правил. Кожен ідентифікатор слугує вказівником на головний екземпляр бізнес-правила. Таким чином, вам не доведеться турбуватися про те, що специфікація варіанту використання застаріє, якщо правило зміниться.

Присвоение каждому бизнес-правилу уникального идентификатора позволяет связать требования с конкретным правилом. Например, некоторые шаблоны для вариантов использования содержат поле для бизнес-правил, которые влияют на вариант использования. Вместо того, чтобы включать определение правила в описание варианта использования, просто введите идентификаторы для соответствующих правил. Каждый идентификатор служит указателем на главный экземпляр бизнес-правила. Таким образом, вам не придется беспокоиться о том, что спецификация варианта использования устареет, если правило изменится.

The “Type of rule” column identifies each business rule as being a fact, constraint, action enabler, inference, or computation. The “Static or dynamic” column indicates how likely the rule is to change over time. This information is helpful to developers. If they know that certain rules are subject to periodic change, they can structure the software to make the affected functionality or data easy to update. Income tax calculations change at least every year. If the developer structures the income tax information into tables of a database, rather than hard-coding it into the software, it's a lot easier to update those values when necessary. It's safe to hard-code laws of nature, such as calculations based on the laws of thermodynamics; laws of humans are much more volatile.

Стовпець "Тип правила" визначає кожне бізнес-правило як факт, обмеження, активатор операцій, висновок чи обчислення. У стовпці "Статичний або динамічний" вказується, наскільки ймовірно, що правило змінюється з часом. Ця інформація корисна розробникам. Якщо вони знають, що певні правила можуть періодично змінюватися, вони можуть структурувати програмне забезпечення для спрощення функціональних можливостей чи даних, які легко оновлювати. Розрахунки з податку на прибуток змінюються щонайменше щороку. Якщо розробник структурує інформацію про податок на прибуток у таблиці бази даних, а не жорстко кодує її у програмне забезпечення, набагато простіше оновити ці значення, коли це необхідно. Безпечно жорстко кодувати закони природи, такі як обчислення на основі законів термодинаміки; закони людей набагато мінливіші.

217

Столбец «Тип правила» определяет каждое бизнес-правило как факт, ограничение, средство действия, вывод или вычисление. Столбец «Статический или динамический» указывает, насколько вероятно правило со временем измениться. Эта информация полезна для разработчиков. Если они знают, что определенные правила подлежат периодическому изменению, они могут структурировать программное обеспечение, чтобы упростить обновление затронутых функций или данных. Расчет подоходного налога меняется не реже одного раза в год. Если разработчик структурирует информацию о подоходном налоге в виде таблиц базы данных, а не жестко закодирует ее в программном обеспечении, при необходимости будет намного проще обновить эти значения. Безопаснее жестко кодировать законы природы, такие как вычисления, основанные на законах термодинамики; человеческие законы гораздо более изменчивы.

218

Discovering business rules

Just as asking “What are your requirements?” doesn’t help much when eliciting user requirements, asking users “What are your business rules?” doesn’t get you very far. Sometimes you invent business rules as you go along, sometimes they come up during requirements discussions, and sometimes you need to hunt for them.

Так само, як запитання «Які ваші вимоги?» не дуже допомагає при виявленні користувальських вимог, запитання «Які ваші бізнес-правила?» також не приводить дуже далеко. Іноді ви вигадуєте бізнес-правила по ходу справи, іноді вони з’являються під час обговорення вимог, а іноді вам потрібно їх шукати.

Так же как и вопрос «Каковы ваши требования?» не очень помогает при выявлении требований пользователей, вопрос «Каковы ваши бизнес-правила?» не приводит слишком далеко. Иногда вы придумываете бизнес-правила по ходу дела, иногда они всплывают во время обсуждения требований, а иногда вам нужно их искать.

219

Where to look for business rules?

- “Common knowledge” from the organization, often collected from individuals who have worked with the business for a long time and know the details of how it operates.
- Legacy systems that embed business rules in their requirements and code. This requires reverse-engineering the rationale behind the requirements or code to understand the pertinent rules. This sometimes yields incomplete knowledge about the business rules.
- Business process modeling, which leads the analyst to look for rules that can affect each process step: constraints, triggering events, computational rules, and relevant facts.
- Analysis of existing documentation, including requirements specifications from earlier projects, regulations, industry standards, corporate policy documents, contracts, and business plans.
- Analysis of data, such as the various states that a data object can have and the conditions under which a user or a system event can change the object’s state.
- Compliance departments in companies building systems subject to regulation.

220

Де шукати бізнес-правила?

- «Загальні знання» від організації, які часто збираються від осіб, які тривалий час працювали у бізнесі та знають подробиці того, як він працює.
- Старі системи, які вбудовують бізнес-правила у свої вимоги та код. Це вимагає зворотного проектування вимог або коду для розуміння відповідних правил. Це іноді дає неповні знання про бізнес-правила.
- Моделювання бізнес-процесів, що змушує аналітика шукати правила, які можуть впливати на кожен крок процесу: обмеження, ініціюючі події, обчислювальні правила та відповідні факти.
- Аналіз існуючої документації, включаючи вимоги до попередніх проектів, нормативних актів, галузевих стандартів, документів корпоративної політики, контрактів та бізнес-планів.
- Аналіз даних, таких як різні стани, які може мати об'єкт даних, та умови, за яких користувач або подія системи можуть змінити стан об'єкта.
- Відділи відповідності у компаніях, що будують системи, що підлягають регулюванню.

Где искать бизнес-правила?

- «Общие знания» организации, часто собираемые от людей, которые работали с бизнесом в течение длительного времени и знают подробности того, как он работает.
- Унаследованные системы, которые включают бизнес-правила в свои требования и код. Это требует обратного инжиниринга требований или кода, чтобы понять соответствующие правила. Иногда это приводит к неполному знанию бизнес-правил.
- Моделирование бизнес-процессов, которое заставляет аналитика искать правила, которые могут влиять на каждый шаг процесса: ограничения, запускающие события, вычислительные правила и соответствующие факты.
- Анализ существующей документации, включая спецификации требований из более ранних проектов, нормативные акты, отраслевые стандарты, документы корпоративной политики, контракты и бизнес-планы.
- Анализ данных, таких как различные состояния, которые может иметь объект данных, и условия, при которых пользователь или системное событие могут изменить состояние объекта.
- Отделы соответствия в компаниях, создающих системы, подлежащие регулированию.

Just because you found some business rules in these various sources doesn't mean they necessarily apply to your current project or that they are even still valid. Computational formulas implemented in the code of legacy applications could be obsolete. Be sure to confirm whether rules gleaned from older documents and applications need to be updated. Assess the scope of applicability of rules you discover. Are they local to the project, or do they span a business domain or the entire enterprise.

Те, що ви знайшли деякі бізнес-правила у цих різних джерелах, не означає, що вони обов'язково стосуються вашого поточного проекту або навіть діють. Обчислювальні формулі, реалізовані в коді успадкованих додатків, можуть бути застарілими. Обов'язково переконайтеся, що правила, зібрані зі старих документів та програм, потребують оновлення. Оцініть сферу застосовності відкритих вами правил. Чи є вони місцевими для проекту, чи охоплюють сферу бізнесу або все підприємство.

Тот факт, что вы нашли какие-то бизнес-правила в этих различных источниках, не означает, что они обязательно применимы к вашему текущему проекту или что они еще действительны. Вычислительные формулы, реализованные в коде унаследованных приложений, могут оказаться устаревшими. Убедитесь, что вы подтвердили необходимость обновления правил, взятых из старых документов и приложений. Оцените сферу применения обнаруженных вами правил. Являются ли они локальными по отношению к проекту или охватывают сферу бизнеса или все предприятие.

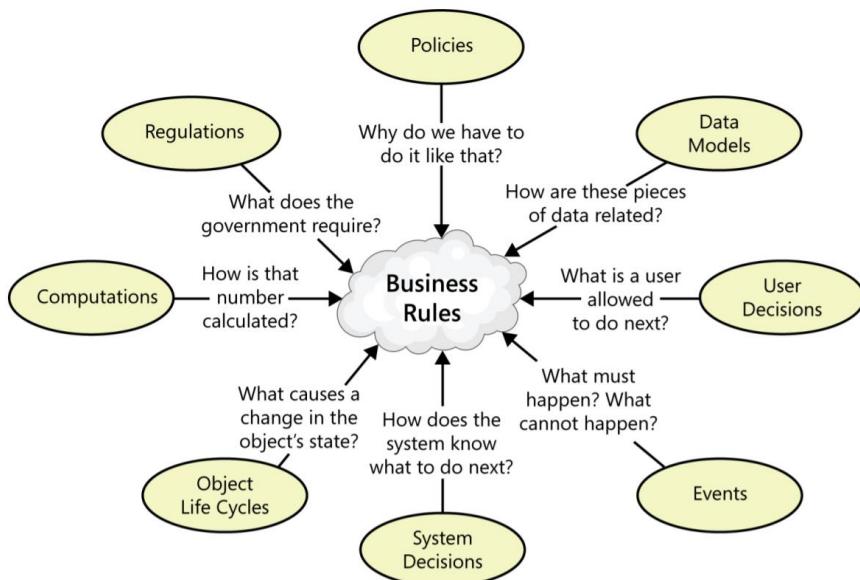
223

Often, project stakeholders already know about business rules that will influence the application. Certain employees sometimes deal with particular types or classes of rules. If that's the case in your environment, find out who those people are and bring them into the discussion.

Часто зацікавлені сторони проекту вже знають про бізнес-правила, які впливатимуть на застосунок. Деякі працівники іноді мають справу з певними типами або класами правил. Якщо це так у вашому оточенні, з'ясуйте, хто такі люди, і залучіть їх до обговорення.

Часто заинтересованные стороны проекта уже знают о бизнес-правилах, которые будут влиять на приложение. Некоторые сотрудники иногда имеют дело с определенными типами или классами правил. Если это так в вашем окружении, выясните, кто эти люди, и вовлеките их в обсуждение.

224



225

DATA MODELS

226

Data Modeling and Data Models

Data modeling: Iterative and progressive process of creating a specific data model for a determined problem domain

Data models: Simple representations of complex real-world data structures

Model – Abstraction of a real-world object or event

Моделювання даних: ітераційний та прогресивний процес створення конкретної моделі даних для визначеної проблемної області

Моделі даних: прості подання складних реальних структур даних

Модель – абстракція реального об'єкта чи події

Моделирование данных: итеративный и прогрессивный процесс создания конкретной модели данных для определенной проблемной области.

Модели данных: простые представления сложных реальных структур данных.

Модель – абстракция реального объекта или события

227

- Are a communication tool
- Give an overall view of the database
- Organize data for various users
- Are an abstraction for the creation of good database

- Є інструментом спілкування
- Дає загальний огляд бази даних
- Впорядковує дані для різних користувачів
- Є абстракцією для створення хорошої бази даних

- Инструмент коммуникации
- Дает общее представление о базе данных
- Организует данные для разных пользователей
- Абстракция для создания хорошей базы данных

228

Entity: Unique and distinct object used to collect and store data

Attribute: Characteristic of an entity

Relationship: Describes an association among entities

- One-to-many (1:M)
- Many-to-many (M:N or M:M)
- One-to-one (1:1)

Constraint: Set of rules to ensure data integrity

Сутність: Унікальний та чіткий об'єкт, що використовується для збору та зберігання даних

Атрибут: Характеристика сутності

Взаємозв'язок: описує асоціацію між сутностями

- Один-до-багатьох (1:M)
- Багато-до-багатьох (M:N або M:M)
- Один-до-одного (1:1)

Обмеження: набір правил для забезпечення цілісності даних

229

Сущность: уникальный и четкий объект, используемый для сбора и хранения данных

Атрибут: характеристика объекта

Связь: описывает связь между сущностями

- Один-ко-многим (1:M)
- Многие-ко-многим (M:N или M:M)
- Один-к-одному (1:1)

Ограничение: набор правил для обеспечения целостности данных

230

Business Rules

Brief, precise, and unambiguous description of a policy, procedure, or principle

Enable defining the basic building blocks

Describe main and distinguishing characteristics of the data

Короткий, точний і однозначний опис політики, процедури чи принципу

Містять визначення основних будівельних блоків

Описує основні та відмінні характеристики даних

Краткое, точное и недвусмысленное описание политики, процедуры или принципа

Включает определение основных строительных блоков

Описывает основные и отличительные характеристики данных

231

Reasons for Identifying and Documenting Business Rules

Help standardize company's view of data

Communications tool between users and designers

Allow designer to:

- Understand the nature, role, scope of data, and business processes
- Develop appropriate relationship participation rules and constraints
- Create an accurate data model

Причини виявлення та документування бізнес-правил

Допомагають стандартизувати погляд компанії на дані

Інструмент комунікації між користувачами та дизайнерами

Дозволяють дизайнери:

- Зрозуміти природу, роль, обсяг даних та бізнес-процеси
- Розробити відповідні правила та обмеження щодо участі сущностей у відношеннях
- Створити точну модель даних

232

Причины выявления и документирования бизнес-правил

Помогают стандартизировать взгляд компании на данные

Инструмент коммуникации между пользователями и дизайнерами

Позволяют дизайнеру:

- Понять природу, роль, объем данных и бизнес-процессы
- Разработать соответствующие правила и ограничения участия сущностей в отношениях
- Создать точную модель данных

233

Translating Business Rules into Data Model Components

Nouns translate into entities

Verbs translate into relationships among entities

Relationships are bidirectional

Questions to identify the relationship type

- How many instances of B are related to one instance of A?
- How many instances of A are related to one instance of B?

Перетворення бізнес-правил у компоненти моделі даних

Іменники трансформуються у сутності

Дієслова трансформуються у відношення між сущностями

Відношення двунаправлені

Питання для визначення типу стосунків

- Скільки екземплярів В пов'язано з одним екземпляром А?
- Скільки екземплярів А пов'язано з одним екземпляром В?

234

Преобразование бизнес-правил в компоненты модели данных

Существительные трансформируются в сущности

Глаголы трансформируются в отношения между сущностями

Отношения двунаправленные

Вопросы для определения типа отношений

- Сколько экземпляров В связано с одним экземпляром А?
- Сколько экземпляров А связано с одним экземпляром В?

235

Naming Conventions

Entity names - Required to:

- Be descriptive of the objects in the business environment
- Use terminology that is familiar to the users

Attribute name - Required to be descriptive of the data represented by the attribute

Proper naming:

- Facilitates communication between parties
- Promotes self-documentation

Конвенції іменування

Назви сущностей повинні:

- Бути описовими для об'єктів у бізнес-середовищі
- Використовувати звичну для користувачів термінологію

Назва атрибута повинна бути описова для даних, представлених атрибутом

Правильне найменування:

- Сприяє спілкуванню між сторонами
- Сприяє самодокументуванню

236

Конвенции именования

Имена сущностей должны:

- Описывать объекты в бизнес-среде
- Использовать терминологию, знакомую пользователям

Имя атрибута должно быть описательным для данных, представленных атрибутом

Правильное название:

- Облегчает общение между сторонами
- Способствует самодокументированию

237

Standard Database Concepts

Schema

Conceptual organization of the entire database as viewed by the database administrator

Subschema

Portion of the database seen by the application programs that produce the desired information from the data within the database

Схема

Концептуальна організація всієї бази даних, яка доступна адміністратору бази даних

Підсхема

Частина бази даних, яку бачать прикладні програми, що отримують бажану інформацію з даних у базі даних

238

Схема

Концептуальная организация всей базы данных, доступная администратору базы данных

Подсхема

Часть базы данных, видимая прикладными программами, которые получают желаемую информацию из данных в базе данных

239

Data manipulation language (DML)

Environment in which data can be managed and is used to work with the data in the database

Data definition language (DDL)

Enables the database administrator to define the schema components

Мова обробки даних (DML)

Середовище, в якому можна управляти даними та яке використовується для роботи з даними в базі даних

Мова визначення даних (DDL)

Дозволяє адміністратору бази даних визначати компоненти схеми

Язык обработки данных (DML)

Среда, в которой можно управлять данными и которая используется для работы с данными в базе данных

Язык определения данных (DDL)

Позволяет администратору базы данных определять компоненты схемы

240

Relational Model

Based on a relation

Relation or table: Matrix composed of intersecting tuple and attribute

Tuple: Rows

Attribute: Columns

Describes a precise set of data manipulation constructs

Базується на відношеннях

Відношення або таблиця: матриця, що складається з кортежів та атрибутів, що перетинаються

Кортеж: Рядки

Атрибут: Столпці

Описує точний набір конструкцій для обробки даних

Базируется на отношениях

Отношение или таблица: матрица, состоящая из пересекающихся кортежей и атрибутов

Кортеж: строки

Атрибут: столбцы

Описывает точный набор конструкций манипулирования данными

241

Advantages	Disadvantages
<ul style="list-style-type: none"> Structural independence is promoted using independent tables Tabular view improves conceptual simplicity Ad hoc query capability is based on SQL Isolates the end user from physical-level details Improves implementation and management simplicity 	<ul style="list-style-type: none"> Requires substantial hardware and system software overhead Conceptual simplicity gives untrained people the tools to use a good system poorly May promote information problems
Переваги	Недоліки
<ul style="list-style-type: none"> Структурній незалежності сприяє використання незалежних таблиць Табличний вигляд покращує концептуальну простоту Можливість спеціальних запитів базується на SQL Ізolare кінцевого користувача від деталей фізичного рівня Покращує простоту впровадження та управління 	<ul style="list-style-type: none"> Потрібні значні накладні витрати на апаратне та системне програмне забезпечення Концептуальна простота дає непідготовленим людям інструменти поганого використання хорошої системи Може сприяти інформаційним проблемам

242

Преимущества	Недостатки
<ul style="list-style-type: none"> Структурная независимость достигается с помощью независимых таблиц Табличный вид улучшает концептуальную простоту Возможность специальных запросов основана на SQL Изолирует конечного пользователя от деталей физического уровня Повышает простоту внедрения и управления 	<ul style="list-style-type: none"> Требуются значительные накладные расходы на оборудование и системное программное обеспечение Концептуальная простота дает неподготовленным людям инструменты для плохого использования хорошей системы Может способствовать информационным проблемам

243

SQL-Based Relational Database Application

End-user interface

- Allows end user to interact with the data
- Collection of tables stored in the database
 - Each table is independent from another
 - Rows in different tables are related based on common values in common attributes

SQL engine

- Executes all queries

Інтерфейс кінцевого користувача

- Дозволяє кінцевому користувачеві взаємодіяти з даними
- Колекція таблиць, що зберігаються в базі даних
 - Кожна таблиця незалежна від іншої
 - Рядки в різних таблицях пов'язані на основі загальних значень у загальних атрибутах
- Двіжок SQL
 - Виконує всі запити

244

Интерфейс конечного пользователя

- Позволяет конечному пользователю взаимодействовать с данными
- Коллекция таблиц, хранящихся в базе данных
 - Каждая таблица не зависит от другой
 - Строки в разных таблицах связаны на основе общих значений общих атрибутов
- Двигок SQL
 - Выполняет все запросы

245

Entity Relationship Model

Graphical representation of entities and their relationships in a database structure

Entity relationship diagram (ERD)

- Uses graphic representations to model database components

Entity instance or entity occurrence

- Rows in the relational table

Connectivity: Term used to label the relationship types

Графічне представлення сущностей та їх взаємозв'язків у структурі бази даних

Діаграма сущність-відношення (ERD)

- Використовує графічні зображення для моделювання компонентів бази даних

Екземпляр сущності або входження сущності

- Рядки в реляційній таблиці

Зв'язок: термін, що використовується для позначення типів відношень

246

Графическое представление сущностей и их взаимосвязей в структуре базы данных

Диаграмма сущность-связь (ERD)

- Использует графические представления для моделирования компонентов базы данных

Экземпляр сущности или вхождение сущности

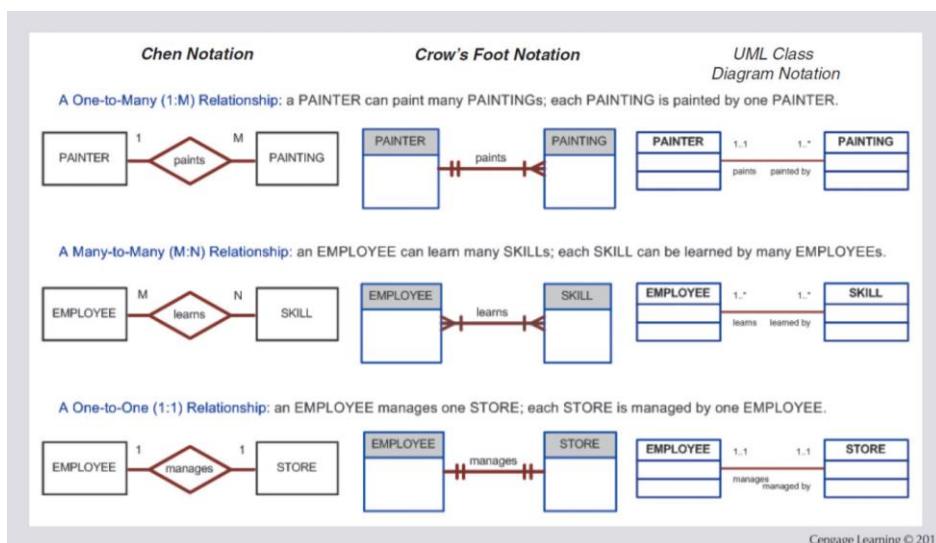
- Строки в реляционной таблице

Связь: термин, используемый для обозначения типов отношений

247

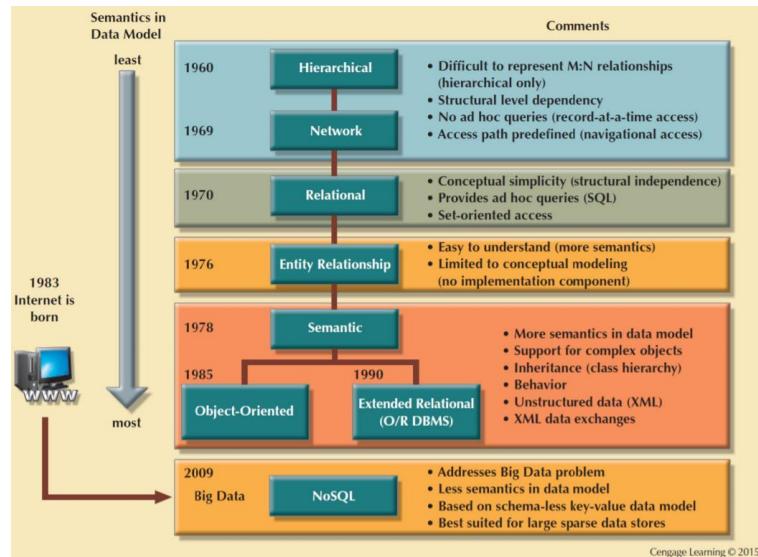
Advantages	Disadvantages
<ul style="list-style-type: none"> • Visual modeling yields conceptual simplicity • Visual representation makes it an effective communication tool • Is integrated with the dominant relational model 	<ul style="list-style-type: none"> • Limited constraint representation • Limited relationship representation • No data manipulation language • Loss of information content occurs when attributes are removed from entities to avoid crowded displays
Переваги	Недоліки
<ul style="list-style-type: none"> • Візуальне моделювання дає концептуальну простоту • Візуальне подання робить його ефективним інструментом спілкування • Інтегрована з домінуючою реляційною моделью 	<ul style="list-style-type: none"> • Спрощене представлення обмежень • Спрощене представлення відношень • Немає мови для обробки даних • Втрата інформаційного вмісту відбувається, коли атрибути видаляються з сущностей, щоб уникнути переповнених дисплеїв
Преимущества	Недостатки
<ul style="list-style-type: none"> • Визуальное моделирование дает концептуальную простоту • Визуальное представление делает его эффективным инструментом коммуникации • Интегрирован с доминирующей реляционной моделью 	<ul style="list-style-type: none"> • Упрощенное представление ограничений • Упрощенное представление отношений • Нет языка для обработки данных • Потеря информационного содержания происходит, когда атрибуты удаляются из сущностей, чтобы избежать переполненных дисплеев

248



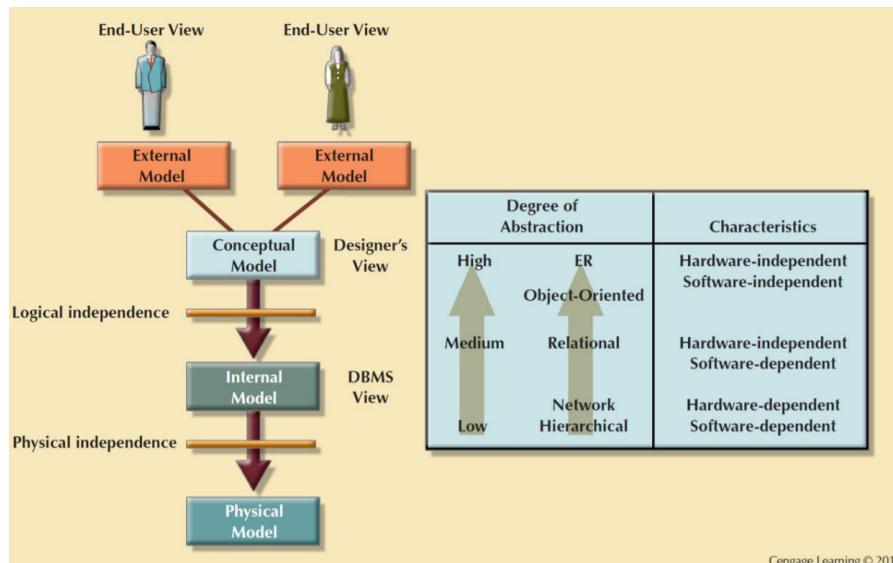
249

The Evolution of Data Models



250

Data Abstraction Levels



251

Attributes

Characteristics of entities

Required attribute: Must have a value, cannot be left empty

Optional attribute: Does not require a value, can be left empty

Domain - Set of possible values for a given attribute

Identifiers: One or more attributes that uniquely identify each entity instance

Характеристики сущностей

Обов'язковий атрибут: має мати значення, не може залишатися порожнім

Необов'язковий атрибут: не вимагає значення, його можна залишити порожнім

Домен – набір можливих значень для даного атрибута

Ідентифікатори: один або більше атрибутів, які однозначно ідентифікують кожен екземпляр сущності

252

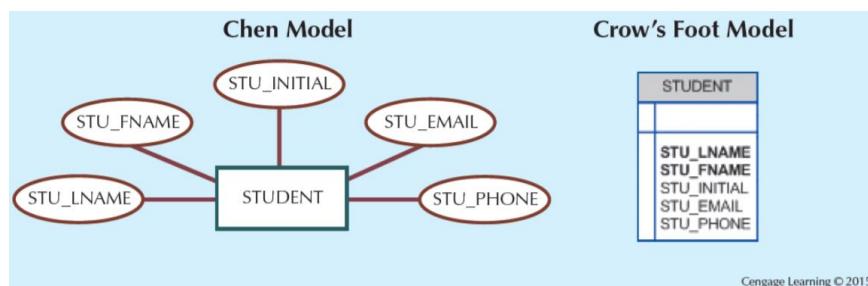
Характеристики сущностей

Обязательный атрибут: должен иметь значение, не может быть оставлен пустым

Необязательный атрибут: не требует значения, можно оставить пустым

Домен – набор возможных значений для данного атрибута

Идентификаторы: один или несколько атрибутов, которые однозначно идентифицируют каждый экземпляр объекта



253

Composite identifier: Primary key composed of more than one attribute

Composite attribute: Attribute that can be subdivided to yield additional attributes

Simple attribute: Attribute that cannot be subdivided

Single-valued attribute: Attribute that has only a single value

Multivalued attributes: Attributes that have many values

Складений ідентифікатор: первинний ключ, що складається з більш ніж одного атрибута

Складений атрибут: атрибут, який можна розділити, щоб отримати додаткові атрибути

Простий атрибут: атрибут, який не можна розділити

Однозначний атрибут: атрибут, який має лише одне значення

Багатозначні атрибути: атрибути, що мають багато значень

254

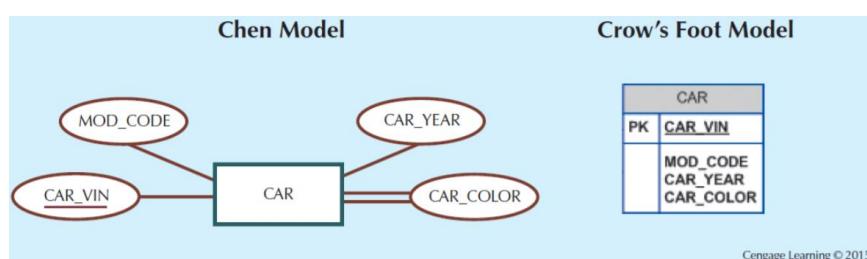
Составной идентификатор: первичный ключ, состоящий из нескольких атрибутов

Составной атрибут: атрибут, который можно разделить для получения дополнительных атрибутов

Простой атрибут: атрибут, который нельзя разделить

Однозначный атрибут: атрибут, имеющий только одно значение

Многозначные атрибуты: атрибуты, которые имеют много значений



255

Multivalued attributes: Attributes that have many values and require creating:

- Several new attributes, one for each component of the original multivalued attribute
- A new entity composed of the original multivalued attribute's components

Derived attribute: Attribute whose value is calculated from other attributes

- Derived using an algorithm

Багатозначні атрибути: атрибути, що мають багато значень і потребують створення:

- Кількох нових атрибутів, по одному для кожного компонента вихідного багатозначного атрибута
- Нової сутності, що складається з вихідних багатозначних компонентів атрибута

Похідний атрибут: атрибут, значення якого обчислюється з інших атрибутів

- Отриманого за допомогою алгоритму

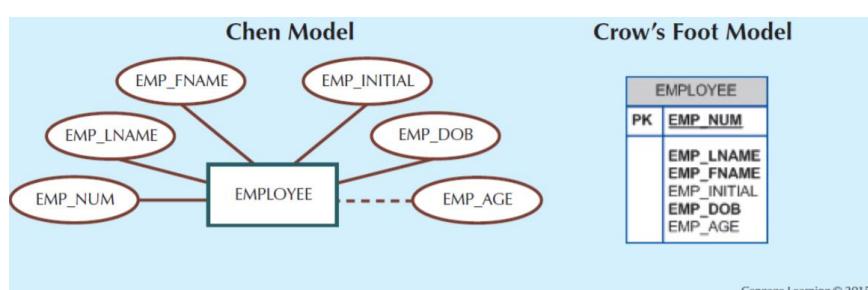
256

Многозначные атрибуты: атрибуты, которые имеют много значений и требуют создания:

- Нескольких новых атрибутов, по одному для каждого компонента исходного многозначного атрибута
- Нового объекта, состоящего из компонентов исходного многозначного атрибута

Производный атрибут: атрибут, значение которого вычисляется из других атрибутов

- Полученного с использованием алгоритма



Cengage Learning © 2015

257

Relationships

Association between entities that always operate in both directions

Participants: Entities that participate in a relationship

Connectivity: Describes the relationship classification

Cardinality: Expresses the minimum and maximum number of entity occurrences associated with one occurrence of related entity

Асоціації між сущностями, які завжди діють в обох напрямках

Учасники: сущності, які беруть участь у відношеннях

Зв'язок: описує класифікацію відношень

Потужність: виражає мінімальну та максимальну кількість екземплярів сущності, пов'язаних з одним екземпляром пов'язаної сущності

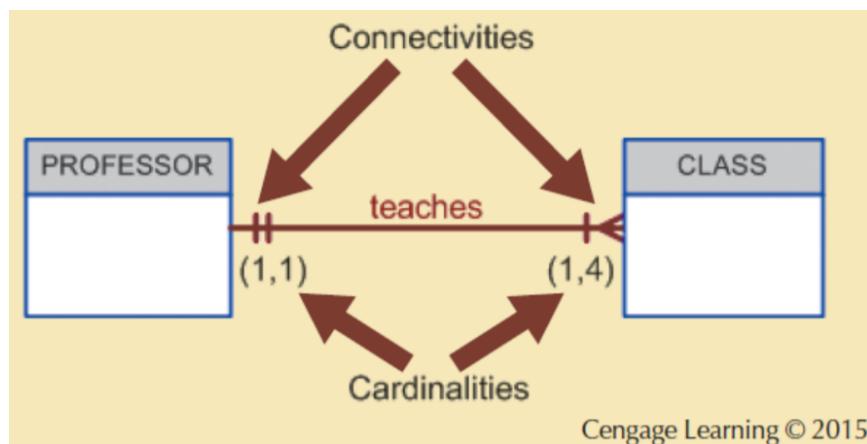
Связи между сущностями, которые всегда действуют в обоих направлениях

Участники: сущности, которые участвуют в отношениях

Связь: описывает классификацию отношений

Мощность: выражает минимальное и максимальное количество экземпляров сущности, связанных с одним экземпляром связанной сущности

258



259

Relationship Strength

Weak (non-identifying) relationship - Primary key of the related entity does not contain a primary key component of the parent entity

Strong (identifying) relationships - Primary key of the related entity contains a primary key component of the parent entity

Сила відношень

Слабкі (неідентифікуючі) відношення - первинний ключ пов'язаної сущності не містить компонента первинного ключа батьківської сущності

Міцні (ідентифікуючі) відношення - первинний ключ пов'язаної сущності містить компонент первинного ключа батьківської сущності

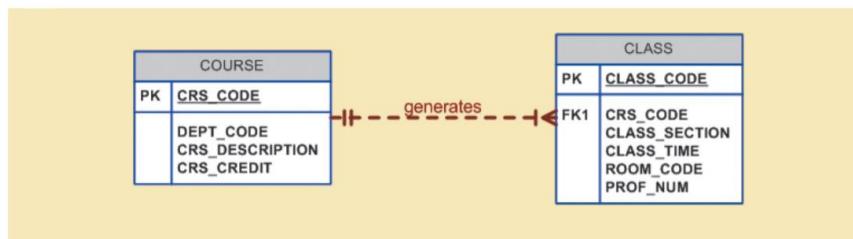
Сила отношений

Слабая (неидентифицирующая) связь - первичный ключ связанной сущности не содержит компонент первичного ключа родительской сущности

Сильная (идентифицирующие) связь - первичный ключ связанной сущности содержит компонент первичного ключа родительской сущности

260

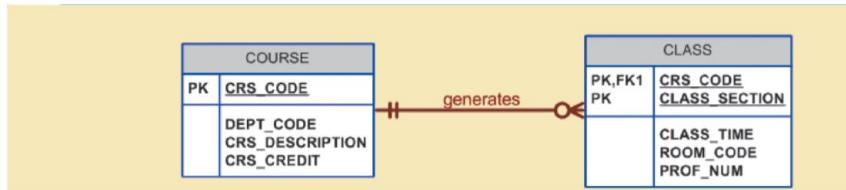
Non-Identifying



Cengage Learning © 2015

261

Identifying



Cengage Learning © 2015

262

Weak Entity

Conditions

- Existence-dependent
- Has a primary key that is partially or totally derived from parent entity in the relationship

Database designer determines whether an entity is weak based on business rules

Слабка сутність

Умови

- Залежна від існування
- Має первинний ключ, який частково або повністю походить від батьківської сутності у відношенні

Проектувальник баз даних визначає, чи є сутність слабкою на основі бізнес-правил

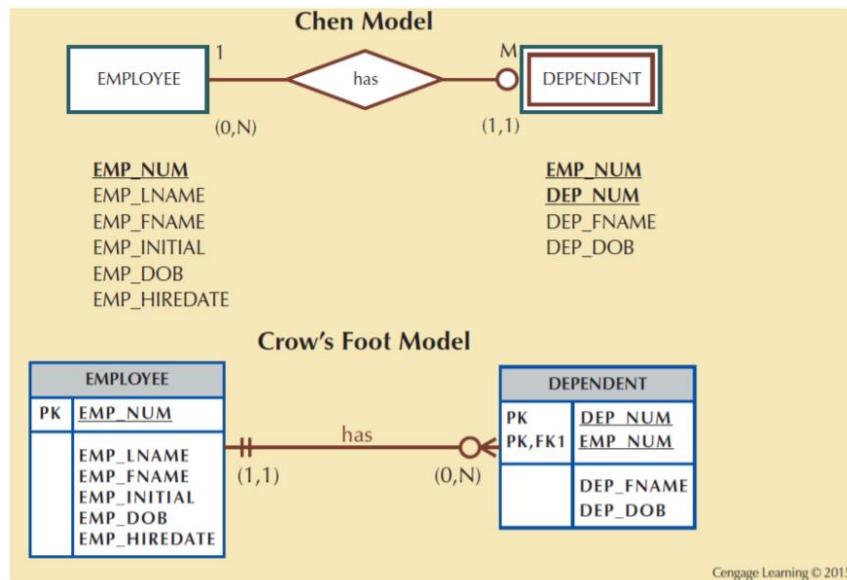
Слабая сущность

Условия

- Зависимая от существования
- Имеет первичный ключ, частично или полностью производный от родительской сущности в отношении

Проектировщик базы данных определяет, является ли объект слабым, на основе бизнес-правил

263



Cengage Learning © 2015

264

Table name: EMPLOYEE

Database name: Ch04_ShortCo

EMP_NUM	EMP_LNAME	EMP_FNAME	EMP_INITIAL	EMP_DOB	EMP_HIREDATE
1001	Callifante	Jeanine	J	12-Mar-64	25-May-97
1002	Smithson	William	K	23-Nov-70	28-May-97
1003	Washington	Herman	H	15-Aug-68	28-May-97
1004	Chen	Lydia	B	23-Mar-74	15-Oct-98
1005	Johnson	Melanie		28-Sep-66	20-Dec-98
1006	Ortega	Jorge	G	12-Jul-79	05-Jan-02
1007	O'Donnell	Peter	D	10-Jun-71	23-Jun-02
1008	Brzinski	Barbara	A	12-Feb-70	01-Nov-03

Table name: DEPENDENT

EMP_NUM	DEP_NUM	DEP_FNAME	DEP_DOB
1001	1	Annelise	05-Dec-97
1001	2	Jorge	30-Sep-02
1003	1	Suzanne	25-Jan-04
1006	1	Carlos	25-May-01
1008	1	Michael	19-Feb-95
1008	2	George	27-Jun-98
1008	3	Katherine	18-Aug-03

Cengage Learning © 2015

Relationship Participation

Optional participation

- One entity occurrence does not require a corresponding entity occurrence in a particular relationship

Mandatory participation

- One entity occurrence requires a corresponding entity occurrence in a particular relationship

Участь у відношеннях

Необов'язкова участь

- Входження однієї сущності не вимагає відповідного входження іншої сущності в певне відношення

Обов'язкова участь

- Входження однієї сущності вимагає відповідного входження іншої сущності в певне відношення

Участие в отношениях

Необязательное участие

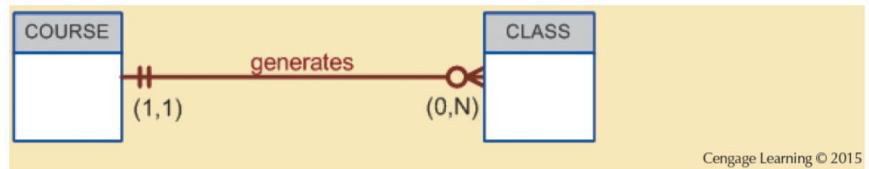
- Вхождение одной сущности не требует соответствующего вхождения другой сущности в конкретном отношении

Обязательное участие

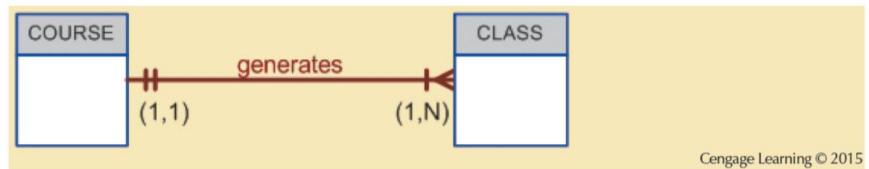
- Для экземпляра одной сущности требуется соответствующее вхождение экземпляра другой сущности в конкретном отношении

CROW'S FOOT SYMBOLS	CARDINALITY
	(0,N)
	(1,N)
	(1,1)
	(0,1)

267



268



269

Relationship Degree

Indicates the number of entities or participants associated with a relationship

Unary relationship: Association is maintained within a single entity

- Recursive relationship: Relationship exists between occurrences of the same entity set

Binary relationship: Two entities are associated

Ternary relationship: Three entities are associated

Ступінь відношень

Позначає кількість сущностей або учасників, пов'язаних із відношеннями

Унарні відношення: асоціація підтримується в межах єдиного цілого

- Рекурсивні відношення: існує взаємозв'язок між екземплярами одного і того ж набору сущностей

Бінарні відношення: пов'язані дві сущності

Тернарні відношення: пов'язані три сущності

Степень отношений

Указывает количество существ и или участников, связанных с отношениями

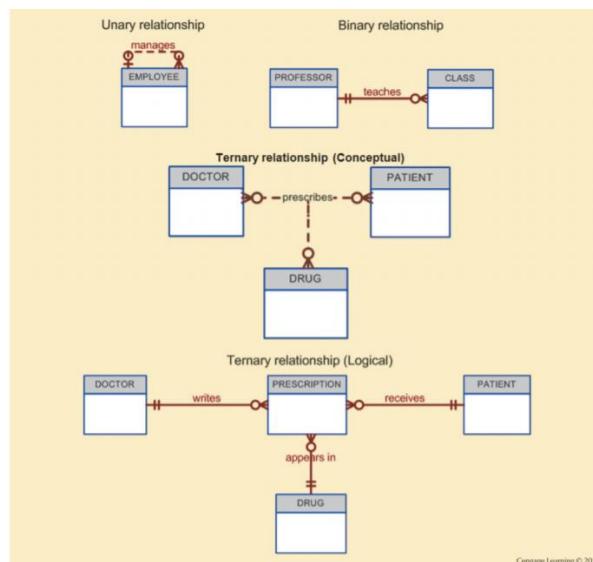
Унарные отношения: ассоциация поддерживается в рамках одного объекта

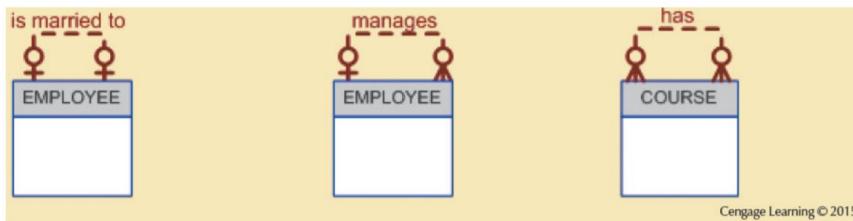
- Рекурсивная связь: существует связь между экземплярами одного и того же набора существ

Бинарные отношения: связаны две сущности

Тернарные отношения: связаны три сущности

270





Associative Entities

Also known as composite or bridge entities

Used to represent an M:N relationship between two or more entities

Is in a 1:M relationship with the parent entities

- Composed of the primary key attributes of each parent entity

May also contain additional attributes that play no role in connective process

Асоціативні сутності

Також відомі як композитні або зв'язуючі сутності

Використовуються для представлення взаємозв'язку M:N між двома або більше сутностями

Перебувають у співвідношенні 1:M з батьківськими сутностями

- Складається з атрибутів первинного ключа кожної батьківської сутності

Також може містити додаткові атрибути, які не відіграють жодної ролі в процесі зв'язку

Ассоциативные сущности

Также известны как составные или связывающие сущности

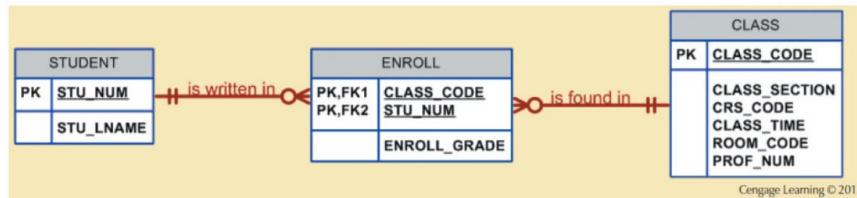
Используются для представления отношения M:N между двумя или более сущностями

Находятся в отношениях 1:M с родительскими сущностями

- Состоит из атрибутов первичного ключа каждой родительской сущности

Также может содержать дополнительные атрибуты, которые не играют роли в связывающем процессе

273



Cengage Learning © 2015

274

Table name: STUDENT		Database name: Ch04_CollegeTry	
STU_NUM			
321452 Bowser			
324257 Smithson			
Table name: ENROLL			
CLASS_CODE			
STU_NUM			
ENROLL_GRADE			
10014	321452 C		
10014	324257 B		
10018	321452 A		
10018	324257 B		
10021	321452 C		
10021	324257 C		
Table name: CLASS			
CLASS_CODE			
CRS_CODE			
CLASS_SECTION			
CLASS_TIME			
ROOM_CODE			
PROF_NUM			
10014	ACCT-211	3	TTh 2:30-3:45 p.m.
10018	CIS-220	2	MWF 9:00-9:50 a.m.
10021	QM-261	1	MWF 8:00-8:50 a.m.
			KLR211
			KLR200
			342
			114
			114

Cengage Learning © 2015

275

Developing an ER Diagram

- Create a detailed narrative of the organization's description of operations
- Identify business rules based on the descriptions
- Identify main entities and relationships from the business rules
- Develop the initial ERD
- Identify the attributes and primary keys that adequately describe entities
- Revise and review ERD

Розробка діаграми ЕР

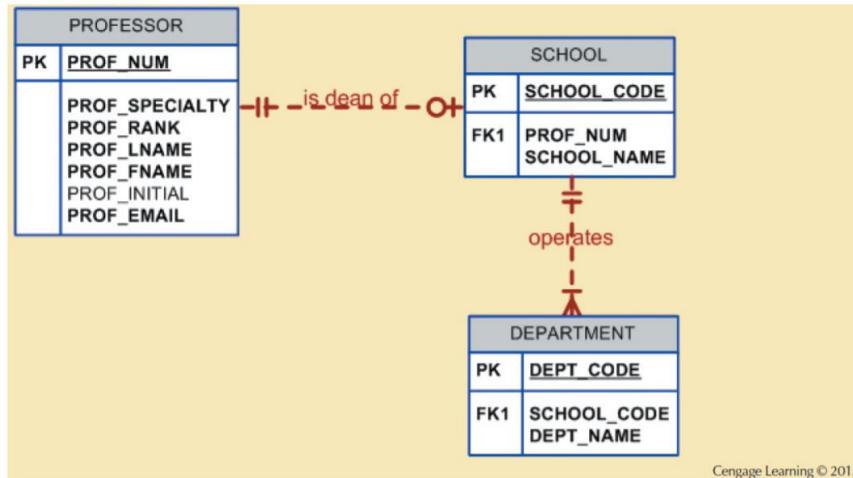
- Створіть детальний опис діяльності організації
- Визначте бізнес-правила на основі описів
- Визначте основні сущності та відношення з бізнес-правил
- Розробіть початкову ERD
- Визначте атрибути та первинні ключі, які адекватно описують сущності
- Перегляньте та перевірте ERD

276

Разработка диаграммы ЕР

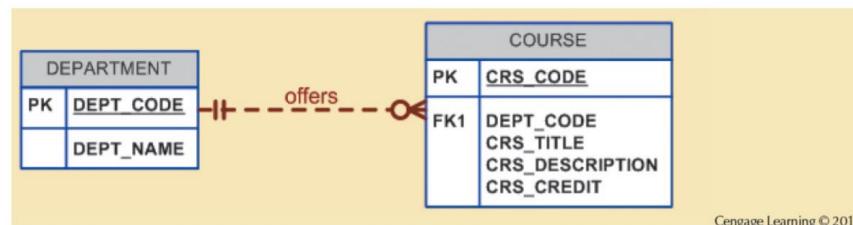
- Создайте детальное описание деятельности организации
- Определите бизнес-правила на основе описаний
- Определите основные сущности и отношения из бизнес-правил
- Разработайте начальную ERD
- Определите атрибуты и первичные ключи, которые адекватно описывают сущности
- Просмотрите и проверьте ERD

277



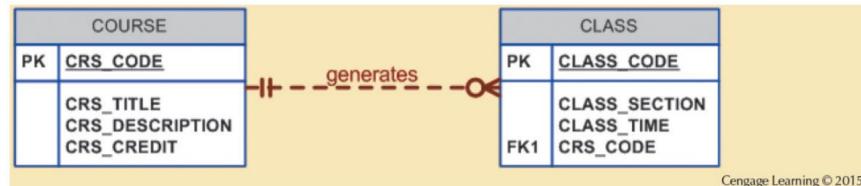
Cengage Learning © 2015

278



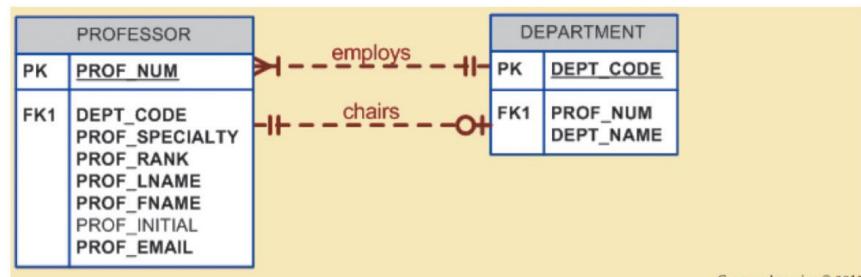
Cengage Learning © 2015

279



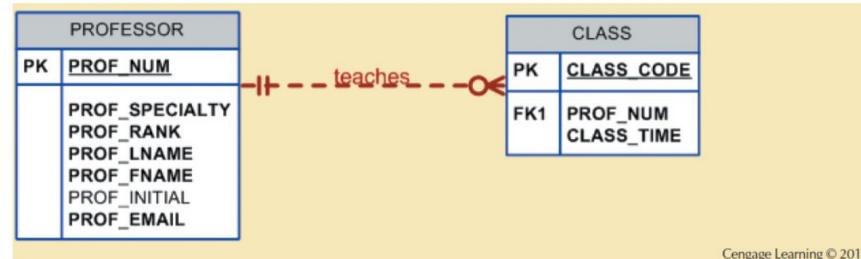
Cengage Learning © 2015

280



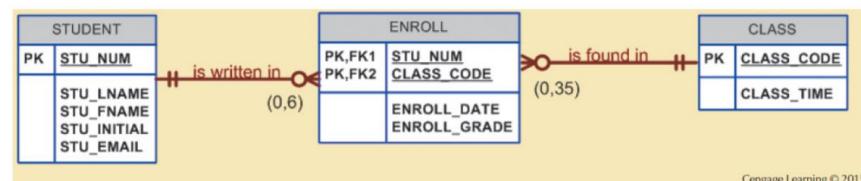
Cengage Learning © 2015

281



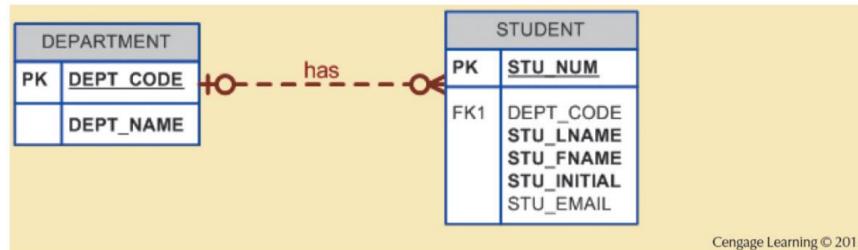
Cengage Learning © 2015

282



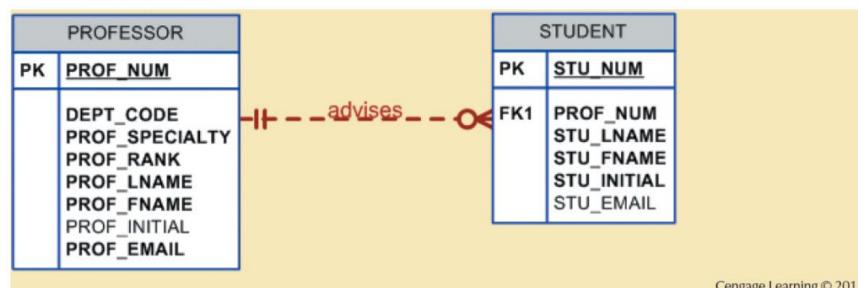
Cengage Learning © 2015

283



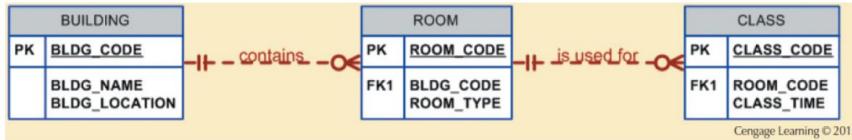
Cengage Learning © 2015

284



Cengage Learning © 2015

285



286

ENTITY	RELATIONSHIP	CONNECTIVITY	ENTITY
SCHOOL	operates	1:M	DEPARTMENT
DEPARTMENT	has	1:M	STUDENT
DEPARTMENT	employs	1:M	PROFESSOR
DEPARTMENT	offers	1:M	COURSE
COURSE	generates	1:M	CLASS
PROFESSOR	is dean of	1:1	SCHOOL
PROFESSOR	chairs	1:1	DEPARTMENT
PROFESSOR	teaches	1:M	CLASS
PROFESSOR	advises	1:M	STUDENT
STUDENT	enrolls in	M:N	CLASS
BUILDING	contains	1:M	ROOM
ROOM	is used for	1:M	CLASS

Note: ENROLL is the composite entity that implements the M:N relationship "STUDENT enrolls in CLASS."

Cengage Learning © 2015

287

- Database design must conform to design standards
- Need for high processing speed may limit the number and complexity of logically desirable relationships
- Need for maximum information generation may lead to loss of clean design structures and high transaction speed

- Дизайн бази даних повинен відповідати стандартам проектування
- Потреба у високій швидкості обробки може обмежити кількість і складність логічно бажаних відносин
- Потреба в максимальному формуванні інформації може привести до втрати «чистоти» спроектованих конструкцій та високої швидкості транзакцій

- Дизайн базы данных должен соответствовать стандартам проектирования
- Потребность в высокой скорости обработки может ограничить количество и сложность логически желаемых отношений
- Потребность в максимальном генерировании информации может привести к потере «чистых» спроектированных структур и высокой скорости транзакций

288

Контрольні питання

1. Діаграми потоків даних (DFD). Види та елементи.
2. Діаграми потоків даних (DFD). Правила моделювання.
3. Бізнес-правила. Типи.
4. Бізнес-правила. Кращі практики.
5. Перетворення бізнес-правил у компоненти моделі даних.
6. Стандартні концепції баз даних.
7. Модель сутність-зв'язок. Атрибути.
8. Модель сутність-зв'язок. Відношення.
9. Модель сутність-зв'язок. Асоціативні сутності.
10. Розробка діаграми ER.

289

Assessment questions

1. Data Flow Diagrams (DFD). Types and elements.
2. Data Flow Diagrams (DFD). Design rules.
3. Business rules. Types.
4. Business rules. Best practices.
5. Translating Business Rules into Data Model Components.
6. Standard Database Concepts.
7. Entity Relationship Model. Attributes.
8. Entity Relationship Model. Relationships.
9. Entity Relationship Model. Associative Entities.
10. Developing an ER Diagram.

290

Контрольные вопросы

1. Диаграммы потоков данных (DFD). Виды и элементы.
2. Диаграммы потоков данных (DFD). Правила моделирования.
3. Бизнес-правила. Типы.
4. Бизнес-правила. Лучшие практики.
5. Преобразование бизнес-правил в компоненты модели данных.
6. Стандартные концепции баз данных.
7. Модель сущность-связь. Атрибуты.
8. Модель сущность-связь. Отношения.
9. Модель сущность-связь. Ассоциативные сущности.
10. Разработка диаграммы ER.

291

Створення БД засобами мови SQL Creating a database using SQL language Создание БД средствами языка SQL

Тема 3
Topic 3

292

SQL DATA DEFINITION LANGUAGE (DDL)

293

SQL CREATE DATABASE

`CREATE DATABASE databasename;`

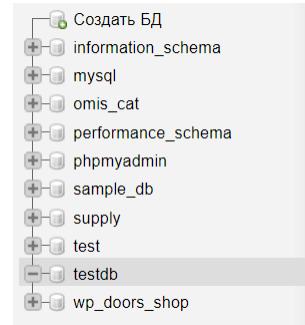
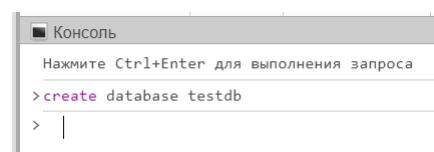
`CREATE DATABASE testDB;`

Make sure you have admin privilege before creating any database. Once a database is created, you can check it in the list of databases with the following SQL command: `SHOW DATABASES`.

Перед створенням будь-якої бази даних переконайтесь, що у вас є права адміністратора. Після створення бази даних ви можете перевірити її у списку баз даних за допомогою наступної команди SQL: `SHOW DATABASES`.

Перед созданием любой базы данных убедитесь, что у вас есть права администратора. После создания базы данных вы можете проверить ее в списке баз данных с помощью следующей команды SQL: `SHOW DATABASES`.

294



295

SQL DROP DATABASE

`DROP DATABASE databasename;`

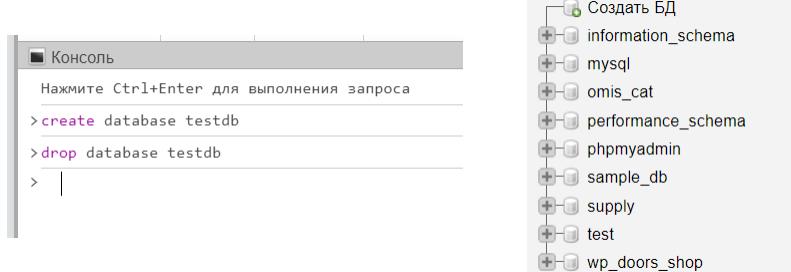
`DROP DATABASE testDB;`

Be careful before dropping a database. Deleting a database will result in loss of complete information stored in the database!

Будьте обережні, перш ніж видаляти базу даних. Видалення бази даних призведе до втрати повної інформації, що зберігається в базі даних!

Будьте осторожны перед удалением базы данных. Удаление базы данных приведет к потере всей информации, хранящейся в базе данных!

296



SQL CREATE TABLE

```
CREATE TABLE table_name (
    column1 datatype,
    column2 datatype,
    column3 datatype,
    ...
);
```

The CREATE TABLE statement is used to create a new table in a database. The column parameters specify the names of the columns of the table. The datatype parameter specifies the type of data the column can hold (e.g. varchar, integer, date, etc.).

Оператор CREATE TABLE використовується для створення нової таблиці в базі даних. Параметри стовпців визначають імена стовпців таблиці. Параметр типу даних визначає тип даних, які може містити стовпець (наприклад, varchar, ціле число, дата тощо).

Оператор CREATE TABLE используется для создания новой таблицы в базе данных. Параметры столбца определяют имена столбцов таблицы. Параметр datatype указывает тип данных, которые столбец может содержать (например, varchar, integer, date и т.д.).

The following example creates a table called "Persons" that contains five columns: PersonID, LastName, FirstName, Address, and City:

У наступному прикладі створюється таблиця "Persons", яка містить п'ять стовпців: PersonID, LastName, FirstName, Address і City:

В следующем примере создается таблица с именем «Persons», которая содержит пять столбцов: PersonID, LastName, FirstName, Address и City:

```
CREATE TABLE Persons (
    PersonID int,
    LastName varchar(255),
    FirstName varchar(255),
    Address varchar(255),
    City varchar(255)
);
```

299

```
>create database testdb
>use testdb
>create table persons ( personid int, lastname varchar(255), firstname varchar(255), address varchar(255), city varchar(255) )
```

#	Имя	Тип	Сравнение	Атрибуты	Null	По умолчанию
1	personid	int(11)		Да	NULL	
2	lastname	varchar(255)	latin1_swedish_ci	Да	NULL	
3	firstname	varchar(255)	latin1_swedish_ci	Да	NULL	
4	address	varchar(255)	latin1_swedish_ci	Да	NULL	
5	city	varchar(255)	latin1_swedish_ci	Да	NULL	

300

- The PersonID column is of type int and will hold an integer.
- The LastName, FirstName, Address, and City columns are of type varchar and will hold characters, and the maximum length for these fields is 255 characters.
- Стовпець PersonID має тип int і міститиме ціле число.
- Стовпці LastName, FirstName, Address та City мають тип varchar і містять символи, а максимальна довжина цих полів 255 символів.
- Столбец PersonID имеет тип int и содержит целое число.
- Столбцы LastName, FirstName, Address и City относятся к типу varchar и содержат символы, а максимальная длина этих полей составляет 255 символов.

301

A copy of an existing table can also be created using CREATE TABLE. The new table gets the same column definitions. All columns or specific columns can be selected.

If you create a new table using an existing table, the new table will be filled with the existing values from the old table.

Копію існуючої таблиці також можна створити за допомогою CREATE TABLE.

Нова таблиця отримує ті самі визначення стовпців. Можна вибрати всі стовпці або окремі стовпці.

Якщо ви створюєте нову таблицю з використанням існуючої таблиці, нова таблиця буде заповнена наявними значеннями зі старої таблиці.

Копию существующей таблицы также можно создать с помощью CREATE TABLE.

Новая таблица получит те же определения столбцов. Можно выбрать все столбцы или определенные столбцы.

Если вы создаете новую таблицу с использованием существующей таблицы, новая таблица будет заполнена существующими значениями из старой таблицы.

302

```
CREATE TABLE new_table_name AS
  SELECT column1, column2, ...
    FROM existing_table_name
   WHERE ....;
```

- The following SQL creates a new table called "TestTables" (which is a copy of the "Customers" table):
- Наступний SQL створює нову таблицю під назвою "TestTables" (яка є копією таблиці "Клієнти"):
- Следующий SQL создает новую таблицу с именем «TestTables» (которая является копией таблицы «Customers»):

```
CREATE TABLE TestTable AS
  SELECT customername, contactname
    FROM customers;
```

303

```
>create table persons ( personid int, lastname varchar(255), firstname varchar(255), address varchar(255), city varchar(255) )
>create table testtable as select lastname,firstname from persons
```

The screenshot shows the MySQL Workbench interface. On the left, there's a tree view of databases: Создать БД, information_schema, mysql, omis_cat, performance_schema, phpmyadmin, sample_db, supply, test, testdb (which contains Новая, persons, and testitable), and wp_doors_shop. On the right, a table definition for 'persons' is shown in a grid:

#	Имя	Тип	Сравнение	Атрибуты	Null	По умолчанию
1	lastname	varchar(255)	latin1_swedish_ci		Да	NULL
2	firstname	varchar(255)	latin1_swedish_ci		Да	NULL

304

SQL DROP TABLE

- The DROP TABLE statement is used to drop an existing table in a database.
- Оператор DROP TABLE використовується для видалення існуючої таблиці в базі даних.
- Оператор DROP TABLE используется для удаления существующей таблицы в базе данных.

DROP TABLE *table_name*;

- Be careful before dropping a table. Deleting a table will result in loss of complete information stored in the table!
- Будьте обережні, перш ніж видаляти таблицю. Видалення таблиці призведе до втрати повної інформації, що зберігається в таблиці!
- Будьте осторожны, прежде чем удалять таблицу. Удаление таблицы приведет к потере всей информации, хранящейся в таблице!

305

- The following SQL statement drops the existing table "Shippers":
- Наступний оператор SQL видаляє існуючу таблицю "Shippers":
- Следующий оператор SQL удаляет существующую таблицу «Shippers»:

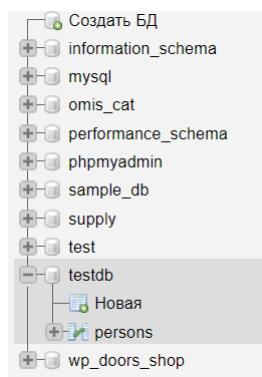
DROP TABLE Shippers;

- The TRUNCATE TABLE statement is used to delete the data inside a table, but not the table itself.
- Оператор TRUNCATE TABLE використовується для видалення даних всередині таблиці, але не самої таблиці.
- Оператор TRUNCATE TABLE используется для удаления данных внутри таблицы, но не самой таблицы.

TRUNCATE TABLE *table_name*;

306

```
> create table testtable as select lastname, firstname from persons
> drop table testtable
```



307

SQL ALTER TABLE

The ALTER TABLE statement is used to add, delete, or modify columns in an existing table.

The ALTER TABLE statement is also used to add and drop various constraints on an existing table.

Оператор ALTER TABLE використовується для додавання, видалення або модифікації стовпців у існуючій таблиці.

Оператор ALTER TABLE також використовується для додавання та видалення різних обмежень існуючої таблиці.

Оператор ALTER TABLE используется для добавления, удаления или изменения столбцов в существующей таблице.

Оператор ALTER TABLE также используется для добавления и удаления различных ограничений существующей таблицы.

308

- To add a column in a table, use the following syntax:
- Щоб додати стовпець у таблицю, використовуйте такий синтаксис:
- Чтобы добавить столбец в таблицу, используйте следующий синтаксис:

```
ALTER TABLE table_name
ADD column_name datatype;
```

- The following SQL adds an "Email" column to the "Customers" table:
- Наступний SQL додає стовпець "Електронна пошта" до таблиці "Клієнти":
- Следующий SQL добавляет столбец «Электронная почта» в таблицу «Клиенты»:

```
ALTER TABLE Customers
ADD Email varchar(255);
```

309

```

XAMPP for Windows - mysql -u root -p
MariaDB [testdb]> alter table persons add email varchar(100);
Query OK, 0 rows affected (0.108 sec)
Records: 0  Duplicates: 0  Warnings: 0

MariaDB [testdb]> describe persons;
+-----+-----+-----+-----+-----+-----+
| Field | Type   | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| personid | int(11) | YES |   | NULL    |       |
| lastname | varchar(255) | YES |   | NULL    |       |
| firstname | varchar(255) | YES |   | NULL    |       |
| address | varchar(255) | YES |   | NULL    |       |
| city | varchar(255) | YES |   | NULL    |       |
| email | varchar(100) | YES |   | NULL    |       |
+-----+-----+-----+-----+-----+-----+
6 rows in set (0.051 sec)

MariaDB [testdb]>

```

310

- To change the data type of a column in a table, use the following syntax:
- Щоб змінити тип даних стовпця в таблиці, використовуйте такий синтаксис:
- Чтобы изменить тип данных столбца в таблице, используйте следующий синтаксис:

MySQL:

```

ALTER TABLE table_name
MODIFY COLUMN column_name datatype;

```

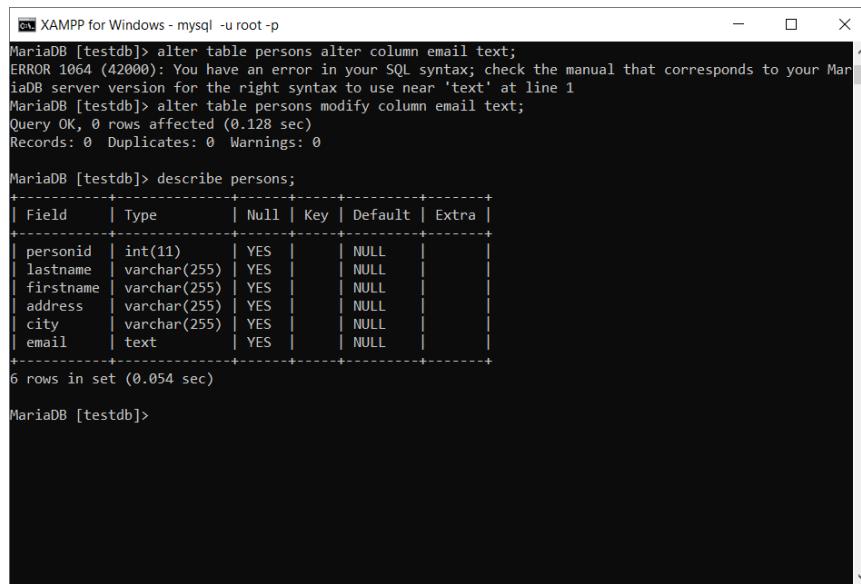
SQL Server:

```

ALTER TABLE table_name
ALTER COLUMN column_name datatype;

```

311



```

XAMPP for Windows - mysql -u root -p
MariaDB [testdb]> alter table persons alter column email text;
ERROR 1064 (42000): You have an error in your SQL syntax; check the manual that corresponds to your MariaDB server version for the right syntax to use near 'text' at line 1
MariaDB [testdb]> alter table persons modify column email text;
Query OK, 0 rows affected (0.128 sec)
Records: 0  Duplicates: 0  Warnings: 0

MariaDB [testdb]> describe persons;
+-----+-----+-----+-----+-----+-----+
| Field | Type  | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| personid | int(11) | YES |   | NULL    |       |
| lastname | varchar(255) | YES |   | NULL    |       |
| firstname | varchar(255) | YES |   | NULL    |       |
| address | varchar(255) | YES |   | NULL    |       |
| city | varchar(255) | YES |   | NULL    |       |
| email | text    | YES |   | NULL    |       |
+-----+-----+-----+-----+-----+-----+
6 rows in set (0.054 sec)

MariaDB [testdb]>

```

312

- Next, we want to delete the column named "DateOfBirth" in the "Persons" table.
- We use the following SQL statement:

- Далі ми хочемо видалити стовпець "DateOfBirth" у таблиці "Persons".
- Ми використовуємо наступний оператор SQL:

- Затем мы хотим удалить столбец с именем «DateOfBirth» в таблице «Persons».
- Мы используем следующий оператор SQL:

```

ALTER TABLE Persons
DROP COLUMN DateOfBirth;

```

313

```

XAMPP for Windows - mysql -u root -p

MariaDB [testdb]> alter table persons drop column email;
Query OK, 0 rows affected (0.037 sec)
Records: 0  Duplicates: 0  Warnings: 0

MariaDB [testdb]> describe persons;
+-----+-----+-----+-----+-----+
| Field | Type   | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+
| personid | int(11) | YES |   | NULL    |       |
| lastname | varchar(255) | YES |   | NULL    |       |
| firstname | varchar(255) | YES |   | NULL    |       |
| address  | varchar(255) | YES |   | NULL    |       |
| city     | varchar(255) | YES |   | NULL    |       |
+-----+-----+-----+-----+-----+
5 rows in set (0.056 sec)

MariaDB [testdb]>

```

314

- **SQL constraints** are used to specify rules for data in a table.
- Constraints can be specified when the table is created with the CREATE TABLE statement, or after the table is created with the ALTER TABLE statement.
- **Обмеження SQL** використовуються для визначення правил для даних у таблиці.
- Обмеження можна вказати, коли таблиця створюється за допомогою оператора CREATE TABLE, або після створення таблиці за допомогою оператора ALTER TABLE.
- **Ограничения SQL** используются для определения правил для данных в таблице.
- Ограничения могут быть указаны при создании таблицы с помощью оператора CREATE TABLE или после создания таблицы с помощью оператора ALTER TABLE.

```

CREATE TABLE table_name (
  column1 datatype constraint,
  column2 datatype constraint,
  column3 datatype constraint,
  ...
);

```

315

SQL constraints are used to specify rules for the data in a table.

Constraints are used to limit the type of data that can go into a table. This ensures the accuracy and reliability of the data in the table. If there is any violation between the constraint and the data action, the action is aborted.

Constraints can be column level or table level. Column level constraints apply to a column, and table level constraints apply to the whole table.

Обмеження SQL використовуються для визначення правил для даних у таблиці.

Обмеження використовуються для обмеження типу даних, які можуть потрапляти в таблицю. Це забезпечує точність і надійність даних у таблиці. Якщо між обмеженням та дією з даними є порушення, дія переривається.

Обмеження можуть бути на рівні стовпців або на рівні таблиці. Обмеження рівня стовпця застосовуються до стовпця, а обмеження рівня таблиці застосовуються до всієї таблиці.

Ограничения SQL используются для определения правил для данных в таблице.

Ограничения используются для ограничения типа данных, которые могут входить в таблицу. Это обеспечивает точность и надежность данных в таблице. Если есть какое-либо нарушение между ограничением и действием с данными, действие прерывается.

Ограничения могут быть на уровне столбца или таблицы. Ограничения уровня столбца применяются к столбцу, а ограничения уровня таблицы применяются ко всей таблице.

316

The following constraints are commonly used in SQL:

- NOT NULL - Ensures that a column cannot have a NULL value
- UNIQUE - Ensures that all values in a column are different
- PRIMARY KEY - A combination of a NOT NULL and UNIQUE. Uniquely identifies each row in a table
- FOREIGN KEY - Uniquely identifies a row/record in another table
- CHECK - Ensures that all values in a column satisfies a specific condition
- DEFAULT - Sets a default value for a column when no value is specified
- INDEX - Used to create and retrieve data from the database very quickly

Наступні обмеження зазвичай використовуються в SQL:

- NOT NULL - гарантує, що стовпець не може мати значення NULL
- UNIQUE - гарантує, що всі значення в стовпці різні
- PRIMARY KEY - комбінація NOT NULL та UNIQUE. Унікально ідентифікує кожен рядок у таблиці
- FOREIGN KEY - однозначно ідентифікує рядок / запис в іншій таблиці
- CHECK - гарантує, що всі значення в стовпці відповідають певній умові
- DEFAULT - Встановлює значення за замовчуванням для стовпця, коли значення не вказано
- INDEX - Використовується для створення та отримання даних з бази даних дуже швидко

317

В SQL обычно используются следующие ограничения:

- NOT NULL - гарантирует, что столбец не может иметь значение NULL
- UNIQUE - гарантирует, что все значения в столбце различны.
- PRIMARY KEY - комбинация NOT NULL и UNIQUE. Однозначно идентифицирует каждую строку в таблице
- FOREIGN KEY - однозначно определяет строку / запись в другой таблице
- CHECK - проверяет, что все значения в столбце удовлетворяют определенному условию
- DEFAULT - устанавливает значение по умолчанию для столбца, если значение не указано
- INDEX - используется для очень быстрого создания и извлечения данных из базы данных

318

SQL NOT NULL

By default, a column can hold NULL values.

The NOT NULL constraint enforces a column to NOT accept NULL values. This enforces a field to always contain a value, which means that you cannot insert a new record, or update a record without adding a value to this field.

За замовчуванням стовпець може містити NULL значення.
Обмеження NOT NULL примушує стовпець НЕ приймати значення NULL.

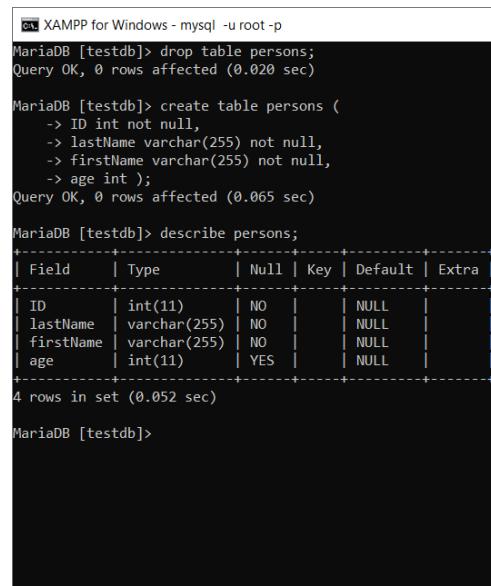
Це примушує поле завжди містити значення, а це означає, що ви не можете вставити новий запис або оновити запис, не додавши значення в це поле.

По умолчанию столбец может содержать значения NULL.
Ограничение NOT NULL заставляет столбец НЕ принимать значения NULL.

Это заставляет поле всегда содержать значение, что означает, что вы не можете вставить новую запись или обновить запись без добавления значения в это поле.

- The following SQL ensures that the "ID", "LastName", and "FirstName" columns will NOT accept NULL values when the "Persons" table is created:
- Наступний SQL гарантує, що стовпці "ID", "LastName" та "FirstName" НЕ прийматимуть значення NULL при створенні таблиці "Persons":
- Следующий SQL гарантирует, что столбцы «ID», «LastName» и «FirstName» НЕ будут принимать значения NULL при создании таблицы «Persons»:

```
CREATE TABLE Persons (
    ID int NOT NULL,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255) NOT NULL,
    Age int
);
```



XAMPP for Windows - mysql -u root -

```
MariaDB [testdb]> drop table persons;
Query OK, 0 rows affected (0.020 sec)

MariaDB [testdb]> create table persons (
    -> ID int not null,
    -> lastName varchar(255) not null,
    -> firstName varchar(255) not null,
    -> age int );
Query OK, 0 rows affected (0.065 sec)

MariaDB [testdb]> describe persons;
+-----+-----+-----+-----+-----+
| Field | Type  | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+
| ID   | int(11) | NO  |   | NULL   |       |
| lastName | varchar(255) | NO  |   | NULL   |       |
| firstName | varchar(255) | NO  |   | NULL   |       |
| age  | int(11) | YES |   | NULL   |       |
+-----+-----+-----+-----+-----+
4 rows in set (0.052 sec)

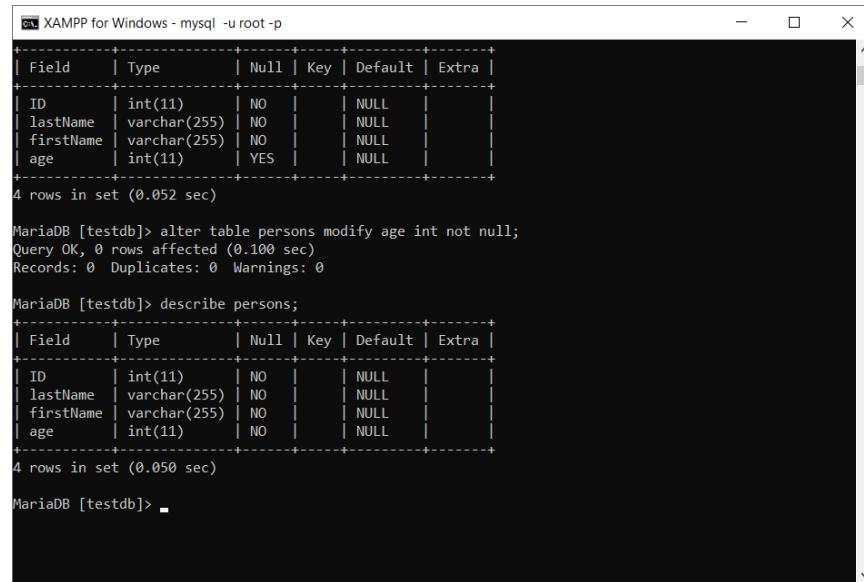
MariaDB [testdb]>
```

321

- To create a NOT NULL constraint on the "Age" column when the "Persons" table is already created, use the following SQL:
- Щоб створити обмеження NOT NULL у стовпці "Age", коли таблиця "Persons" уже створена, використовуйте такий SQL:
- Чтобы создать ограничение NOT NULL для столбца «Age», когда таблица «Persons» уже создана, используйте следующий SQL:

```
ALTER TABLE Persons
MODIFY Age int NOT NULL;
```

322



The screenshot shows a terminal window titled "XAMPP for Windows - mysql -u root -p". It displays the following MySQL session:

```
MariaDB [testdb]> alter table persons modify age int not null;
Query OK, 0 rows affected (0.100 sec)
Records: 0  Duplicates: 0  Warnings: 0

MariaDB [testdb]> describe persons;
+-----+-----+-----+-----+-----+
| Field | Type   | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+
| ID    | int(11) | NO   |   | NULL    |       |
| lastName | varchar(255) | NO   |   | NULL    |       |
| firstName | varchar(255) | NO   |   | NULL    |       |
| age   | int(11)  | YES  |   | NULL    |       |
+-----+-----+-----+-----+-----+
4 rows in set (0.052 sec)
```

After running the ALTER TABLE command, the DESCRIBE output shows that the 'age' column now has a NOT NULL constraint applied.

SQL UNIQUE

The UNIQUE constraint ensures that all values in a column are different. Both the UNIQUE and PRIMARY KEY constraints provide a guarantee for uniqueness for a column or set of columns.

A PRIMARY KEY constraint automatically has a UNIQUE constraint. However, you can have many UNIQUE constraints per table, but only one PRIMARY KEY constraint per table.

Обмеження UNIQUE гарантує, що всі значення в стовпці різні. Обмеження UNIQUE та PRIMARY KEY забезпечують унікальність стовпця або набору стовпців.

Обмеження PRIMARY KEY автоматично має обмеження UNIQUE.

Однак ви можете мати багато обмежень UNIQUE для таблиці, але лише одне обмеження PRIMARY KEY для таблиці.

Ограничение UNIQUE гарантирует, что все значения в столбце различны. Ограничения UNIQUE и PRIMARY KEY обеспечивают уникальность столбца или набора столбцов.

Ограничение PRIMARY KEY автоматически имеет ограничение UNIQUE.

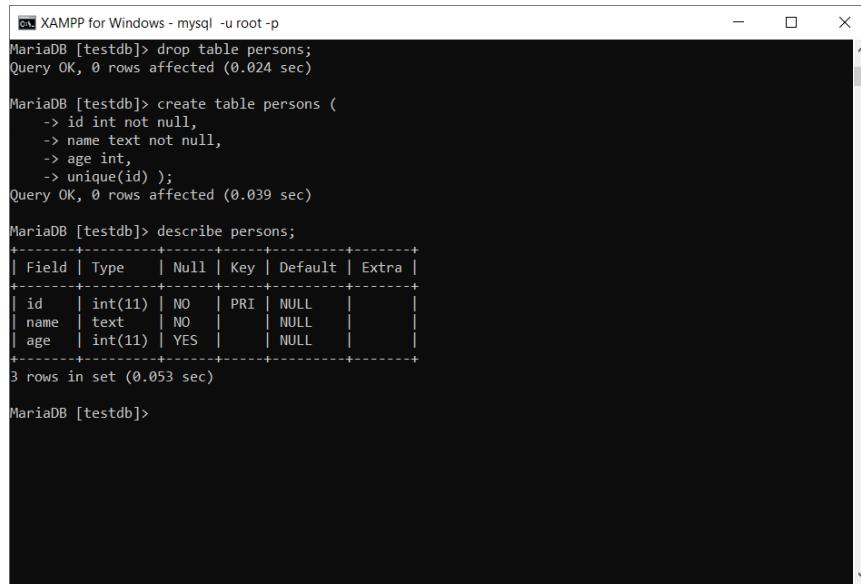
Однако у вас может быть много ограничений UNIQUE для каждой таблицы, но только одно ограничение PRIMARY KEY для каждой таблицы.

- The following SQL creates a UNIQUE constraint on the "ID" column when the "Persons" table is created:
- Наступний SQL створює UNIQUE обмеження для стовпця "ID", коли створюється таблиця "Persons":
- Следующий SQL создает ограничение UNIQUE для столбца «ID» при создании таблицы «Persons»:

MySQL:

```
CREATE TABLE Persons (
    ID int NOT NULL,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int,
    UNIQUE (ID)
);
```

325



```

XAMPP for Windows - mysql -u root -p
MariaDB [testdb]> drop table persons;
Query OK, 0 rows affected (0.024 sec)

MariaDB [testdb]> create table persons (
    -> id int not null,
    -> name text not null,
    -> age int,
    -> unique(id) ;
Query OK, 0 rows affected (0.039 sec)

MariaDB [testdb]> describe persons;
+-----+-----+-----+-----+-----+
| Field | Type   | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+
| id   | int(11) | NO   | PRI | NULL    |       |
| name | text    | NO   |     | NULL    |       |
| age  | int(11) | YES  |     | NULL    |       |
+-----+-----+-----+-----+-----+
3 rows in set (0.053 sec)

MariaDB [testdb]>

```

326

SQL Server:

```

CREATE TABLE Persons (
    ID int NOT NULL UNIQUE,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int
);

```

- To name a UNIQUE constraint, and to define a UNIQUE constraint on multiple columns, use the following SQL syntax:
- Щоб назвати UNIQUE обмеження та визначити UNIQUE обмеження для декількох стовпців, використовуйте такий синтаксис SQL:
- Чтобы назвать ограничение UNIQUE и определить ограничение UNIQUE для нескольких столбцов, используйте следующий синтаксис SQL:

```

CREATE TABLE Persons (
    ID int NOT NULL,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int,
    CONSTRAINT UC_Person UNIQUE (ID,LastName)
);

```

327

```

XAMPP for Windows - mysql -u root -p
MariaDB [testdb]> create table persons (
    -> id int not null,
    -> name text not null,
    -> age int,
    -> constraint person_id unique (id) );
Query OK, 0 rows affected (0.064 sec)

MariaDB [testdb]> describe persons;
+-----+-----+-----+-----+
| Field | Type   | Null | Key | Default | Extra |
+-----+-----+-----+-----+
| id    | int(11)| NO   | PRI | NULL    |       |
| name  | text    | NO   |     | NULL    |       |
| age   | int(11)| YES  |     | NULL    |       |
+-----+-----+-----+-----+
3 rows in set (0.059 sec)

MariaDB [testdb]> show create table persons;
+-----+
| Table | Create Table
+-----+
| persons | CREATE TABLE `persons` (
  `id` int(11) NOT NULL,
  `name` text NOT NULL,
  `age` int(11) DEFAULT NULL,
  UNIQUE KEY `person_id` (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1 |
+-----+
1 row in set (0.000 sec)

```

328

- To create a UNIQUE constraint on the "ID" column when the table is already created, use the following SQL:
- Щоб створити UNIQUE обмеження в стовіці "ID", коли таблиця вже створена, використовуйте такий SQL:
- Чтобы создать ограничение UNIQUE для столбца «ID», когда таблица уже создана, используйте следующий SQL:

`ALTER TABLE Persons
ADD UNIQUE (ID);`

- To name a UNIQUE constraint, and to define a UNIQUE constraint on multiple columns, use the following SQL syntax:
- Щоб назвати UNIQUE обмеження та визначити UNIQUE обмеження для декількох стовіців, використовуйте такий синтаксис SQL:
- Чтобы назвать ограничение UNIQUE и определить ограничение UNIQUE для нескольких столбцов, используйте следующий синтаксис SQL:

`ALTER TABLE Persons
ADD CONSTRAINT UC_Person UNIQUE (ID,LastName);`

329

```

XAMPP for Windows - mysql -u root -p
MariaDB [testdb]> alter table persons add constraint person_uname unique (name);
Query OK, 0 rows affected (0.137 sec)
Records: 0  Duplicates: 0  Warnings: 0

MariaDB [testdb]> describe persons;
+-----+-----+-----+-----+-----+
| Field | Type   | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+
| id    | int(11) | NO   | PRI | NULL    |       |
| name  | text    | NO   | UNI | NULL    |       |
| age   | int(11) | YES  |     | NULL    |       |
+-----+-----+-----+-----+-----+
3 rows in set (0.052 sec)

MariaDB [testdb]> show create table persons;
+-----+-----+
| Table | Create Table
+-----+-----+
| persons | CREATE TABLE `persons` (
  `id` int(11) NOT NULL,
  `name` text NOT NULL,
  `age` int(11) DEFAULT NULL,
  UNIQUE KEY `person_id` (`id`),
  UNIQUE KEY `person_uname` (`name`) USING HASH
) ENGINE=InnoDB DEFAULT CHARSET=latin1 |
+-----+-----+

```

330

- To drop a UNIQUE constraint, use the following SQL:
- Щоб видалити UNIQUE обмеження, використовуйте такий SQL:
- Чтобы удалить ограничение UNIQUE, используйте следующий SQL:

MySQL:

```
ALTER TABLE Persons
DROP INDEX UC_Person;
```

SQL Server:

```
ALTER TABLE Persons
DROP CONSTRAINT UC_Person;
```

331

```
c:\ XAMPP for Windows - mysql -u root -p

MariaDB [testdb]> describe persons;
+-----+-----+-----+-----+-----+
| Field | Type   | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+
| id    | int(11) | NO   | PRI | NULL    |       |
| name  | text    | NO   | UNI | NULL    |       |
| age   | int(11) | YES  |     | NULL    |       |
+-----+-----+-----+-----+-----+
3 rows in set (0.034 sec)

MariaDB [testdb]> alter table persons
    > drop index person_uname;
Query OK, 0 rows affected (0.049 sec)
Records: 0  Duplicates: 0  Warnings: 0

MariaDB [testdb]> describe persons;
+-----+-----+-----+-----+-----+
| Field | Type   | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+
| id    | int(11) | NO   | PRI | NULL    |       |
| name  | text    | NO   | UNI | NULL    |       |
| age   | int(11) | YES  |     | NULL    |       |
+-----+-----+-----+-----+-----+
3 rows in set (0.034 sec)

MariaDB [testdb]>
```

332

SQL PRIMARY KEY

The PRIMARY KEY constraint uniquely identifies each record in a table. Primary keys must contain UNIQUE values, and cannot contain NULL values. A table can have only ONE primary key; and in the table, this primary key can consist of single or multiple columns (fields).

Обмеження PRIMARY KEY однозначно ідентифікує кожен запис у таблиці. Первінні ключі повинні містити УНІКАЛЬНІ значення і не можуть містити NULL значень.

Таблиця може мати лише ОДИН первинний ключ; а в таблиці цей первинний ключ може складатися з одного або декількох стовпців (полів).

Ограничение PRIMARY KEY однозначно идентифицирует каждую запись в таблице.

Первичные ключи должны содержать УНИКАЛЬНЫЕ значения и не могут содержать значения NULL.

Таблица может иметь только ОДИН первичный ключ; а в таблице этот первичный ключ может состоять из одного или нескольких столбцов (полей).

333

- The following SQL creates a PRIMARY KEY on the "ID" column when the "Persons" table is created:
- Наступний SQL створює PRIMARY KEY у стовпці "ID", коли створюється таблиця "Persons":
- Следующий SQL создает PRIMARY KEY в столбце «ID» при создании таблицы «Persons»:

MySQL:

```
CREATE TABLE Persons (
    ID int NOT NULL,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int,
    PRIMARY KEY (ID)
);
```

SQL Server:

```
CREATE TABLE Persons (
    ID int NOT NULL PRIMARY KEY,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int
);
```

334

```
c:\ XAMPP for Windows - mysql -u root -p
MariaDB [testdb]> create table persons (
-> id int not null,
-> name text not null,
-> age int,
-> primary key (id));
Query OK, 0 rows affected (0.070 sec)

MariaDB [testdb]> describe persons;
+-----+-----+-----+-----+
| Field | Type  | Null | Key  |
+-----+-----+-----+-----+
| id   | int(11)| NO   | PRI  |
| name | text   | NO   |       |
| age  | int(11)| YES  |       |
+-----+-----+-----+-----+
3 rows in set (0.063 sec)

MariaDB [testdb]> -
```

335

- To allow naming of a PRIMARY KEY constraint, and for defining a PRIMARY KEY constraint on multiple columns, use the following SQL syntax:
- Щоб дозволити іменування обмеження PRIMARY KEY та для визначення обмеження PRIMARY KEY для декількох стовпців, використовуйте такий синтаксис SQL:
- Чтобы разрешить именование ограничения PRIMARY KEY и для определения ограничения PRIMARY KEY для нескольких столбцов, используйте следующий синтаксис SQL:

```
CREATE TABLE Persons (
    ID int NOT NULL,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int,
    CONSTRAINT PK_Person PRIMARY KEY (ID,LastName)
);
```

336

```
c:\XAMPP for Windows - mysql -u root -p
MariaDB [testdb]> create table persons (
->     id int not null,
->     name text not null,
->     age int,
->     constraint persons_pk primary key (id) );
Query OK, 0 rows affected, 1 warning (0.061 sec)

MariaDB [testdb]> show create table persons;
+-----+
| Table | Create Table
+-----+
| persons | CREATE TABLE `persons` (
`id` int(11) NOT NULL,
`name` text NOT NULL,
`age` int(11) DEFAULT NULL,
PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1 |
+-----+
1 row in set (0.002 sec)

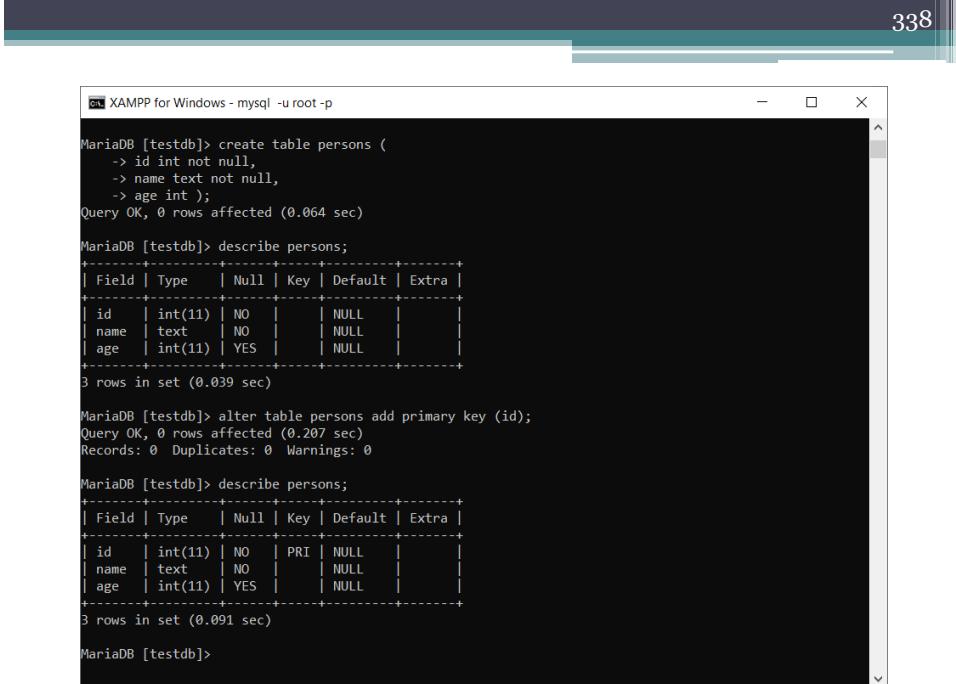
MariaDB [testdb]> describe persons;
+-----+-----+-----+-----+-----+
| Field | Type   | Null | Key  | Default | Extra |
+-----+-----+-----+-----+-----+
| id   | int(11) | NO   | PRI  | NULL    |       |
| name | text    | NO   |      | NULL    |       |
| age  | int(11) | YES  |      | NULL    |       |
+-----+-----+-----+-----+-----+
3 rows in set (0.085 sec)
```

- To create a PRIMARY KEY constraint on the "ID" column when the table is already created, use the following SQL:
- Щоб створити обмеження PRIMARY KEY у стовпці "ID", коли таблиця вже створена, використовуйте такий SQL:
- Чтобы создать ограничение PRIMARY KEY для столбца «ID», когда таблица уже создана, используйте следующий SQL:

```
ALTER TABLE Persons
ADD PRIMARY KEY (ID);
```

- To allow naming of a PRIMARY KEY constraint, and for defining a PRIMARY KEY constraint on multiple columns, use the following SQL syntax:
- Щоб дозволити іменування обмеження PRIMARY KEY та для визначення обмеження PRIMARY KEY для декількох стовпців, використовуйте такий синтаксис SQL:
- Чтобы разрешить именование ограничения PRIMARY KEY и для определения ограничения PRIMARY KEY для нескольких столбцов, используйте следующий синтаксис SQL:

```
ALTER TABLE Persons
ADD CONSTRAINT PK_Person PRIMARY KEY (ID,LastName);
```



```
c:\ XAMPP for Windows - mysql -u root -p
MariaDB [testdb]> create table persons (
    -> id int not null,
    -> name text not null,
    -> age int );
Query OK, 0 rows affected (0.064 sec)

MariaDB [testdb]> describe persons;
+-----+-----+-----+-----+
| Field | Type  | Null | Key  |
+-----+-----+-----+-----+
| id   | int(11)| NO   |       |
| name | text   | NO   |       |
| age  | int(11)| YES  |       |
+-----+-----+-----+-----+
3 rows in set (0.039 sec)

MariaDB [testdb]> alter table persons add primary key (id);
Query OK, 0 rows affected (0.207 sec)
Records: 0  Duplicates: 0  Warnings: 0

MariaDB [testdb]> describe persons;
+-----+-----+-----+-----+
| Field | Type  | Null | Key  |
+-----+-----+-----+-----+
| id   | int(11)| NO   | PRI  |
| name | text   | NO   |       |
| age  | int(11)| YES  |       |
+-----+-----+-----+-----+
3 rows in set (0.091 sec)

MariaDB [testdb]>
```

339

- To drop a PRIMARY KEY constraint, use the following SQL:
- Щоб видалити обмеження PRIMARY KEY, використовуйте такий SQL:
- Чтобы удалить ограничение PRIMARY KEY, используйте следующий SQL:

MySQL:

```
ALTER TABLE Persons
DROP PRIMARY KEY;
```

SQL Server:

```
ALTER TABLE Persons
DROP CONSTRAINT PK_Person;
```

340

```
MariaDB [testdb]> alter table persons add primary key (id);
Query OK, 0 rows affected (0.207 sec)
Records: 0  Duplicates: 0  Warnings: 0

MariaDB [testdb]> describe persons;
+-----+-----+-----+-----+
| Field | Type  | Null | Key | Default | Extra |
+-----+-----+-----+-----+
| id   | int(11)| NO  | PRI | NULL    |       |
| name | text   | NO  |     | NULL    |       |
| age  | int(11)| YES |     | NULL    |       |
+-----+-----+-----+-----+
3 rows in set (0.091 sec)

MariaDB [testdb]> alter table persons
-> drop primary key;
Query OK, 0 rows affected (0.170 sec)
Records: 0  Duplicates: 0  Warnings: 0

MariaDB [testdb]> describe persons;
+-----+-----+-----+-----+
| Field | Type  | Null | Key | Default | Extra |
+-----+-----+-----+-----+
| id   | int(11)| NO  |     | NULL    |       |
| name | text   | NO  |     | NULL    |       |
| age  | int(11)| YES |     | NULL    |       |
+-----+-----+-----+-----+
3 rows in set (0.052 sec)

MariaDB [testdb]>
```

341

SQL FOREIGN KEY

A FOREIGN KEY is a key used to link two tables together.

A FOREIGN KEY is a field (or collection of fields) in one table that refers to the PRIMARY KEY in another table.

The table containing the foreign key is called the child table, and the table containing the candidate key is called the referenced or parent table.

FOREIGN KEY - це ключ, який використовується для зв'язування двох таблиць.

FOREIGN KEY - це поле (або сукупність полів) в одній таблиці, яке посилається на PRIMARY KEY в іншій таблиці.

Таблиця, що містить зовнішній ключ, називається дочірньою таблицею, а таблиця, що містить ключ-кандидат, називається посилальною або батьківською таблицею.

FOREIGN KEY - это ключ, используемый для связи двух таблиц вместе.

FOREIGN KEY - это поле (или набор полей) в одной таблице, которое ссылается на PRIMARY KEY в другой таблице.

Таблица, содержащая внешний ключ, называется дочерней таблицей, а таблица, содержащая ключ-кандидат, называется ссылочной или родительской таблицей.

342

"Persons" table:

PersonID	LastName	FirstName	Age
1	Hansen	Ola	30
2	Svendson	Tove	23
3	Pettersen	Karl	20

"Orders" table:

OrderID	OrderNumber	PersonID
1	77895	3
2	44678	3
3	22456	2
4	24562	1

343

- The following SQL creates a FOREIGN KEY on the "PersonID" column when the "Orders" table is created:
- Наступний SQL створює FOREIGN KEY у стовпці "PersonID", коли створюється таблиця "Orders":
- Следуючий SQL создает FOREIGN KEY в столбце «PersonID» при создании таблицы «Orders»:

MySQL:

```
CREATE TABLE Orders (
    OrderID int NOT NULL,
    OrderNumber int NOT NULL,
    PersonID int,
    PRIMARY KEY (OrderID),
    FOREIGN KEY (PersonID) REFERENCES Persons(PersonID)
);
```

SQL Server:

```
CREATE TABLE Orders (
    OrderID int NOT NULL PRIMARY KEY,
    OrderNumber int NOT NULL,
    PersonID int FOREIGN KEY REFERENCES Persons(PersonID)
);
```

344

```
c:\XAMPP for Windows - mysql -u root -p
MariaDB [testdb]> alter table persons add primary key (id);
Query OK, 0 rows affected (0.095 sec)
Records: 0  Duplicates: 0  Warnings: 0

MariaDB [testdb]> describe persons;
+-----+-----+-----+-----+
| Field | Type  | Null | Key  | Default | Extra |
+-----+-----+-----+-----+
| id   | int(11)| NO   | PRI  | NULL    |       |
| name | text   | NO   |      | NULL    |       |
| age  | int(11)| YES  |      | NULL    |       |
+-----+-----+-----+-----+
3 rows in set (0.057 sec)

MariaDB [testdb]> create table orders (
    -> id int not null,
    -> note text not null,
    -> person_id int not null,
    -> primary key (id),
    -> foreign key (person_id) references persons(id) );
Query OK, 0 rows affected (0.068 sec)

MariaDB [testdb]> describe orders;
+-----+-----+-----+-----+
| Field | Type  | Null | Key  | Default | Extra |
+-----+-----+-----+-----+
| id   | int(11)| NO   | PRI  | NULL    |       |
| note | text   | NO   |      | NULL    |       |
| person_id | int(11)| NO   | MUL  | NULL    |       |
+-----+-----+-----+-----+
3 rows in set (0.072 sec)
```

345

```

c:\ XAMPP for Windows - mysql -u root -p
MariaDB [testdb]> describe orders;
+-----+-----+-----+-----+-----+
| Field | Type   | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+
| id    | int(11) | NO   | PRI | NULL    |       |
| note  | text    | NO   |     | NULL    |       |
| person_id | int(11) | NO   | MUL | NULL    |       |
+-----+-----+-----+-----+-----+
3 rows in set (0.072 sec)

MariaDB [testdb]> show create table orders;
+-----+-----+
| Table | Create Table
+-----+-----+
| orders | CREATE TABLE `orders` (
`id` int(11) NOT NULL,
`note` text NOT NULL,
`person_id` int(11) NOT NULL,
PRIMARY KEY (`id`),
KEY `person_id`(`person_id`),
CONSTRAINT `orders_ibfk_1` FOREIGN KEY (`person_id`) REFERENCES `persons` (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1 |
+-----+-----+
1 row in set (0.000 sec)

```

346

- To allow naming of a FOREIGN KEY constraint, and for defining a FOREIGN KEY constraint on multiple columns, use the following SQL syntax:
- Щоб дозволити іменування обмеження FOREIGN KEY та визначення обмеження FOREIGN KEY для декількох стовпців, використовуйте такий синтаксис SQL:
- Чтобы разрешить именование ограничения FOREIGN KEY и для определения ограничения FOREIGN KEY для нескольких столбцов, используйте следующий синтаксис SQL:

```

CREATE TABLE Orders (
    OrderID int NOT NULL,
    OrderNumber int NOT NULL,
    PersonID int,
    PRIMARY KEY (OrderID),
    CONSTRAINT FK_PersonOrder FOREIGN KEY (PersonID)
        REFERENCES Persons(PersonID)
);

```

- To create a FOREIGN KEY constraint on the "PersonID" column when the "Orders" table is already created, use the following SQL:
- Щоб створити обмеження FOREIGN KEY у стовпці "PersonID", коли таблиця "Orders" вже створена, використовуйте такий SQL:
- Чтобы создать ограничение FOREIGN KEY для столбца «PersonID», когда таблица «Orders» уже создана, используйте следующий SQL:

```
ALTER TABLE Orders
ADD FOREIGN KEY (PersonID) REFERENCES Persons(PersonID);
```

- To allow naming of a FOREIGN KEY constraint, and for defining a FOREIGN KEY constraint on multiple columns, use the following SQL syntax:
- Щоб дозволити іменування обмеження FOREIGN KEY та визначення обмеження FOREIGN KEY для декількох стовпців, використовуйте такий синтаксис SQL:
- Чтобы разрешить именование ограничения FOREIGN KEY и для определения ограничения FOREIGN KEY для нескольких столбцов, используйте следующий синтаксис SQL:

```
ALTER TABLE Orders
ADD CONSTRAINT FK_PersonOrder
FOREIGN KEY (PersonID) REFERENCES Persons(PersonID);
```

```
MariaDB [testdb]> create table orders (
-> id int not null,
-> note text not null,
-> person_id int not null,
-> primary key (id) );
Query OK, 0 rows affected (0.037 sec)

MariaDB [testdb]> alter table orders
-> add foreign key (person_id) references persons(id);
Query OK, 0 rows affected (0.145 sec)
Records: 0 Duplicates: 0 Warnings: 0

MariaDB [testdb]> show create table orders;
+-----+-----+
| Table | Create Table
+-----+-----+
| orders | CREATE TABLE `orders` (
`id` int(11) NOT NULL,
`note` text NOT NULL,
`person_id` int(11) NOT NULL,
PRIMARY KEY (`id`),
KEY `person_id` (`person_id`),
CONSTRAINT `orders_ibfk_1` FOREIGN KEY (`person_id`) REFERENCES `persons` (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1 |
+-----+-----+
```

349

- To drop a FOREIGN KEY constraint, use the following SQL:
- Щоб видалити обмеження FOREIGN KEY, використовуйте такий SQL:
- Чтобы удалить ограничение FOREIGN KEY, используйте следующий SQL:

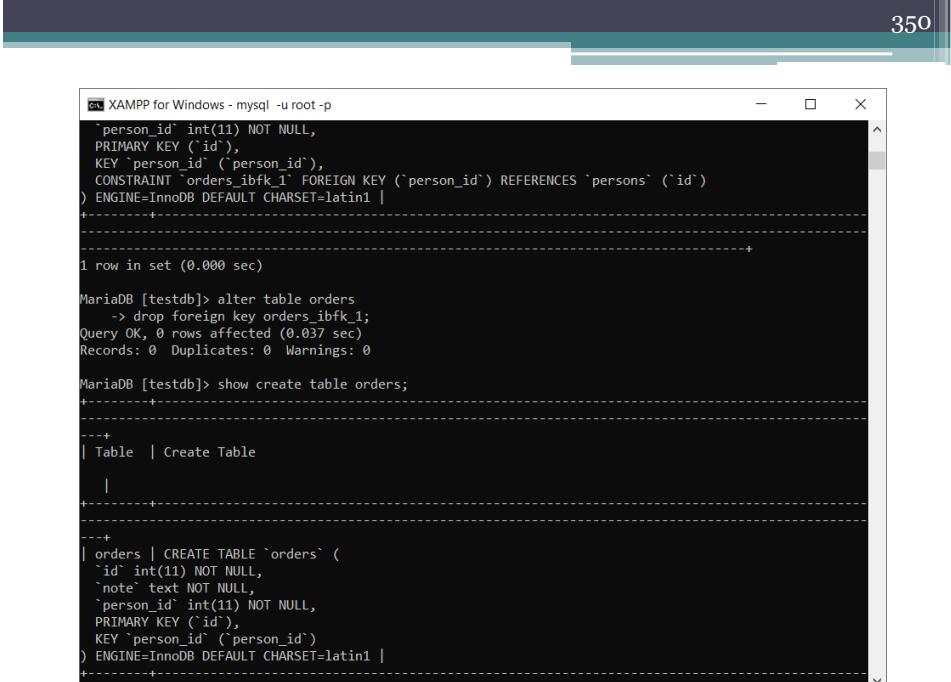
MySQL:

```
ALTER TABLE Orders
DROP FOREIGN KEY FK_PersonOrder;
```

SQL Server:

```
ALTER TABLE Orders
DROP CONSTRAINT FK_PersonOrder;
```

350



```
c:\ XAMPP for Windows - mysql -u root -p
`person_id` int(11) NOT NULL,
PRIMARY KEY (`id`),
KEY `person_id` (`person_id`),
CONSTRAINT `orders_ibfk_1` FOREIGN KEY (`person_id`) REFERENCES `persons` (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1 |
+-----+
-----+
-----+
1 row in set (0.000 sec)

MariaDB [testdb]> alter table orders
    -> drop foreign key orders_ibfk_1;
Query OK, 0 rows affected (0.037 sec)
Records: 0  Duplicates: 0  Warnings: 0

MariaDB [testdb]> show create table orders;
+-----+
| Table | Create Table
|
+-----+
| orders | CREATE TABLE `orders` (
`id` int(11) NOT NULL,
`note` text NOT NULL,
`person_id` int(11) NOT NULL,
PRIMARY KEY (`id`),
KEY `person_id` (`person_id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1 |
+-----+
```

SQL CHECK

The CHECK constraint is used to limit the value range that can be placed in a column.

If you define a CHECK constraint on a single column it allows only certain values for this column.

If you define a CHECK constraint on a table it can limit the values in certain columns based on values in other columns in the row.

Обмеження CHECK використовується для обмеження діапазону значень, який може бути розміщений у стовпці.

Якщо ви визначите обмеження CHECK для одного стовпця, це дозволить лише певні значення для цього стовпця.

Якщо ви визначите обмеження CHECK для таблиці, це може обмежити значення в певних стовпцях на основі значень в інших стовпцях рядка.

Ограничение CHECK используется для ограничения диапазона значений, которые могут быть помещены в столбец.

Если вы определяете ограничение CHECK для одного столбца, оно допускает только определенные значения для этого столбца.

Если вы определяете ограничение CHECK для таблицы, оно может ограничивать значения в определенных столбцах на основе значений в других столбцах в строке.

The following SQL creates a CHECK constraint on the "Age" column when the "Persons" table is created. The CHECK constraint ensures that the age of a person must be 18, or older:

Наступний SQL створює обмеження CHECK у стовпці "Age", коли створюється таблиця "Persons". Обмеження CHECK гарантує, що вік людини повинен бути 18 років або старше:

Следующий SQL создает ограничение CHECK для столбца «Age» при создании таблицы «Persons». Ограничение CHECK гарантирует, что возраст человека должен быть 18 или старше:

MySQL:

```
CREATE TABLE Persons (
    ID int NOT NULL,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int,
    CHECK (Age>=18)
);
```

353

```

XAMPP for Windows - mysql -u root -p

MariaDB [testdb]> drop table orders;
Query OK, 0 rows affected (0.023 sec)

MariaDB [testdb]> drop table persons;
Query OK, 0 rows affected (0.037 sec)

MariaDB [testdb]> create table persons (
    -> id int not null,
    -> name text not null,
    -> age int,
    -> check (age >= 18) );
Query OK, 0 rows affected (0.062 sec)

MariaDB [testdb]> describe persons;
+-----+-----+-----+-----+
| Field | Type  | Null | Key | Default | Extra |
+-----+-----+-----+-----+
| id   | int(11)| NO   |     | NULL    |          |
| name | text   | NO   |     | NULL    |          |
| age  | int(11)| YES  |     | NULL    |          |
+-----+-----+-----+-----+
3 rows in set (0.060 sec)

MariaDB [testdb]> insert into persons values (1, 'John Smith', 16);
ERROR 4025 (23000): CONSTRAINT `CONSTRAINT_1` failed for `testdb`.`persons`
MariaDB [testdb]> -

```

354

SQL Server:

```

CREATE TABLE Persons (
    ID int NOT NULL,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int CHECK (Age>=18)
);

```

- To allow naming of a CHECK constraint, and for defining a CHECK constraint on multiple columns, use the following SQL syntax:
- Щоб дозволити іменування обмеження CHECK та для визначення обмеження CHECK для декількох стовпців, використовуйте такий синтаксис SQL:
- Чтобы разрешить именование ограничения CHECK и определить ограничение CHECK для нескольких столбцов, используйте следующий синтаксис SQL:

```

CREATE TABLE Persons (
    ID int NOT NULL,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int,
    City varchar(255),
    CONSTRAINT CHK_Person CHECK (Age>=18 AND City='Sandnes')
);

```

355

```

c:\ XAMPP for Windows - mysql -u root -p
# mysql -u root -p
Enter password:
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 1759
Server version: 10.4.17-MariaDB mariadb.org binary distribution

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MariaDB [(none)]> use testdb;
Database changed
MariaDB [testdb]> drop table recruitment;
Query OK, 0 rows affected (0.038 sec)

MariaDB [testdb]> create table recruitment (
    -> id int not null,
    -> name text not null,
    -> age int,
    -> citizenship varchar(50),
    -> check (age >= 18 and age <= 27 and citizenship = 'Ukraine') );
Query OK, 0 rows affected (0.129 sec)

MariaDB [testdb]> insert into recruitment values (1, 'John Smith', 22, 'UK');
ERROR 4025 (23000): CONSTRAINT `CONSTRAINT_1` failed for `testdb`.`recruitment`
MariaDB [testdb]>

```

356

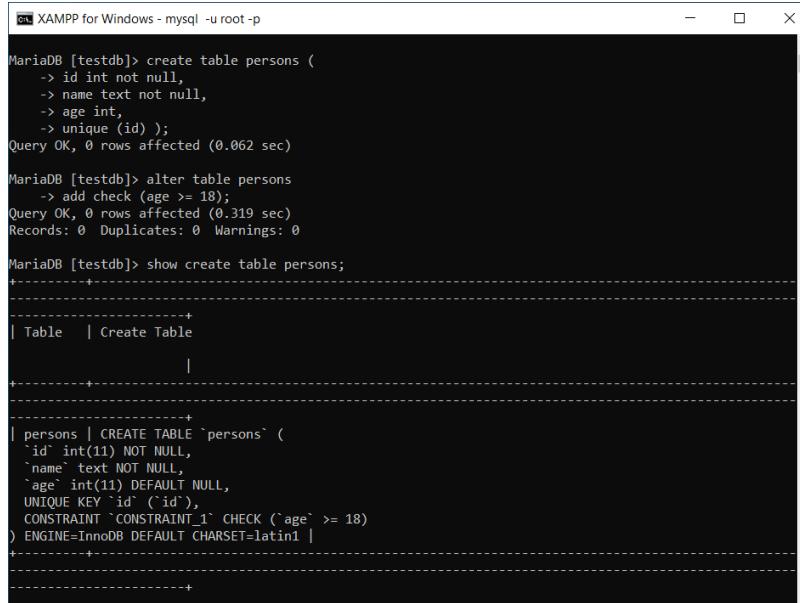
- To create a CHECK constraint on the "Age" column when the table is already created, use the following SQL:
- Щоб створити обмеження CHECK у стовпці "Age", коли таблиця вже створена, використовуйте такий SQL:
- Чтобы создать ограничение CHECK для столбца «Age», когда таблица уже создана, используйте следующий SQL:

```
ALTER TABLE Persons
ADD CHECK (Age>=18);
```

- To allow naming of a CHECK constraint, and for defining a CHECK constraint on multiple columns, use the following SQL syntax:
- Щоб дозволити іменування обмеження CHECK та для визначення обмеження CHECK для декількох стовпців, використовуйте такий синтаксис SQL:
- Чтобы разрешить именование ограничения CHECK и определить ограничение CHECK для нескольких столбцов, используйте следующий синтаксис SQL:

```
ALTER TABLE Persons
ADD CONSTRAINT CHK_PersonAge
    CHECK (Age>=18 AND City='Sandnes');
```

357



```

c:\ XAMPP for Windows - mysql -u root -p

MariaDB [testdb]> create table persons (
-> id int not null,
-> name text not null,
-> age int,
-> unique (id));
Query OK, 0 rows affected (0.062 sec)

MariaDB [testdb]> alter table persons
-> add check (age >= 18);
Query OK, 0 rows affected (0.319 sec)
Records: 0 Duplicates: 0 Warnings: 0

MariaDB [testdb]> show create table persons;
+-----+
| Table | Create Table
+-----+
| persons | CREATE TABLE `persons` (
`id` int(11) NOT NULL,
`name` text NOT NULL,
`age` int(11) DEFAULT NULL,
UNIQUE KEY `id` (`id`),
CONSTRAINT `CONSTRAINT_1` CHECK (`age` >= 18)
) ENGINE=InnoDB DEFAULT CHARSET=latin1 |
+-----+

```

358

- To drop a CHECK constraint, use the following SQL:
- Щоб видалити обмеження CHECK, використовуйте такий SQL:
- Чтобы удалить ограничение CHECK, используйте следующий SQL:

MySQL:

```

ALTER TABLE Persons
DROP CONSTRAINT CHK_PersonAge;

```

SQL Server:

```

ALTER TABLE Persons
DROP CONSTRAINT CHK_PersonAge;

```

359

```

| persons | CREATE TABLE `persons` (
`id` int(11) NOT NULL,
`name` text NOT NULL,
`age` int(11) DEFAULT NULL,
UNIQUE KEY `id` (`id`),
CONSTRAINT `CONSTRAINT_1` CHECK (`age` >= 18)
) ENGINE=InnoDB DEFAULT CHARSET=latin1 |
+-----+
-----+
1 row in set (0.000 sec)

MariaDB [testdb]> alter table persons drop constraint `CONSTRAINT_1`;
Query OK, 0 rows affected (0.036 sec)
Records: 0  Duplicates: 0  Warnings: 0

MariaDB [testdb]> show create table persons;
+-----+
-----+
| Table | Create Table
+-----+
-----+
| persons | CREATE TABLE `persons` (
`id` int(11) NOT NULL,
`name` text NOT NULL,
`age` int(11) DEFAULT NULL,
UNIQUE KEY `id` (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1 |
+-----+
-----+

```

360

SQL DEFAULT

The DEFAULT constraint is used to provide a default value for a column.

The default value will be added to all new records IF no other value is specified.

Обмеження DEFAULT використовується для надання значення за замовчуванням для стовпця.

Значення за замовчуванням буде додано до всіх нових записів, ЯКЩО не вказано інше значення.

Ограничение DEFAULT используется для предоставления значения по умолчанию для столбца.

Значение по умолчанию будет добавлено ко всем новым записям, ЕСЛИ не указано другое значение.

361

- The following SQL sets a DEFAULT value for the "City" column when the "Persons" table is created:
- Наступний SQL встановлює значення DEFAULT для стовпця "City" коли створюється таблиця "Persons":
- Следующий SQL устанавливает значение DEFAULT для столбца «Город» при создании таблицы «Persons»:

```
CREATE TABLE Persons (
    ID int NOT NULL,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int,
    City varchar(255) DEFAULT 'Sandnes'
);
```

362



```
c:\ XAMPP for Windows - mysql -u root -p
MariaDB [testdb]> drop table persons;
Query OK, 0 rows affected (0.037 sec)

MariaDB [testdb]> create table persons (
-> id int not null,
-> name text not null,
-> age int,
-> city text default 'Kharkiv' );
Query OK, 0 rows affected (0.068 sec)

MariaDB [testdb]> insert into persons (id, name, age)
-> values (1, 'John Doe', 27);
Query OK, 1 row affected (0.024 sec)

MariaDB [testdb]> select * from persons;
+----+-----+-----+-----+
| id | name  | age   | city   |
+----+-----+-----+-----+
| 1  | John Doe | 27 | Kharkiv |
+----+-----+-----+-----+
1 row in set (0.000 sec)

MariaDB [testdb]>
```

363

- The DEFAULT constraint can also be used to insert system values, by using functions like GETDATE():
- Обмеження DEFAULT також можна використовувати для вставки системних значень, використовуючи такі функції, як GETDATE():
- Ограничение DEFAULT также можно использовать для вставки системных значений с помощью таких функций, как GETDATE():

```
CREATE TABLE Orders (
    ID int NOT NULL,
    OrderNumber int NOT NULL,
    OrderDate date DEFAULT GETDATE()
);
```

364

```

c:\ XAMPP for Windows - mysql -u root -p
MariaDB [testdb]> create table orders (
-> id int not null,
-> note text not null,
-> order_date date default now(),
-> unique (id);
Query OK, 0 rows affected (0.057 sec)

MariaDB [testdb]> describe orders;
+-----+-----+-----+-----+
| Field | Type  | Null | Key | Default   |
+-----+-----+-----+-----+
| id    | int(11)| NO   | PRI | NULL      |
| note  | text   | NO   |     | NULL      |
| order_date | date | YES  |     | current_timestamp() |
+-----+-----+-----+-----+
3 rows in set (0.063 sec)

MariaDB [testdb]> insert into orders (id, note)
-> values (1, 'something nice');
Query OK, 1 row affected, 1 warning (0.002 sec)

MariaDB [testdb]> select * from orders;
+-----+-----+
| id | note      | order_date |
+-----+-----+
| 1  | something nice | 2021-02-18 |
+-----+-----+
1 row in set (0.000 sec)

MariaDB [testdb]>
```

365

- To create a DEFAULT constraint on the "City" column when the table is already created, use the following SQL:
- Щоб створити обмеження DEFAULT у стовпці "Місто", коли таблиця вже створена, використовуйте такий SQL:
- Чтобы создать ограничение DEFAULT для столбца «Город», когда таблица уже создана, используйте следующий SQL:

MySQL

```
ALTER TABLE Persons
ALTER City SET DEFAULT 'Sandnes';
```

SQL Server

```
ALTER TABLE Persons
ADD CONSTRAINT df_City
DEFAULT 'Sandnes' FOR City;
```

MS Access

```
ALTER TABLE Persons
ALTER COLUMN City SET DEFAULT 'Sandnes';
```

Oracle

```
ALTER TABLE Persons
MODIFY City DEFAULT 'Sandnes';
```

366

```
c:\ XAMPP for Windows - mysql -u root -p
MariaDB [testdb]> create table person (
    -> id int not null,
    -> name text not null,
    -> age int,
    -> unique (id);
Query OK, 0 rows affected (0.068 sec)

MariaDB [testdb]> describe person;
+-----+-----+-----+-----+
| Field | Type   | Null | Key | Default | Extra |
+-----+-----+-----+-----+
| id   | int(11) | NO  | PRI | NULL    |       |
| name | text    | NO  |     | NULL    |       |
| age  | int(11) | YES |     | NULL    |       |
+-----+-----+-----+-----+
3 rows in set (0.055 sec)

MariaDB [testdb]> alter table person
    -> alter age set default 18;
Query OK, 0 rows affected (0.036 sec)
Records: 0  Duplicates: 0  Warnings: 0

MariaDB [testdb]> describe person;
+-----+-----+-----+-----+
| Field | Type   | Null | Key | Default | Extra |
+-----+-----+-----+-----+
| id   | int(11) | NO  | PRI | NULL    |       |
| name | text    | NO  |     | NULL    |       |
| age  | int(11) | YES |     | 18      |       |
+-----+-----+-----+-----+
3 rows in set (0.053 sec)
```

367

- To drop a DEFAULT constraint, use the following SQL:
- Щоб видалити обмеження DEFAULT, використовуйте такий SQL:
- Чтобы удалить ограничение DEFAULT, используйте следующий SQL:

MySQL

```
ALTER TABLE Persons
ALTER City DROP DEFAULT;
```

SQL Server / Oracle / MS Access

```
ALTER TABLE Persons
ALTER COLUMN City DROP DEFAULT;
```

368

```
XAMPP for Windows - mysql -u root -p
| Table | Create Table
+-----+
| person | CREATE TABLE `person` (
  `id` int(11) NOT NULL,
  `name` text NOT NULL,
  `age` int(11) DEFAULT 18,
  UNIQUE KEY `id` (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1 |
+-----+
1 row in set (0.000 sec)

MariaDB [testdb]> alter table person
      >     drop default;
Query OK, 0 rows affected (0.036 sec)
Records: 0  Duplicates: 0  Warnings: 0

MariaDB [testdb]> describe person;
+-----+-----+-----+-----+-----+
| Field | Type  | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+
| id   | int(11)| NO  | PRI | NULL    |       |
| name | text   | NO  |     | NULL    |       |
| age  | int(11)| YES |     | NULL    |       |
+-----+-----+-----+-----+-----+
3 rows in set (0.053 sec)

MariaDB [testdb]> -
```

369

SQL INDEX

The CREATE INDEX statement is used to create indexes in tables. Indexes are used to retrieve data from the database more quickly than otherwise. The users cannot see the indexes, they are just used to speed up searches/queries.

Оператор CREATE INDEX використовується для створення індексів у таблицях.

Індекси використовуються для отримання даних з бази даних швидше, ніж без індексів. Користувачі не можуть бачити індекси, вони просто використовуються для пришвидшення пошуку / запитів.

Оператор CREATE INDEX используется для создания индексов в таблицах.

Индексы используются для более быстрого извлечения данных из базы данных. Пользователи не могут видеть индексы, они просто используются для ускорения поиска / запросов.

370

Updating a table with indexes takes more time than updating a table without (because the indexes also need an update). So, only create indexes on columns that will be frequently searched against.

Оновлення таблиці з індексами займає більше часу, ніж оновлення таблиці без (оскільки індекси також потребують оновлення). Отже, створюйте лише індекси в стовпцях, за якими часто шукатимуть.

Обновление таблицы с индексами занимает больше времени, чем обновление таблицы без (потому что индексы также нуждаются в обновлении). Таким образом, создавайте индексы только для столбцов, по которым будет часто выполняться поиск.

- Creates an index on a table. Duplicate values are allowed:
- Створює індекс для таблиці. Допускаються повторювані значення:
- Создает индекс для таблицы. Допускаются повторяющиеся значения:

```
CREATE INDEX index_name
ON table_name (column1, column2, ...);
```

- Creates a unique index on a table. Duplicate values are not allowed:
- Створює унікальний індекс для таблиці. Повторювані значення не допускаються:
- Создает уникальный индекс для таблицы. Повторяющиеся значения не допускаются:

```
CREATE UNIQUE INDEX index_name
ON table_name (column1, column2, ...);
```

- The SQL statement below creates an index named "idx_lastname" on the "LastName" column in the "Persons" table:
- Оператор SQL нижче створює індекс "idx_lastname" у стовпці "LastName" у таблиці "Persons":
- Приведенный ниже оператор SQL создает индекс с именем «idx_lastname» в столбце «LastName» в таблице «Persons»:

```
CREATE INDEX idx_lastname
ON Persons (LastName);
```

- If you want to create an index on a combination of columns, you can list the column names within the parentheses, separated by commas:
- Якщо ви хочете створити індекс для комбінації стовпців, ви можете вказати імена стовпців у дужках, роздливши їх комами:
- Если вы хотите создать индекс для комбинации столбцов, вы можете перечислить имена столбцов в круглых скобках, разделенных запятыми:

```
CREATE INDEX idx_pname
ON Persons (LastName, FirstName);
```

373

```

XAMPP for Windows - mysql -u root -p

MariaDB [testdb]> describe person;
+-----+-----+-----+-----+-----+
| Field | Type  | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+
| id   | int(11) | NO  | PRI | NULL   |       |
| name | text   | NO  |     | NULL   |       |
| age  | int(11) | YES |     | NULL   |       |
+-----+-----+-----+-----+-----+
3 rows in set (0.059 sec)

MariaDB [testdb]> create index idx_person_age
    -> on person (age);
Query OK, 0 rows affected (0.050 sec)
Records: 0  Duplicates: 0  Warnings: 0

MariaDB [testdb]> describe person;
+-----+-----+-----+-----+-----+
| Field | Type  | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+
| id   | int(11) | NO  | PRI | NULL   |       |
| name | text   | NO  |     | NULL   |       |
| age  | int(11) | YES | MUL | NULL   |       |
+-----+-----+-----+-----+-----+
3 rows in set (0.052 sec)

MariaDB [testdb]>

```

374

- The **DROP INDEX** statement is used to delete an index in a table.
- Оператор **DROP INDEX** використовується для видалення індексу в таблиці.
- Оператор **DROP INDEX** используется для удаления индекса в таблице.

MS Access

DROP INDEX index_name ON table_name;

SQL Server

DROP INDEX table_name.index_name;

DB2/Oracle

DROP INDEX index_name;

MySQL

ALTER TABLE table_name
DROP INDEX index_name;

375

```
c:\ XAMPP for Windows - mysql -u root -p

MariaDB [testdb]> describe person;
+-----+-----+-----+-----+-----+
| Field | Type   | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+
| id    | int(11) | NO   | PRI  | NULL    |       |
| name  | text    | NO   |     | NULL    |       |
| age   | int(11) | YES  | MUL  | NULL    |       |
+-----+-----+-----+-----+-----+
3 rows in set (0.052 sec)

MariaDB [testdb]> alter table person
    >     drop index idx_person_age;
Query OK, 0 rows affected (0.045 sec)
Records: 0  Duplicates: 0  Warnings: 0

MariaDB [testdb]> describe person;
+-----+-----+-----+-----+-----+
| Field | Type   | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+
| id    | int(11) | NO   | PRI  | NULL    |       |
| name  | text    | NO   |     | NULL    |       |
| age   | int(11) | YES  | MUL  | NULL    |       |
+-----+-----+-----+-----+-----+
3 rows in set (0.052 sec)

MariaDB [testdb]>
```

376

SQL AUTO INCREMENT

Auto-increment allows a unique number to be generated automatically when a new record is inserted into a table.

Often this is the primary key field that we would like to be created automatically every time a new record is inserted.

Автоінкремент дозволяє автоматично генерувати унікальний номер при вставці нового запису в таблицю.

Часто це поле первинного ключа, яке ми хотіли б створювати автоматично кожного разу, коли вставляється новий запис.

Автоинкремент позволяет автоматически генерировать уникальный номер при вставке новой записи в таблицу.

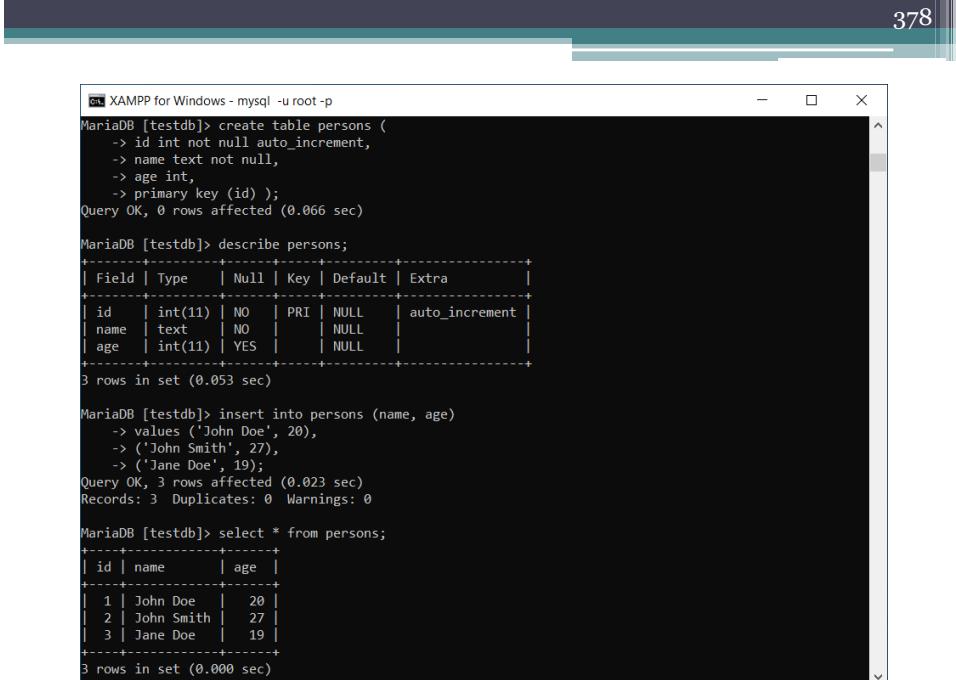
Часто это поле первичного ключа, которое мы хотели бы создавать автоматически при каждой вставке новой записи.

377

- The following SQL statement defines the "Personid" column to be an auto-increment primary key field in the "Persons" table:
- Наступний оператор SQL визначає стовпець "Personid" як поле первинного ключа з автоінкрементом у таблиці "Persons":
- Следующий оператор SQL определяет столбец «Personid» как поле первичного ключа с автоинкрементом в таблице «Persons»:

```
CREATE TABLE Persons (
    Personid int NOT NULL AUTO_INCREMENT,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int,
    PRIMARY KEY (Personid)
);
```

378



```
c:\XAMPP for Windows - mysql -u root -p
MariaDB [testdb]> create table persons (
-> id int not null auto_increment,
-> name text not null,
-> age int,
-> primary key (id) );
Query OK, 0 rows affected (0.066 sec)

MariaDB [testdb]> describe persons;
+-----+-----+-----+-----+-----+
| Field | Type  | Null | Key  | Default |
+-----+-----+-----+-----+-----+
| id   | int(11)| NO   | PRI  | NULL    |
| name | text   | NO   |      | NULL    |
| age  | int(11)| YES  |      | NULL    |
+-----+-----+-----+-----+-----+
3 rows in set (0.053 sec)

MariaDB [testdb]> insert into persons (name, age)
-> values ('John Doe', 20),
-> ('John Smith', 27),
-> ('Jane Doe', 19);
Query OK, 3 rows affected (0.023 sec)
Records: 3  Duplicates: 0  Warnings: 0

MariaDB [testdb]> select * from persons;
+----+-----+-----+
| id | name  | age  |
+----+-----+-----+
| 1  | John Doe | 20 |
| 2  | John Smith | 27 |
| 3  | Jane Doe | 19 |
+----+-----+-----+
3 rows in set (0.000 sec)
```

379

MySQL uses the AUTO_INCREMENT keyword to perform an auto-increment feature.

By default, the starting value for AUTO_INCREMENT is 1, and it will increment by 1 for each new record.

To let the AUTO_INCREMENT sequence start with another value, use the following SQL statement:

MySQL використовує ключове слово AUTO_INCREMENT для виконання функції автоматичного збільшення.

За замовчуванням початкове значення для AUTO_INCREMENT дорівнює 1, і воно збільшується на 1 для кожного нового запису.

Щоб дозволити послідовність AUTO_INCREMENT починатися з іншого значення, використовуйте такий оператор SQL:

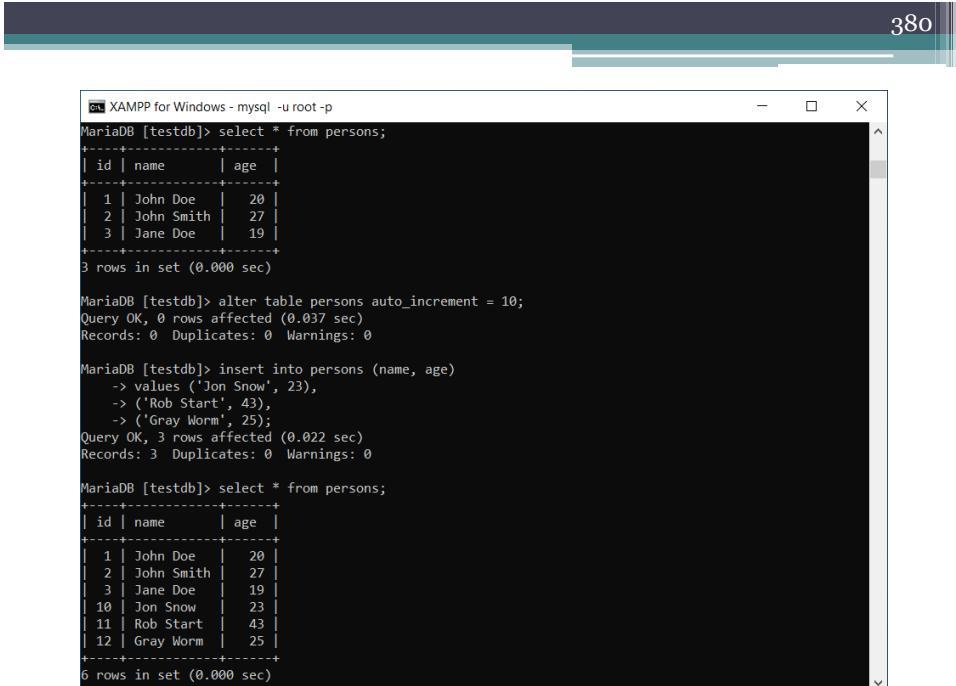
MySQL использует ключевое слово AUTO_INCREMENT для выполнения функции автоматического увеличения.

По умолчанию начальное значение для AUTO_INCREMENT равно 1, и оно будет увеличиваться на 1 для каждой новой записи.

Чтобы позволить последовательности AUTO_INCREMENT начинаться с другого значения, используйте следующий оператор SQL:

```
ALTER TABLE Persons AUTO_INCREMENT=100;
```

380



```
c:\XAMPP for Windows - mysql -u root -p
MariaDB [testdb]> select * from persons;
+---+-----+-----+
| id | name   | age  |
+---+-----+-----+
| 1  | John Doe | 20   |
| 2  | John Smith | 27   |
| 3  | Jane Doe | 19   |
+---+-----+-----+
3 rows in set (0.000 sec)

MariaDB [testdb]> alter table persons auto_increment = 10;
Query OK, 0 rows affected (0.037 sec)
Records: 0  Duplicates: 0  Warnings: 0

MariaDB [testdb]> insert into persons (name, age)
-> values ('Jon Snow', 23),
-> ('Rob Start', 43),
-> ('Gray Worm', 25);
Query OK, 3 rows affected (0.022 sec)
Records: 3  Duplicates: 0  Warnings: 0

MariaDB [testdb]> select * from persons;
+---+-----+-----+
| id | name   | age  |
+---+-----+-----+
| 1  | John Doe | 20   |
| 2  | John Smith | 27   |
| 3  | Jane Doe | 19   |
| 10 | Jon Snow | 23   |
| 11 | Rob Start | 43   |
| 12 | Gray Worm | 25   |
+---+-----+-----+
6 rows in set (0.000 sec)
```

381

- To insert a new record into the "Persons" table, we will NOT have to specify a value for the "PersonId" column (a unique value will be added automatically):
- Щоб вставити новий запис у таблицю "Persons", НЕ доведеться вказувати значення для стовпця "PersonId" (унікальне значення буде додано автоматично):
- Чтобы вставить новую запись в таблицу «Persons», нам НЕ нужно указывать значение в столбце «PersonsId» (уникальное значение будет добавлено автоматически):

```
INSERT INTO Persons (FirstName,LastName)  
VALUES ('Lars','Monsen');
```

382

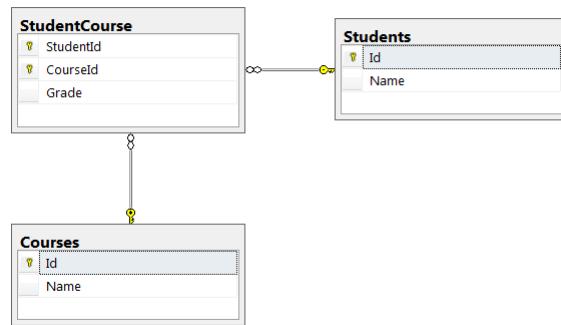
USING DDL TO DEVELOP A DATABASE

383

Data model

Модель данных

Модель данных



384

```
XAMPP for Windows - mysql -u root -p
MariaDB [(none)]> create database student_courses;
Query OK, 1 row affected (0.010 sec)

MariaDB [(none)]> use student_courses;
Database changed
MariaDB [student_courses]> show tables;
Empty set (0.001 sec)

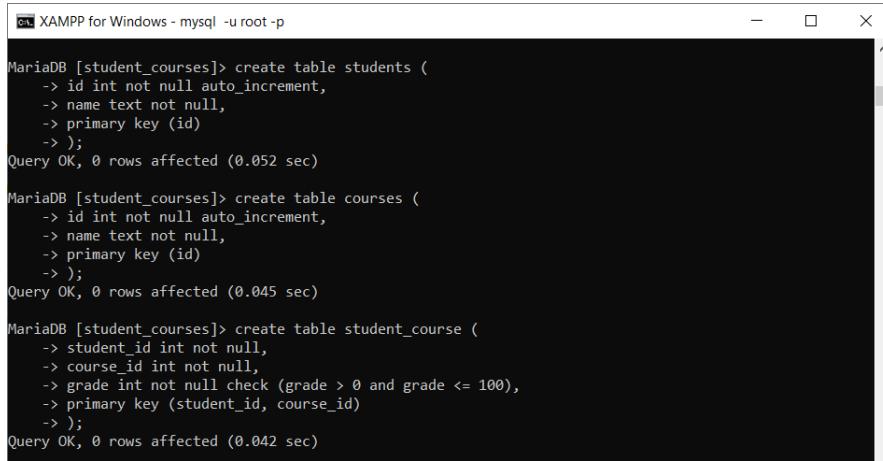
MariaDB [student_courses]>
```

385

Create tables

Створення таблиць

Создание таблиц



```

XAMPP for Windows - mysql -u root -p

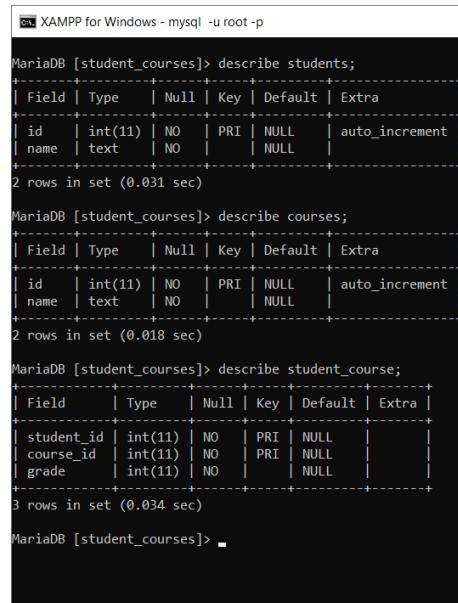
MariaDB [student_courses]> create table students (
    -> id int not null auto_increment,
    -> name text not null,
    -> primary key (id)
    -> );
Query OK, 0 rows affected (0.052 sec)

MariaDB [student_courses]> create table courses (
    -> id int not null auto_increment,
    -> name text not null,
    -> primary key (id)
    -> );
Query OK, 0 rows affected (0.045 sec)

MariaDB [student_courses]> create table student_course (
    -> student_id int not null,
    -> course_id int not null,
    -> grade int not null check (grade > 0 and grade <= 100),
    -> primary key (student_id, course_id)
    -> );
Query OK, 0 rows affected (0.042 sec)

```

386



```

XAMPP for Windows - mysql -u root -p

MariaDB [student_courses]> describe students;
+-----+-----+-----+-----+-----+
| Field | Type   | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+
| id    | int(11) | NO   | PRI  | NULL    | auto_increment |
| name  | text    | NO   |      | NULL    |               |
+-----+-----+-----+-----+-----+
2 rows in set (0.031 sec)

MariaDB [student_courses]> describe courses;
+-----+-----+-----+-----+-----+
| Field | Type   | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+
| id    | int(11) | NO   | PRI  | NULL    | auto_increment |
| name  | text    | NO   |      | NULL    |               |
+-----+-----+-----+-----+-----+
2 rows in set (0.018 sec)

MariaDB [student_courses]> describe student_course;
+-----+-----+-----+-----+-----+
| Field     | Type   | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+
| student_id | int(11) | NO   | PRI  | NULL    |               |
| course_id  | int(11) | NO   | PRI  | NULL    |               |
| grade      | int(11) | NO   |      | NULL    |               |
+-----+-----+-----+-----+-----+
3 rows in set (0.034 sec)

MariaDB [student_courses]> -

```

387

Create relationships Створення зв'язків Создание связей

```
XAMPP for Windows - mysql -u root -p

MariaDB [student_courses]> alter table student_course
    -> add foreign key (student_id) references students (id);
Query OK, 0 rows affected (0.100 sec)
Records: 0  Duplicates: 0  Warnings: 0

MariaDB [student_courses]> alter table student_course
    -> add foreign key (course_id) references courses (id);
Query OK, 0 rows affected (0.081 sec)
Records: 0  Duplicates: 0  Warnings: 0

MariaDB [student_courses]> describe student_course;
+-----+-----+-----+-----+
| Field | Type | Null | Key | Default | Extra |
+-----+-----+-----+-----+
| student_id | int(11) | NO | PRI | NULL |
| course_id | int(11) | NO | PRI | NULL |
| grade | int(11) | NO | | NULL |
+-----+-----+-----+-----+
3 rows in set (0.036 sec)
```

388

```
XAMPP for Windows - mysql -u root -p

MariaDB [student_courses]> show create table student_course;
+-----+-----+
| Table | Create Table
+-----+-----+
| student_course | CREATE TABLE `student_course` (
  `student_id` int(11) NOT NULL,
  `course_id` int(11) NOT NULL,
  `grade` int(11) NOT NULL CHECK (`grade` > 0 AND `grade` <= 100),
  PRIMARY KEY (`student_id`,`course_id`),
  KEY `course_id` (`course_id`),
  CONSTRAINT `student_course_ibfk_1` FOREIGN KEY (`student_id`) REFERENCES `students` (`id`),
  CONSTRAINT `student_course_ibfk_2` FOREIGN KEY (`course_id`) REFERENCES `courses` (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1 |
+-----+-----+
1 row in set (0.000 sec)
```

389

Create indexes**Створення індексів****Создание индексов**

```
cmd XAMPP for Windows - mysql -u root -p

MariaDB [student_courses]> create index idx_grade
    -> on student_course (grade);
Query OK, 0 rows affected (0.025 sec)
Records: 0  Duplicates: 0  Warnings: 0

cmd XAMPP for Windows - mysql -u root -p

MariaDB [student_courses]> describe student_course;
+-----+-----+-----+-----+-----+
| Field | Type   | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+
| student_id | int(11) | NO  | PRI | NULL    |       |
| course_id  | int(11) | NO  | PRI | NULL    |       |
| grade       | int(11) | NO  | MUL | NULL    |       |
+-----+-----+-----+-----+-----+
3 rows in set (0.025 sec)

MariaDB [student_courses]>
```

390

Insert initial records**Заповнення початковими даними****Заполнение начальными данными**

```
cmd XAMPP for Windows - mysql -u root -p

MariaDB [student_courses]> insert into students (name)
    -> values ('Jon Snow'), ('Rob Stark'), ('Gray Worm');
Query OK, 3 rows affected (0.004 sec)
Records: 3  Duplicates: 0  Warnings: 0

MariaDB [student_courses]> insert into courses (name)
    -> values ('Mathematics'), ('Programming'), ('Databases');
Query OK, 3 rows affected (0.002 sec)
Records: 3  Duplicates: 0  Warnings: 0

MariaDB [student_courses]> insert into student_course
    -> values (1, 1, 78), (2, 3, 94), (2, 2, 100), (3, 2, 72), (3, 1, 86);
Query OK, 5 rows affected (0.005 sec)
Records: 5  Duplicates: 0  Warnings: 0

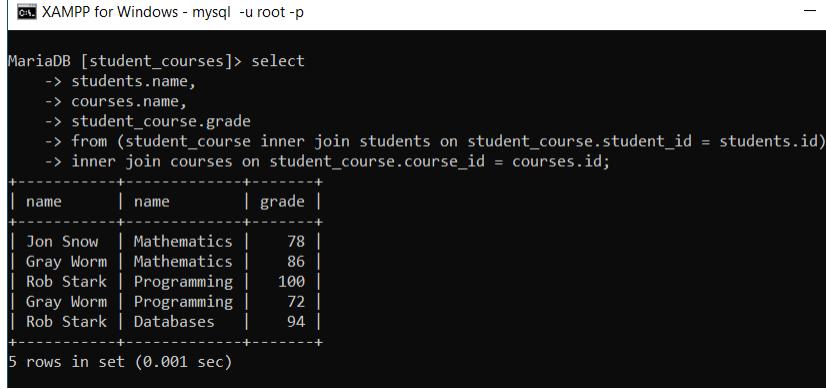
MariaDB [student_courses]>
```

391

Query database tables

Вилучення даних з таблиць

Получение данных из таблиц



```
XAMPP for Windows - mysql -u root -p

MariaDB [student_courses]> select
    -> students.name,
    -> courses.name,
    -> student_course.grade
    -> from (student_course inner join students on student_course.student_id = students.id)
    -> inner join courses on student_course.course_id = courses.id;
+-----+-----+-----+
| name | name | grade |
+-----+-----+-----+
| Jon Snow | Mathematics | 78 |
| Gray Worm | Mathematics | 86 |
| Rob Stark | Programming | 100 |
| Gray Worm | Programming | 72 |
| Rob Stark | Databases | 94 |
+-----+-----+-----+
5 rows in set (0.001 sec)
```

392

Контрольні питання

1. Вираз SQL CREATE DATABASE
2. Вираз SQL DROP DATABASE
3. Вираз SQL CREATE TABLE
4. Вираз SQL DROP TABLE
5. Вираз SQL ALTER TABLE
6. Вираз SQL NOT NULL
7. Вираз SQL UNIQUE
8. Вираз SQL PRIMARY KEY
9. Вираз SQL FOREIGN KEY
10. Вираз SQL CHECK
11. Вираз SQL DEFAULT
12. Вираз SQL INDEX
13. Вираз SQL AUTO INCREMENT

393

Assessment questions

1. SQL CREATE DATABASE statement
2. SQL DROP DATABASE statement
3. SQL CREATE TABLE statement
4. SQL DROP TABLE statement
5. SQL ALTER TABLE statement
6. SQL NOT NULL statement
7. SQL UNIQUE statement
8. SQL PRIMARY KEY statement
9. SQL FOREIGN KEY statement
10. SQL CHECK statement
11. SQL DEFAULT statement
12. SQL INDEX statement
13. SQL AUTO INCREMENT statement

394

Контрольные вопросы

1. Выражение SQL CREATE DATABASE
2. Выражение SQL DROP DATABASE
3. Выражение SQL CREATE TABLE
4. Выражение SQL DROP TABLE
5. Выражение SQL ALTER TABLE
6. Выражение SQL NOT NULL
7. Выражение SQL UNIQUE
8. Выражение SQL PRIMARY KEY
9. Выражение SQL FOREIGN KEY
10. Выражение SQL CHECK
11. Выражение SQL DEFAULT
12. Выражение SQL INDEX
13. Выражение SQL AUTO INCREMENT

395

Реалізація бізнес-логіки у БД Business logic implementation in a DB Реализация бизнес-логики в БД

Тема 4
Topic 4

396

DATABASE VIEWS

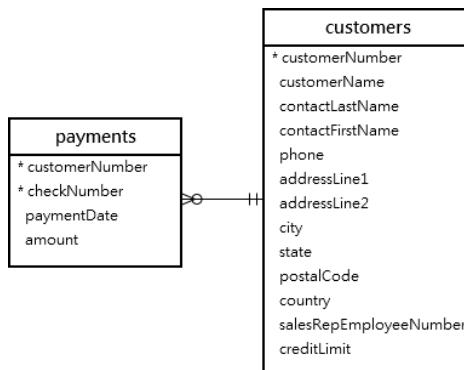
397

Introduction to MySQL Views

Let's see the following tables customers and payments.

Розглянемо наведені нижче таблиці клієнтів та платежів.

Рассмотрим следующие таблицы клиентов и платежей.



398

This query returns data from both tables customers and payments using the inner join:

Цей запит повертає дані з таблиць клієнтів та платежів за допомогою внутрішнього об'єднання:

Этот запрос возвращает данные из таблиц клиентов и платежей с использованием внутреннего соединения:

```

SELECT
    customerName,
    checkNumber,
    paymentDate,
    amount
FROM
    customers
INNER JOIN
    payments USING (customerNumber);
  
```

399

Here is the output:

Ось результат:

Вот результат:

	customerName	checkNumber	paymentDate	amount
▶	Atelier graphique	HQ336336	2004-10-19	6066.78
	Atelier graphique	JM555205	2003-06-05	14571.44
	Atelier graphique	OM314933	2004-12-18	1676.14
	Signal Gift Stores	BO864823	2004-12-17	14191.12
	Signal Gift Stores	HQ55022	2003-06-06	32641.98
	Signal Gift Stores	ND748579	2004-08-20	33347.88
	Australian Collectors, Co.	GG31455	2003-05-20	45864.03
	Australian Collectors, Co.	MA765515	2004-12-15	82261.22
	Australian Collectors, Co.	NP603840	2003-05-31	7565.08
	Australian Collectors, Co.	NR27552	2004-03-10	44894.74
	La Rochelle Gifts	DB933704	2004-11-14	19501.82
	La Rochelle Gifts	LN373447	2004-08-08	47924.19

400

Next time, if you want to get the same information including customer name, check number, payment date, and amount, you need to issue the same query again.

One way to do this is to save the query in a file, either .txt or .sql file so that later you can open and execute it from any MySQL client tool.

A better way to do this is to save the query in the database server and assign a name to it. This named query is called a database view, or simply, view.

By definition, a view is a named query stored in the database catalog.

Наступного разу, якщо буде потрібно отримати ту ж саму інформацію, включаючи ім'я клієнта, номер чека, дату платежу та суму, вам потрібно буде виконати той самий запит ще раз.

Одній із способів зробити це - зберегти запит у файлі .txt, або .sql, щоб згодом ви могли відкрити та виконати його за допомогою будь-якого іншого клієнтського інструментарію MySQL.

Кращий спосіб зробити це - зберегти запит на сервері бази даних і призначити йому ім'я. Цей іменованій запит називається представлением бази даних або просто представлением.

За визначенням представлена - це іменований запит, що зберігається в каталогі бази даних.

В следующий раз, если потребуется получить ту же информацию, включая имя клиента, номер чека, дату платежа и сумму, вам нужно будет выполнить тот же запрос еще раз.

Один из способов сделать это - сохранить запрос в файле .txt или .sql, чтобы впоследствии вы могли открыть и выполнить его с помощью любого другого клиентского инструментария MySQL.

Лучший способ сделать это - сохранить запрос на сервере базы данных и назначить ему имя. Этот именеменный запрос называется представлением базы данных или просто представлением. По определению представление - это именеменный запрос, который сохраняется в каталоге базы данных.

401

To create a new view you use the CREATE VIEW statement. This statement creates a view customerPayments based on the above query above:

Для створення нового представлення використовується оператор CREATE VIEW. Цей вираз створює представлення customerPayments на основі наведеного вище запиту:

Для создания нового представления используется оператор CREATE VIEW. Это выражение создает представление customerPayments на основе приведенного выше запроса:

```
CREATE VIEW customerPayments AS
SELECT
    customerName, checkNumber, paymentDate, amount
FROM
    customers
INNER JOIN
    payments USING (customerNumber);
```

402

Once you execute the CREATE VIEW statement, MySQL creates the view and stores it in the database.

Now, you can reference the view as a table in SQL statements. For example, you can query data from the customerPayments view using the SELECT statement:

Після виконання оператора CREATE VIEW MySQL створює представлення та зберігає його в базі даних.

Тепер можна посилатися на представлення як на таблицю в операторах SQL. Наприклад, можна вилучати дані з представлення customerPayments, використовуючи оператор SELECT:

После выполнения оператора CREATE VIEW MySQL создает представление и сохраняет его в базе данных.

Теперь можно ссылаться на представление как на таблицу в операторах SQL. Например, можно извлекать данные из представления customerPayments, используя оператор SELECT:

```
SELECT * FROM customerPayments;
```

Note that a view does not physically store the data. When you issue the SELECT statement against the view, MySQL executes the underlying query specified in the view's definition and returns the result set. For this reason, sometimes, a view is referred to as a virtual table.

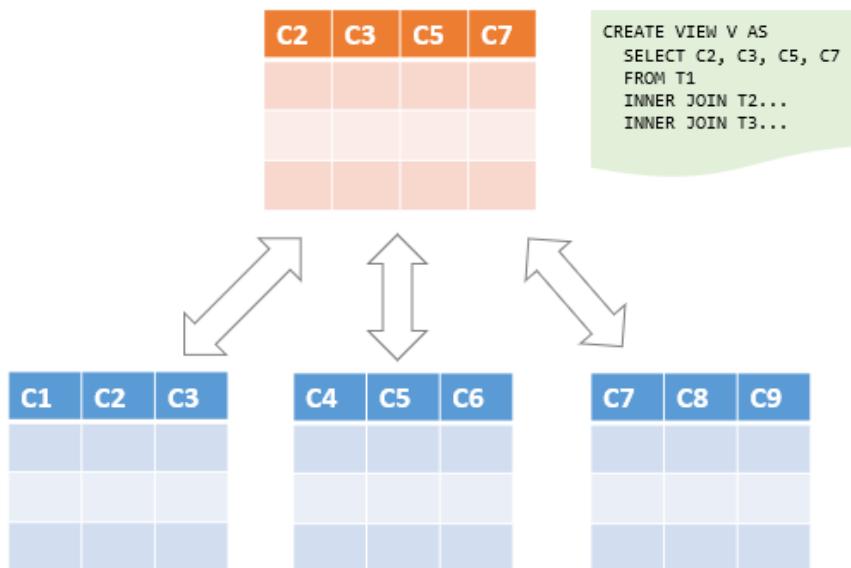
MySQL allows you to create a view based on a SELECT statement that retrieves data from one or more tables.

Зверніть увагу, що представлення фізично не зберігає дані. Коли виконується оператор SELECT для запиту представлення, MySQL виконує базовий запит, зазначений у визначенні представлення, і повертає набір результатів. З цієї причини інколи представлення називають віртуальною таблицею.

MySQL дозволяє створити представлення на основі оператора SELECT, який отримує дані з однієї або декількох таблиць.

Обратите внимание, что представление физически не сохраняет данные. Когда выполняется оператор SELECT для запроса представления, MySQL выполняет базовый запрос, указанный в определении представления, и возвращает набор результатов. По этой причине иногда представление называют виртуальной таблицей.

MySQL позволяет создать представление на основе оператора SELECT, который получает данные из одной или нескольких таблиц.



405

In addition, MySQL even allows you to create a view that does not refer to any table. But you will rarely find this kind of view in practice. For example, you can create a view called daysofweek that return 7 days of a week by executing the following query.

Крім того, MySQL дозволяє навіть створити представлення, яке не стосується жодної таблиці. Але таке представлення можна зрідка зустріти на практиці.

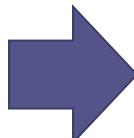
Наприклад, можна створити представлення із назвою daysofweek, яке повертає 7 днів тижня, виконавши наступний запит.

Кроме того, MySQL позволяет даже создать представление, которое не касается ни одной таблицы. Но такое представление можно редко встретить на практике.

Например, можно создать представление под названием daysofweek, которое возвращает 7 дней недели, выполнив следующий запрос.

406

```
CREATE VIEW daysofweek (day) AS
    SELECT 'Mon'
    UNION
    SELECT 'Tue'
    UNION
    SELECT 'Wed'
    UNION
    SELECT 'Thu'
    UNION
    SELECT 'Fri'
    UNION
    SELECT 'Sat'
    UNION
    SELECT 'Sun';
```



	day
▶	Mon
	Tue
	Web
	Thu
	Fri
	Sat
	Sun

```
SELECT * FROM daysofweek;
```

Advantages of MySQL Views

1) Simplify complex query

Views help simplify complex queries. If you have any frequently used complex query, you can create a view based on it so that you can reference to the view by using a simple SELECT statement instead of typing the query all over again.

2) Make the business logic consistent

Suppose you have to repeatedly write the same formula in every query. Or you have a query that has complex business logic. To make this logic consistent across queries, you can use a view to store the calculation and hide the complexity.

3) Add extra security layers

A table may expose a lot of data including sensitive data such as personal and banking information.

By using views and privileges, you can limit which data users can access by exposing only the necessary data to them.

4) Enable backward compatibility

In legacy systems, views can enable backward compatibility.

Suppose, you want to normalize a big table into many smaller ones. And you don't want to impact the current applications that reference the table.

In this case, you can create a view whose name is the same as the table based on the new tables so that all applications can reference the view as if it were a table.

Переваги представлень MySQL

1) Спрощують складні запити

Представлення допомагають спростити складні запити. Якщо існує якийсь часто використовуваний складний запит, можна створити представлення на його основі, щоб посилатися на нього, використовуючи простий оператор SELECT, замість того, щоб вводити запит знову.

2) Роблять бізнес-логіку цілісною

Припустимо, доводиться неодноразово писати одну і ту ж формулу в кожному запиті. Або існує запит із складною бізнес-логікою. Щоб зробити цю логіку цілісною для запитів, можна використовувати представлення для зберігання розрахунків та приховування складності.

3) Додають додаткові рівні захисту

У таблиці може бути багато даних, включаючи конфіденційні дані, такі як особиста та банківська інформація.

Використовуючи представлення та привілеї, можна обмежити доступ до даних, надаючи користувачам лише необхідні дані.

4) Забезпечують зворотну сумісність

У наслідуваннях системах представлення можуть забезпечувати зворотну сумісність.

Припустимо, необхідно декомпозувати велику таблицю на декілька менших. І не бажано впливати на існуючі програми, які посилаються на таблицю.

У цьому випадку можна створити представлення, ім'я якого збігається з таблицею, та яке базується на нових таблицях, щоб усі програми могли посилатися на представлення так, як якщо б це була таблиця.

Преимущества представлений MySQL

1) Упрощают сложные запросы

Представления помогают упростить сложные запросы. Если существует какой-то часто используемый сложный запрос, можно создать представление на его основе, чтобы ссылаться на него, используя простой оператор SELECT, вместо того, чтобы вводить запрос снова.

2) Делают бизнес-логику целостной

Предположим, приходится неоднократно писать одну и ту же формулу в каждом запросе. Или существует запрос со сложной бизнес-логикой. Чтобы сделать эту логику целостной для запросов, можно использовать представления для хранения расчетов и сокрытия сложности.

3) Добавляют дополнительные уровни защиты

В таблице может быть много данных, включая конфиденциальные данные, такие как личная и банковская информация.

Используя представления и привилегии, можно ограничить доступ к данным, предоставляя пользователям только необходимые данные.

4) Обеспечивают обратную совместимость

В унаследованных системах представления могут обеспечивать обратную совместимость.

Допустим, необходимо декомпозировать большую таблицу на несколько меньших. И не желательно влиять на существующие программы, которые ссылаются на таблицу.

В этом случае можно создать представление, имя которого совпадает с таблицей, и основанное на новых таблицах, чтобы все программы могли ссылаться на представление так, как если бы это была таблица.

MySQL CREATE VIEW

The CREATE VIEW statement creates a new view in the database. Here is the basic syntax of the CREATE VIEW statement:

Оператор CREATE VIEW створює нове представлення в базі даних. Ось основний синтаксис виразу CREATE VIEW:

Оператор CREATE VIEW создает новое представление в базе данных. Вот основной синтаксис оператора CREATE VIEW:

CREATE [OR REPLACE] VIEW

[db_name.]view_name [(column_list)]

AS

select-statement;

411

First, specify the name of the view that you want to create after the CREATE VIEW keywords. The name of the view is unique in a database. Because views and tables in the same database share the same namespace, the name a view cannot be the same as the name of an existing table.

Second, use the OR REPLACE option if you want to replace an existing view if the view already exists. If the view does not exist, the OR REPLACE has no effect.

Third, specify a list of columns for the view. By default, the columns of the view are derived from the select list of the SELECT statement. However, you can explicitly specify the column list for the view by listing them in parentheses following the view name.

Finally, specify a SELECT statement that defines the view. The SELECT statement can query data from tables or views. MySQL allows you to use the ORDER BY clause in the SELECT statement but ignores it if you select from the view with a query that has its own ORDER BY clause.

By default, the CREATE VIEW statement creates a view in the current database. If you want to explicitly create a view in a given database, you can qualify the view name with the database name.

412

Спочатку вказується назва представлення, яке потрібно створити після ключових слів CREATE VIEW. Ім'я представлення є унікальним в базі даних. Оскільки представлення та таблиці в одній базі даних мають спільний простір імен, ім'я представлення не може збігатися з ім'ям існуючої таблиці.

По-друге, можна використовувати опцію OR REPLACE, якщо ви хочете замінити існуюче представлення, якщо воно вже існує. Якщо представлення не існує, OR REPLACE не вплине на виконання команди.

По-третє, вказується список стовпців для представлення. За замовчуванням стовпці представлення походять із списку вибору оператора SELECT. Однак можна явно вказати список стовпців для представлення, перелічивши їх у дужках після імені представлення.

Нарешті, вказується оператор SELECT, який визначає представлення. Оператор SELECT може отримувати дані з таблиць або представлень. MySQL дозволяє використовувати вираз ORDER BY в операторі SELECT, але ігнорує його, якщо завертатися до представлення за допомогою запиту, який має власний вираз ORDER BY.

За замовчуванням оператор CREATE VIEW створює представлення у поточній базі даних. Якщо потрібно явно створити представлення в заданій базі даних, можна визначити ім'я представлення з префіксом у вигляді імені бази даних.

413

Сначала указывается название представления, которое нужно создать после ключевых слов CREATE VIEW. Имя представления является уникальным в базе данных. Поскольку представления и таблицы в одной базе данных имеют общее пространство имен, имя представления не может совпадать с именем существующей таблицы.

Во-вторых, можно использовать опцию OR REPLACE, если вы хотите заменить существующее представление, если оно уже существует. Если представление не существует, OR REPLACE не повлияет на выполнение команды.

В-третьих, указывается список столбцов для представления. По умолчанию столбцы представления происходят из списка выбора оператора SELECT. Однако можно явно указать список столбцов для представления, перечислив их в скобках после имени представления.

Наконец, указывается оператор SELECT, который определяет представление. Оператор SELECT может получать данные из таблиц или представлений. MySQL позволяет использовать выражение ORDER BY в операторе SELECT, но игнорирует его, если обращаться к представлению с помощью запроса, который имеет собственное выражение ORDER BY.

По умолчанию оператор CREATE VIEW создает представление в текущей базе данных. Если нужно явно создать представление в заданной базе данных, можно определить имя представления с префиксом в виде имени базы данных.

414

Creating a simple view example

```
CREATE VIEW salePerOrder AS
SELECT
    orderNumber,
    SUM(quantityOrdered * priceEach) total
FROM
    orderDetails
GROUP by orderNumber
ORDER BY total DESC;
```

```
SHOW FULL TABLES;
```

orderdetails	
*	orderNumber
*	productCode
	quantityOrdered
	priceEach
	orderLineNumber

Tables_in_classicmodels	Table_type
customers	BASE TABLE
employees	BASE TABLE
offices	BASE TABLE
orderdetails	BASE TABLE
orders	BASE TABLE
payments	BASE TABLE
productlines	BASE TABLE
products	BASE TABLE
saleperorder	VIEW

415

```
SELECT * FROM salePerOrder;
```

	orderNumber	total
▶	10165	67392.85
	10287	61402.00
	10310	61234.67
	10212	59830.55
	10207	59265.14
	10127	58841.35
	10204	58793.53
	10126	57131.92
	10222	56822.65
	10142	56052.56
	10390	55902.50

416

Creating a view based on another view example

```
CREATE VIEW bigSalesOrder AS
  SELECT
    orderNumber,
    ROUND(total,2) as total
  FROM
    salePerOrder
  WHERE
    total > 60000;
```

```
SELECT orderNumber, total
FROM bigSalesOrder;
```

	orderNumber	total
▶	10165	67392.85
	10287	61402.00
	10310	61234.67

417

Creating a view with join example

```
CREATE OR REPLACE VIEW customerOrders AS
SELECT orderNumber,
      (customerName,
        SUM(quantityOrdered * priceEach) total
     ) AS total
  FROM orderDetails
 INNER JOIN orders
    USING (orderNumber)
 INNER JOIN customers
    USING (customerNumber)
 GROUP BY orderNumber;

SELECT * FROM customerOrders
 ORDER BY total DESC;
```

orderNumber	customerName	total
10165	Dragon Souveniers, Ltd.	67392.85
10287	Vida Sport, Ltd	61402.00
10310	Toms Spezialitäten, Ltd	61234.67
10212	Euro+ Shopping Channel	59830.55
10207	Diecast Collectables	59265.14
10127	Muscle Machine Inc	58841.35
10204	Muscle Machine Inc	58793.53
10126	Corrida Auto Replicas, Ltd	57131.92
10222	Collectable Mini Designs Co.	56822.65
10142	Mini Gifts Distributors Ltd.	56052.56
10390	Mini Gifts Distributors Ltd.	55902.50

418

Creating a view with a subquery example

```
CREATE VIEW aboveAvgProducts AS
SELECT
       productCode,
       productName,
       buyPrice
  FROM
       products
 WHERE
       buyPrice > (
           SELECT
               AVG(buyPrice)
          FROM
               products
          ORDER BY buyPrice DESC;
```

	productCode	productName	buyPrice
▶	S10_4962	1962 LanciaA Delta 16V	103.42
	S18_2238	1998 Chrysler Plymouth Prowler	101.51
	S10_1949	1952 Alpine Renault 1300	98.58
	S24_3856	1956 Porsche 356A Coupe	98.3
	S12_1108	2001 Ferrari Enzo	95.59
	S12_1099	1968 Ford Mustang	95.34
	S18_1984	1995 Honda Civic	93.89
	S18_4027	1970 Triumph Spitfire	91.92

```
SELECT * FROM aboveAvgProducts;
```

419

Creating a view with explicit view columns example

```
CREATE VIEW customerOrderStats (
   (customerName ,
    orderCount
)
AS
    SELECT
        customerName,
        COUNT(orderNumber)
    FROM
        customers
        INNER JOIN
            orders USING (customerNumber)
    GROUP BY customerName;
```

	customerName	orderCount
▶	Bavarian Collectables Imports, Co.	1
	Amica Models & Co.	2
	Auto Associés & Cie.	2
	Boards & Toys Co.	2
	CAF Imports	2
	Cambridge Collectables Co.	2
	Canadian Gift Exchange Network	2
	Classic Gift Ideas, Inc	2
	Clover Collections, Co.	2
	Collectable Mini Designs Co.	2
	Daedalus Designs Imports	2

```
SELECT customerName, orderCount
FROM customerOrderStats
ORDER BY orderCount, customerName;
```

420

MySQL updatable views

In MySQL, views are not only query-able but also updatable. It means that you can use the INSERT or UPDATE statement to insert or update rows of the base table through the updatable view. In addition, you can use DELETE statement to remove rows of the underlying table through the view.

У MySQL до представлень можна не тільки робити запити, але й оновлювати представлення. Це означає, що можна використовувати оператор INSERT або UPDATE, щоб вставити або оновити рядки базової таблиці через оновлюване представлення. Крім того, можна використовувати оператор DELETE, щоб видалити рядки базової таблиці через представлення.

В MySQL к представлениям можно не только делать запросы, но и обновлять представления. Это означает, что можно использовать оператор INSERT или UPDATE, чтобы вставить или обновить строки базовой таблицы через обновляемое представление. Кроме того, можно использовать оператор DELETE, чтобы удалить строки базовой таблицы через представление.

However, to create an updatable view, the SELECT statement that defines the view must not contain any of the following elements:

- Aggregate functions such as MIN, MAX, SUM, AVG, and COUNT.
- DISTINCT.
- GROUP BY clause.
- HAVING clause.
- UNION or UNION ALL clause.
- Left join or outer join.
- Subquery in the SELECT clause or in the WHERE clause that refers to the table appeared in the FROM clause.
- Reference to non-updatable view in the FROM clause.
- Reference only to literal values.
- Multiple references to any column of the base table.

Однак для створення оновлюваного представлення оператор SELECT, який визначає представлення, не повинен містити жодного з наступних елементів:

- Функції агрегації, такі як MIN, MAX, SUM, AVG і COUNT.
- DISTINCT.
- Вирази GROUP BY.
- Вирази HAVING.
- Вирази UNION або UNION ALL.
- Ліве або зовнішнє з'єднання.
- Підзапит в операторі SELECT або у виразі WHERE, що посилається на таблицю, вказану у виразі FROM.
- Посилання на не оновлюване представлення у виразі FROM.
- Посилання лише на символічні значення.
- Декілька посилань на будь-який стовпець базової таблиці.

Однако для создания обновляемого представления оператор SELECT, который определяет представление, не должен содержать ни одного из следующих элементов:

- Функции агрегации, такие как MIN, MAX, SUM, AVG и COUNT.
- DISTINCT.
- Выражения GROUP BY.
- Выражения HAVING.
- Выражения UNION или UNION ALL.
- Левое или внешнее соединение.
- Подзапрос в операторе SELECT или в выражении WHERE, ссылающегося на таблицу, указанную в выражении FROM.
- Ссылка на НЕ обновляемое представление в выражении FROM.
- Ссылка только на символьные значения.
- Несколько ссылок на любой столбец базовой таблицы.

423

MySQL updatable view example

```
CREATE VIEW officeInfo
AS
    SELECT officeCode, phone, city
    FROM offices;
```

```
SELECT
    *
FROM
    officeInfo;
```

	officeCode	phone	city
▶	1	+1 650 219 4782	San Francisco
	2	+1 215 837 0825	Boston
	3	+1 212 555 3000	NYC
	4	+33 14 723 4404	Paris
	5	+81 33 224 5000	Tokyo
	6	+61 2 9264 2451	Sydney
	7	+44 20 7877 2041	London

424

```
UPDATE officeInfo
SET
    phone = '+33 14 723 5555'
WHERE
    officeCode = 4;
```

```
SELECT
    *
FROM
    officeInfo
WHERE
    officeCode = 4;
```

	officeCode	phone	city
▶	4	+33 14 723 5555	Paris

425

Checking updatable view information

```
SELECT
    table_name,
    is_updatable
FROM
    information_schema.views
WHERE
    table_schema = 'classicmodels';
```

	table_name	is_updatable
▶	aboveavgproducts	YES
	customerorders	NO
	officeinfo	YES
	saleperorder	NO

426

Removing rows through the view

```
-- create a new table named items
CREATE TABLE items (
    id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(100) NOT NULL,
    price DECIMAL(11 , 2 ) NOT NULL
);

-- insert data into the items table
INSERT INTO items(name,price)
VALUES('Laptop',700.56),('Desktop',699.99),('iPad',700.50) ;

-- create a view based on items table
CREATE VIEW LuxuryItems AS
    SELECT
        *
    FROM
        items
    WHERE
        price > 700;
-- query data from the LuxuryItems view
SELECT
    *
FROM
    LuxuryItems;
```

	id	name	price
▶	1	Laptop	700.56
	3	iPad	700.50

427

```
DELETE FROM LuxuryItems
WHERE
    id = 3;
```

MySQL returns a message saying that 1 row(s) affected.

MySQL повертає повідомлення про те, що це вплинуло на 1 рядок.

MySQL возвращает сообщение о том, что затронута 1 строка (строки).

```
SELECT * FROM LuxuryItems;
```

	id	name	price
▶	1	Laptop	700.56

```
SELECT * FROM items;
```

	id	name	price
▶	1	Laptop	700.56
	2	Desktop	699.99

428

WITH CHECK OPTION

The WITH CHECK OPTION is an optional clause of the CREATE VIEW statement. The WITH CHECK OPTION prevents a view from updating or inserting rows that are not visible through it. In other words, whenever you update or insert a row of the base tables through a view, MySQL ensures that the insert or update operation is conformed with the definition of the view.

WITH CHECK OPTION є необов'язковим виразом оператора CREATE VIEW. WITH CHECK OPTION запобігає оновленню або вставці рядків, які не видно через нього. Іншими словами, кожного разу, коли ви оновлюєте або вставляєте рядки базових таблиць через представлення, MySQL гарантує, що операція вставки або оновлення відповідає визначеню представлення.

WITH CHECK OPTION является необязательным выражением оператора CREATE VIEW. WITH CHECK OPTION предотвращает обновление или вставку строк, которые не видны из-за него. Иными словами, каждый раз, когда вы обновляете или вставляете строки базовых таблиц через представление, MySQL гарантирует, что операция вставки или обновления соответствует определению представления.

```
CREATE [OR REPLACE] VIEW view_name
AS
    select_statement
    WITH CHECK OPTION;
```

429

```
CREATE OR REPLACE VIEW vps AS
```

```
SELECT
```

```
    employeeNumber,  
    lastname,  
    firstname,  
    jobtitle,  
    extension,  
    email,  
    officeCode,  
    reportsTo
```

	employeeNumber	lastname	firstname	jobtitle
▶	1056	Phan	Mary	VP Sales
	1076	Firrelli	Jeff	VP Marketing

```
FROM
```

```
employees
```

```
WHERE
```

```
    jobTitle LIKE '%VP%';
```

```
SELECT * FROM vps;
```

430

```
INSERT INTO vps ( employeeNumber, firstName, lastName,  
jobTitle, extension, email, officeCode, reportsTo )  
VALUES ( 1703, 'Lily', 'Bush', 'IT Manager', 'x9111',  
'lilybush@classicmodelcars.com', 1, 1002 );
```

```
SELECT * FROM employees  
ORDER BY employeeNumber DESC;
```

	employeeNumber	lastName	firstName	extension	email	officeCode	reportsTo	jobTitle
▶	1703	Bush	Lily	x9111	lilybush@classicmodelcars.com	1	1002	IT Manager
	1702	Gerard	Martin	x2312	mgerard@classicmodelcars.com	4	1102	Sales Rep
	1625	Kato	Yoshimi	x102	ykato@classicmodelcars.com	5	1621	Sales Rep
	1621	Nishi	Mami	x101	mnishi@classicmodelcars.com	5	1056	Sales Rep
	1619	King	Tom	x103	tking@classicmodelcars.com	6	1088	Sales Rep

431

```

CREATE OR REPLACE VIEW vps AS
SELECT
    employeeNumber,
    lastName,
    firstName,
    jobTitle,
    extension,
    email,
    officeCode,
    reportsTo
FROM
    employees
WHERE
    jobTitle LIKE '%VP%'
WITH CHECK OPTION;

INSERT INTO
vps(employeeNumber,firstname,lastname,jobtitle,extension,email,officeCode,repo
rtsTo)
VALUES(1704,'John','Smith','IT
Staff','x9112','johnsmith@classicmodelcars.com',1,1703);

Error Code: 1369. CHECK OPTION failed 'classicmodels.vps'

```

432

```

INSERT INTO
vps(employeeNumber,firstname,lastname,jobtitle,extension,email,offi
ceCode,reportsTo)
VALUES(1704,'John','Smith','SVP
Marketing','x9112','johnsmith@classicmodelcars.com',1,1076);

```

1 rows(s) affected.

```
SELECT * FROM vps;
```

	employeeNumber	lastname	firstname	jobtitle	extension	email	officeCode	reportsTo
▶	1056	Phan	Mary	VP Sales	x4611	mpatterso@classicmodelcars.com	1	1002
	1076	Firrelli	Jeff	VP Marketing	x9273	jfirrelli@classicmodelcars.com	1	1002
	1704	Smith	John	SVP Marketing	x9112	johnsmith@classicmodelcars.com	1	1076

Extra: <https://www.mysqltutorial.org/mysql-view-local-cascaded-in-with-check-option>

MySQL DROP VIEW

The DROP VIEW statement deletes a view from the database. Here is the basic syntax of the DROP VIEW statement:

Вираз DROP VIEW видаляє представлення з бази даних. Ось основний синтаксис виразу DROP VIEW:

Оператор DROP VIEW удаляет представление из базы данных. Вот основной синтаксис оператора DROP VIEW:

DROP VIEW [IF EXISTS] view_name;

To remove multiple views at once, you can use the following syntax:

Щоб видалити кілька представлень одночасно, ви можете використовувати такий синтаксис:

Чтобы удалить сразу несколько представлений, вы можете использовать следующий синтаксис:

DROP VIEW [IF EXISTS] view_name1 [,view_name2]...;

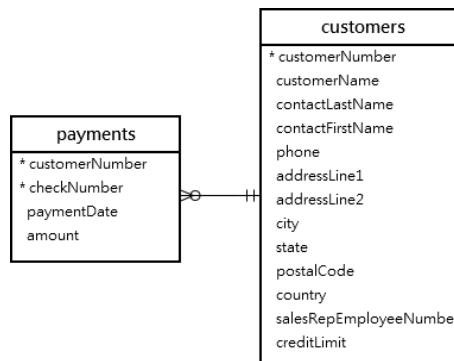
In this syntax, if any view specified after the DROP VIEW clause does not exist, the DROP VIEW statement fails and does not delete any view.

У цьому синтаксисі, якщо жодне представлення, вказане після виразу DROP VIEW, не існує, оператор DROP VIEW не виконується і не видаляє жодного представлення.

В этом синтаксисе, если ни одно представление, указанное после выражения DROP VIEW, не существует, оператор DROP VIEW не выполняется и не удаляет ни представления.

MySQL DROP VIEW – drop a view example

```
CREATE VIEW customerPayments
AS
SELECT
   (customerName,
    SUM(amount)) payment
FROM
    customers
INNER JOIN payments
    USING (customerNumber)
GROUP BY
    customerName;
```



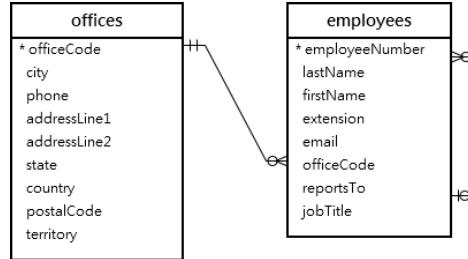
DROP VIEW IF EXISTS customerPayments;

435

MySQL DROP VIEW – drop multiple views example

```
CREATE VIEW employeeOffices AS
```

```
SELECT
    firstName, lastName,
    addressLine1, city
FROM
    employees
    INNER JOIN
    offices
    USING (officeCode);
```



DROP VIEW employeeOffices, eOffices;

Error Code: 1051. Unknown table 'classicmodels.eoffices'

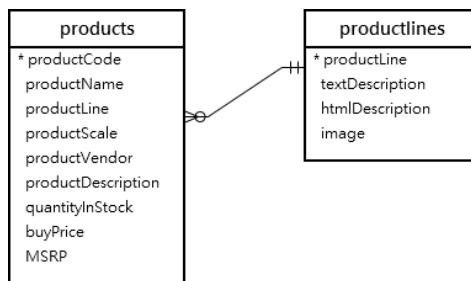
DROP VIEW IF EXISTS employeeOffices, eOffices;

1 warning(s): 1051 Unknown table 'classicmodels.eoffices'

436

```
CREATE VIEW productCatalogs AS
```

```
SELECT
    productLine, productName, msrp
FROM
    products
    INNER JOIN
    productLines USING (productLine);
```



DROP VIEW employeeOffices, productCatalogs;

437

RENAME TABLE

Because views and tables share the same namespace, you can use the RENAME TABLE statement to change the name of a view.

Оскільки представлення даних і таблиць мають один і той самий простір імен, ви можете використовувати оператор RENAME TABLE, щоб змінити назву представлення.

Поскольку представления и таблицы используют одно и то же пространство имен, вы можете использовать оператор RENAME TABLE, чтобы изменить имя представления.

```
RENAME TABLE original_view_name  
TO new_view_name;
```

Another indirect way to rename a view is to use a sequence of the DROP VIEW and CREATE VIEW statement. By using a sequence of DROP and CREATE VIEW statements, you can move a view from one database to another.

Іншим непрямим способом перейменування представлення є використання послідовності операторів DROP VIEW і CREATE VIEW. Використовуючи послідовність операторів DROP і CREATE VIEW, ви можете перемістити представлення з однієї бази даних в іншу.

Другой косвенный способ переименовать представление - использовать последовательность операторов DROP VIEW и CREATE VIEW. Используя последовательность операторов DROP и CREATE VIEW, вы можете перемещать представление из одной базы данных в другую.

438

Renaming a view using the RENAME TABLE statement example

Перейменування представлення на прикладі оператора RENAME TABLE
Переименование представления с использованием оператора RENAME TABLE

```
CREATE VIEW productLineSales AS  
SELECT  
    productLine,  
    SUM(quantityOrdered) totalQtyOrdered  
FROM  
    productLines  
        INNER JOIN  
    products USING (productLine)  
        INNER JOIN  
    orderdetails USING (productCode)  
GROUP BY productLine;
```

```
RENAME TABLE productLineSales  
TO productLineQtySales;
```

```
SHOW FULL TABLES WHERE table_type = 'VIEW';
```

439

Renaming a view using the DROP VIEW and CREATE VIEW sequence example

Перейменування представлення на прикладі DROP VIEW та CREATE VIEW

Переименование представления с использованием примера последовательности DROP VIEW и CREATE VIEW

SHOW CREATE VIEW productLineQtySales;

	View	Create View	character_set_client	collation_connection
▶	productlineqtsales	<pre>CREATE ALGORITHM=UNDEFINED DEFINER = 'root' @ 'localhost' SQL SECURITY DEFINER VIEW `productlineqtsales` AS select `productlines`.`productline` AS `productLine`,sum(`orderdetails`.`quantityOrdered` AS `totalQtyOrdered` from (`productlines` join `products` ...</pre>	utf8mb4	utf8mb4_0900_ai_ci

DROP VIEW productLineQtySales;

Then change the name of the view in DDL and execute it.

Потім змініть ім'я представлення в DDL і запустіть його.

Затем измените имя представления в DDL и выполните его.

440

STORED FUNCTIONS

441

A stored function is a special kind of stored procedure (program) that returns a single value. Typically, you use stored functions to encapsulate common formulas or business rules that are reusable among SQL statements or stored procedures.

Збережена функція - це спеціальна програма, яка повертає єдине значення. Як правило, збережені функції використовуються для інкапсуляції загальних формул або бізнес-правил, які багаторазово використовуються серед операторів SQL або збережених процедур.

Хранимая функция - это специальная программа, которая возвращает единственное значение. Как правило, хранимые функции используются для инкапсуляции общих формул или бизнес-правил, которые многократно используются среди операторов SQL или хранимых процедур.

442

MySQL CREATE FUNCTION

```
DELIMITER $$
```

```
CREATE FUNCTION function_name(
    param1,
    param2,...
)
RETURNS datatype
[NOT] DETERMINISTIC
BEGIN
    -- statements
END $$
```

```
DELIMITER ;
```

443

- First, specify the name of the stored function that you want to create after CREATE FUNCTION keywords.
- Second, list all parameters of the stored function inside the parentheses followed by the function name.
- Third, specify the data type of the return value in the RETURNS statement, which can be any valid MySQL data types.
- Fourth, specify if a function is deterministic or not using the DETERMINISTIC keyword. A deterministic function always returns the same result for the same input parameters whereas a non-deterministic function returns different results for the same input parameters. If you don't use DETERMINISTIC or NOT DETERMINISTIC, MySQL uses the NOT DETERMINISTIC option by default.
- Fifth, write the code in the body of the stored function in the BEGIN END block. Inside the body section, you need to specify at least one RETURN statement. The RETURN statement returns a value to the calling programs. Whenever the RETURN statement is reached, the execution of the stored function is terminated immediately.

444

- Спочатку вказується ім'я збереженої функції, яку потрібно створити після створення ключових слів CREATE FUNCTION.
- По-друге, вказуються усі параметри збереженої функції всередині дужок, за якими слідує ім'я функції.
- По-третє, вказується тип даних значення, що повертається, у операторі RETURNS, який може бути будь-яким дійсним типом даних MySQL.
- По-четверте, вказується, чи є функція детермінованою, використовуючи ключове слово DETERMINISTIC. Детермінована функція завжди повертає одинаковий результат для тих самих входних параметрів, тоді як недетермінована функція повертає різні результати для тих самих входних параметрів. Якщо не використовується DETERMINISTIC або NOT DETERMINISTIC, MySQL за замовчуванням використовує опцію NOT DETERMINISTIC.
- По-п'яте, записується код у тілі збереженої функції в блоці BEGIN END. Всередині тіла функції потрібно вказати принаймні один оператор RETURN. Оператор RETURN повертає значення для програм, що викликають функції. Щоразу, коли досягається оператор RETURN, виконання збереженої функції негайно припиняється.

445

- Сначала указывается имя хранимой функции, которую нужно создать после создания ключевых слов CREATE FUNCTION.
- Во-вторых, указываются все параметры хранимой функции внутри скобок, за которыми следует имя функции.
- В-третьих, указывается тип данных возвращаемого значения, в операторе RETURNS, который может быть любым действительным типом данных MySQL.
- В-четвертых, указывается, является ли функция детерминированной, используя ключевое слово DETERMINISTIC. Детерминирована функция всегда возвращает одинаковый результат для тех же входных параметров, тогда как недетерминированными функция возвращает разные результаты для тех же входных параметров. Если не используется DETERMINISTIC или NOT DETERMINISTIC, MySQL по умолчанию использует опцию NOT DETERMINISTIC.
- В-пятых, записывается код в теле хранимой функции в блоке BEGIN END. Внутри тела функции нужно указать по крайней мере один оператор RETURN. Оператор RETURN возвращает значение для программ, которые вызывают функции. Каждый раз, когда достигается оператор RETURN, выполнение хранимой функции немедленно прекращается.

446

```

DELIMITER $$

CREATE FUNCTION CustomerLevel(
    credit DECIMAL(10,2)
)
RETURNS VARCHAR(20)
DETERMINISTIC
BEGIN
    DECLARE customerLevel VARCHAR(20);

    IF credit > 50000 THEN
        SET customerLevel = 'PLATINUM';
    ELSEIF (credit >= 50000 AND
            credit <= 10000) THEN
        SET customerLevel = 'GOLD';
    ELSEIF credit < 10000 THEN
        SET customerLevel = 'SILVER';
    END IF;
    -- return the customer level
    RETURN (customerLevel);
END$$
DELIMITER ;

```

customers
* customerNumber
customerName
contactLastName
contactFirstName
phone
addressLine1
addressLine2
city
state
postalCode
country
salesRepEmployeeNumber
creditLimit

447



**SHOW FUNCTION STATUS
WHERE db = 'classicmodels';**

```
SELECT
    customerName,
    CustomerLevel(creditLimit)
FROM
    customers
ORDER BY
    customerName;
```

Db	Name	Type	Definer
classicmodels	CustomerLevel	FUNCTION	root@localhost

	customerName	CustomerLevel(creditLimit)
▶	Alpha Cognac	PLATINUM
	American Souvenirs Inc	SILVER
	Amica Models & Co.	PLATINUM
	ANG Resellers	SILVER
	Anna's Decorations, Ltd	PLATINUM
	Anton Designs, Ltd.	SILVER
	Asian Shopping Network, Co	SILVER
	Asian Treasures, Inc.	SILVER
	Atelier graphique	GOLD
	Australian Collectables, Ltd	PLATINUM
	Australian Collectors, Co.	PLATINUM

448

MySQL DROP FUNCTION

DROP FUNCTION [IF EXISTS] function_name;

In this syntax, you specify the name of the stored function that you want to drop after the **DROP FUNCTION** keywords.

The IF EXISTS option allows you to conditionally drop a stored function if it exists. It prevents an error from arising if the function does not exist.

У цьому синтаксисі вказується ім'я збереженої функції, яку потрібно видалити, після ключових слів **DROP FUNCTION**.

Опція IF EXISTS дозволяє видалити збережену функцію, за умови якщо вона існує. Це запобігає виникненню помилки, якщо функція не існує.

В этом синтаксисе указывается имя хранимой функции, которую нужно удалить, после ключевых слов **DROP FUNCTION**.

Опция IF EXISTS позволяет удалить хранимую функцию, при условии если она существует. Это предотвращает ошибки, если функция не существует.

449

```
DELIMITER $$
```

```
CREATE FUNCTION OrderLeadTime (
    orderDate DATE,
    requiredDate DATE
)
RETURNS INT
DETERMINISTIC
BEGIN
    RETURN requiredDate - orderDate;
END$$
```

```
DELIMITER ;
```

orders
* orderNumber
orderDate
requiredDate
shippedDate
status
comments
customerNumber

```
DROP FUNCTION OrderLeadTime;
```

```
DROP FUNCTION IF EXISTS NonExistingFunction;
```

0 row(s) affected, 1 warning(s): 1305 FUNCTION classicmodels.NonExistingFunction does not exist

450

SHOW FUNCTION STATUS

```
SHOW FUNCTION STATUS
[LIKE 'pattern' | WHERE search_condition];
```

The SHOW FUNCTION STATUS statement returns all characteristics of stored functions. The following statement shows all stored functions in the current MySQL server:

Оператор SHOW FUNCTION STATUS повертає всі характеристики збережених функцій. Наступний вираз показує всі збережені функції на поточному сервері MySQL:

Оператор SHOW FUNCTION STATUS возвращает все характеристики сохранившихся функций. Следующее выражение показывает все хранимые функции на текущем сервере MySQL:

```
SHOW FUNCTION STATUS;
```

451

If you just want to show stored functions in a particular database, you can use a WHERE clause in the SHOW FUNCTION STATUS as shown in the following statement:

Якщо потрібно показати збережені функції в певній базі даних, можна використати вираз WHERE у SHOW FUNCTION STATUS, як показано в наступному запиті:

Если нужно показать хранимые функции в определенной базе данных, можно использовать выражение WHERE в SHOW FUNCTION STATUS, как показано в следующем запросе:

```
SHOW FUNCTION STATUS
WHERE db = 'supply';
```

If you want to find the stored functions whose names contain a specific word, you can use the LIKE clause:

Якщо потрібно знайти збережені функції, імена яких містять конкретне слово, можна використати речення LIKE:

Если нужно найти хранимые функции, имена которых содержат конкретное слово, можно использовать предложение LIKE:

```
SHOW FUNCTION STATUS LIKE '%Customer%';
```

452

MySQL data dictionary has a routines table that stores information about the stored functions of all databases in the current MySQL server.

Словник даних MySQL містить таблицю підпрограм, яка зберігає інформацію про збережені функції всіх баз даних на поточному сервері MySQL.

Словарь данных MySQL содержит таблицу подпрограмм, которая хранит информацию о хранимых функциях всех баз данных на текущей сервере MySQL.

```
SELECT
    routine_name
FROM
    information_schema.routines
WHERE
    routine_type = 'FUNCTION'
        AND routine_schema = 'supply';
```

453

STORED PROCEDURES

454

The following SELECT statement returns all rows in the table customers from the sample database:

Наступний оператор SELECT повертає всі рядки з таблиці customers бази даних:

Следующий оператор SELECT возвращает все строки из таблицы customers базы данных:

```

SELECT
    customerName,
    city,
    state,
    postalCode,
    country
FROM
    customers
ORDER BY customerName;
  
```

	customerName	city	state	postalCode	country
▶	Alpha Cognac	Toulouse	NULL	31000	France
	American Souvenirs Inc	New Haven	CT	97823	USA
	Amica Models & Co.	Torino	NULL	10100	Italy
	ANG Resellers	Madrid	NULL	28001	Spain
	Anna's Decorations, Ltd	North Sydney	NSW	2060	Australia
	Anton Designs, Ltd.	Madrid	NULL	28023	Spain
	Asian Shopping Network, Co	Singapore	NULL	038988	Singapore
	Asian Treasures, Inc.	Cork	Co. Cork	NULL	Ireland
	Atelier graphique	Nantes	NULL	44000	France
	Australian Collectables, Ltd	Glen Waverly	Victoria	3150	Australia
	Australian Collectors, Co.	Melbourne	Victoria	3004	Australia

455

The following CREATE PROCEDURE statement creates a new stored procedure that wraps the query above:

Наступний оператор CREATE PROCEDURE створює нову збережену процедуру, яка приховує наведений вище запит:

Следующий оператор CREATE PROCEDURE создает новую хранимую процедуру, которая скрывает вышеприведенный запрос:

```
DELIMITER $$
```

```
CREATE PROCEDURE GetCustomers()
BEGIN
    SELECT
        customerName,
        city,
        state,
        postalCode,
        country
    FROM
        customers
    ORDER BY customerName;
END$$
DELIMITER ;
```

456

Once you save the stored procedure, you can invoke it by using the CALL statement:

Після створення збереженої процедури її можна викликати за допомогою оператора CALL:

После создания хранимой процедуры ее можно вызвать с помощью оператора CALL:

```
CALL GetCustomers();
```

And the statement returns the same result as the query.

The first time you invoke a stored procedure, MySQL looks up for the name in the database catalog, compiles the stored procedure's code, place it in a memory area known as a cache, and execute the stored procedure.

If you invoke the same stored procedure in the same session again, MySQL just executes the stored procedure from the cache without having to recompile it.

457

```
CALL GetCustomers();
```

Оператор поверне той самий результат, що й запит.

Перший раз, коли збережену процедуру викликають, MySQL шукає ім'я в каталозі бази даних, компілює код збереженої процедури, поміщає його в область пам'яті, відому як кеш, і виконує збережену процедуру.

Якщо знову викликати ту ж саму збережену процедуру в тому ж сеансі, MySQL просто виконає збережену процедуру з кешу без необхідності її перекомпіляції.

Оператор вернет тот же результат, что и запрос.

Первый раз, когда хранимую процедуру вызывают, MySQL ищет имя в каталоге базы данных, компилирует код хранимой процедуры, помещает его в область памяти, известную как кэш, и выполняет хранимую процедуру.

Если снова вызвать ту же хранимую процедуру в том же сеансе, MySQL просто выполнит хранимую процедуру из кэша без необходимости ее перекомпиляции.

458

A stored procedure can have parameters so you can pass values to it and get the result back. For example, you can have a stored procedure that returns customers by country and city. In this case, the country and city are parameters of the stored procedure.

A stored procedure may contain control flow statements such as IF, CASE, and LOOP that allow you to implement the code in the procedural way.

A stored procedure can call other stored procedures or stored functions, which allows you to modularize your code.

Збережена процедура може мати параметри, тому можна передавати їй значення і повертати результат. Наприклад, може існувати збережена процедура, яка повертає клієнтів за країною та містом. У цьому випадку країна та місто є параметрами збереженої процедури. Збережена процедура може містити оператори потоку керування, такі як IF, CASE та LOOP, які дозволяють реалізувати код процедурним способом.

Збережена процедура може викликати інші збережені процедури або збережені функції, що дозволяє модулювати вихідний код.

459

Хранимая процедура может иметь параметры, поэтому можно передавать ей значения и возвращать результат. Например, может существовать хранимая процедура, которая возвращает клиентов по странам и городам. В этом случае страна и город являются параметрами хранимой процедуры.

Хранимая процедура может содержать операторы потока управления, такие как IF, CASE и LOOP, которые позволяют реализовать код процедурным способом.

Хранимая процедура может вызывать другие хранимые процедуры или хранимые функции, что позволяет модулировать исходный код.

460

Advantages	Disadvantages
<ul style="list-style-type: none"> Reduce network traffic <p>Stored procedures help reduce the network traffic between applications and MySQL Server. Because instead of sending multiple lengthy SQL statements, applications have to send only the name and parameters of stored procedures.</p> <ul style="list-style-type: none"> Centralize business logic in the database <p>You can use the stored procedures to implement business logic that is reusable by multiple applications. The stored procedures help reduce the efforts of duplicating the same logic in many applications and make your database more consistent.</p> <ul style="list-style-type: none"> Make database more secure <p>The database administrator can grant appropriate privileges to applications that only access specific stored procedures without giving any privileges on the underlying tables.</p>	<ul style="list-style-type: none"> Resource usages <p>If you use many stored procedures, the memory usage of every connection will increase substantially. Besides, overusing a large number of logical operations in the stored procedures will increase the CPU usage because the MySQL is not well-designed for logical operations.</p> <ul style="list-style-type: none"> Troubleshooting <p>It's difficult to debug stored procedures. Unfortunately, MySQL does not provide any facilities to debug stored procedures like other enterprise database products such as Oracle and SQL Server.</p> <ul style="list-style-type: none"> Maintainances <p>Developing and maintaining stored procedures often requires a specialized skill set that not all application developers possess. This may lead to problems in both application development and maintenance.</p>

Переваги	Недоліки
<ul style="list-style-type: none"> Зменшують мережевий трафік Збережені процедури допомагають зменшити мережевий трафік між програмами та сервером MySQL. Оскільки замість надсилання кількох тривалих операторів SQL програми повинні надсилати лише ім'я та параметри збережених процедур. Централизують бізнес-логіку в базі даних Можна використовувати збережені процедури для реалізації бізнес-логіки, яка багаторазово використовується багатьма програмами. Збережені процедури допомагають зменшити зусилля для дублювання тієї самої логіки у багатьох додатках та роблять базу даних більш цілісною. Роблять базу даних більш захищеною Адміністратор бази даних може надавати відповідні привілеї програмам, які отримують доступ лише до певних збережених процедур, не надаючи жодних привілеїв для таблиць, що лежать в основі. 	<ul style="list-style-type: none"> Використання ресурсів Якщо використовується багато збережених процедур, використання пам'яті кожного з'єднання значно зросте. Крім того, надмірне використання великої кількості логічних операцій у збережених процедурах збільшить використання центрального процесора, оскільки MySQL не дуже придатний для логічних операцій. Вирішення проблем Важко відлагодити збережені процедури. На жаль, MySQL не надає жодних можливостей для відлагодження збережених процедур, як інші продукти корпоративних баз даних, такі як Oracle та SQL Server. Технічне обслуговування Розробка та підтримка збережених процедур часто вимагає спеціального набору навичок, яким володіють не всі розробники програм. Це може привести до проблем як у розробці додатків, так і в обслуговуванні.

Преимущества	Недостатки
<ul style="list-style-type: none"> Уменьшают сетевой трафик Хранимые процедуры помогают уменьшить сетевой трафик между приложениями и сервером MySQL. Поскольку вместо отправки нескольких длительных операторов SQL программы должны передавать только имя и параметры хранимых процедур. Централизуют бизнес-логику в базе данных Можно использовать хранимые процедуры для реализации бизнес-логики, которая многократно используется многими программами. Хранимые процедуры помогают уменьшить усилия для дублирования той же логики во многих приложениях и делают базу данных более целостной. Делают базу данных более защищенной Администратор базы данных может предоставлять соответствующие привилегии программам, которые получают доступ только к определенным хранимым процедурам, не оказывая никаких привилегий для таблиц, лежащих в основе. 	<ul style="list-style-type: none"> Использование ресурсов Если используется много хранимых процедур, использование памяти каждого соединения значительно возрастет. Кроме того, чрезмерное использование большого количества логических операций в хранимых процедурах увеличит использование центрального процессора, поскольку MySQL не слишком пригоден для логических операций. Решение проблем Трудно отлаживать хранимые процедуры. К сожалению, MySQL не предоставляет никаких возможностей для отладки хранимых процедур, как другие корпоративные базы данных, такие как Oracle и SQL Server. Техническое обслуживание Разработка и поддержка хранимых процедур часто требует специального набора навыков, которым обладают не все разработчики программ. Это может привести к проблемам как в разработке приложений, так и в обслуживании.

MySQL Delimiter

When you write SQL statements, you use the semicolon (;) to separate two statements like the following example:

Під час запису операторів SQL, використовується крапка з комою (;), щоб відокремити два оператори, як у наступному прикладі:

Во время записи операторов SQL, используется точка с запятой (;), чтобы отделить два оператора, как в следующем примере:

```
SELECT * FROM products;
```

```
SELECT * FROM customers;
```

A MySQL client program such as MySQL Workbench or mysql program uses the (;) delimiter to separate statements and executes each statement separately.

Клієнтська програма MySQL, така як MySQL Workbench або програма mysql, використовує розділювач (;) щоб відокремити вирази та виконати кожен з них окремо.

Клиентская программа MySQL, такая как MySQL Workbench или программа mysql, использует разделитель (;) чтобы отделить выражения и выполнить каждое из них в отдельности.

A stored procedure, however, consists of multiple statements separated by a semicolon (;).

If you use a MySQL client program to define a stored procedure that contains semicolon characters, the MySQL client program will not treat the whole stored procedure as a single statement, but many statements.

Therefore, you must redefine the delimiter temporarily so that you can pass the whole stored procedure to the server as a single statement.

Однак збережена процедура складається з декількох операторів, розділених крапкою з комою (;).

Якщо використовується клієнтська програма MySQL для визначення збереженої процедури, що містить крапку з комою, програма-клієнт MySQL буде трактувати всю збережену процедуру не як один вираз, а як множину операторів. Отже, необхідно тимчасово перевизначити роздільник, щоб можна було передати всю збережену процедуру на сервер як єдиний оператор.

Однако хранимая процедура состоит из нескольких операторов, разделенных точкой с запятой (;).

Если используется клиентская программа MySQL для определения хранимой процедуры, содержащей точку с запятой, программа-клиент MySQL будет трактовать всю хранимую процедуру не как одно выражение, а как множество операторов.

Следовательно, необходимо временно переопределить разделитель, чтобы можно было передать всю хранимую процедуру на сервер как единственный оператор.

465

To redefine the default delimiter, you use the DELIMITER command:

Щоб перевизначити роздільник за замовчуванням, потрібно використовувати команду DELIMITER:

Чтобы переопределить разделитель по умолчанию, нужно использовать команду DELIMITER:

`DELIMITER delimiter_character`

The delimiter character may consist of a single character or multiple characters e.g., // or \$\$. However, you should avoid using the backslash (\) because this is the escape character in MySQL.

Розділовач може складатися з одного символу або декількох символів, наприклад, // або \$\$. Однак слід уникати використання зворотної косої риски (\), оскільки це символ екранування в MySQL.

Разделитель может состоять из одного символа или нескольких символов, например, // или \$\$. Однако следует избегать использования обратной косой черты (\), поскольку это символ экранирования в MySQL.

466

Once change the delimiter, you can use the new delimiter to end a statement as follows:

Змінивши роздільник, можна використовувати новий роздільник, щоб закінчити оператор наступним чином:

Изменив разделитель, можно использовать новый разделитель, чтобы закончить оператор следующим образом:

```
DELIMITER //
```

```
SELECT * FROM customers //
```

```
SELECT * FROM products //
```

To change the delimiter back to semicolon, you should use this statement:

Щоб змінити роздільник назад на крапку з комою, необхідно використовувати наступний вираз:

Чтобы изменить разделитель обратно на точку с запятой, необходимо использовать следующее выражение:

```
DELIMITER ;
```

467

A stored procedure typically contains multiple statements separated by semicolon (;). To use compile of the whole stored procedure as a single compound statement, you need to temporarily change the delimiter from the semicolon (;) to another delimiter such as \$\$ or //:

Збережена процедура зазвичай містить кілька операторів, розділених крапкою з комою (;). Щоб використовувати компіляцію всієї збереженої процедури як єдиного складеного оператора, необхідно тимчасово змінити роздільник із крапки з комою (;) на роздільник, наприклад \$\$ або //:

Хранимая процедура обычно содержит несколько операторов, разделенных точкой с запятой (;). Чтобы использовать компиляцию всей хранимой процедуры как единого составного оператора, необходимо временно изменить разделитель с точки с запятой (;) на разделитель, например \$\$ или //:

```
DELIMITER $$
```

```
CREATE PROCEDURE sp_name()
BEGIN
    -- statements
END $$

DELIMITER ;
```

468

MySQL CREATE PROCEDURE

This query returns all products in the products table from the sample database.
Цей запит повертає всі продукти з таблиці products бази даних.
Этот запрос возвращает все продукты из таблицы products базы данных.

```
SELECT * FROM products;
```

The following statement creates a new stored procedure that wraps the query:
Наступний оператор створює нову збережену процедуру, яка обгортася запитом:
Следующий оператор создает новую хранимую процедуру, которая обертывает запрос:

```
DELIMITER //
```

```
CREATE PROCEDURE GetAllProducts()
BEGIN
    SELECT * FROM products;
END //

DELIMITER ;
```

Here is the basic syntax of the CREATE PROCEDURE statement:

Основний синтаксис оператора CREATE PROCEDURE:

Основной синтаксис оператора CREATE PROCEDURE:

```
CREATE PROCEDURE procedure_name(parameter_list)
BEGIN
    statements;
END //
```

First, specify the name of the stored procedure that you want to create after the CREATE PROCEDURE keywords.

Second, specify a list of comma-separated parameters for the stored procedure in parentheses after the procedure name.

Third, write the code between the BEGIN END block. The above example just has a simple SELECT statement. After the END keyword, you place the delimiter character to end the procedure statement.

Спочатку вказується ім'я збереженої процедури, яку потрібно створити після ключових слів CREATE PROCEDURE.

По-друге, вказується перелік розділених комами параметрів для збереженої процедури в дужках після назви процедури.

По-третє, задається код між виразами BEGIN та END. У наведеному вище прикладі вказано лише простий оператор SELECT. Після ключового слова END вказується символ роздільника, щоб закінчити оператор процедури.

Сначала указывается имя хранимой процедуры, которую нужно создать после ключевых слов CREATE PROCEDURE.

Во-вторых, указывается перечень разделенных запятыми параметров для хранимой процедуры в скобках после названия процедуры.

В-третьих, задается код между выражениями BEGIN и END. В приведенном выше примере указано только простой оператор SELECT. После ключевого слова END указывается символ разделителя, чтобы закончить оператор процедуры.

To execute a stored procedure, you use the CALL statement:

Для виконання збереженої процедури можна використовувати оператор CALL:

Для выполнения хранимой процедуры можно использовать оператор CALL:

```
CALL stored_procedure_name(argument_list);
```

In this syntax, you specify the name of the stored procedure after the CALL keyword. If the stored procedure has parameters, you need to pass arguments inside parentheses following the stored procedure name.

У цьому синтаксисі вказується назуব збереженої процедури після ключового слова CALL. Якщо збережена процедура має параметри, потрібно передати аргументи всередині дужок після імені збереженої процедури.

В этом синтаксисе указывается название хранимой процедуры после ключевого слова CALL. Если хранимая процедура имеет параметры, нужно передать аргументы внутри скобок после имени хранимой процедуры.

This example illustrates how to call the GetAllProducts() stored procedure:

Цей приклад ілюструє, як викликати збережену процедуру GetAllProducts():

Этот пример иллюстрирует, как вызывать хранимую процедуру GetAllProducts():

```
CALL GetAllProducts();
```

	productCode	productName	productLine	productScale	productVendor
▶	S10_1678	1969 Harley Davidson Ultimate Chopper	Motorcycles	1:10	Min Lin Diecast
	S10_1949	1952 Alpine Renault 1300	Classic Cars	1:10	Classic Metal Creations
	S10_2016	1996 Moto Guzzi 1100i	Motorcycles	1:10	Highway 66 Mini Classics
	S10_4698	2003 Harley-Davidson Eagle Drag Bike	Motorcycles	1:10	Red Start Diecast
	S10_4757	1972 Alfa Romeo GTA	Classic Cars	1:10	Motor City Art Classics
	S10_4962	1962 LanciaA Delta 16V	Classic Cars	1:10	Second Gear Diecast
	S12_1099	1968 Ford Mustang	Classic Cars	1:12	Autoart Studio Design
	S12_1108	2001 Ferrari Enzo	Classic Cars	1:12	Second Gear Diecast

473

MySQL DROP PROCEDURE

The **DROP PROCEDURE** deletes a stored procedure from the database.

DROP PROCEDURE видаляє збережену процедуру з бази даних.

DROP PROCEDURE удаляет хранимую процедуру из базы данных.

```
DROP PROCEDURE [IF EXISTS] stored_procedure_name;
```

When you drop a procedure that does not exist without using the IF EXISTS option, MySQL issues an error. In this case, if you use the IF EXISTS option, MySQL issues a warning instead.

При видаленні процедури, яка не існує, не використовуючи опцію IF EXISTS, MySQL видаватиме помилку. У тому випадку, якщо використовувати опцію IF EXISTS, замість помилки MySQL видаватиме попередження.

При удалении процедуры, которая не существует, не используя опцию IF EXISTS, MySQL будет выдавать ошибку. В том случае, если использовать опцию IF EXISTS, вместо ошибки MySQL будет выдавать предупреждение.

474

```
DELIMITER $$
```

```
CREATE PROCEDURE GetEmployees()
```

```
BEGIN
```

```
    SELECT
```

```
        firstName,  
        lastName,  
        city,  
        state,  
        country
```

```
    FROM employees
```

```
    INNER JOIN offices using (officeCode);
```

```
END$$
```

```
DELIMITER ;
```

```
DROP PROCEDURE GetEmployees;
```

475

```
DROP PROCEDURE abc;
```

Error Code: 1305. PROCEDURE test.abc does not exist

```
DROP PROCEDURE IF EXISTS abc;
```

0 row(s) affected, 1 warning(s): 1305 PROCEDURE test.abc does not exist

476

MySQL stored procedure parameters

IN parameters

IN is the default mode. When you define an IN parameter in a stored procedure, the calling program has to pass an argument to the stored procedure. In addition, the value of an IN parameter is protected. It means that even the value of the IN parameter is changed inside the stored procedure, its original value is retained after the stored procedure ends. In other words, the stored procedure only works on the copy of the IN parameter.

Параметри IN

IN – режим параметрів за замовчуванням. Коли визначається параметр IN у збереженій процедурі, програма, що її викликає, повинна передати аргумент збереженої процедури. Крім того, значення параметра IN захищено. Це означає, що навіть якщо значення параметра IN змінюється всередині збереженої процедури, його початкове значення зберігається після виконання збереженої процедури. Іншими словами, збережена процедура працює лише на копії параметра IN.

Параметри IN

IN – режим параметров по умолчанию. Когда определяется параметр IN в хранимой процедуре, программа, которая ее вызывает, должна передать аргумент хранимой процедуре. Кроме того, значения параметра IN защищены. Это означает, что даже если значение параметра IN меняется внутри хранимой процедуры, его первоначальное значение сохраняется после выполнения хранимой процедуры. Иными словами, хранимая процедура работает только на копии параметра IN.

OUT parameters

The value of an OUT parameter can be changed inside the stored procedure and its new value is passed back to the calling program. Notice that the stored procedure cannot access the initial value of the OUT parameter when it starts.

OUT параметри

Значення параметра OUT можна змінити всередині збереженої процедури, а його нове значення передається назад програмі, що викликає процедуру. Зверніть увагу, що збережена процедура не може отримати доступ до початкового значення параметра OUT при її запуску.

OUT параметри

Значение параметра OUT можно изменить внутри хранимой процедуры, а его новое значение передается обратно программе, которая вызывает процедуру. Обратите внимание, что хранимая процедура не может получить доступ к исходному значению параметра OUT при ее запуске.

INOUT parameters

An INOUT parameter is a combination of IN and OUT parameters. It means that the calling program may pass the argument, and the stored procedure can modify the INOUT parameter, and pass the new value back to the calling program.

Параметри INOUT

Параметр INOUT – це поєднання параметрів IN та OUT. Це означає, що програма, яка викликає процедуру, може передавати аргумент, а збережена процедура може змінювати параметр INOUT і передавати нове значення назад програмі, що викликає процедуру.

Параметри INOUT

Параметр INOUT – это сочетание параметров IN и OUT. Это означает, что программа, которая вызывает процедуру, может передавать аргумент, а хранимая процедура может изменять параметр INOUT и передавать новое значение назад программе, которая вызывает процедуру.

479

The following example creates a stored procedure that finds all offices that locate in a country specified by the input parameter `countryName`:

Наступний приклад створює збережену процедуру, яка знаходить усі відділення, які знаходяться у країні, зазначеній входним параметром `countryName`:

Следующий пример создает хранимую процедуру, которая находит все отделения, которые находятся в стране, указанной входным параметром `countryName`:

```
DELIMITER //

CREATE PROCEDURE GetOfficeByCountry(
    IN countryName VARCHAR(255)
)
BEGIN
    SELECT *
    FROM offices
    WHERE country = countryName;
END //

DELIMITER ;
```

480

Suppose that you want to find offices locating in the USA, you need to pass an argument (USA) to the stored procedure as shown in the following query:

Припустимо, що потрібно знайти офіси, що знаходяться в США, тобто потрібно передати аргумент (США) збереженій процедурі, як показано в наступному запиті:

Предположим, что нужно найти офисы, которые находятся в США, то есть нужно передать аргумент (США) хранимой процедуре, как показано в следующем запросе:

```
CALL GetOfficeByCountry('USA');
```

	officeCode	city	phone	addressLine1	addressLine2	state	country	postalCode	territory
▶	1	San Francisco	+1 650 219 4782	100 Market Street	Suite 300	CA	USA	94080	NA
	2	Boston	+1 215 837 0825	1550 Court Place	Suite 102	MA	USA	02107	NA
	3	NYC	+1 212 555 3000	523 East 53rd Street	apt. 5A	NY	USA	10022	NA

481

To find offices in France, you pass the literal string France to the GetOfficeByCountry stored procedure as follows:

Щоб знайти офіси у Франції, рядок France передається до збереженої процедури GetOfficeByCountry таким чином:

Чтобы найти офисы во Франции, строка France передается хранимой процедуре GetOfficeByCountry следующим образом:

```
CALL GetOfficeByCountry('France')
```

	officeCode	city	phone	addressLine1	addressLine2	state	country	postalCode	territory
▶	4	Paris	+33 14 723 4404	43 Rue Jouffroy D'abbans	NULL	NULL	France	75017	EMEA

Because the countryName is the IN parameter, you must pass an argument. Fail to do so will result in an error:

Оскільки countryName є параметром IN, потрібно передати аргумент. Якщо цього не зробити, це приведе до помилки:

Поскольку countryName является параметром IN, нужно передать аргумент. Если этого не сделать, это приведет к ошибке:

```
CALL GetOfficeByCountry();
```

```
Error Code: 1318. Incorrect number of arguments for PROCEDURE
test.GetOfficeByCountry; expected 1, got 0
```

482

The following stored procedure returns the number of orders by order status.

Наступна збережена процедура повертає кількість замовлень за статусом замовлення.

Следующая хранимая процедура возвращает количество заказов по статусу заказа.

```
DELIMITER $$
```

```
CREATE PROCEDURE GetOrderCountByStatus (
    IN orderStatus VARCHAR(25),
    OUT total INT
)
BEGIN
    SELECT COUNT(orderNumber)
    INTO total
    FROM orders
    WHERE status = orderStatus;
END$$
```

```
DELIMITER ;
```

483

To find the number of orders that already shipped, you call GetOrderCountByStatus and pass the order status as of Shipped, and also pass a session variable (@total) to receive the return value.

Щоб знайти кількість замовлень, які вже відправлені, необхідно викликати GetOrderCountByStatus і передати статус замовлення як відправленого, а також передати сесійну змінну (@total), щоб отримати повернене значення.

Чтобы найти количество заказов, которые уже отправлены, необходимо вызвать GetOrderCountByStatus и передать статус заказа как отправленного, а также передать сессионную переменную (@total), чтобы получить возвращенное значение.

```
CALL GetOrderCountByStatus('Shipped',@total);
SELECT @total;
```

	@total
▶	303

```
CALL GetOrderCountByStatus('in process',@total);
SELECT @total AS total_in_process;
```

	total_in_process
▶	6

484

In this example, the stored procedure SetCounter() accepts one INOUT parameter (counter) and one IN parameter (inc). It increases the counter (counter) by the value of specified by the inc parameter.

У цьому прикладі збережена процедура SetCounter () приймає один параметр INOUT (counter) та один параметр IN (inc). Він збільшує лічильник (counter) на значення, вказане параметром inc.

В этом примере хранимая процедура SetCounter () принимает один параметр INOUT (counter) и один параметр IN (inc). Он увеличивает счетчик (counter) на значение, указанное параметром inc.

```
DELIMITER $$
```

```
CREATE PROCEDURE SetCounter(
    INOUT counter INT,
    IN inc INT
)
BEGIN
    SET counter = counter + inc;
END$$
```

```
DELIMITER ;
```

485

```

SET @counter = 1;

CALL SetCounter(@counter,1); -- 2
CALL SetCounter(@counter,1); -- 3
CALL SetCounter(@counter,5); -- 8

SELECT @counter; -- 8

```

	@counter
▶	8

486

If you want to change the query, body or parameters of stored procedure in MySQL command line, then you need to DROP PROCEDURE and then CREATE PROCEDURE with new definition.

Якщо виникла необхідність змінити запит, тіло чи параметри збереженої процедури в командному рядку MySQL, то потрібно видалити процедуру, використовуючи DROP PROCEDURE, а потім створити процедуру, використовуючи CREATE PROCEDURE з новим визначенням.

Если возникла необходимость изменить запрос, тело или параметры хранимой процедуры в командной строке MySQL, то нужно удалить процедуру, используя DROP PROCEDURE, а затем создать процедуру, используя CREATE PROCEDURE с новым определением.

MySQL Stored Procedure Variables

A variable is a named data object whose value can change during the stored procedure execution. You typically use variables in stored procedures to hold immediate results. These variables are local to the stored procedure.

Змінна - це іменований об'єкт даних, значення якого може змінюватися під час виконання збереженої процедури. Зазвичай змінні використовуються в збережених процедурах для отримання негайних результатів. Ці змінні є локальними для збереженої процедури.

Переменная - это именуемый объект данных, значение которого может изменяться во время выполнения хранимой процедуры. Обычно переменные используются в хранимых процедурах для получения немедленных результатов. Эти переменные являются локальными для хранимой процедуры.

```
DECLARE variable_name datatype(size) [DEFAULT default_value];
```

The following example declares a variable named totalSale with the data type DEC(10,2) and default value 0.0 as follows:

У наступному прикладі оголошується змінна з іменем totalSale з типом даних DEC(10,2) та значенням за замовчуванням 0.0 наступним чином:

В следующем примере объявляется переменная с именем totalSale с типом данных DEC (10,2) и значением по умолчанию 0.0 следующим образом:

```
DECLARE totalSale DEC(10,2) DEFAULT 0.0;
```

MySQL allows you to declare two or more variables that share the same data type using a single DECLARE statement. The following example declares two integer variables x and y, and set their default values to zero.

MySQL дозволяє оголосити дві або більше змінних, що мають одинаковий тип даних, використовуючи один оператор DECLARE. Наступний приклад оголошує дві цілочисельні змінні x та y та встановлює значення за замовчуванням на нуль.

MySQL позволяет объявить две или более переменных, имеющих одинаковый тип данных, используя один оператор DECLARE. Следующий пример объявляет две целочисленные переменные x и y и устанавливает значение по умолчанию на ноль.

```
DECLARE x, y INT DEFAULT 0;
```

Once a variable is declared, it is ready to use. To assign a variable a value, you use the SET statement:

Після оголошення змінної вона готова до використання. Щоб призначити змінний значення, використовується оператор SET:

После объявления переменной она готова к использованию. Чтобы присвоить переменной значение, используется оператор SET:

```
DECLARE total INT DEFAULT 0;
SET total = 10;
```

In addition to the SET statement, you can use the SELECT INTO statement to assign the result of a query to a variable as shown in the following example:

На додаток до оператора SET, можна використовувати оператор SELECT INTO, щоб призначити результат запиту змінній, як показано в наступному прикладі:

В дополнение к оператору SET, можно использовать оператор SELECT INTO, чтобы присвоить результат запроса переменной, как показано в следующем примере:

```
DECLARE productCount INT DEFAULT 0;
```

```
SELECT COUNT(*)
INTO productCount
FROM products;
```

A variable has its own scope that defines its lifetime. If you declare a variable inside a stored procedure, it will be out of scope when the END statement of stored procedure reaches.

When you declare a variable inside the block BEGIN END, it will be out of scope if the END is reached.

MySQL allows you to declare two or more variables that share the same name in different scopes. Because a variable is only effective in its scope. However, declaring variables with the same name in different scopes is not good programming practice.

A variable whose name begins with the @ sign is a session variable. It is available and accessible until the session ends.

Змінна має власну область визначення часу її життя. Якщо оголосити змінну всередині збереженої процедури, вона буде поза областю визначення при досягненні оператора END збереженої процедури.

Якщо оголосити змінну всередині блоку BEGIN END, вона буде поза областю визначення, якщо буде досягнуто END.

MySQL дозволяє оголосити дві або більше змінних, що мають однакові назви в різних областях визначення, оскільки змінна доступна лише у своїй області визначення. Однак оголошення змінних з однаковою назвою в різних областях визначення не є доброю практикою програмування.

Змінна, назва якої починається зі знака @, є сесійною змінною. Вона доступна до закінчення сесії.

491

Переменная имеет свою область определения времени ее жизни. Если объявить переменную внутри хранимой процедуры, она будет вне области определения при достижении оператора END хранимой процедуры.

Если объявить переменную внутри блока BEGIN END, она будет вне области определения, если будет достигнуто END.

MySQL позволяет объявить две или более переменных, имеющих одинаковые названия в различных областях определения, поскольку переменная доступна только в своей области определения. Однако объявления переменных с одинаковым названием в разных областях определение не является хорошей практикой программирования.

Переменная, название которой начинается со знака @, является сеансовой переменной. Она доступна до окончания сеанса.

492

```
DELIMITER $$
```

```
CREATE PROCEDURE GetTotalOrder()
BEGIN
    DECLARE totalOrder INT DEFAULT 0;

    SELECT COUNT(*)
    INTO totalOrder
    FROM orders;

    SELECT totalOrder;
END$$
```

```
DELIMITER ;
```

493

First, declare a variable totalOrder with a default value of zero. This variable will hold the number of orders from the orders table.

Спочатку оголошується змінну totalOrder з нульовим значенням за замовчуванням. Ця змінна міститиме кількість замовлень з таблиці замовлень.

Сначала объявляется переменная totalOrder с нулевым значением по умолчанию. Эта переменная будет содержать количество заказов из таблицы заказов.

```
DECLARE totalOrder INT DEFAULT 0;
```

Second, use the SELECT INTO statement to assign the variable totalOrder the number of orders selected from the orders table:

По-друге, використовується оператор SELECT INTO, щоб призначити змінній totalOrder кількість замовлень, вибраних із таблиці замовлень:

Во-вторых, используется оператор SELECT INTO, чтобы присвоить переменной totalOrder количество заказов, выбранных из таблицы заказов:

```
SELECT COUNT(*)
INTO totalOrder
FROM orders;
```

494

Third, select the value of the variable totalOrder.

По-третє, виводиться значення змінної totalOrder.

В-третьих, выводится значение переменной totalOrder.

```
SELECT totalOrder;
```

```
CALL GetTotalOrder();
```

	totalOrder
▶	326

495

SHOW PROCEDURE STATUS

The SHOW PROCEDURE STATUS statement shows all characteristic of stored procedures including stored procedure names. It returns stored procedures that you have a privilege to access.

Оператор SHOW PROCEDURE STATUS відображає всі характеристики збережених процедур, включаючи імена збережених процедур. Він повертає збережені процедури, право на доступ до яких має користувач.

Оператор SHOW PROCEDURE STATUS отражает все характеристики хранимых процедур, включая имена хранимых процедур. Он возвращает хранимые процедуры, право на доступ к которым имеет пользователь.

SHOW PROCEDURE STATUS;

	Db	Name	Type	Definer
▶	classicmodels	GetAllCustomers	PROCEDURE	root@localhost
	classicmodels	GetAllProducts	PROCEDURE	root@localhost
	classicmodels	GetOfficeByCountry	PROCEDURE	root@localhost
	classicmodels	GetOrderAmount	PROCEDURE	root@localhost
	classicmodels	GetOrderCountByStatus	PROCEDURE	root@localhost
	classicmodels	GetTotalOrder	PROCEDURE	root@localhost
	classicmodels	SetCounter	PROCEDURE	root@localhost

496

For example, this statement lists all stored procedures in the database "test":

Наприклад, цей оператор перераховує всі збережені процедури в базі даних "test":

Например, этот оператор перечисляет все хранимые процедуры в базе данных "test":

SHOW PROCEDURE STATUS WHERE db = 'test';

The following statement shows all stored procedure whose names contain the wordOrder:

Наступний вираз показує всі збережені процедури, імена яких містять слово Order:

Следующее выражение показывает все хранимые процедуры, имена которых содержат слово Order:

SHOW PROCEDURE STATUS LIKE '%Order%'

	Db	Name	Type	Definer
▶	classicmodels	GetOrderAmount	PROCEDURE	root@localhost
	classicmodels	GetOrderCountByStatus	PROCEDURE	root@localhost
	classicmodels	GetTotalOrder	PROCEDURE	root@localhost

497

More for MySQL stored procedures

<https://www.mysqltutorial.org/mysql-stored-procedure-tutorial.aspx>

Section 2. Conditional Statements

- IF statement – show you how to use the `IF THEN` statement in stored procedures.
- CASE statement – introduce you to the `CASE` statements including simple `CASE` and searched `CASE` statements.

Section 3. Loops

- LOOP – learn how to execute a list of statements repeatedly based on a condition.
- WHILE Loop – show you how to execute a loop as long as a condition is true.
- REPEAT Loop – show you how to execute a loop until a search condition is true.
- LEAVE statement – guide you on how to exit a loop immediately.

Section 4. Error Handling

- Handling exceptions – show you how to handle exception and errors in stored procedures.
- Raising errors – learn how to use `SIGNAL` and `RESIGNAL` to raise errors in stored procedures.

Section 5. Cursors

- Cursors – learn how to use cursors to process row by row in a result set.

498

DATABASE TRIGGERS

499

In MySQL, a trigger is a stored program invoked automatically in response to an event such as insert, update, or delete that occurs in the associated table. For example, you can define a trigger that is invoked automatically before a new row is inserted into a table.

MySQL supports triggers that are invoked in response to the INSERT, UPDATE or DELETE event.

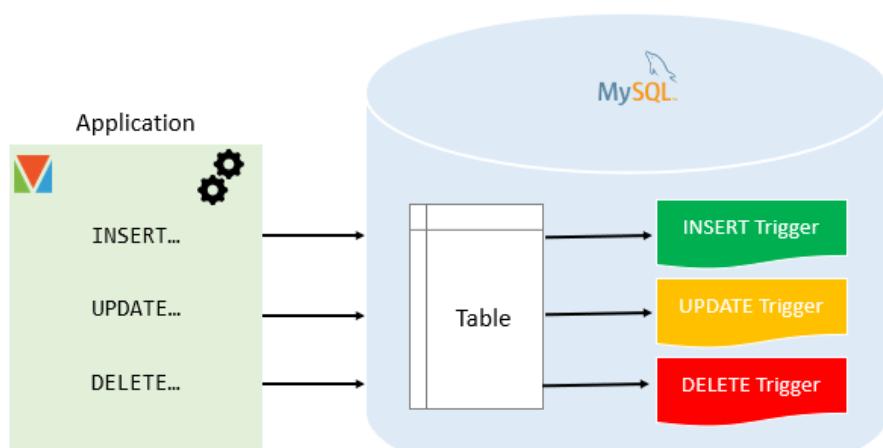
У MySQL тригер – це збережена програма, що викликається автоматично у відповідь на подію, таку як вставка, оновлення чи видалення рядку, що відбувається у відповідній таблиці. Наприклад, можна визначити тригер, який запускається автоматично перед тим, як новий рядок буде вставлений у таблицю.

MySQL підтримує тригери, які викликаються у відповідь на подію INSERT, UPDATE або DELETE.

В MySQL триггер – это хранимая программа, вызываемая автоматически в ответ на событие, такое как вставка, обновление или удаление строки, которое происходит в соответствующей таблице. Например, можно определить триггер, который запускается автоматически перед тем, как новая строка будет вставлена в таблицу.

MySQL поддерживает триггеры, которые вызываются в ответ на событие INSERT, UPDATE или DELETE.

500



501

Advantages of triggers	Disadvantages of triggers
<ul style="list-style-type: none"> Triggers provide another way to check the integrity of data. Triggers handle errors from the database layer. Triggers give an alternative way to run scheduled tasks. By using triggers, you don't have to wait for the scheduled events to run because the triggers are invoked automatically before or after a change is made to the data in a table. Triggers can be useful for auditing the data changes in tables. 	<ul style="list-style-type: none"> Triggers can only provide extended validations, not all validations. For simple validations, you can use the NOT NULL, UNIQUE, CHECK and FOREIGN KEY constraints. Triggers can be difficult to troubleshoot because they execute automatically in the database, which may not be visible to the client applications. Triggers may increase the overhead of the MySQL Server.

502

Переваги тригерів	Недоліки тригерів
<ul style="list-style-type: none"> Тригери забезпечують інший спосіб перевірки цілісності даних. Тригери обробляють помилки на рівні бази даних. Тригери дають альтернативний спосіб виконання запланованих завдань. Використовуючи тригери, не потрібно чекати запуску запланованих подій, оскільки тригери викликаються автоматично до або після внесення змін до даних у таблиці. Тригери можуть бути корисними для перевірки змін даних у таблицях. 	<ul style="list-style-type: none"> Тригери можуть надавати лише розширені перевірки, а не всі перевірки даних. Для простих перевірок можна використовувати обмеження NOT NULL, UNIQUE, CHECK і FOREIGN KEY. Усунення помилок може викликати труднощі, оскільки тригери виконуються автоматично в базі даних, що може бути непомітним для клієнтських програм. Тригери можуть збільшити навантаження на сервер MySQL.

503

Advantages of triggers	Disadvantages of triggers
<ul style="list-style-type: none"> Триггеры обеспечивают другой способ проверки целостности данных. Триггеры обрабатывают ошибки на уровне базы данных. Триггеры дают альтернативный способ выполнения запланированных задач. Используя триггеры, не нужно ждать запуска запланированных событий, поскольку триггеры вызываются автоматически до или после внесения изменений в данные в таблице. Триггеры могут быть полезными для проверки изменений данных в таблицах. 	<ul style="list-style-type: none"> Триггеры могут предоставлять только расширенные проверки, а не все проверки данных. Для простых проверок можно использовать ограничения NOT NULL, UNIQUE, CHECK и FOREIGN KEY. Устранение ошибок может вызвать трудности, так как триггеры выполняются автоматически в базе данных, что может быть незаметным для клиентских программ. Триггеры могут увеличить нагрузку на сервер MySQL.

504

MySQL CREATE TRIGGER

The CREATE TRIGGER statement creates a new trigger. Here is the basic syntax of the CREATE TRIGGER statement:

Оператор CREATE TRIGGER створює новий триггер. Ось основний синтаксис виразу CREATE TRIGGER:

Оператор CREATE TRIGGER создает новый триггер. Вот основной синтаксис выражения CREATE TRIGGER:

```
CREATE TRIGGER trigger_name
{BEFORE | AFTER} {INSERT | UPDATE| DELETE }
ON table_name FOR EACH ROW
trigger_body;
```

The trigger body can access the values of the column being affected by the DML statement.

Тіло тригера може отримати доступ до значень стовпця, на які впливає оператор DML.

Тело триггера может получить доступ к значениям столбца, на которые влияет оператор DML.

505

To distinguish between the value of the columns BEFORE and AFTER the DML has fired, you use the NEW and OLD modifiers.

For example, if you update the column description, in the trigger body, you can access the value of the description before the update OLD.description and the new value NEW.description.

Щоб розрізнати значення стовпців до та після запуску DML, потрібно використовувати модифікатори NEW та OLD.

Наприклад, якщо оновлюється стовпець description, у тілі тригера можна отримати доступ до значення description до оновлення OLD.description та до нового значення NEW.description.

Чтобы различить значения столбцов до и после запуска DML, нужно использовать модификаторы NEW и OLD.

Например, если обновляется столбец description, в теле триггера можно получить доступ к значению description до обновления OLD.description и к новому значению NEW.description.

506

The following table illustrates the availability of the OLD and NEW modifiers:

Наступна таблиця ілюструє доступність модифікаторів OLD і NEW:

Следующая таблица иллюстрирует доступность модификаторов OLD и NEW:

Trigger Event	OLD	NEW
INSERT	No	Yes
UPDATE	Yes	Yes
DELETE	Yes	No

507

Let's create a trigger in MySQL to log the changes of the employees table.
 Створимо тригер в MySQL для реєстрації змін у таблиці співробітників.

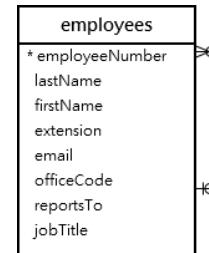
Создадим триггер в MySQL для регистрации изменений в таблице сотрудников.

First, create a new table named employees_audit to keep the changes to the employees table:

Спочатку створимо нову таблицю з назвою staff_audit, щоб зберігати зміни у таблиці співробітників:

Сначала создадим новую таблицу с называнием staff_audit, чтобы сохранять изменения в таблице сотрудников:

```
CREATE TABLE employees_audit (
    id INT AUTO_INCREMENT PRIMARY KEY,
    employeeNumber INT NOT NULL,
    lastName VARCHAR(50) NOT NULL,
    changedat DATETIME DEFAULT NULL,
    action VARCHAR(50) DEFAULT NULL
);
```



508

Next, create a BEFORE UPDATE trigger that is invoked before a change is made to the employees table.

Далі створимо тригер BEFORE UPDATE, який викликається до внесення змін до таблиці співробітників.

Далее создадим триггер BEFORE UPDATE, который вызывается до внесения изменений в таблицы сотрудников.

```
CREATE TRIGGER before_employee_update
    BEFORE UPDATE ON employees
    FOR EACH ROW
    INSERT INTO employees_audit
    SET action = 'update',
        employeeNumber = OLD.employeeNumber,
        lastname = OLD.lastname,
        changedat = NOW();
```

Inside the body of the trigger, we used the OLD keyword to access values of the columns employeeNumber and lastname of the row affected by the trigger.

Усередині тіла тригера використовувалося ключове слово OLD для доступу до значень стовпців workerNumber та lastname рядка, на який впливає триггер.

Внутри тела триггера использовалось ключевое слово OLD для доступа к значениям столбцов workerNumber и lastname строки, на которую влияет триггер.

509

Then, show all triggers in the current database by using the SHOW TRIGGERS statement:

Отримаємо усі тригери в поточній базі даних за допомогою оператора SHOW TRIGGERS:

Получим все триггеры в текущей базе данных с помощью оператора SHOW TRIGGERS:

SHOW TRIGGERS;

Trigger	Event	Table	Statement	Timing
▶ before_employee_update	UPDATE	employees	INSERT INTO employees_audit SET action = 'update', employeeNumber = OLD.employeeNumber, lastname = OLD.lastname, changedat = NOW()	BEFORE

510

After that, update a row in the employees table:

Після цього оновимо рядок у таблиці співробітників:
После этого обновим строку в таблице сотрудников:

```
UPDATE employees
SET
    lastName = 'Phan'
WHERE
    employeeNumber = 1056;
```

Finally, query the employees_audit table to check if the trigger was fired by the UPDATE statement:

Нарешті, переглянемо таблицю staff_audit, щоб перевірити, чи активовано триггер за допомогою оператора UPDATE:
Наконец, просмотрим таблицу staff_audit, чтобы проверить, был ли активирован триггер с помощью оператора UPDATE:

SELECT * FROM employees_audit;

	id	employeeNumber	lastname	changedat	action
▶	1	1056	Patterson	2019-09-06 15:38:30	update

511

MySQL DROP TRIGGER

The DROP TRIGGER statement deletes a trigger from the database.

Оператор DROP TRIGGER видаляє тригер з бази даних.

Оператор DROP TRIGGER удаляет триггер из базы данных.

```
DROP TRIGGER [IF EXISTS] [schema_name.]trigger_name;
```

If you drop a trigger that does not exist without using the IF EXISTS clause, MySQL issues an error. However, if you use the IF EXISTS clause, MySQL issues a NOTE instead.

Якщо спробувати видалити тригер, який не існує, не використовуючи вираз IF EXISTS, MySQL видаватиме помилку. Однак, якщо використовувати вираз IF EXISTS, MySQL видаватиме попередження.

Если попытаться удалить триггер, который не существует, используя выражение IF EXISTS, MySQL выдаст ошибку. Однако, если использовать выражение IF EXISTS, MySQL выдаст предупреждение.

512

```
CREATE TABLE billings (
    billingNo INT AUTO_INCREMENT,
    customerNo INT,
    billingDate DATE,
    amount DEC(10, 2),
    PRIMARY KEY (billingNo)
);
```

The trigger activates before any update. If the new amount is 10 times greater than the current amount, the trigger raises an error.

Тригер активується перед будь-яким оновленням. Якщо нова сума в 10 разів перевищує поточну, тригер викликає помилку.

Тригер активируется перед любым обновлением. Если новая сумма в 10 раз превышает текущую, триггер вызывает ошибку.

```
DELIMITER $$  
CREATE TRIGGER before_billing_update  
BEFORE UPDATE  
ON billings FOR EACH ROW  
BEGIN  
    IF new.amount > old.amount * 10 THEN  
        SIGNAL SQLSTATE '45000'  
        SET MESSAGE_TEXT = 'New amount cannot be 10 times greater than the current  
amount.';  
    END IF;  
END$$  
DELIMITER ;
```

513

SHOW TRIGGERS;

	Trigger	Event	Table	Statement
▶	before_billing_update	UPDATE	billings	BEGIN IF new.amount > old.amount * 10 THEN SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'New amount cannot be times greater than the current amount.'; END IF; END
	before_employee_update	UPDATE	employees	INSERT INTO employees_audit SET action = 'update', employeeNumber = OLD.employeeNumber, lastname = OLD.lastname, changedat = NOW()

DROP TRIGGER before_billing_update;
 SHOW TRIGGERS;

	Trigger	Event	Table	Statement	Timing
▶	before_employee_update	UPDATE	employees	INSERT INTO employees_audit SET action = 'update', employeeNumber = OLD.employeeNumber, lastname = OLD.lastname, changedat = NOW()	BEFORE

514

More for MySQL triggers

<https://www.mysqltutorial.org/mysql-triggers.aspx>

- Create a BEFORE INSERT trigger – show you how to create a `BEFORE INSERT` trigger to maintain a summary table from another table.
- Create an AFTER INSERT trigger – describe how to create an `AFTER INSERT` trigger to insert data into a table after inserting data into another table.
- Create a BEFORE UPDATE trigger – learn how to create a `BEFORE UPDATE` trigger that validates data before it is updated to the table.
- Create an AFTER UPDATE trigger – show you how to create an `AFTER UPDATE` trigger to log the changes of data in a table.
- Create a BEFORE DELETE trigger – show how to create a `BEFORE DELETE` trigger.
- Create an AFTER DELETE trigger – describe how to create an `AFTER DELETE` trigger.
- Create multiple triggers for a table that have the same trigger event and time – MySQL 8.0 allows you to define multiple triggers for a table that have the same trigger event and time.
- Show triggers – list triggers in a database, table by specific patterns.

515

Контрольні питання

- | | |
|-------------------------------|---------------------------------|
| 1. Вираз CREATE VIEW | 12. Вираз IN, OUT, INOUT |
| 2. Вираз DROP VIEW | 13. Вираз DECLARE, SET |
| 3. Вираз RENAME TABLE | 14. Вираз SHOW PROCEDURE STATUS |
| 4. Вираз SHOW CREATE VIEW | 15. Вираз CREATE TRIGGER |
| 5. Вираз CREATE FUNCTION | 16. Вираз OLD, NEW |
| 6. Вираз DELIMITER | 17. Вираз SHOW TRIGGERS |
| 7. Вираз DROP FUNCTION | 18. Вираз DROP TRIGGER |
| 8. Вираз SHOW FUNCTION STATUS | |
| 9. Вираз CREATE PROCEDURE | |
| 10. Вираз CALL | |
| 11. Вираз DROP PROCEDURE | |

516

Assessment questions

- | | |
|-----------------------------------|-------------------------------------|
| 1. CREATE VIEW statement | statement |
| 2. DROP VIEW statement | 10. CALL statement |
| 3. RENAME TABLE statement | 11. DROP PROCEDURE statement |
| 4. SHOW CREATE VIEW statement | 12. IN, OUT, INOUT statement |
| 5. CREATE FUNCTION statement | 13. DECLARE, SET statement |
| 6. DELIMITER statement | 14. SHOW PROCEDURE STATUS statement |
| 7. DROP FUNCTION statement | 15. CREATE TRIGGER statement |
| 8. SHOW FUNCTION STATUS statement | 16. OLD, NEW statement |
| 9. CREATE PROCEDURE | 17. SHOW TRIGGERS statement |
| | 18. DROP TRIGGER statement |

Контрольные вопросы

1. Выражение CREATE VIEW
2. Выражение DROP VIEW
3. Выражение RENAME TABLE
4. Выражение SHOW CREATE VIEW
5. Выражение CREATE FUNCTION
6. Выражение DELIMITER
7. Выражение DROP FUNCTION
8. Выражение SHOW FUNCTION STATUS
9. Выражение CREATE PROCEDURE
10. Выражение CALL
11. Выражение DROP PROCEDURE
12. Выражение IN, OUT, INOUT
13. Выражение DECLARE, SET
14. Выражение SHOW PROCEDURE STATUS
15. Выражение CREATE TRIGGER
16. Выражение OLD, NEW
17. Выражение SHOW TRIGGERS
18. Выражение DROP TRIGGER

Цілісність. Транзакції. Користувачі
Integrity. Transactions. Users
Целостность. Транзакции.
Пользователи

Тема 5

Topic 5

DATABASE INTEGRITY

MySQL Primary Key

A primary key is a column or a set of columns that uniquely identifies each row in the table. The primary key follows these rules:

- A primary key must contain unique values. If the primary key consists of multiple columns, the combination of values in these columns must be unique.
- A primary key column cannot have NULL values. Any attempt to insert or update NULL to primary key columns will result in an error. Note that MySQL implicitly adds a NOT NULL constraint to primary key columns.
- A table can have one and only one primary key.

Первинний ключ – це стовпець або набір стовпців, який однозначно ідентифікує кожен рядок у таблиці. Первинний ключ відповідає таким правилам:

- Первинний ключ повинен містити унікальні значення. Якщо первинний ключ складається з декількох стовпців, комбінація значень у цих стовпцях повинна бути унікальною.
- Стовпець першого ключа не може мати значення NULL. Будь-яка спроба вставити або оновити NULL для стовпців першого ключа призведе до помилки. Крім того, MySQL неявно додає обмеження NOT NULL до стовпців першого ключа.
- У таблиці може бути лише один первинний ключ.

Первичный ключ – это столбец или набор столбцов, который однозначно идентифицирует каждую строку в таблице.

Первичный ключ соответствует таким правилам:

- Первичный ключ должен содержать уникальные значения. Если первичный ключ состоит из нескольких столбцов, комбинация значений в этих столбцах должна быть уникальной.
- Столбец первичного ключа не может иметь значение NULL. Любая попытка вставить или обновить NULL для столбцов первичного ключа приведет к ошибке. Обратите внимание, что MySQL неявно добавляет ограничение NOT NULL для столбцов первичного ключа.
- В таблице может быть только один первичный ключ.

Because MySQL works faster with integers, the data type of the primary key column should be the integer e.g., INT, BIGINT. And you should ensure sure that value ranges of the integer type for the primary key are sufficient for storing all possible rows that the table may have.

A primary key column often has the AUTO_INCREMENT attribute that automatically generates a sequential integer whenever you insert a new row into the table.

When you define a primary key for a table, MySQL automatically creates an index called PRIMARY.

Оскільки MySQL працює швидше із цілими числами, типом даних стовпця первинного ключа має бути ціле число, наприклад, INT, BIGINT. І необхідно переконатися, що діапазони значень цілочисельного типу для первинного ключа є достатніми для зберігання всіх можливих рядків, які може мати таблиця.

Стовпець первинного ключа часто має атрибут AUTO_INCREMENT, який автоматично генерує послідовне ціле число, коли додається новий рядок у таблицю.

Коли визначається первинний ключ для таблиці, MySQL автоматично створює індекс під назвою PRIMARY.

Поскольку MySQL работает быстрее с целыми числами, типом данных столбца первичного ключа должно быть целое число, например, INT, BIGINT. И необходимо убедиться, что диапазоны значений целочисленного типа для первичного ключа достаточно для хранения всех возможных строк, которые может иметь таблица.

Столбец первичного ключа часто имеет атрибут AUTO_INCREMENT, который автоматически генерирует последовательное целое число, когда добавляется новая строка в таблицу.

Когда определяется первичный ключ для таблицы, MySQL автоматически создает индекс под названием PRIMARY.

523

Typically, you define the primary key for a table in the CREATE TABLE statement.

If the primary key has one column, you can use the PRIMARY KEY constraint as a column constraint:

Як правило, ви визначаєте первинний ключ для таблиці в операторі CREATE TABLE.

Якщо первинний ключ має один стовпець, ви можете використовувати обмеження PRIMARY KEY як обмеження стовпця:

Обычно вы определяете первичный ключ для таблицы в операторе CREATE TABLE.

Если первичный ключ имеет один столбец, вы можете использовать ограничение PRIMARY KEY в качестве ограничения столбца:

```
CREATE TABLE table_name (
    primary_key_column datatype PRIMARY KEY,
    ...
);
```

524

When the primary key has more than one column, you must use the PRIMARY KEY constraint as a table constraint.

Коли первинний ключ має більше одного стовпця, ви повинні використовувати обмеження PRIMARY KEY як обмеження таблиці.

Если первичный ключ имеет более одного столбца, вы должны использовать ограничение PRIMARY KEY в качестве ограничения таблицы.

```
CREATE TABLE table_name (
    primary_key_column1 datatype,
    primary_key_column2 datatype,
    ...
    PRIMARY KEY (column_list)
);
```

In this syntax, you separate columns in the column_list by commas (,).
У цьому синтаксисі стовпці у column_list ви відокремлюєте комами (,).

В этом синтаксисе вы разделяете столбцы в column_list запятыми (,).

525

The PRIMARY KEY table constraint can be used when the primary key has one column:

Обмеження таблиці PRIMARY KEY можна використовувати, коли первинний ключ має один стовпець:

Ограничение таблицы PRIMARY KEY можно использовать, когда первичный ключ имеет один столбец:

```
CREATE TABLE table_name (
    primary_key_column datatype,
    ...,
    PRIMARY KEY (primary_key_column)
);
```

526

The following example creates a table named users whose primary key is the user_id column:
Наступний приклад створює таблицю з іменем users, первинним ключем якої є стовпець user_id:

В следующем примере создается таблица с именем users, первичным ключом которой является столбец user_id:

```
CREATE TABLE users (
    user_id INT AUTO_INCREMENT PRIMARY KEY,
    username VARCHAR(40),
    password VARCHAR(255),
    email VARCHAR(255)
);
```

This statement creates the roles table that has the PRIMARY KEY constraint as the table constraint:
Цей оператор створює таблицю ролей, яка має обмеження PRIMARY KEY як обмеження таблиці:

Этот оператор создает таблицу ролей, которая имеет ограничение PRIMARY KEY в качестве ограничения таблицы:

```
CREATE TABLE roles (
    role_id INT AUTO_INCREMENT,
    role_name VARCHAR(50),
    PRIMARY KEY (role_id)
);
```

In case the primary key consists of multiple columns, you must specify them at the end of the CREATE TABLE statement. You put a comma-separated list of primary key columns inside parentheses followed the PRIMARY KEY keywords.

Якщо первинний ключ складається з декількох стовпців, їх потрібно вказати в кінці оператора CREATE TABLE. Ви поміщаєте розділений комами список стовпців первинного ключа всередині дужок після ключових слів PRIMARY KEY.

Если первичный ключ состоит из нескольких столбцов, вы должны указать их в конце оператора CREATE TABLE. Вы помещаете разделенный запятыми список столбцов первичного ключа в круглые скобки после ключевых слов PRIMARY KEY.

```
CREATE TABLE user_roles (
    user_id INT,
    role_id INT,
    PRIMARY KEY (user_id, role_id),
    FOREIGN KEY (user_id)
        REFERENCES users (user_id),
    FOREIGN KEY (role_id)
        REFERENCES roles (role_id)
);
```

If a table, for some reasons, does not have a primary key, you can use the ALTER TABLE statement to add a primary key to the table as follows:

Якщо таблиця з якихось причин не має первинного ключа, ви можете використовувати оператор ALTER TABLE, щоб додати первинний ключ до таблиці наступним чином:

Если таблица по каким-либо причинам не имеет первичного ключа, вы можете использовать оператор ALTER TABLE для добавления первичного ключа в таблицу следующим образом:

```
ALTER TABLE table_name
ADD PRIMARY KEY (column_list);

CREATE TABLE pkdemos (
    id INT,
    title VARCHAR(255) NOT NULL
);

ALTER TABLE pkdemos
ADD PRIMARY KEY (id);
```

If you add a primary key to a table that already has data, the data in the column(s), which will be included in the primary key, must be unique and not NULL.

Якщо ви додаєте первинний ключ до таблиці, яка вже має дані, дані у стовпцях, які будуть включені до первинного ключа, повинні бути унікальними та не мати значення NULL.

Если вы добавляете первичный ключ в таблицу, в которой уже есть данные, данные в столбцах, которые будут включены в первичный ключ, должны быть уникальными, и не содержать значения NULL.

PRIMARY KEY vs. UNIQUE KEY vs. KEY

KEY is the synonym for INDEX. You use the KEY when you want to create an index for a column or a set of columns that is not the part of a primary key or unique key. A UNIQUE index ensures that values in a column must be unique. Unlike the PRIMARY index, MySQL allows NULL values in the UNIQUE index. In addition, a table can have multiple UNIQUE indexes.

KEY – це синонім INDEX. Ви використовуєте KEY, коли хочете створити індекс для стовпця або набору стовпців, який не є частиною первинного ключа або унікального ключа.

UNIQUE індекс гарантує, що значення в стовпці повинні бути унікальними. На відміну від індексу PRIMARY, MySQL дозволяє значення NULL в індексі UNIQUE. Крім того, таблиця може мати кілька UNIQUE індексів.

KEY – это синоним INDEX. Вы используете KEY, если хотите создать индекс для столбца или набора столбцов, который не является частью первичного ключа или уникального ключа.

UNIQUE индекс гарантирует, что значения в столбце должны быть уникальными. В отличие от индекса PRIMARY, MySQL допускает значение NULL в индексе UNIQUE. Кроме того, таблица может иметь несколько UNIQUE индексов.

Suppose that email and username of users in the users table must be unique. To enforce these rules, you can define UNIQUE indexes for the email and username columns as the following statement:

Припустимо, що електронна пошта та ім'я користувача у таблиці користувачів повинні бути унікальними. Щоб застосувати ці правила, ви можете визначити UNIQUE індекси для стовпців електронної пошти та імені користувача наступним чином:

Предположим, что электронная почта и имя пользователя в таблице пользователей должны быть уникальными. Чтобы обеспечить соблюдение этих правил, вы можете определить UNIQUE индексы для столбцов электронной почты и имени пользователя следующим образом:

```
ALTER TABLE users
ADD UNIQUE INDEX username_unique (username ASC);
```

```
ALTER TABLE users
ADD UNIQUE INDEX email_unique (email ASC);
```

531

MySQL Foreign Key

A foreign key is a column or group of columns in a table that links to a column or group of columns in another table. The foreign key places constraints on data in the related tables, which allows MySQL to maintain referential integrity.

Зовнішній ключ – це стовпець або група стовпців у таблиці, яка посилається на стовпець або групу стовпців в іншій таблиці. Зовнішній ключ накладає обмеження на дані у відповідних таблицях, що дозволяє MySQL підтримувати цілісність посилань.

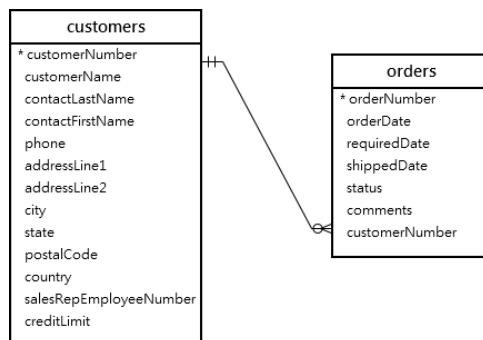
Внешний ключ – это столбец или группа столбцов в таблице, которые связаны со столбцом или группой столбцов в другой таблице. Внешний ключ накладывает ограничения на данные в связанных таблицах, что позволяет MySQL поддерживать ссылочную целостность.

532

In this diagram, each customer can have zero or many orders and each order belongs to one customer.

На цій схемі кожен замовник може мати нуль або багато замовлень, і кожне замовлення належить одному замовнику.

На этой диаграмме у каждого покупателя может быть ноль или несколько заказов, и каждый заказ принадлежит одному покупателю.



533

The relationship between customers table and orders table is one-to-many. And this relationship is established by the foreign key in the orders table specified by the customerNumber column.

The customerNumber column in the orders table links to the customerNumber primary key column in the customers table.

The customers table is called the parent table or referenced table, and the orders table is known as the child table or referencing table.

Відношення між таблицею customers та таблицею orders має тип "один до багатьох". І це відношення встановлюється зовнішнім ключем у таблиці orders, заданим у стовпці customerNumber.

Стовпець customerNumber у таблиці orders посилається на стовпець первинного ключа customerNumber у таблиці customers.

Таблиця customers називається батьківською таблицею або головною таблицею, а таблиця orders відома як дочірня таблиця або залежна таблиця.

Отношение между таблицей customers и таблицей orders имеет тип "один ко многим". И это отношение устанавливается внешним ключом в таблице orders, заданным в столбце customerNumber.

Столбец customerNumber в таблице orders ссылается на столбец первичного ключа customerNumber в таблице customers.

Таблица customers называется родительской таблицей или главной таблицей, а таблица orders известна как дочерняя таблица или зависимая таблица.

534

Typically, the foreign key columns of the child table often refer to the primary key columns of the parent table.

A table can have more than one foreign key where each foreign key references to a primary key of the different parent tables.

Зазвичай стовпці зовнішнього ключа дочірньої таблиці часто посилаються на стовпці первинного ключа батьківської таблиці. Таблиця може мати більше одного зовнішнього ключа, де кожен зовнішній ключ посилається на первинний ключ різних батьківських таблиць.

Как правило, столбцы внешнего ключа дочерней таблицы часто ссылаются на столбцы первичного ключа родительской таблицы.

Таблица может иметь несколько внешних ключей, каждый из которых ссылается на первый ключ разных родительских таблиц.

535

Once a foreign key constraint is in place, the foreign key columns from the child table must have the corresponding row in the primary key columns of the parent table or values in these foreign key column must be NULL.

For example, each row in the orders table has a customerNumber that exists in the customerNumber column of the customers table. Multiple rows in the orders table can have the same customerNumber.

Як тільки встановлено обмеження зовнішнього ключа, стовпці зовнішнього ключа з дочірньої таблиці повинні мати відповідний рядок у стовпцях первинного ключа батьківської таблиці або значення в цьому стовпці зовнішнього ключа повинні мати значення NULL.

Наприклад, у кожному рядку таблиці orders є customerNumber, який існує у стовпці customerNumber таблиці customers. Кілька рядків у таблиці orders можуть мати одинаковий customerNumber.

Как только установлено ограничение внешнего ключа, столбцы внешнего ключа дочерней таблицы должны иметь соответствующую строку в столбцах первичного ключа родительской таблицы или значение в этом столбце внешнего ключа должны иметь значение NULL.

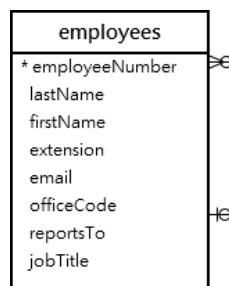
Например, в каждой строке таблицы orders есть customerNumber, который существует в столбце customerNumber таблицы customers. Несколько строк в таблице orders могут иметь одинаковый customerNumber.

536

Sometimes, the child and parent tables may refer to the same table. In this case, the foreign key references back to the primary key within the same table.

Іноді дочірня та батьківська таблиці можуть посилятися на одну й ту ж саму таблицю. У цьому випадку зовнішній ключ посилається на первинний ключ у тій же самій таблиці.

Иногда дочерняя и родительская таблицы могут ссылаться на одну и ту же таблицу. В этом случае внешний ключ ссылается на первичный ключ в той же таблице.



537

The reportTo column is a foreign key that refers to the employeeNumber column which is the primary key of the employees table.

This relationship allows the employees table to store the reporting structure between employees and managers. Each employee reports to zero or one employee and an employee can have zero or many subordinates.

The foreign key on the column reportTo is known as a recursive or self-referencing foreign key.

Стовпець reportTo – зовнішній ключ, який посилається на стовпець workerNumber, який є первинним ключем таблиці employees.

Це відношення дозволяє таблиці співробітників зберігати структуру звітності між працівниками та менеджерами. Кожен працівник звітус перед нулем або одним працівником, і працівник може мати нуль або багато підлеглих.

Зовнішній ключ у стовпці reportTo відомий як рекурсивний або зовнішній ключ, що посилається сам на себе.

Столбец reportTo – внешний ключ, который ссылается на столбец workerNumber, который является первичным ключом таблицы employees.

Это отношение позволяет таблице сотрудников сохранять структуру отчетности между работниками и менеджерами. Каждый работник отчитывается перед нулем или одним работником, и работник может иметь ноль или много подчиненных.

Внешний ключ в столбце reportTo известный как рекурсивный или внешний ключ, ссылающийся сам на себя.

538

MySQL FOREIGN KEY

Here is the basic syntax of defining a foreign key constraint in the CREATE TABLE or ALTER TABLE statement:

Ось основний синтаксис визначення обмеження зовнішнього ключа в операторі CREATE TABLE або ALTER TABLE:

Вот основной синтаксис определения ограничения внешнего ключа в операторе CREATE TABLE или ALTER TABLE:

[CONSTRAINT constraint_name]

FOREIGN KEY [foreign_key_name] (column_name, ...)

REFERENCES parent_table(column_name,...)

[ON DELETE reference_option]

[ON UPDATE reference_option]

539

First, specify the name of foreign key constraint that you want to create after the CONSTRAINT keyword. If you omit the constraint name, MySQL automatically generates a name for the foreign key constraint.

Second, specify a list of comma-separated foreign key columns after the FOREIGN KEY keywords. The foreign key name is also optional and is generated automatically if you skip it.

Third, specify the parent table followed by a list of comma-separated columns to which the foreign key columns reference.

Спочатку вкажіть ім'я обмеження зовнішнього ключа, яке ви хочете створити після ключевого слова CONSTRAINT. Якщо пропустите ім'я обмеження, MySQL автоматично згенерує ім'я для обмеження зовнішнього ключа.

По-друге, вкажіть список стовпців зовнішнього ключа, розділених комами, після ключевих слів FOREIGN KEY. Ім'я зовнішнього ключа також є необов'язковим і генерується автоматично, якщо його пропустити.

По-третє, вкажіть батьківську таблицю, за якою слідує список стовпців, розділених комами, на які посилаються стовпці зовнішнього ключа.

Сначала укажите имя ограничения внешнего ключа, которое вы хотите создать, после ключевого слова CONSTRAINT. Если вы опустите имя ограничения, MySQL автоматически сгенерирует имя для ограничения внешнего ключа.

Во-вторых, укажите список разделенных запятыми столбцов внешнего ключа после ключевых слов FOREIGN KEY. Имя внешнего ключа также является необязательным и создается автоматически, если вы его пропустите.

В-третьих, укажите родительскую таблицу, а затем список разделенных запятыми столбцов, на которые ссылается столбцы внешнего ключа.

540

Finally, specify how foreign key maintains the referential integrity between the child and parent tables by using the ON DELETE and ON UPDATE clauses. The reference_option determines action which MySQL will take when values in the parent key columns are deleted (ON DELETE) or updated (ON UPDATE).

Нарешті, вкажіть, як зовнішній ключ підтримує посилальну цілісність між дочірньою та батьківською таблицями, використовуючи вирази ON DELETE та ON UPDATE. Параметр reference_option визначає дію, яку буде виконувати MySQL, коли значення у стовпцях батьківського ключа будуть видалені (ON DELETE) або оновлені (ON UPDATE).

Наконец, укажите, как внешний ключ поддерживает ссылочную целостность между дочерней и родительской таблицами, используя выражения ON DELETE и ON UPDATE. Параметр reference_option определяет действие, которое MySQL предпримет, когда значения в столбцах родительского ключа будут удалены (ON DELETE) или обновлены (ON UPDATE).

541

MySQL has five reference options: CASCADE, SET NULL, NO ACTION, RESTRICT, and SET DEFAULT.

- CASCADE: if a row from the parent table is deleted or updated, the values of the matching rows in the child table automatically deleted or updated.
- SET NULL: if a row from the parent table is deleted or updated, the values of the foreign key column (or columns) in the child table are set to NULL.
- RESTRICT: if a row from the parent table has a matching row in the child table, MySQL rejects deleting or updating rows in the parent table.
- NO ACTION: is the same as RESTRICT.
- SET DEFAULT: is recognized by the MySQL. However, this action is rejected.

MySQL має п'ять варіантів посилань: CASCADE, SET NULL, NO ACTION, RESTRICT, та SET DEFAULT.

- CASCADE: якщо рядок з батьківської таблиці видаляється або оновлюється, значення відповідних рядків у дочірній таблиці автоматично видаляються або оновлюються.
- SET NULL: якщо рядок з батьківської таблиці видалено або оновлено, значенням стовиця зовнішнього ключа (або стовиців) у дочірній таблиці встановлено значення NULL.
- RESTRICT: якщо рядок з батьківської таблиці має відповідний рядок у дочірній таблиці, MySQL відхиляє видалення або оновлення рядків у батьківській таблиці.
- NO ACTION: те саме, що RESTRICT.
- SET DEFAULT: розпізнається MySQL. Однак ця дія відхиляється.

542

MySQL имеет пять вариантов ссылок: CASCADE, SET NULL, NO ACTION, RESTRICT и SET DEFAULT.

- CASCADE: если строка из родительской таблицы удаляется или обновляется, значения соответствующих строк в дочерней таблице автоматически удаляются или обновляются.
- SET NULL: если строка из родительской таблицы удаляется или обновляется, значения столбца (или столбцов) внешнего ключа в дочерней таблице устанавливаются в NULL.
- RESTRICT: если строка из родительской таблицы имеет соответствующую строку в дочерней таблице, MySQL отклоняет удаление или обновление строк в родительской таблице.
- NO ACTION: то же самое, что и RESTRICT.
- SET DEFAULT: распознается MySQL. Однако это действие отклоняется.

543

In fact, MySQL fully supports three actions: RESTRICT, CASCADE and SET NULL. If you don't specify the ON DELETE and ON UPDATE clause, the default action is RESTRICT.

Насправді MySQL повністю підтримує три дії: RESTRICT, CASCADE і SET NULL. Якщо ви не вказали вирази ON DELETE та ON UPDATE, дією за замовчуванням є RESTRICT.

Фактически, MySQL повністю підтримує три дії: RESTRICT, CASCADE і SET NULL. Якщо ви не вказали вирази ON DELETE та ON UPDATE, дією за замовчуванням є RESTRICT.

544

```
CREATE DATABASE fkdemo;
USE fkdemo;
CREATE TABLE categories (
    categoryId INT AUTO_INCREMENT PRIMARY KEY,
    categoryName VARCHAR(100) NOT NULL
) ENGINE=INNODB;
CREATE TABLE products (
    productId INT AUTO_INCREMENT PRIMARY KEY,
    productName varchar(100) not null,
    categoryId INT,
    CONSTRAINT fk_category
    FOREIGN KEY (categoryId)
    REFERENCES categories(categoryId)
) ENGINE=INNODB;
```

Because we don't specify any ON UPDATE and ON DELETE clauses, the default action is RESTRICT for both update and delete operation.

Оскільки ми не вказуємо жодних виразів ON UPDATE і ON DELETE, дією за замовчуванням є RESTRICT для операцій оновлення та видалення.

Поскольку мы не указываем никаких предложений ON UPDATE и ON DELETE, действием по умолчанию является RESTRICT как для операции обновления, так и для операции удаления.

545

```
INSERT INTO categories(categoryName)
VALUES
('Smartphone'),
('Smartwatch');
```

```
SELECT * FROM categories;
```

	categoryId	categoryName
▶	1	Smartphone
	2	Smartwatch

```
INSERT INTO products(productName, categoryId)
VALUES('iPhone',1);
```

```
INSERT INTO products(productName, categoryId)
VALUES('iPad',3);
```

Error Code: 1452. Cannot add or update a child row: a foreign key constraint fails (`fkdemo`.`products`, CONSTRAINT `fk_category` FOREIGN KEY (`categoryId`) REFERENCES `categories` (`categoryId`) ON DELETE RESTRICT ON UPDATE RESTRICT)

546

```
UPDATE categories
SET categoryId = 100
WHERE categoryId = 1;
```

Error Code: 1451. Cannot delete or update a parent row: a foreign key constraint fails (`fkdemo`.`products`, CONSTRAINT `fk_category` FOREIGN KEY (`categoryId`) REFERENCES `categories` (`categoryId`) ON DELETE RESTRICT ON UPDATE RESTRICT)

Because of the **RESTRICT** option, you cannot delete or update categoryId 1 since it is referenced by the productId 1 in the products table.

Через параметр **RESTRICT** не можна видалити або оновити categoryId 1, оскільки на нього посилається productId 1 у таблиці products.

Из-за опции **RESTRICT** вы не можете удалить или обновить categoryId 1, поскольку на нее ссылается productId 1 в таблице products.

547

```
CREATE TABLE products(
    productId INT AUTO_INCREMENT PRIMARY KEY,
    productName varchar(100) not null,
    categoryId INT NOT NULL,
    CONSTRAINT fk_category
    FOREIGN KEY (categoryId)
    REFERENCES categories(categoryId)
        ON UPDATE CASCADE
        ON DELETE CASCADE
) ENGINE=INNODB;
```

```
INSERT INTO products(productName, categoryId)
VALUES
    ('iPhone', 1),
    ('Galaxy Note', 1),
    ('Apple Watch', 2),
    ('Samsung Galaxy Watch', 2);
```

548

```
SELECT * FROM products;
```

	productId	productName	categoryId
▶	1	iPhone	1
	2	Galaxy Note	1
	3	Apple Watch	2
	4	Samsung Galaxy Watch	2

```
UPDATE categories
SET categoryId = 100
WHERE categoryId = 1;
```

```
SELECT * FROM categories;
```

	categoryId	categoryName
▶	2	Smartwatch
	100	Smartphone

```
SELECT * FROM products;
```

	productId	productName	categoryId
▶	1	iPhone	100
	2	Galaxy Note	100
	3	Apple Watch	2
	4	Samsung Galaxy Watch	2

549

```
DELETE FROM categories
WHERE categoryId = 2;
```

```
SELECT * FROM categories;
```

	categoryId	categoryName
▶	100	Smartphone

```
SELECT * FROM products;
```

	productId	productName	categoryId
▶	1	iPhone	100
	2	Galaxy Note	100

Two rows with value 1 in the categoryId column of the products table were automatically updated to 100 because of the **ON UPDATE CASCADE** action. All products with categoryId 2 from the products table were automatically deleted because of the **ON DELETE CASCADE** action.

Два рядки зі значенням 1 у стовиці categoryId таблиці products були автоматично оновлені до 100 через дію **ON UPDATE CASCADE**. Усі товари з ідентифікатором categoryId 2 із таблиці products були автоматично видалені через дію **ON DELETE CASCADE**.

Две строки со значением 1 в столбце categoryId таблицы products были автоматически обновлены до 100 из-за действия **ON UPDATE CASCADE**. Все товары с categoryId 2 из таблицы products были автоматически удалены из-за действия **ON DELETE CASCADE**.

550

```
CREATE TABLE categories(
    categoryId INT AUTO_INCREMENT PRIMARY KEY,
    categoryName VARCHAR(100) NOT NULL
)ENGINE=INNODB;
```

```
CREATE TABLE products(
    productId INT AUTO_INCREMENT PRIMARY KEY,
    productName varchar(100) not null,
    categoryId INT,
    CONSTRAINT fk_category
    FOREIGN KEY (categoryId)
        REFERENCES categories(categoryId)
        ON UPDATE SET NULL
        ON DELETE SET NULL
)ENGINE=INNODB;
```

551

```
INSERT INTO categories(categoryName)
VALUES
('Smartphone'),
('Smartwatch');
```

```
INSERT INTO products(productName, categoryId)
VALUES
('iPhone', 1),
('Galaxy Note', 1),
('Apple Watch', 2),
('Samsung Galaxy Watch', 2);
```

```
UPDATE categories
SET categoryId = 100
WHERE categoryId = 1;
```

552

SELECT * FROM categories;

	categoryId	categoryName
▶	2	Smartwatch
	100	Smartphone

SELECT * FROM products;

	productId	productName	categoryId
▶	1	iPhone	NULL
	2	Galaxy Note	NULL
	3	Apple Watch	2
	4	Samsung Galaxy Watch	2

The rows with the categoryId 1 in the products table were automatically set to NULL due to the **ON UPDATE SET NULL** action.

Рядки з categoryId 1 у таблиці products були автоматично встановлені на NULL завдяки дії **ON UPDATE SET NULL**.

Строки с categoryId 1 в таблице products были автоматически установлены в NULL из-за действия **ON UPDATE SET NULL**.

553

```
DELETE FROM categories
WHERE categoryId = 2;
```

```
SELECT * FROM products;
```

	productId	productName	categoryId
▶	1	iPhone	NULL
	2	Galaxy Note	NULL
	3	Apple Watch	NULL
	4	Samsung Galaxy Watch	NULL

The values in the categoryId column of the rows with categoryId 2 in the products table were automatically set to NULL due to the **ON DELETE SET NULL** action.

Значення у стовпці categoryId рядків з categoryId 2 у таблиці товарів були автоматично встановлені на NULL завдяки дії **ON DELETE SET NULL**.

Значения в столбце categoryId строк с categoryId 2 в таблице продуктов были автоматически установлены на NULL из-за действия **ON DELETE SET NULL**.

554

To drop a foreign key constraint, you use the ALTER TABLE statement:

Щоб видалити обмеження зовнішнього ключа, використовуйте оператор ALTER TABLE:

Чтобы удалить ограничение внешнего ключа, используйте ALTER TABLE:

```
ALTER TABLE table_name
DROP FOREIGN KEY constraint_name;
```

To obtain the generated constraint name of a table, you use the SHOW CREATE TABLE statement:

Щоб отримати сформоване ім'я обмеження таблиці, використовуйте оператор SHOW CREATE TABLE:

Чтобы получить сгенерированное имя ограничения таблицы, используйте оператор SHOW CREATE TABLE:

```
SHOW CREATE TABLE table_name;
```

555

SHOW CREATE TABLE products;

Table	Create Table
products	<pre>CREATE TABLE `products` (`productId` int(11) NOT NULL AUTO_INCREMENT, `productName` varchar(100) NOT NULL, `categoryId` int(11) DEFAULT NULL, PRIMARY KEY (`productId`), KEY `fk_category` (`categoryId`), CONSTRAINT `fk_category` FOREIGN KEY (`categoryId`) REFERENCES `categories`(`categoryId`) ON DELETE SET NULL ON UPDATE SET NULL) ENGINE=InnoDB AUTO_INCREMENT=5 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci</pre>

ALTER TABLE products
DROP FOREIGN KEY fk_category;

SHOW CREATE TABLE products;

Table	Create Table
products	<pre>CREATE TABLE `products` (`productId` int(11) NOT NULL AUTO_INCREMENT, `productName` varchar(100) NOT NULL, `categoryId` int(11) DEFAULT NULL, PRIMARY KEY (`productId`), KEY `fk_category` (`categoryId`)) ENGINE=InnoDB AUTO_INCREMENT=5 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci</pre>

556

Sometimes, it is very useful to disable foreign key checks e.g., when you import data from a CSV file into a table. If you don't disable foreign key checks, you have to load data into a proper order i.e., you have to load data into parent tables first and then child tables, which can be tedious. However, if you disable the foreign key checks, you can load data into tables in any order.

Іноді дуже корисно вимкнути перевірку зовнішнього ключа, наприклад, коли ви імпортуєте дані з файлу CSV у таблицю. Якщо ви не вимкнете перевірку зовнішнього ключа, вам доведеться завантажувати дані у правильному порядку, тобто спочатку потрібно завантажувати дані у батьківські таблиці, а потім дочірні таблиці, що може бути незручно. Однак, якщо ви вимкнете перевірку зовнішнього ключа, ви можете завантажувати дані в таблиці в будь-якому порядку.

Иногда очень полезно отключить проверку внешнего ключа, например, когда вы импортируете данные из файла CSV в таблицу. Если вы не отключите проверку внешнего ключа, вам придется загружать данные в правильном порядке, т. е. сначала нужно загрузить данные в родительские таблицы, а затем в дочерние, что может быть неудобным. Однако, если вы отключите проверку внешнего ключа, вы сможете загружать данные в таблицы в любом порядке.

557

To disable foreign key checks, you use the following statement:

Щоб вимкнути перевірку зовнішнього ключа, використовуйте такий вираз:

Чтобы отключить проверку внешнего ключа, используйте следующую инструкцию:

SET foreign_key_checks = 0;

And you can enable it by using the following statement:

І ви можете увімкнути її, використовуючи такий вираз:

И вы можете включить ее, используя следующую инструкцию:

SET foreign_key_checks = 1;

558

DATABASE TRANSACTIONS

559

To understand what a transaction in MySQL is, let's take a look at an example of adding a new sales order in our sample database. The steps of adding a sales order are as described as follows:

- First, query the latest sales order number from the orders table and use the next sales order number as the new sales order number.
- Next, insert a new sales order into the orders table.
- Then, get the newly inserted sales order number.
- After that, insert the new sales order items into the order details table with the sales order number.
- Finally, select data from both orders and order details tables to confirm the changes.

Щоб зрозуміти, що таке транзакція в MySQL, давайте розглянемо приклад додавання нового замовлення на продаж у деякій базі даних. Етапи додавання замовлення на продаж описані наступним чином:

- Спочатку вилучити останній номер замовлення на продаж із таблиці замовлень і взяти номер наступного замовлення на продаж (інкремент) як номер нового замовлення на продаж.
- Потім вставити нове замовлення на продаж у таблицю замовлень.
- Потім отримати новий номер замовлення на продаж.
- Після цього вставити нові елементи замовлення на продаж у таблицю деталей замовлення з номером замовлення на продаж.
- Нарешті, вибрати дані із таблиць замовлень та деталей замовлень, щоб підтвердити зміни.

560

Чтобы понять, что такое транзакция в MySQL, давайте рассмотрим пример добавления нового заказа на продажу в некоторой базе данных. Этапы добавления заказ на продажу описаны следующим образом:

- Сначала получить последний номер заказа на продажу из таблицы заказов и взять номер следующего заказа на продажу (инкремент) как номер нового заказа на продажу.
- Затем вставить новый заказ на продажу в таблицу заказов.
- Затем получить новый номер заказа на продажу.
- После этого вставить новые элементы заказ на продажу в таблицу деталей заказа с номером заказа на продажу.
- Наконец, выбрать данные из таблиц заказов и деталей заказов, чтобы подтвердить изменения.

561

Now, imagine what would happen to the sales order data if one or more steps above fail due to some reasons? For example, if the step of adding order's items into order details table fails, you will have an empty sales order.

That is why the transaction processing comes to the rescue. MySQL ***transaction*** allows you to execute a set of MySQL operations to ensure that the database never contains the result of partial operations. In a set of operations, if one of them fails, the rollback occurs to restore the database to its original state. If no error occurs, the entire set of statements is committed to the database.

А тепер уявіть, що трапиться з даними замовлення на продаж, якщо один чи кілька кроків вище не вдається виконати з якихось причин? Наприклад, якщо крок додавання елементів замовлення в таблицю деталей замовлення не вдається, у нас буде пусте замовлення на продаж.

Тому на допомогу приходить обробка транзакцій. ***Транзакція*** MySQL дозволяє виконувати набір операцій MySQL, щоб база даних ніколи не містила результату часткових операцій. У наборі операцій, якщо одна з них не вдається, відбувається відкат для відновлення бази даних до початкового стану. Якщо помилки не виникає, весь набір операторів фіксується в базі даних.

562

А тепер представьте, что случится с данными заказа на продажу, если один или несколько шагов выше не удастся выполнить по каким-то причинам? Например, если шаг добавления элементов заказа в таблицу деталей заказа не удастся, у нас будет пустой заказ на продажу.

Поэтому на помощь приходит обработка транзакций.

Транзакция MySQL позволяет выполнять набор операций MySQL, чтобы база данных никогда не содержала результата частичных операций. В наборе операций, если одна из них не удаётся, происходит откат для восстановления базы данных в исходное состояние. Если ошибки не возникает, весь набор операторов фиксируется в базе данных.

563

MySQL provides us with the following important statement to control transactions:

- To start a transaction, you use the START TRANSACTION statement. The BEGIN or BEGIN WORK are the aliases of the START TRANSACTION.
- To commit the current transaction and make its changes permanent, you use the COMMIT statement.
- To roll back the current transaction and cancel its changes, you use the ROLLBACK statement.
- To disable or enable the auto-commit mode for the current transaction, you use the SET autocommit statement.

MySQL надає нам наступні важливі вирази для управління транзакціями:

- Щоб розпочати транзакцію, використовуйте оператор START TRANSACTION. BEGIN або BEGIN WORK – це псевдоніми START TRANSACTION.
- Щоб зафіксувати поточну транзакцію та зробити її зміни постійними, використовуйте оператор COMMIT.
- Щоб повернути поточну транзакцію та скасувати її зміни, використовуйте оператор ROLLBACK.
- Щоб вимкнути або ввімкнути режим автоматичного фіксування для поточної транзакції, використовуйте оператор SET autocommit.

564

MySQL предоставляет нам следующие важные выражения для управления транзакциями:

- Чтобы начать транзакцию, используйте оператор START TRANSACTION. BEGIN или BEGIN WORK - это псевдонимы START TRANSACTION.
- Чтобы зафиксировать текущую транзакцию и сделать ее изменения постоянными, используйте оператор COMMIT.
- Чтобы вернуть текущую транзакцию и отменить ее изменения, используйте оператор ROLLBACK.
- Чтобы включить или выключить режим автоматического фиксирования для текущей транзакции, используйте оператор SET autocommit.

565

By default, MySQL automatically commits the changes permanently to the database. To force MySQL not to commit changes automatically, you use the following statement:

За замовчуванням MySQL автоматично постійно фіксує зміни в базі даних. Щоб змусити MySQL не фіксувати зміни автоматично, використовуйте наступне твердження:

По умолчанию MySQL автоматически фиксирует изменения в базе данных на постоянной основе. Чтобы заставить MySQL не фиксировать изменения автоматически, используйте следующий оператор:

```
SET autocommit = 0; / SET autocommit = OFF;
```

You use the following statement to enable the autocommit mode explicitly:

Використовуйте наступну заяву, щоб явно ввімкнути режим автоматичної фіксації:

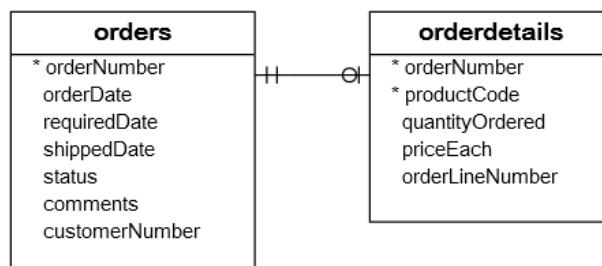
Для явного включення режима автоматической фиксации используется следующий оператор:

```
SET autocommit = 1; / SET autocommit = ON;
```

566

The following illustrates the step of creating a new sales order:

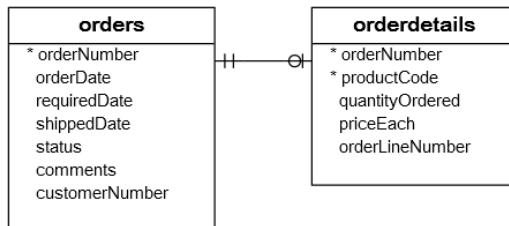
- First, start a transaction by using the START TRANSACTION statement.
- Next, select the latest sales order number from the orders table and use the next sales order number as the new sales order number.
- Then, insert a new sales order into the orders table.
- After that, insert sales order items into the orderdetails table.
- Finally, commit the transaction using the COMMIT statement.



567

Далі ілюструється крок створення нового замовлення на продаж:

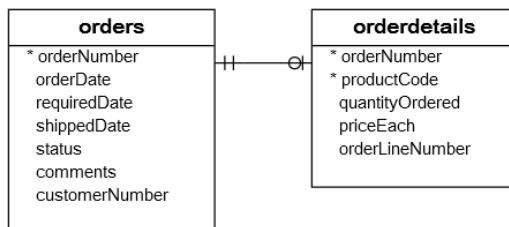
- Спочатку розпочніть транзакцію за допомогою оператора START TRANSACTION.
- Далі виберіть останній номер замовлення на продаж із таблиці orders і використовуйте номер наступного замовлення на продаж як номер нового замовлення на продаж.
- Потім вставте нове замовлення на продаж у таблицю orders.
- Після цього вставте елементи замовлення на продаж у таблицю orderdetails.
- Нарешті, зафіксуйте транзакцію за допомогою оператора COMMIT.



568

Ниже показан этап создания нового заказа на продажу:

- Сначала запустите транзакцию с помощью оператора START TRANSACTION.
- Затем выберите последний номер заказа на продажу из таблицы orders и используйте следующий номер заказа на продажу в качестве нового номера заказа на продажу.
- Затем вставьте новый заказ на продажу в таблицу orders.
- После этого вставьте позиции заказа на продажу в таблицу orderdetails.
- Наконец, зафиксируйте транзакцию с помощью оператора COMMIT.



569

```
-- 1. start a new transaction
START TRANSACTION;

-- 2. Get the latest order number
SELECT
    @orderNumber:=MAX(orderNUmber)+1
FROM
    orders;

-- 3. insert a new order for customer 145
INSERT INTO orders(orderNumber,
    orderDate,
    requiredDate,
    shippedDate,
    status,
    customerNumber)
VALUES(@orderNumber,
    '2005-05-31',
    '2005-06-10',
    '2005-06-11',
    'In Process',
    145);
```

570

```
-- 4. Insert order line items
INSERT INTO orderdetails(orderNumber,
    productCode,
    quantityOrdered,
    priceEach,
    orderLineNumber)
VALUES(@orderNumber,'S18_1749', 30, '136', 1),
    (@orderNumber,'S18_2248', 50, '55.09', 2);

-- 5. commit changes
COMMIT;
```

	@orderNumber:=IFNULL(MAX(orderNUmber),0)+1
▶	10426

	orderNumber	orderDate	requiredDate	shippedDate	status	comments	customerNumber	orderLineNumber	productCode	quantityOrdered	priceEach
▶	10426	2005-05-31	2005-06-10	2005-06-11	In Process	NULL	145	1	S18_1749	30	136.00
	10426	2005-05-31	2005-06-10	2005-06-11	In Process	NULL	145	2	S18_2248	50	55.09

571

Session 1	Session 2
<pre>mysql> START TRANSACTION; Query OK, 0 rows affected (0.00 sec) mysql> DELETE FROM orders; Query OK, 327 rows affected (0.03 sec) mysql> ROLLBACK; Query OK, 0 rows affected (0.04 sec) mysql> SELECT COUNT(*) FROM orders; +-----+ COUNT(*) +-----+ 327 +-----+ 1 row in set (0.00 sec)</pre>	<pre>mysql> SELECT COUNT(*) FROM orders; +-----+ COUNT(*) +-----+ 327 +-----+ 1 row in set (0.00 sec)</pre>

572

A **transaction** is a single logical unit of work which accesses and possibly modifies the contents of a database. Transactions access data using read and write operations.

In order to maintain consistency in a database, before and after the transaction, certain properties are followed. These are called ACID properties.

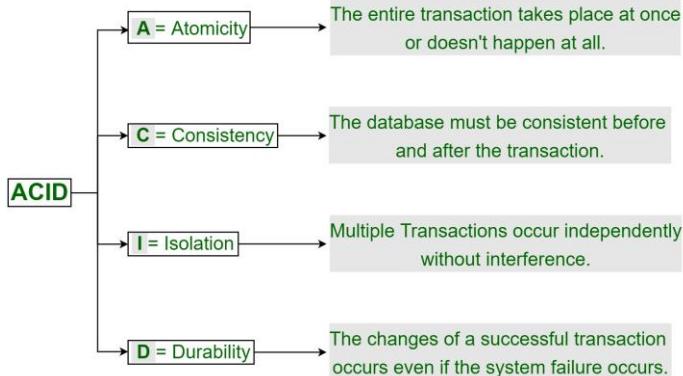
Транзакція – це одна логічна одиниця роботи, яка отримує доступ і, можливо, змінює вміст бази даних. Транзакції отримують доступ до даних за допомогою операцій читання та запису.

Для того, щоб зберегти узгодженість у базі даних до і після змін, транзакції дотримуються певних властивостей. Вони називаються властивостями ACID.

Транзакция – это одна логическая единица работы, которая получает доступ и, возможно, изменяет содержимое базы данных. Транзакции получают доступ к данным с помощью операций чтения и записи.

Для того, чтобы сохранить согласованность в базе данных до и после изменений, транзакции придерживаются определенных свойств. Они называются свойствами ACID.

ACID Properties in DBMS



ЕG

Atomicity

By this, we mean that either the entire transaction takes place at once or doesn't happen at all. There is no midway i.e. transactions do not occur partially. Each transaction is considered as one unit and either runs to completion or is not executed at all. It involves the following two operations.

- Rollback: If a transaction aborts, changes made to database are not visible.
- Commit: If a transaction commits, changes made are visible.

Atomicity is also known as the 'All or nothing rule'.

Атомарність

Під цим ми маємо на увазі, що або вся транзакція відбувається одночасно, або взагалі не відбувається. Середини не існує, тобто транзакції не відбуваються частково. Кожна транзакція розглядається як одна одиниця і або виконується до завершення, або взагалі не виконується. Вона передбачає наступні дві операції.

- Rollback: Якщо транзакція перервана, зміни, внесені до бази даних, не видно.
- Commit: Якщо транзакція здійснюється, внесені зміни видно.

Атомарність також відома як «правило все або нічого».

Атомарность

Под этим мы подразумеваем, что либо вся транзакция выполняется сразу, либо не происходит вообще. Промежуточного пути нет, т.е. транзакции не происходят частично. Каждая транзакция рассматривается как одна единица и либо выполняется до завершения, либо не выполняется вообще. Оно включает в себя следующие две операции.

- Rollback: если транзакция прерывается, изменения, внесенные в базу данных, не видны.
- Commit: если транзакция фиксируется, внесенные изменения видны.

Атомарность также известна как правило «все или ничего».

575

Consistency

This means that integrity constraints must be maintained so that the database is consistent before and after the transaction. It refers to the correctness of a database.

Узгодженість

Це означає, що обмеження цілісності повинні підтримуватися, щоб база даних була узгодженою до і після транзакції. Це стосується цілісності бази даних.

Согласованність

Это означает, что ограничения целостности должны поддерживаться, чтобы база данных была согласованной до и после транзакции. Это касается правильности базы данных.

576

Isolation

This property ensures that multiple transactions can occur concurrently without leading to the inconsistency of database state. Transactions occur independently without interference. Changes occurring in a particular transaction will not be visible to any other transaction until that particular change in that transaction is committed. This property ensures that the execution of transactions concurrently will result in a state that is equivalent to a state achieved these were executed serially in some order.

Ізоляція

Ця властивість гарантує, що кілька транзакцій можуть відбуватися одночасно, не приводячи до неузгодженості стану бази даних. Транзакції відбуваються самостійно без втручання. Зміни, що відбуваються в певній транзакції, не будуть видимими для будь-якої іншої транзакції, поки ця конкретна зміна в цій транзакції не буде зафіксована. Ця властивість гарантує, що виконання транзакцій одночасно призведе до стану, еквівалентного стану, досягнутому, що вони виконувались постійно в якомусь порядку.

Изоляция

Это свойство гарантирует, что несколько транзакций могут происходить одновременно, не приводя к несогласованности состояния базы данных. Транзакции происходят самостоятельно без вмешательства. Изменения, происходящие в определенной транзакции, не видны для любой другой транзакции, пока это конкретная изменение в этой транзакции не будет зафиксировано. Это свойство гарантирует, что выполнение транзакций одновременно приведет к состоянию, эквивалентному такому состоянию, как если бы они выполнялись последовательно в каком-либо порядке.

Durability

This property ensures that once the transaction has completed execution, the updates and modifications to the database are stored in and written to disk and they persist even if a system failure occurs. These updates now become permanent and are stored in non-volatile memory. The effects of the transaction, thus, are never lost.

Довговічність

Ця властивість гарантує, що після завершення транзакції оновлення та модифікації бази даних зберігаються та записуються на диск, і вони зберігаються, навіть якщо трапляється збій системи. Тепер ці оновлення стають постійними і зберігаються в енергонезалежній пам'яті. Отже, наслідки транзакції ніколи не втрачаються.

Долговечность

Это свойство гарантирует, что после завершения транзакции обновления и модификации базы данных будут сохраняться и записываться на диск, и они сохраняются даже в случае сбоя системы. Эти обновления теперь становятся постоянными и хранятся в энергонезависимой памяти. Таким образом, последствия транзакции никогда не теряются.

The ACID properties, in totality, provide a mechanism to ensure correctness and consistency of a database in a way such that each transaction is a group of operations that acts a single unit, produces consistent results, acts in isolation from other operations and updates that it makes are durably stored.

Властивості ACID, у цілому, забезпечують механізм забезпечення коректності та узгодженості бази даних таким чином, що кожна транзакція є групою операцій, яка діє як одна одиниця, дає послідовні результати, діє ізольовано від інших операцій, а здійснені оновлення надійно зберігаються.

Свойства ACID в совокупности обеспечивают механизм для обеспечения правильности и согласованности базы данных таким образом, что каждая транзакция представляет собой группу операций, которые действуют как единое целое, производят согласованные результаты, действуют изолированно от других операций, а сделанные обновления надежно хранятся.

Isolation levels define the degree to which a transaction must be isolated from the data modifications made by any other transaction in the database system. A transaction isolation level is defined by the following phenomena:

- **Dirty Read** – is the situation when a transaction reads a data that has not yet been committed. For example, Let's say transaction T1 updates a row and leaves it uncommitted, meanwhile, Transaction T2 reads the updated row. If transaction T1 rolls back the change, transaction T2 will have read data that is considered never to have existed.
- **Non Repeatable Read** – occurs when a transaction reads same row twice, and get a different value each time. For example, suppose transaction T1 reads data. Due to concurrency, another transaction T2 updates the same data and commit, Now if transaction T1 rereads the same data, it will retrieve a different value.
- **Phantom Read** – occurs when two same queries are executed, but the rows retrieved by the two, are different. For example, suppose transaction T1 retrieves a set of rows that satisfy some search criteria. Now, Transaction T2 generates some new rows that match the search criteria for transaction T1. If transaction T1 re-executes the statement that reads the rows, it gets a different set of rows this time.

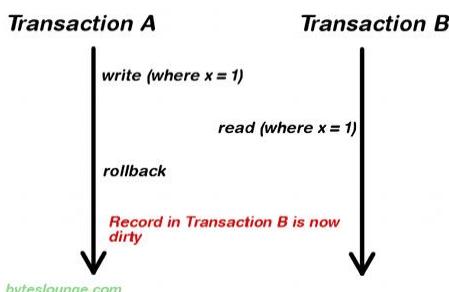
Рівні ізоляції визначають ступінь, до якого транзакція повинна бути ізольована від модифікацій даних, зроблених будь-якою іншою транзакцією в системі управління базами даних. Рівень ізоляції транзакції визначається наступними аномаліями:

- **Брудне читання** – це ситуація, коли транзакція зчитує дані, які ще не були зафіксовані. Наприклад, припустимо, що транзакція T1 оновлює рядок і залишає його незафіксованим, тим часом транзакція T2 читає оновлений рядок. Якщо транзакція T1 відкатує зміни, транзакція T2 матиме прочитані дані, які, як вважається, ніколи не існували.
- **Неповторюване читання** – відбувається, коли транзакція зчитує один і той же рядок двічі та отримує різне значення кожного разу. Наприклад, припустимо, транзакція T1 зчитує дані. Через паралельність інша транзакція T2 оновлює ті самі дані та фіксує. Тепер, якщо транзакція T1 перечитає ті самі дані, вона отримає інше значення.
- **Фантомне читання** – відбувається, коли виконуються два однакові запити, але рядки, отримані цими запитами, відрізняються. Наприклад, припустимо, що транзакція T1 отримує набір рядків, які задовільняють певним критеріям пошуку. Тепер транзакція T2 генерує кілька нових рядків, які відповідають критеріям пошуку транзакції T1. Якщо транзакція T1 повторно виконує оператор, який читає рядки, цього разу вона отримує інший набір рядків.

Уровни изоляции определяют степень, до которой транзакция должна быть изолирована от модификаций данных, сделанных любой другой транзакцией в системе управления базами данных. Уровень изоляции транзакции определяется следующими аномалиями:

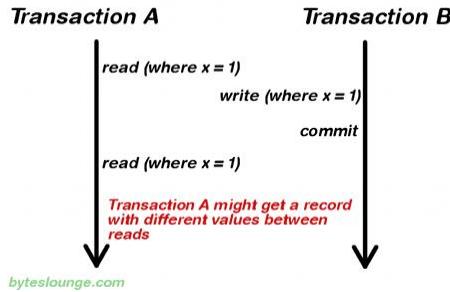
- **Грязное чтение** – это ситуация, когда транзакция считывает данные, которые еще не были зафиксированы. Например, предположим, что транзакция T1 обновляет строку и оставляет ее незафиксированной, между тем транзакция T2 читает обновленную строку. Если транзакция T1 откатывает изменения, транзакцией T2 будут прочитаны данные, которые, как считается, никогда не существовали.
- **Неповторяющееся чтение** – происходит, когда транзакция считывает одну и ту же строку дважды и получает различное значение каждый раз. Например, предположим, транзакция T1 считывает данные. В силу параллельности, другая транзакция T2 обновляет те же данные и фиксирует. Теперь, если транзакция T1 прочитает те же данные, она получит другое значение.
- **Фантомное чтение** – происходит, когда выполняются два одинаковых запроса, но строки, полученные этими запросами, отличаются. Например, предположим, что транзакция T1 получает набор строк, которые удовлетворяют определенным критериям поиска. Теперь транзакция T2 генерирует несколько новых строк, соответствующих критериям поиска транзакции T1. Если транзакция T1 повторно выполняет оператор, читает строки, то на этот раз она получает другой набор строк.

Dirty read



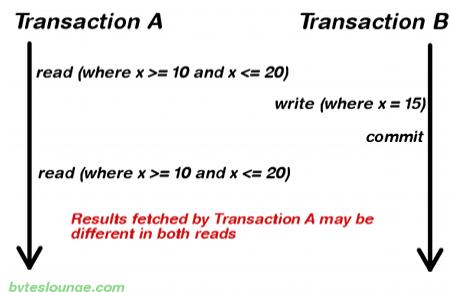
583

Non-repeatable read



584

Phantom read



585

The SQL standard defines four **isolation levels**:

- **Read Uncommitted** – is the lowest isolation level. In this level, one transaction may read not yet committed changes made by other transaction, thereby allowing dirty reads. In this level, transactions are not isolated from each other.
- **Read Committed** – guarantees that any data read is committed at the moment it is read. Thus it does not allow dirty read. The transaction holds a read or write lock on the current row, and thus prevent other transactions from reading, updating or deleting it.
- **Repeatable Read** – the transaction holds read locks on all rows it references and writes locks on all rows it inserts, updates, or deletes. Since other transaction cannot read, update or delete these rows, consequently it avoids non-repeatable read.
- **Serializable** – is defined to be an execution of operations in which concurrently executing transactions appears to be serially executing.

Стандарт SQL визначає чотири **рівні ізоляції**:

- **Read Uncommitted** – це найнижчий рівень ізоляції. На цьому рівні одна транзакція може читати ще не зафіксовані зміни, внесені іншою транзакцією, дозволяючи тим самим брудні читання. На цьому рівні транзакції не ізольовані один від одного.
- **Read Committed** – гарантує, що будь-які прочитані дані зафіксовані в момент їх прочитання. Таким чином, він не дозволяє брудне читання. Транзакція утримує блокування читання або запису в поточному рядку, і таким чином запобігає її читання, оновлення або видалення.
- **Repeatable Read** – транзакція зберігає блокування читання у всіх рядках, на які посилається, та записує блокування у всіх рядках, які вона вставляє, оновлює чи видаляє. Оскільки інша транзакція не може прочитати, оновити або видалити ці рядки, отже, це дозволяє уникнути неповторного читання.
- **Serializable** – найвищий рівень ізоляції, при якому виконання операцій, в яких одночасно виконуються транзакції, насправді є послідовним.

586

Стандарт SQL определяет четыре уровня изоляции:

- **Read Uncommitted** – это самый низкий уровень изоляции. На этом уровне одна транзакция может читать еще не зафиксированные изменения, внесенные другой транзакцией, позволяя тем самым грязное чтение. На этом уровне транзакции не изолированы друг от друга.
- **Read Committed** – гарантирует, что любые прочитанные данные зафиксированы в момент их прочтения. Таким образом, он не позволяет грязное чтение. Транзакция удерживает блокировку чтения или записи в текущей строке, и таким образом предотвращает ее чтение, обновление или удаление.
- **Repeatable Read** – транзакция сохраняет блокировки чтения во всех строках, на которые ссылается, и записывает блокировки во всех строках, которые она вставляет, обновляет или удаляет. Поскольку другая транзакция не может прочитать, обновить или удалить эти строки, значит, это позволяет избежать неповторяющегося чтения.
- **Serializable** – самый высокий уровень изоляции, при котором выполнение операций, в которых одновременно выполняются транзакции, на самом деле является последовательным.

587

Isolation Level	Dirty reads	Non-repeatable reads	Phantoms
Read Uncommitted	May occur	May occur	May occur
Read Committed	Don't occur	May occur	May occur
Repeatable Read	Don't occur	Don't occur	May occur
Serializable	Don't occur	Don't occur	Don't occur

The default isolation level is REPEATABLE READ
 REPEATABLE READ використовується за замовчуванням
 REPEATABLE READ используется по умолчанию

588

SET [GLOBAL | SESSION] TRANSACTION ISOLATION LEVEL { READ UNCOMMITED | READ COMMITED | REPEATABLE READ | SERIALIZABLE }

By default, the isolation level is set for a subsequent (non-initial) transaction
 За замовчуванням рівень ізоляції встановлюється для наступної (непочаткової) транзакції

По умолчанию установлен уровень изоляции для последующей (не начальной) транзакции

When using the **GLOBAL** keyword, this command sets the default isolation level globally for all new connections created from this moment

При використанні ключевого слова **GLOBAL** ця команда встановлює глобальний рівень ізоляції за замовчуванням для всіх нових підключень, створених з цього моменту

При использовании ключевого слова **GLOBAL** эта команда устанавливает глобальный уровень изоляции по умолчанию для всех новых подключений, созданных с этого момента

Using the **SESSION** keyword sets the default isolation level for all future transactions performed on the current connection

Використання ключевого слова **SESSION** встановлює рівень ізоляції за замовчуванням для всіх майбутніх транзакцій, що виконуються у поточному з'єднанні

Использование ключевого слова **SESSION** устанавливает уровень изоляции по умолчанию для всех будущих транзакций, выполняемых в текущем соединении.

589

For InnoDB tables, there are SAVEPOINT and ROLLBACK TO
SAVEPOINT statements that allow you to work with named
transaction start points

Для таблиць InnoDB існують оператори SAVEPOINT і ROLLBACK
TO SAVEPOINT, які дозволяють працювати з названими
початковими точками транзакцій

Для таблиц InnoDB есть операторы SAVEPOINT и ROLLBACK TO
SAVEPOINT, которые позволяют работать с именованными
точками начала транзакции.

SAVEPOINT identifier

...

ROLLBACK [WORK] TO [SAVEPOINT] identifier

...

RELEASE SAVEPOINT identifier

590

```
mysql> SELECT * FROM names;
+----+-----+
| id | first_name |
+----+-----+
| 4 | Harry    |
| 33 | Tom      |
| 8 | Shelly   |
| 10 | Ricky   |
+----+-----+
4 rows in set (0.00 sec)
```

```
mysql> START TRANSACTION;
Query OK, 0 rows affected (0.00 sec)
```

```
mysql> SAVEPOINT initial_save;
Query OK, 0 rows affected (0.00 sec)
```

```
mysql> UPDATE names
-> SET id = 414
-> WHERE first_name = 'Harry';
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0
```

591

```
mysql> SELECT *  
-> FROM names  
-> WHERE first_name = 'Harry';  
+----+-----+  
| id | first_name |  
+----+-----+  
| 414 | Harry |  
+----+-----+  
1 row in set (0.00 sec)
```

```
mysql> ROLLBACK TO initial_save;  
Query OK, 0 rows affected (0.04 sec)
```

```
mysql> SELECT *  
-> FROM names  
-> WHERE first_name = 'Harry';  
+----+-----+  
| id | first_name |  
+----+-----+  
| 4 | Harry |  
+----+-----+  
1 row in set (0.00 sec)
```

...
COMMIT / ROLLBACK;

592

DATABASE USERS

MySQL CREATE USER

The CREATE USER statement creates a new user in the database server.

Оператор CREATE USER створює нового користувача на сервері баз даних.

Оператор CREATE USER создает нового пользователя на сервере базы данных.

```
CREATE USER [IF NOT EXISTS] account_name
IDENTIFIED BY 'password';
```

First, specify the account name after the CREATE USER keywords. The account name has two parts: username and hostname, separated by the @ sign:

Спочатку вкажіть називу облікового запису після ключових слів CREATE USER. Ім'я облікового запису складається з двох частин: імені користувача та імені хосту, розділених знаком @:

Спочатку вкажіть називу облікового запису після ключових слів CREATE USER. Ім'я облікового запису складається з двох частин: імені користувача та імені хосту, розділених знаком @:

username@hostname

The username is the name of the user. And hostname is the name of the host from which the user connects to the MySQL Server.

The hostname part of the account name is optional. If you omit it, the user can connect from any host.

username – це ім'я користувача, hostname – це ім'я хосту, з якого користувач під'єднується до сервера MySQL.

Частина імені хосту імені облікового запису є необов'язковою.

Якщо ви це опустите, користувач може підключитися з будь-якого хосту.

username – это имя пользователя, hostname – это имя хоста, с которого пользователь подключается к серверу MySQL.

Часть имени хоста имени учетной записи является необязательной. Если вы его опустите, пользователь сможет подключиться с любого хоста.

username@%

595

If the username and hostname contains special characters such as space or -, you need to quote the username and hostname separately as follows:

Якщо ім'я користувача та ім'я хосту містять спеціальні символи, такі як пробіл або -, вам потрібно вказати ім'я користувача та ім'я хосту окремо у лапках таким чином:

Если имя пользователя и имя хоста содержат специальные символы, такие как пробел или -, вам необходимо отдельно заключить имя пользователя и имя хоста в кавычки, как показано ниже:

'username'@'hostname'

Besides the single quote ('), you can use backticks (`) or double quotation mark (").

Окрім одинарної лапки ('), ви можете використовувати зворотні позначки (`) або подвійні лапки (").

Помимо одинарної кавычки ('), вы можете использовать обратные кавычки (`) или двойные кавычки (").

596

Second, specify the password for the user after the IDENTIFIED BY keywords.

The IF NOT EXISTS option conditionally create a new user only if it does not exist.

Note that the CREATE USER statement creates a new user without any privileges. To grant privileges to the user, you use the GRANT statement.

По-друге, вкажіть пароль користувача після ключових слів IDENTIFIED BY.

Опція IF NOT EXISTS умовно створює нового користувача, лише якщо він не існує.

Зверніть увагу, что оператор CREATE USER створює нового користувача без будь-яких привілеїв. Щоб надати користувачеві привілеї, використовуйте оператор GRANT.

Во-вторых, укажите пароль для пользователя после ключевых слов IDENTIFIED BY.

Параметр IF NOT EXISTS условно создает нового пользователя, только если он не существует.

Обратите внимание, что оператор CREATE USER создает нового пользователя без каких-либо привилегий. Чтобы предоставить пользователю привилегии, используйте оператор GRANT.

597

```

mysql> select user from mysql.user;
+-----+
| user      |
+-----+
| mysql.infoschema |
| mysql.saession |
| mysql.sys      |
| root          |
+-----+
mysql> create user bob@localhost identified by 'Secure1pass!';
mysql> select user from mysql.user;

mysql> select user from mysql.user;
+-----+
| user      |
+-----+
| bob       |
| mysql.infoschema |
| mysql.session   |
| mysql.sys      |
| root          |
+-----+
5 rows in set (0.00 sec)

```

598

MySQL GRANT

GRANT privilege [,privilege]..
ON privilege_level
TO account_name;

The following example grants UPDATE, DELETE, and INSERT privileges on the table employees to bob@localhost:

Наступний приклад надає привілеї UPDATE, DELETE та INSERT для таблиці employees для bob@localhost:

В следующем примере пользователю bob@localhost предоставляются права UPDATE, DELETE и INSERT для таблицы employees:

```

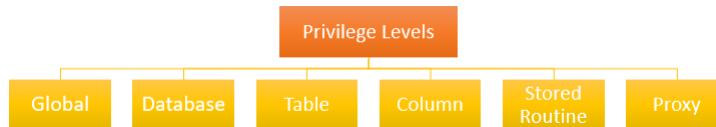
GRANT INSERT, UPDATE, DELETE
ON employees
TO bob@localhost;

```

MySQL supports the following main privilege levels:

MySQL підтримує такі основні рівні привілеїв:

MySQL поддерживает следующие основные уровни привилегий:



Global privileges apply to all databases in a MySQL Server. To assign global privileges, you use the `*.*` syntax, for example:

Глобальні привілеї застосовуються до всіх баз даних на сервері MySQL. Щоб призначити глобальні привілеї, використовуйте синтаксис `*.*`, наприклад:

Глобальные привилегии применяются ко всем базам данных на сервере MySQL. Чтобы назначить глобальные привилегии, используйте синтаксис `*.*`, например:

```
GRANT SELECT
ON *.*
TO bob@localhost;
```

Database privileges apply to all objects in a database. To assign database-level privileges, use the `ON database_name.*` syntax, for example:

Привілеї бази даних застосовуються до всіх об'єктів бази даних. Щоб призначити привілеї на рівні бази даних, використовуйте синтаксис `ON database_name.*`, наприклад:

Привилегии базы данных применяются ко всем объектам в базе данных. Чтобы назначить права на уровне базы данных, используйте синтаксис `ON database_name.*`, например:

```
GRANT INSERT
ON classicmodels.*
TO bob@localhost;
```

Table privileges apply to all columns in a table. To assign table-level privileges, use the `ON database_name.table_name` syntax, for example:

Привілеї таблиці застосовуються до всіх стовпців таблиці. Щоб призначити привілеї на рівні таблиці, використовуйте синтаксис `ON database_name.table_name`, наприклад:

Привилегии таблицы применяются ко всем столбцам в таблице. Чтобы назначить привилегии на уровне таблицы, используйте синтаксис `ON database_name.table_name`, например:

```
GRANT DELETE
ON classicmodels.employees
TO bob@localhost;
```

601

Column privileges apply to single columns in a table. You must specify the column or columns for each privilege, for example:

Привілеї на стовпці застосовуються до окремих стовпців у таблиці. Ви повинні вказати стовпець або стовпці для кожної привілеї, наприклад:

Привилегии столбца применяются к отдельным столбцам в таблице. Вы должны указать столбец или столбцы для каждой привилегии, например:

```
GRANT
  SELECT (employeeNummer,lastName, firstName, email),
  UPDATE(lastName)
ON employees
TO bob@localhost;
```

In this example, bob@localhost can select data from four columns employeeNumber, lastName, firstName, and email and update only the lastName column in the employees table.

У цьому прикладі bob @ localhost може вибирати дані з чотирьох стовпців workerNumber, lastName, firstName та email та оновлювати лише стовпець lastName у таблиці employees.

В этом примере bob @ localhost может выбирать данные из четырех столбцов employeeNumber, lastName, firstName и email и обновлять только столбец lastName в таблице employees.

602

Stored routine privileges apply to stored procedures and stored functions, for example:

Привілеї збережених процедур поширюються на збережені процедури та збережені функції, наприклад:

Привилегии хранимых процедур применяются к хранимым процедурам и хранимым функциям, например:

```
GRANT EXECUTE
ON PROCEDURE CheckCredit
TO bob@localhost;
```

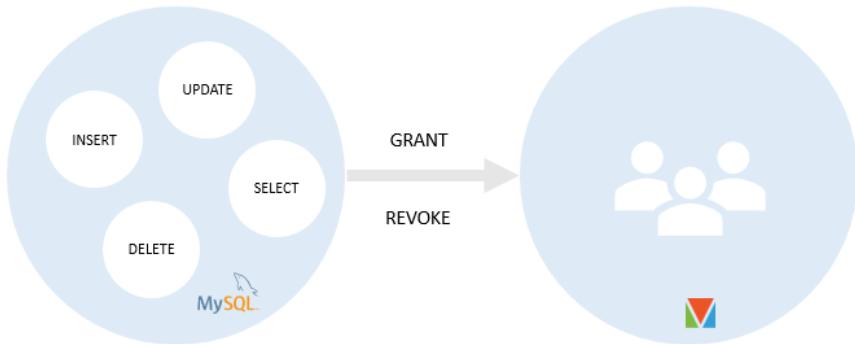
Proxy user privileges allow one user to be a proxy for another. The proxy user gets all privileges of the proxied user. For example:

Привілеї користувачів-проксі дозволяють одному користувачеві бути проксі іншого. Користувач-проксі отримує всі привілеї іншого користувача. Наприклад:

Привилегии пользователей-прокси позволяют одному пользователю быть прокси другого. Пользователь-прокси получает все привилегии другого пользователя. Например:

```
GRANT PROXY
ON root
TO alice@localhost;
```

603



604

```
CREATE USER super@localhost
IDENTIFIED BY 'Secure1Pass!';
```

```
SHOW GRANTS FOR super@localhost;
```

	Grants for super@localhost
▶	GRANT USAGE ON *.* TO `super` @`localhost`

```
GRANT ALL
ON classicmodels.*
TO super@localhost;
```

```
SHOW GRANTS FOR super@localhost;
```

	Grants for super@localhost
▶	GRANT USAGE ON *.* TO `super` @`localhost`
	GRANT ALL PRIVILEGES ON `classicmodels`.* TO `super` @`localhost`

Permissible privileges for GRANT statement
<https://www.mysqltutorial.org/mysql-grant.aspx>

MySQL REVOKE

The following illustrates the basic syntax of the REVOKE statement that revokes one or more privileges from user accounts:

Далі ілюструється основний синтаксис оператора REVOKE, який відміняє одну або кілька привілеїв з облікових записів користувачів:

Нижче показан базовий синтаксис оператора REVOKE, который отменяет одну или несколько привилегий у учетных записей пользователей:

REVOKE

```
privilegee [,privilege]..  
ON [object_type] privilege_level  
FROM user1 [, user2] ..;
```

To revoke all privileges from a user, use the following form of the REVOKE ALL statement:

Щоб скасувати всі привілеї у користувача, використовуйте таку форму оператора REVOKE ALL:

Чтобы отозвать все привилегии у пользователя, используйте следующую форму оператора REVOKE ALL:

REVOKE

```
ALL [PRIVILEGES],  
GRANT OPTION  
FROM user1 [, user2];
```

To revoke a proxy user, use the REVOKE PROXY command:

Щоб відкликати проксі-користувача, використовуйте команду REVOKE PROXY:

Чтобы отозвать прокси-пользователя, используйте команду REVOKE PROXY:

REVOKE PROXY

```
ON proxied_user  
FROM proxy_user1[,proxy_user1]...;
```

A proxy user is a valid user in MySQL who can impersonate another user, therefore, the proxy user has all privileges of the user that it impersonates.

Користувач-проксі-сервер – це дійсний користувач у MySQL, який може видавати себе за іншого користувача, отже, користувач-проксі має всі привілеї користувача, якого він втілює.

Пользователь-прокси – это действительный пользователь в MySQL, который может выдавать себя за другого пользователя, следовательно, у пользователя-посредника есть все привилегии пользователя, которого он олицетворяет.

607

```
CREATE USER rfc@localhost
IDENTIFIED BY 'Secret1Pass!';
```

```
GRANT SELECT, UPDATE, INSERT
ON classicmodels.*
TO rfc@localhost;
```

	Grants for rfc@localhost
▶	GRANT USAGE ON *.* TO 'rfc' @'localhost'
	GRANT SELECT, INSERT, UPDATE ON `classicmodels`.* TO 'rfc' @'localhost'

```
SHOW GRANTS FOR rfc@localhost;
```

```
REVOKE INSERT, UPDATE
ON classicmodels.*
FROM rfc@localhost;
```

```
SHOW GRANTS FOR rfc@localhost;
```

	Grants for rfc@localhost
▶	GRANT USAGE ON *.* TO 'rfc' @'localhost'
	GRANT SELECT ON `classicmodels`.* TO 'rfc' @'localhost'

608

```
GRANT EXECUTE
ON classicmodels.*
TO rfc@localhost;
```

	Grants for rfc@localhost
▶	GRANT USAGE ON *.* TO 'rfc' @'localhost'
	GRANT SELECT, EXECUTE ON `classicmodels`.* TO 'rfc' @'localhost'

```
SHOW GRANTS FOR rfc@localhost;
```

```
REVOKE ALL, GRANT OPTION
FROM rfc@localhost;
```

```
SHOW GRANTS FOR rfc@localhost;
```

	Grants for rfc@localhost
▶	GRANT USAGE ON *.* TO 'rfc' @'localhost'

609

```
GRANT PROXY
```

```
ON root
```

```
TO rfc@localhost;
```

```
SHOW GRANTS FOR rfc@localhost;
```

	Grants for rfc@localhost
▶	GRANT USAGE ON *.* TO 'rfc' @'localhost'
	GRANT PROXY ON 'root'@'%' TO 'rfc'@'localhost'

```
REVOKE PROXY
```

```
ON root
```

```
FROM rfc@localhost;
```

```
SHOW GRANTS FOR rfc@localhost;
```

	Grants for rfc@localhost
▶	GRANT USAGE ON *.* TO 'rfc' @'localhost'

610

MySQL SHOW GRANTS

SHOW GRANTS

[**FOR** {user | role}]

[**USING** role [, role] ...]]

The following statement uses the SHOW GRANTS statement to display the privileges granted for the current user:

У наступному операторі використовується оператор SHOW GRANTS для відображення привілеїв, наданих поточному користувачеві:

Следующий оператор использует оператор SHOW GRANTS для отображения привилегий, предоставленных текущему пользователю:

```
SHOW GRANTS;
```

```
SHOW GRANTS FOR CURRENT_USER;
```

```
SHOW GRANTS FOR CURRENT_USER();
```

611

MySQL DROP USER

To remove a user account from the MySQL Server, you use the DROP USER statement as follows:

Щоб видалити обліковий запис користувача з сервера MySQL, використовуйте оператор DROP USER наступним чином:

Чтобы удалить учетную запись пользователя с сервера MySQL, используйте оператор DROP USER следующим образом:

DROP USER [IF EXISTS] account_name [,account_name2]...;

Besides removing the user account, the DROP USER statement also removes all privileges of the user.

Окрім видалення облікового запису користувача, оператор DROP USER також видаляє всі привілеї користувача.

Помимо удаления учетной записи пользователя, оператор DROP USER также удаляет все привилегии пользователя.

612

```
mysql> create user api@localhost, remote, dbadmin@localhost,
alice@localhost identified by 'Secure1Pass!';
```

```
mysql> select user, host from mysql.user;
```

user	host
remote	%
alice	localhost
api	localhost
bob	localhost
dbadmin	localhost
mysql.infoschema	localhost
mysql.session	localhost
mysql.sys	localhost
root	localhost

9 rows in set (0.00 sec)

613

```
mysql> drop user dbadmin@localhost
```

```
mysql> select user, host from mysql.user;
```

user	host
remote	%
alice	localhost
api	localhost
bob	localhost
mysql.infoschema	localhost
mysql.session	localhost
mysql.sys	localhost
root	localhost

8 rows in set (0.00 sec)

614

```
mysql> drop user api@localhost, remote;
```

```
mysql> select user, host from mysql.user;
```

user	host
alice	localhost
bob	localhost
mysql.infoschema	localhost
mysql.session	localhost
mysql.sys	localhost
root	localhost

6 rows in set (0.00 sec)

The first way to change the password is to use the UPDATE statement to update the user table of the mysql database.

After executing the UPDATE statement, you also need to execute the FLUSH PRIVILEGES statement to reload privileges in the mysql database.

Перший спосіб змінити пароль – використання оператора UPDATE для оновлення таблиці користувачів бази даних mysql.

Після виконання оператора UPDATE вам також потрібно виконати оператор FLUSH PRIVILEGES, щоб перезавантажити привілеї в базі даних mysql.

Первый способ изменить пароль – использовать оператор UPDATE для обновления пользовательской таблицы базы данных mysql.

После выполнения оператора UPDATE вам также необходимо выполнить оператор FLUSH PRIVILEGES, чтобы перезагрузить привилегии в базе данных mysql.

```
USE mysql;
```

```
UPDATE user
SET authentication_string = PASSWORD('dolphin')
WHERE user = 'dbadmin' AND
      host = 'localhost';
```

```
FLUSH PRIVILEGES;
```

Notice that the PASSWORD() function computes the hash value from a plain text.

Зверніть увагу, що функція PASSWORD() обчислює хеш-значення з простого тексту.

Обратите внимание, что функция PASSWORD() вычисляет хеш-значение из простого текста.

The second way to change the password is by using the SET PASSWORD statement.

You use the user account in user@host format to update the password. If you need to change the password for other accounts, your account needs to have at least UPDATE privilege.

By using the SET PASSWORD statement, you don't need to execute the FLUSH PRIVILEGES statement to reload privileges.

Другий спосіб змінити пароль – використання оператора SET PASSWORD.

Використовуйте обліковий запис користувача у форматі user@host для оновлення пароля. Якщо вам потрібно змінити пароль для інших облікових записів, ваш обліковий запис повинен мати принаймні привілей на UPDATE.

Використовуючи оператор SET PASSWORD, вам не потрібно виконувати оператор FLUSH PRIVILEGES, щоб перезавантажити привілеї.

Второй способ изменить пароль – использовать инструкцию SET PASSWORD.

Вы используете учетную запись пользователя в формате user@host для обновления пароля. Если вам нужно изменить пароль для других учетных записей, ваша учетная запись должна иметь как минимум привилегию UPDATE. Используя оператор SET PASSWORD, вам не нужно выполнять оператор FLUSH PRIVILEGES для перезагрузки привилегий.

```
SET PASSWORD FOR 'dbadmin'@'localhost' = bigshark;
```

617

The third way to change the password for a user account is to use the ALTER USER statement with the IDENTIFIED BY clause.

Третім способом зміни пароля для облікового запису користувача є використання оператора ALTER USER із виразом IDENTIFIED BY.

Третий способ изменить пароль для учетной записи пользователя – использовать оператор ALTER USER с выражением IDENTIFIED BY.

```
ALTER USER dbadmin@localhost IDENTIFIED BY 'littlewhale';
```

618

For example, to show users and other information such as host, account locking, and password expiration status, you use the following query:

Наприклад, щоб показати користувачів та іншу інформацію, таку як хост, блокування облікового запису та статус закінчення терміну дії пароля, ви використовуєте такий запит:

Например, чтобы показать пользователей и другую информацию, такую как хост, блокировка учетной записи и статус истечения срока действия пароля, вы используете следующий запрос:

```
SELECT
    user,
    host,
    account_locked,
    password_expired
FROM
    user;
```

	user	host	account_locked	password_expired
▶	mysql.sys	localhost	Y	N
	mysqlxsys	localhost	Y	N
	root	localhost	N	N

619

To get the information on the current user, you use the user() function as shown in the following statement:

Щоб отримати інформацію про поточного користувача, використовуйте функцію user(), як показано в наступному твердженні:

Чтобы получить информацию о текущем пользователе, используйте функцию user(), как показано в следующем утверждении:

```
mysql> SELECT user();
+-----+
| user()      |
+-----+
| local@localhost |
+-----+
1 row in set (0.00 sec)

mysql> SELECT current_user();
+-----+
| current_user() |
+-----+
| local@localhost |
+-----+
1 row in set (0.00 sec)
```

620

To list all users that are currently logged in the MySQL database server, you execute the following statement:

Щоб перерахувати всіх користувачів, які на даний момент увійшли на сервер баз даних MySQL, виконайте наступний оператор:

Чтобы вывести список всех пользователей, которые в настоящее время вошли в систему на сервере базы данных MySQL, выполните следующую инструкцию:

```
SELECT
    user,
    host,
    db,
    command
FROM
    information_schema.processlist;

+-----+-----+-----+-----+
| user | host      | db      | command |
+-----+-----+-----+-----+
| local | localhost:50591 | classicmodels | Sleep   |
| root  | localhost:50557  | NULL     | Query   |
+-----+-----+-----+-----+
2 rows in set (0.00 sec)
```

621

The RENAME USER statement renames one or more existing accounts. Here is the basic syntax of the RENAME USER statement:

Оператор RENAME USER перейменовує один або кілька існуючих облікових записів. Ось основний синтаксис оператора RENAME USER:

Оператор RENAME USER переименовывает одну или несколько существующих учетных записей. Вот основной синтаксис оператора RENAME USER:

```
RENAME USER old_user1
TO new_user;
```

```
RENAME USER
old_user1 TO new_user1,
old_user2 TO new_user2,
...
```

The RENAME USER transfers all privileges of the old users to the new users. However, it does not drop or invalidate database objects that are dependent on old users.

RENAME USER передає всі привілеї старих користувачів новим. Однак вона не видаляє та не робить недійсними об'єкти бази даних, які залежать від старих користувачів.

RENAME USER передает все привилегии старых пользователей новым пользователям. Однако она не удаляет и не делает недействительными объекты базы данных, зависящие от старых пользователей.

622

```
CREATE USER john@localhost
IDENTIFIED BY 'Super!pass1';
```

```
RENAME USER john@localhost
TO doe@localhost;
```

```
CREATE USER jill@localhost
IDENTIFIED BY 'Super!pass1';
```

```
CREATE USER hill@localhost
IDENTIFIED BY 'Super!pass1';
```

```
RENAME USER
jill@localhost TO jin@localhost,
hill@localhost TO hank@localhost;
```

```
SELECT host, user
FROM mysql.user
WHERE user IN ('jin', 'hank');
```

	host	user
▶	localhost	hank
	localhost	jin

623

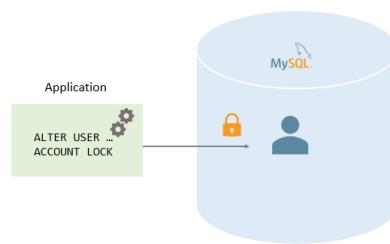
To lock a user account, you can use the CREATE USER .. ACCOUNT LOCK statement:

Щоб заблокувати обліковий запис користувача, ви можете використовувати оператор CREATE USER .. ACCOUNT LOCK:

Чтобы заблокировать учетную запись пользователя, вы можете использовать оператор CREATE USER .. ACCOUNT LOCK:

```
CREATE USER account_name
IDENTIFIED BY 'password'
ACCOUNT LOCK;
```

```
ALTER USER account_name
IDENTIFIED BY 'password'
ACCOUNT LOCK;
```



624

```
CREATE USER david@localhost
IDENTIFIED BY 'Secret!Pass1'
ACCOUNT LOCK;
```

```
SELECT
    user, host, account_locked
FROM
    mysql.user
WHERE
    user = 'david' AND
    host='localhost';
```

	user	host	account_locked
▶	david	localhost	Y

```
mysql -u david -p
Enter password: *****
```

```
ERROR 3118 (HY000): Access denied for user 'david'@'localhost'.
Account is locked.
```

625

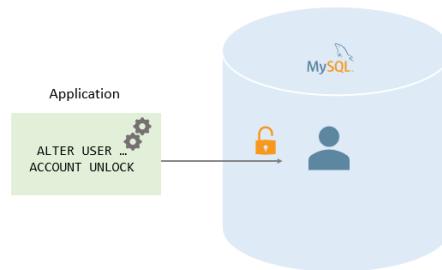
To unlock a user account, use the ALTER USER ACCOUNT LOCK statement:

Щоб розблокувати обліковий запис користувача, використовуйте оператор ALTER USER ACCOUNT LOCK:

Чтобы разблокировать учетную запись пользователя, используйте инструкцию ALTER USER ACCOUNT LOCK:

```
ALTER USER [IF EXISTS] account_name
ACCOUNT UNLOCK;
```

```
ALTER USER [IF EXISTS]
account_name1
[, account_name2, ...]
ACCOUNT UNLOCK;
```



626

```
CREATE USER brad@localhost
IDENTIFIED BY 'Secret!pass1'
ACCOUNT LOCK;
```

```
SELECT
    user,
    host,
    account_locked
FROM
    mysql.user
WHERE
    user = 'brad' AND
    host = 'localhost';
```

	user	host	account_locked
▶	brad	localhost	Y

```
ALTER USER 'brad'@'localhost'
ACCOUNT UNLOCK;
```

627

Контрольні питання

1. MySQL PRIMARY KEY.
2. MySQL UNIQUE KEY.
3. MySQL KEY.
4. MySQL FOREIGN KEY.
5. CASCADE, SET NULL, NO ACTION, RESTRICT, SET DEFAULT.
6. MySQL START TRANSACTION, COMMIT, ROLLBACK.
7. ACID.
8. Брудне читання.
Неповторюване читання.
Фантомне читання.
9. Рівні ізоляції.
10. SAVEPOINT. ROLLBACK TO SAVEPOINT.
11. MySQL CREATE USER.
12. MySQL GRANT.
13. MySQL REVOKE.
14. MySQL SHOW GRANTS.
15. MySQL DROP USER.
16. Зміна паролю користувача.
17. Інформація про користувачів.
18. MySQL RENAME USER.
19. MySQL ACCOUNT LOCK.
ACCOUNT UNLOCK.

628

Assessment questions

1. MySQL PRIMARY KEY.
2. MySQL UNIQUE KEY.
3. MySQL KEY.
4. MySQL FOREIGN KEY.
5. CASCADE, SET NULL, NO ACTION, RESTRICT, SET DEFAULT.
6. MySQL START TRANSACTION, COMMIT, ROLLBACK.
7. ACID.
8. Dirty Read. Non Repeatable Read. Phantom Read.
9. Isolation levels.
10. SAVEPOINT. ROLLBACK TO SAVEPOINT.
11. MySQL CREATE USER.
12. MySQL GRANT.
13. MySQL REVOKE.
14. MySQL SHOW GRANTS.
15. MySQL DROP USER.
16. Change user password.
17. User information.
18. MySQL RENAME USER.
19. MySQL ACCOUNT LOCK.
ACCOUNT UNLOCK.

Контрольные вопросы

1. MySQL PRIMARY KEY.
2. MySQL UNIQUE KEY.
3. MySQL KEY.
4. MySQL FOREIGN KEY.
5. CASCADE, SET NULL, NO ACTION, RESTRICT, SET DEFAULT.
6. MySQL START TRANSACTION, COMMIT, ROLLBACK.
7. ACID.
8. Грязное чтение.
Неповторяющееся чтение.
Фантомное чтение.
9. Уровни изоляции.
10. SAVEPOINT. ROLLBACK TO SAVEPOINT.
11. MySQL CREATE USER.
12. MySQL GRANT.
13. MySQL REVOKE.
14. MySQL SHOW GRANTS.
15. MySQL DROP USER.
16. Смена пароля пользователя.
17. Информация о пользователях.
18. MySQL RENAME USER.
19. MySQL ACCOUNT LOCK.
ACCOUNT UNLOCK.

Побудова клієнт-серверних застосунків для роботи з БД
Development of client-server applications to work with DBs
Построение клиент-серверных приложений для работы с БД

Тема 6

Topic 6

631

PHP DATABASE APPLICATION

632

PHP provides **mysqli** instance or **mysqli_connect()** function to open a database connection. This function takes six parameters and returns a MySQL link identifier on success or FALSE on failure.

```
$mysqli = new mysqli($host, $username, $passwd,  
$dbName, $port, $socket);
```

- **\$host**
- Optional – The host name running the database server. If not specified, then the default value will be **localhost:3306**.
- **\$username**
- Optional – The username accessing the database. If not specified, then the default will be the name of the user that owns the server process.
- **\$passwd**
- Optional – The password of the user accessing the database. If not specified, then the default will be an empty password.

633

- **\$dbName**
- Optional – database name on which query is to be performed.

- **\$port**
- Optional – the port number to attempt to connect to the MySQL server.

- **\$socket**
- Optional – socket or named pipe that should be used.

You can disconnect from the MySQL database anytime using another PHP function **close()**.

```
$mysqli->close();
```

634

```

1 <html>
2   <head>
3     <title>Connecting MySQL Server</title>
4   </head>
5   <body>
6     <?php
7       $dbhost = 'localhost';
8       $dbuser = 'root';
9       $dbpass = 'root@123';
10      $mysqli = new mysqli($dbhost, $dbuser, $dbpass);
11
12      if($mysqli->connect_errno) {
13        printf("Connect failed: %s<br />", $mysqli->connect_error);
14        exit();
15      }
16      printf('Connected successfully.<br />');
17      $mysqli->close();
18    ?>
19  </body>
20 </html>
```

635

PHP uses **`mysqli_query()`** or **`mysql_query()`** function to create or delete a MySQL database. This function takes two parameters and returns TRUE on success or FALSE on failure.

```
$mysqli->query($sql, $resultmode)
```

- **`$sql`**
- Required - SQL query to create a MySQL database.

- **`$resultmode`**
- Optional - Either the constant `MYSQLI_USE_RESULT` or `MYSQLI_STORE_RESULT` depending on the desired behavior. By default, `MYSQLI_STORE_RESULT` is used.

636

```

1 <html>
2   <head><title>Creating MySQL Database</title></head>
3   <body>
4     <?php
5       $dbhost = 'localhost';
6       $dbuser = 'root';
7       $dbpass = 'root@123';
8       $mysqli = new mysqli($dbhost, $dbuser, $dbpass);
9
10      if($mysqli->connect_errno) {
11        printf("Connect failed: %s<br />", $mysqli->connect_error);
12        exit();
13      }
14      printf('Connected successfully.<br />');
15
16      if ($mysqli->query("CREATE DATABASE TUTORIALS")) {
17        printf("Database TUTORIALS created successfully.<br />");
18      }
19      if ($mysqli->errno) {
20        printf("Could not create database: %s<br />", $mysqli->error);
21      }
22      $mysqli->close();
23    ?>
24  </body>
25 </html>
```

637

PHP uses **mysqli_select_db** function to select the database on which queries are to be performed. This function takes two parameters and returns TRUE on success or FALSE on failure.

```
mysqli_select_db(mysqli $link , string $dbname)
```

- **\$link**
- Required - A link identifier returned by mysqli_connect() or mysqli_init().
- **\$dbname**
- Required - Name of the database to be connected.

638

```

1 <html>
2   <head>
3     <title>Selecting MySQL Database</title>
4   </head>
5   <body>
6     <?php
7       $dbhost = 'localhost';
8       $dbuser = 'root';
9       $dbpass = 'root@123';
10      $conn = mysqli_connect($dbhost, $dbuser, $dbpass);
11
12      if(! $conn ) {
13        die('Could not connect: ' . mysqli_error($conn));
14      }
15      echo 'Connected successfully<br />';
16      $retval = mysqli_select_db( $conn, 'TUTORIALS' );
17
18      if(! $retval ) {
19        die('Could not select database: ' . mysqli_error($conn));
20      }
21      echo "Database TUTORIALS selected successfully\n";
22      mysqli_close($conn);
23    ?>
24  </body>
25 </html>
```

639

PHP uses **mysqli_query()** or **mysql_query()** function to create a MySQL table. This function takes two parameters and returns TRUE on success or FALSE on failure.

```
$mysqli->query($sql, $resultmode)
```

- **\$sql**
- Required - SQL query to create a MySQL table.

- **\$resultmode**
- Optional - Either the constant MYSQLI_USE_RESULT or MYSQLI_STORE_RESULT depending on the desired behavior. By default, MYSQLI_STORE_RESULT is used.

640

```
1 <html>
2   <head>
3     <title>Creating MySQL Table</title>
4   </head>
5   <body>
6     <?php
7       $dbhost = 'localhost';
8       $dbuser = 'root';
9       $dbpass = 'root@123';
10      $dbname = 'TUTORIALS';
11      $mysqli = new mysqli($dbhost, $dbuser, $dbpass, $dbname);
12
13      if($mysqli->connect_errno) {
14        printf("Connect failed: %s<br />", $mysqli->connect_error);
15        exit();
16      }
```

641

```

17     printf('Connected successfully.<br />');
18     $sql = "CREATE TABLE tutorials_tbl( ".
19         "tutorial_id INT NOT NULL AUTO_INCREMENT, ".
20         "tutorial_title VARCHAR(100) NOT NULL, ".
21         "tutorial_author VARCHAR(40) NOT NULL, ".
22         "submission_date DATE, ".
23         "PRIMARY KEY ( tutorial_id )); ";
24     if ($mysqli->query($sql)) {
25         printf("Table tutorials_tbl created successfully.<br />");
26     }
27     if ($mysqli->errno) {
28         printf("Could not create table: %s<br />", $mysqli->error);
29     }
30     $mysqli->close();
31     ?>
32   </body>
33 </html>

```

642

PHP uses **mysqli_query** function to insert records in table. This function takes three parameters and returns TRUE on success or FALSE on failure.

```
mysqli_query(mysqli $link, string $query, int
$resultmode = MYSQLI_STORE_RESULT)
```

- **\$link**
- Required - A link identifier returned by mysqli_connect() or mysqli_init().

- **\$query**
- Required - SQL query to drop a table.

- **\$resultmode**
- Optional - Either the constant MYSQLI_USE_RESULT or MYSQLI_STORE_RESULT depending on the desired behavior. By default, MYSQLI_STORE_RESULT is used.

643

This example will take three parameters from the user and will insert them into the MySQL table

Tutorial Title	<input type="text"/>
Tutorial Author	<input type="text"/>
Submission Date [yyyy-mm-dd]	<input type="text"/>
<input type="button" value="Add Tutorial"/>	

- Access the page deployed on Apache web server, enter details and verify the output on submitting the form.
- You can put many validations around to check if the entered data is correct or not and can take the appropriate action.

644

```

1 <html>
2   <head>
3     <title>Add New Record in MySQL Database</title>
4   </head>
5   <body>
6     <?php
7       if(isset($_POST['add'])) {
8         $dbhost = 'localhost';
9         $dbuser = 'root';
10        $dbpass = 'root@123';
11        $conn = mysqli_connect($dbhost, $dbuser, $dbpass);
12
13        if(! $conn ) {
14          die('Could not connect: ' . mysqli_error($conn));
15        }
16        $tutorial_title = $_POST['tutorial_title'];
17        $tutorial_author = $_POST['tutorial_author'];
18        $submission_date = $_POST['submission_date'];

```

645

```

19     $sql = "INSERT INTO tutorials_tbl ".
20         "(tutorial_title,tutorial_author, submission_date) "."VALUES ".
21         "('$tutorial_title','$tutorial_author','$submission_date')";
22 mysqli_select_db( $conn, 'TUTORIALS' );
23 $retval = mysqli_query( $conn, $sql );
24
25 if(! $retval ) {
26     die('Could not enter data: ' . mysqli_error($conn));
27 }
28 echo "Entered data successfully\n";
29 mysqli_close($conn);

```

646

```

30 } else {
31 ?>
32 <form method = "post" action = "<?php $_PHP_SELF ?>">
33     <table width = "600" border = "0" cellspacing = "1" cellpadding = "2">
34         <tr><td width = "250">Tutorial Title</td>
35             <td><input name = "tutorial_title" type = "text" id = "tutorial_title"></td>
36         </tr>
37         <tr><td width = "250">Tutorial Author</td>
38             <td><input name = "tutorial_author" type = "text" id = "tutorial_author"></td>
39         </tr>
40         <tr><td width = "250">Submission Date [ yyyy-mm-dd ]</td>
41             <td><input name = "submission_date" type = "text" id = "submission_date"></td>
42         </tr>
43         <tr><td width = "250"> </td>
44             <td></td>
45         </tr>
46         <tr><td width = "250"> </td>
47             <td><input name = "add" type = "submit" id = "add" value = "Add Tutorial"></td>
48         </tr>
49     </table>
50 </form>
51 <?php
52     ?
53 ?>
54 </body>
55 </html>

```

647

You can use the same SQL SELECT command into a PHP function **mysql_query()**. This function is used to execute the SQL command and then later another PHP function **mysql_fetch_array()** can be used to fetch all the selected data. This function returns the row as an associative array, a numeric array, or both. This function returns FALSE if there are no more rows.

```

1 <html>
2   <head>
3     <title>Selecting Records</title>
4   </head>
5   <body>
6     <?php
7       $dbhost = 'localhost';
8       $dbuser = 'root';
9       $dbpass = 'root@123';
10      $conn = mysqli_connect($dbhost, $dbuser, $dbpass);

```

648

```

11      if(! $conn ) {
12        die('Could not connect: ' . mysqli_error($conn));
13      }
14      echo 'Connected successfully<br />';
15
16      mysqli_select_db( $conn, 'TUTORIALS' );
17      $sql = "SELECT tutorial_id, tutorial_title, tutorial_author, submission_date FROM
18      tutorials_tbl";
19      $retval = mysqli_query( $conn, $sql );
20      if(! $retval ) {
21        die('Could not get data: ' . mysqli_error($conn));
22      }
23
24      while($row = mysqli_fetch_array($retval, MYSQL_ASSOC)) {
25        echo "Tutorial ID :{$row['tutorial_id']} ".
26          "Title: {$row['tutorial_title']} ".
27          "Author: {$row['tutorial_author']} ".
28          "Submission Date : {$row['submission_date']} ".
29          "-----";
30      }
31      echo "Fetched data successfully\n";
32      mysqli_close($conn);
33    ?>
34  </body>
35 </html>

```

649

PHP uses **`mysqli_query()`** or **`mysql_query()`** function to update records in a MySQL table. This function takes two parameters and returns TRUE on success or FALSE on failure.

```
$mysqli->query($sql, $resultmode)
```

- **`$sql`**
- Required - SQL query to update records in a MySQL table.

- **`$resultmode`**
- Optional - Either the constant `MYSQLI_USE_RESULT` or `MYSQLI_STORE_RESULT` depending on the desired behavior. By default, `MYSQLI_STORE_RESULT` is used.

650

```

1 <html>
2   <head>
3     <title>Updating MySQL Table</title>
4   </head>
5   <body>
6     <?php
7       $dbhost = 'localhost';
8       $dbuser = 'root';
9       $dbpass = 'root@123';
10      $dbname = 'TUTORIALS';
11      $mysqli = new mysqli($dbhost, $dbuser, $dbpass, $dbname);
12
13      if($mysqli->connect_errno) {
14        printf("Connect failed: %s<br />", $mysqli->connect_error);
15        exit();
16      }
17      printf('Connected successfully.<br />');

```

651

```
17     printf('Connected successfully.<br />');
18     if ($mysqli->query('UPDATE tutorials_tbl set tutorial_title = "Learning Java" where
19         tutorial_id = 4')) {
20         printf("Table tutorials_tbl updated successfully.<br />");
21     }
22     if ($mysqli->errno) {
23         printf("Could not update table: %s<br />", $mysqli->error);
24     }
25     $sql = "SELECT tutorial_id, tutorial_title, tutorial_author, submission_date FROM
26         tutorials_tbl";
27     $result = $mysqli->query($sql);
```

652

```
28     if ($result->num_rows > 0) {
29         while($row = $result->fetch_assoc()) {
30             printf("Id: %s, Title: %s, Author: %s, Date: %d <br />",
31                 $row["tutorial_id"],
32                 $row["tutorial_title"],
33                 $row["tutorial_author"],
34                 $row["submission_date"]);
35         }
36     } else {
37         printf('No record found.<br />');
38     }
39     mysqli_free_result($result);
40     $mysqli->close();
41     ?>
42     </body>
43 </html>
```

653

- Access the PHP deployed on Apache web server and verify the output. Here we've entered multiple records in the table before running the select script.

```

Connected successfully.
Table tutorials_tbl updated successfully.
Id: 1, Title: MySQL Tutorial, Author: Mahesh, Date: 2021
Id: 2, Title: HTML Tutorial, Author: Mahesh, Date: 2021
Id: 3, Title: PHP Tutorial, Author: Mahesh, Date: 2021
Id: 4, Title: Learning Java, Author: Mahesh, Date: 2021
Id: 5, Title: Apache Tutorial, Author: Suresh, Date: 2021

```

654

PHP uses **mysqli_query()** or **mysql_query()** function to delete records in a MySQL table. This function takes two parameters and returns TRUE on success or FALSE on failure.

```
$mysqli->query($sql, $resultmode)
```

- \$sql**
- Required - SQL query to delete records in a MySQL table.
- \$resultmode**
- Optional - Either the constant MYSQLI_USE_RESULT or MYSQLI_STORE_RESULT depending on the desired behavior. By default, MYSQLI_STORE_RESULT is used.

655

```

1 <html>
2   <head>
3     <title>Deleting MySQL Table record</title>
4   </head>
5   <body>
6     <?php
7       $dbhost = 'localhost';
8       $dbuser = 'root';
9       $dbpass = 'root@123';
10      $dbname = 'TUTORIALS';
11      $mysqli = new mysqli($dbhost, $dbuser, $dbpass, $dbname);
12
13      if($mysqli->connect_errno) {
14        printf("Connect failed: %s<br />", $mysqli->connect_error);
15        exit();
16      }
17      printf('Connected successfully.<br />');

```

656

```

18      if ($mysqli->query('DELETE FROM tutorials_tbl where tutorial_id = 4')) {
19        printf("Table tutorials_tbl record deleted successfully.<br />");
20      }
21      if ($mysqli->errno) {
22        printf("Could not delete record from table: %s<br />", $mysqli->error);
23      }
24      $sql = "SELECT tutorial_id, tutorial_title, tutorial_author, submission_date FROM
25 tutorials_tbl";
26
27      $result = $mysqli->query($sql);
28      if ($result->num_rows > 0) {
29        while($row = $result->fetch_assoc()) {
30          printf("Id: %s, Title: %s, Author: %s, Date: %d <br />",
31            $row["tutorial_id"],
32            $row["tutorial_title"],
33            $row["tutorial_author"],
34            $row["submission_date"]);
35        }
36      } else {
37        printf('No record found.<br />');
38      }
39      mysqli_free_result($result);
40      $mysqli->close();
41    ?>
42  </body>
43 </html>

```

657

```
Connected successfully.  
Table tutorials_tbl updated successfully.  
Id: 1, Title: MySQL Tutorial, Author: Mahesh, Date: 2021  
Id: 2, Title: HTML Tutorial, Author: Mahesh, Date: 2021  
Id: 3, Title: PHP Tutorial, Author: Mahesh, Date: 2021  
Id: 4, Title: Learning Java, Author: Mahesh, Date: 2021  
Id: 5, Title: Apache Tutorial, Author: Suresh, Date: 2021
```



```
Connected successfully.  
Table tutorials_tbl record deleted successfully.  
Id: 1, Title: MySQL Tutorial, Author: Mahesh, Date: 2021  
Id: 2, Title: HTML Tutorial, Author: Mahesh, Date: 2021  
Id: 3, Title: PHP Tutorial, Author: Mahesh, Date: 2021  
Id: 5, Title: Apache Tutorial, Author: Suresh, Date: 2021
```

658

```
1 <html>  
2   <head>  
3     <title>Using Where Clause</title>  
4   </head>  
5   <body>  
6     <?php  
7       $dbhost = 'localhost';  
8       $dbuser = 'root';  
9       $dbpass = 'root@123';  
10      $dbname = 'TUTORIALS';  
11      $mysqli = new mysqli($dbhost, $dbuser, $dbpass, $dbname);  
12  
13      if($mysqli->connect_errno) {  
14          printf("Connect failed: %s<br />", $mysqli->connect_error);  
15          exit();  
16      }  
17      printf('Connected successfully.<br />');
```

659

```

18      $sql = 'SELECT tutorial_id, tutorial_title, tutorial_author, submission_date FROM
19      tutorials_tbl where tutorial_author = "Mahesh"';
20
21      $result = $mysqli->query($sql);
22
23      if ($result->num_rows > 0) {
24          while($row = $result->fetch_assoc()) {
25              printf("Id: %s, Title: %s, Author: %s, Date: %d <br />",
26                  $row["tutorial_id"],
27                  $row["tutorial_title"],
28                  $row["tutorial_author"],
29                  $row["submission_date"]);
30          }
31      } else {
32          printf('No record found.<br />');
33      }
34      mysqli_free_result($result);
35      $mysqli->close();
36  ?>
37  </body>
38 </html>

```

660

Connected successfully.
Table tutorials_tbl updated successfully.
Id: 1, Title: MySQL Tutorial, Author: Mahesh, Date: 2021
Id: 2, Title: HTML Tutorial, Author: Mahesh, Date: 2021
Id: 3, Title: PHP Tutorial, Author: Mahesh, Date: 2021
Id: 5, Title: Apache Tutorial, Author: Suresh, Date: 2021



Connected successfully.
Id: 1, Title: MySQL Tutorial, Author: Mahesh, Date: 2021
Id: 2, Title: HTML Tutorial, Author: Mahesh, Date: 2021
Id: 3, Title: PHP Tutorial, Author: Mahesh, Date: 2021

661

```

18      $sql = 'SELECT tutorial_id, tutorial_title, tutorial_author, submission_date FROM
19      tutorials_tbl where tutorial_author like "Mah%"';
20
21      $result = $mysqli->query($sql);
22
23      if ($result->num_rows > 0) {
24          while($row = $result->fetch_assoc()) {
25              printf("Id: %s, Title: %s, Author: %s, Date: %d <br />",
26                  $row["tutorial_id"],
27                  $row["tutorial_title"],
28                  $row["tutorial_author"],
29                  $row["submission_date"]);
30          }
31      } else {
32          printf('No record found.<br />');
33      }
34      mysqli_free_result($result);
35      $mysqli->close();
36  ?>
37  </body>
38 </html>

```

662

Connected successfully.
Table tutorials_tbl updated successfully.
Id: 1, Title: MySQL Tutorial, Author: Mahesh, Date: 2021
Id: 2, Title: HTML Tutorial, Author: Mahesh, Date: 2021
Id: 3, Title: PHP Tutorial, Author: Mahesh, Date: 2021
Id: 5, Title: Apache Tutorial, Author: Suresh, Date: 2021



Connected successfully.
Id: 1, Title: MySQL Tutorial, Author: Mahesh, Date: 2021
Id: 2, Title: HTML Tutorial, Author: Mahesh, Date: 2021
Id: 3, Title: PHP Tutorial, Author: Mahesh, Date: 2021

663

```

18      $sql = "SELECT tutorial_id, tutorial_title, tutorial_author, submission_date FROM
19      tutorials_tbl order by tutorial_title asc";
20
21      $result = $mysqli->query($sql);
22
23      if ($result->num_rows > 0) {
24          while($row = $result->fetch_assoc()) {
25              printf("Id: %s, Title: %s, Author: %s, Date: %d <br />",
26                  $row["tutorial_id"],
27                  $row["tutorial_title"],
28                  $row["tutorial_author"],
29                  $row["submission_date"]);
30
31      } else {
32          printf('No record found.<br />');
33      }
34      mysqli_free_result($result);
35      $mysqli->close();
36  ?>
37  </body>
38 </html>

```

664

Connected successfully.
Table tutorials_tbl updated successfully.
Id: 1, Title: MySQL Tutorial, Author: Mahesh, Date: 2021
Id: 2, Title: HTML Tutorial, Author: Mahesh, Date: 2021
Id: 3, Title: PHP Tutorial, Author: Mahesh, Date: 2021
Id: 5, Title: Apache Tutorial, Author: Suresh, Date: 2021



Connected successfully.
Id: 5, Title: Apache Tutorial, Author: Suresh, Date: 2021
Id: 2, Title: HTML Tutorial, Author: Mahesh, Date: 2021
Id: 1, Title: MySQL Tutorial, Author: Mahesh, Date: 2021
Id: 3, Title: PHP Tutorial, Author: Mahesh, Date: 2021

665

First create a table in MySQL using following script and insert two records.

```

create table tcount_tbl(
    tutorial_author VARCHAR(40) NOT NULL,
    tutorial_count int
);

insert into tcount_tbl values('Mahesh', 3);
insert into tcount_tbl values('Suresh', 1);

```

666

```

18      $sql = 'SELECT a.tutorial_id, a.tutorial_author, b.tutorial_count
19          FROM tutorials_tbl a, tcount_tbl b
20          WHERE a.tutorial_author = b.tutorial_author';
21      $result = $mysqli->query($sql);
22
23      if ($result->num_rows > 0) {
24          while($row = $result->fetch_assoc()) {
25              printf("Id: %s, Author: %s, Count: %d <br />",
26                  $row["tutorial_id"],
27                  $row["tutorial_author"],
28                  $row["tutorial_count"]);
29          }
30      } else {
31          printf('No record found.<br />');
32      }
33      mysqli_free_result($result);
34      $mysqli->close();
35      ?>
36  </body>
37 </html>

```

667

Access the PHP file deployed on Apache web server and verify the output.

```
Connected successfully.
Id: 1, Author: Mahesh, Count: 3
Id: 2, Author: Mahesh, Count: 3
Id: 3, Author: Mahesh, Count: 3
Id: 5, Author: Suresh, Count: 1
```

668

- You can use the **if...else** condition to prepare a query based on the NULL value.
- The following example takes the tutorial_count from outside and then compares it with the value available in the table.

```
1 <html>
2   <head>
3     <title>Handling NULL</title>
4   </head>
5   <body>
6     <?php
7       $dbhost = 'localhost';
8       $dbuser = 'root';
9       $dbpass = 'root@123';
10      $dbname = 'TUTORIALS';
11      $mysqli = new mysqli($dbhost, $dbuser, $dbpass, $dbname);
12      $tutorial_count = null;
13      if($mysqli->connect_errno) {
14        printf("Connect failed: %s<br />", $mysqli->connect_error);
15        exit();
16      }
17      printf('Connected successfully.<br />');
```

669

```

18     if( isset($tutorial_count) ) {
19         $sql = 'SELECT tutorial_author, tutorial_count
20             FROM tcount_tbl
21             WHERE tutorial_count = ' + $tutorial_count;
22     } else {
23         $sql = 'SELECT tutorial_author, tutorial_count
24             FROM tcount_tbl
25             WHERE tutorial_count IS NULL';
26     }
27     $result = $mysqli->query($sql);
28
29     if ($result->num_rows > 0) {
30         while($row = $result->fetch_assoc()) {
31             printf("Author: %s, Count: %d <br />",
32                 $row["tutorial_author"],
33                 $row["tutorial_count"]);
34         }
35     } else {
36         printf('No record found.<br />');
37     }
38     $mysqli->close();
39     ?>
40     </body>
41 </html>
```

670

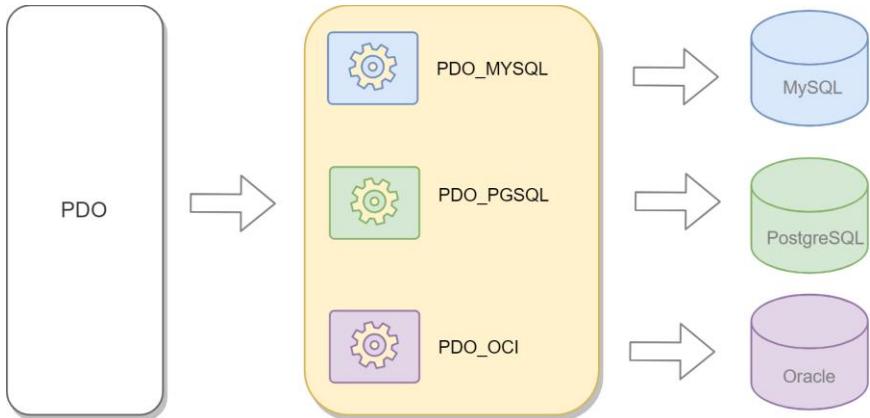
PHP PDO is a database access layer that provides a uniform interface for working with multiple databases.

- PDO simplifies the common database operations including:
- Creating database connections
- Executing queries using prepared statements
- Calling stored procedures
- Performing transactions
- And handling errors

PDO allows you to work with any database that has a PDO driver available. PDO relies on database-specific drivers, e.g., PDO_MYSQL for MySQL, PDO_PGSQ for PostgreSQL, PDO_OCI for Oracle database, etc., to function properly.

Therefore, to use PDO for a specific database, you need to have a corresponding database driver available.

671



672

Suppose you have a local MySQL database server that has the following information:

- The host is localhost.
- The bookdb database on the local database server.
- The account with the user root and password 'S@cr@t1!' that can access the bookdb database.
- In PHP, you can create a config.php file and place the database parameters:

```
<?php
```

```
$host = 'localhost';
$db = 'bookdb';
$user = 'root';
$password = 'S@cr@t1!';
```

673

- To use the database parameters, you can include the config.php file using the require construct:

```
<?php
require 'config.php';
```

PDO_MYSQL is a driver that implements the PDO interface. PDO uses the PDO_MYSQL driver to connect to a MySQL database.

To check if the PDO_MYSQL driver is enabled, you open the php.ini file. The php.ini file is often located under the php directory. For example, you can find the php.ini file under the C:\xampp\php directory if you use XAMPP on Windows.

- The following shows the extension line in the php.ini file:

```
;extension=php_pdo_mysql.dll
```

- To enable the extension, you need to uncomment it by removing the semicolon (;) from the beginning of the line like this:

```
extension=php_pdo_mysql.dll
```

674

PDO uses a data source name (DSN) that contains the following information:

- The database server host
- The database name
- The user
- The password
- and other parameters such as character sets, etc.

PDO uses this information to make a connection to the database server. To connect to the MySQL database server, you use the following data source name format:

```
"mysql:host=host_name;dbname=db_name;charset=UTF8"
```

Note that the charset UTF-8 sets the character set of the database connection to UTF-8.

675

- The following index.php script illustrates how to connect to the bookdb database on the MySQL database server with the root account:

```
<?php

require 'config.php';

$dsn = "mysql:host=$host;dbname=$db;charset=UTF8";

try {
    $pdo = new PDO($dsn, $user, $password);

    if ($pdo) {
        echo "Connected to the $db database
successfully!";
    }
} catch (PDOException $e) {
    echo $e->getMessage();
}
```

- If you have everything set up correctly, you will see the following message:

Connected to the bookdb database successfully!

676

To query data from a table using the query() method, you follow these steps:

- Create a database connection to the database server.
- Execute a SELECT statement by passing it to the query() method of a PDO object.
- The query() method returns a PDOStatement object. If an error occurs, the query() method returns false.

The following illustrates how to query all rows from the publishers table in the bookdb database:

```
<?php

$pdo = require 'connect.php';

$sql = 'SELECT publisher_id, name
       FROM publishers';

$statement = $pdo->query($sql);

// get all publishers
$publishers = $statement->fetchAll(PDO::FETCH_ASSOC);

if ($publishers) {
    // show the publishers
    foreach ($publishers as $publisher) {
        echo $publisher['name'] . '<br>';
    }
}
```

677

The `fetchAll()` method with the `PDO::FETCH_ASSOC` option returns an associative array of data where:

The keys are the names that appear on the select list and the values are the data rows in the result set.

Finally, iterate over the result set and show the array's element:

```
<?php

// show the publishers
if ($publishers) {
    foreach ($publishers as $publisher) {
        echo $publisher['name'] . '<br>';
    }
}
```

678

The following example illustrates how to use a prepared statement to query data from a table:

- Construct an SQL SELECT statement with a named placeholder (`:publisher_id`)

```
$sql = 'SELECT publisher_id, name
       FROM publishers
      WHERE publisher_id = :publisher_id';
```

- Bind the value of the id to the prepared statement:

```
$statement->bindParam(':publisher_id', $publisher_id,
PDO::PARAM_INT);
```

- Execute the prepared statement:

```
$statement->execute();
```

- Fetch a row from the result set into an associative array:

```
$publisher = $statement->fetch(PDO::FETCH_ASSOC);
```

679

```
<?php  
  
$publisher_id = 1;  
  
// connect to the database and select the publisher  
$pdo = require 'connect.php';  
$sql = 'SELECT publisher_id, name  
       FROM publishers  
      WHERE publisher_id = :publisher_id';  
  
$statement = $pdo->prepare($sql);  
$statement->bindParam(':publisher_id', $publisher_id,  
PDO::PARAM_INT);  
$statement->execute();  
$publisher = $statement->fetch(PDO::FETCH_ASSOC);  
  
if ($publisher) {  
    echo $publisher['publisher_id'] . '.' .  
$publisher['name'];  
} else {  
    echo "The publisher with id $publisher_id was not found.";  
}
```

More about PDO: <https://www.phptutorial.net/php-pdo/>

680

JAVA DATABASE APPLICATION

681

JDBC stands for **Java Database Connectivity**, which is a standard Java API for database-independent connectivity between the Java programming language and a wide range of databases.

The JDBC library includes APIs for each of the tasks mentioned below that are commonly associated with database usage.

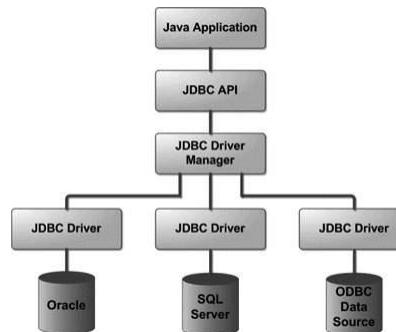
- Making a connection to a database.
- Creating SQL or MySQL statements.
- Executing SQL or MySQL queries in the database.
- Viewing & Modifying the resulting records.

The JDBC API supports both two-tier and three-tier processing models for database access but in general, JDBC Architecture consists of two layers:

- **JDBC API:** This provides the application-to-JDBC Manager connection.
- **JDBC Driver API:** This supports the JDBC Manager-to-Driver Connection.

682

- The JDBC API uses a driver manager and database-specific drivers to provide transparent connectivity to heterogeneous databases.
- The JDBC driver manager ensures that the correct driver is used to access each data source. The driver manager is capable of supporting multiple concurrent drivers connected to multiple heterogeneous databases.



683

The JDBC API provides the following interfaces and classes:

- **DriverManager:** This class manages a list of database drivers. Matches connection requests from the java application with the proper database driver using communication sub protocol. The first driver that recognizes a certain subprotocol under JDBC will be used to establish a database Connection.
- **Driver:** This interface handles the communications with the database server. You will interact directly with Driver objects very rarely. Instead, you use DriverManager objects, which manages objects of this type. It also abstracts the details associated with working with Driver objects.
- **Connection:** This interface with all methods for contacting a database. The connection object represents communication context, i.e., all communication with database is through connection object only.
- **Statement:** You use objects created from this interface to submit the SQL statements to the database. Some derived interfaces accept parameters in addition to executing stored procedures.
- **ResultSet:** These objects hold data retrieved from a database after you execute an SQL query using Statement objects. It acts as an iterator to allow you to move through its data.
- **SQLException:** This class handles any errors that occur in a database application.

684

There are following six steps involved in building a JDBC application:

- **Import the packages:** Requires that you include the packages containing the JDBC classes needed for database programming. Most often, using *import java.sql.** will suffice.
- **Register the JDBC driver:** Requires that you initialize a driver so you can open a communication channel with the database.
- **Open a connection:** Requires using the *DriverManager.getConnection()* method to create a Connection object, which represents a physical connection with the database.
- **Execute a query:** Requires using an object of type Statement for building and submitting an SQL statement to the database.
- **Extract data from result set:** Requires that you use the appropriate *ResultSet.getXXX()* method to retrieve the data from the result set.
- **Clean up the environment:** Requires explicitly closing all database resources versus relying on the JVM's garbage collection.

685

This sample example can serve as a **template** when you need to create your own JDBC application in the future.

```
1 //STEP 1. Import required packages
2 import java.sql.*;
3
4 public class FirstExample {
5     // JDBC driver name and database URL
6     static final String JDBC_DRIVER = "com.mysql.jdbc.Driver";
7     static final String DB_URL = "jdbc:mysql://localhost/EMP";
8
9     // Database credentials
10    static final String USER = "username";
11    static final String PASS = "password";
12
13    public static void main(String[] args) {
14        Connection conn = null;
15        Statement stmt = null;
```

686

```
16    try{
17        //STEP 2: Register JDBC driver
18        Class.forName("com.mysql.jdbc.Driver");
19
20        //STEP 3: Open a connection
21        System.out.println("Connecting to database...");
22        conn = DriverManager.getConnection(DB_URL,USER,PASS);
23
24        //STEP 4: Execute a query
25        System.out.println("Creating statement...");
26        stmt = conn.createStatement();
27        String sql;
28        sql = "SELECT id, first, last, age FROM Employees";
29        ResultSet rs = stmt.executeQuery(sql);
```

687

```

30         //STEP 5: Extract data from result set
31         while(rs.next()){
32             //Retrieve by column name
33             int id = rs.getInt("id");
34             int age = rs.getInt("age");
35             String first = rs.getString("first");
36             String last = rs.getString("last");
37
38             //Display values
39             System.out.print("ID: " + id);
40             System.out.print(", Age: " + age);
41             System.out.print(", First: " + first);
42             System.out.println(", Last: " + last);
43         }
44         //STEP 6: Clean-up environment
45         rs.close();
46         stmt.close();
47         conn.close();

```

688

```

48     }catch(SQLException se){
49         //Handle errors for JDBC
50         se.printStackTrace();
51     }catch(Exception e){
52         //Handle errors for Class.forName
53         e.printStackTrace();
54     }finally{
55         //finally block used to close resources
56         try{
57             if(stmt!=null)
58                 stmt.close();
59         }catch(SQLException se2){
60             }// nothing we can do
61         try{
62             if(conn!=null)
63                 conn.close();
64         }catch(SQLException se){
65             se.printStackTrace();
66         }//end finally try
67     }//end try
68     System.out.println("Goodbye!");
69 } //end main
70 } //end FirstExample

```

689

When you run **FirstExample**, it produces the following result

```
Connecting to database...
Creating statement...
ID: 100, Age: 18, First: Zara, Last: Ali
ID: 101, Age: 25, First: Mahnaz, Last: Fatma
ID: 102, Age: 30, First: Zaid, Last: Khan
ID: 103, Age: 28, First: Sumit, Last: Mittal
```

690

JDBC – Insert Records Example

```
try{
    //STEP 2: Register JDBC driver
    Class.forName("com.mysql.jdbc.Driver");

    //STEP 3: Open a connection
    System.out.println("Connecting to a selected database...");
    conn = DriverManager.getConnection(DB_URL, USER, PASS);
    System.out.println("Connected database successfully...");

    //STEP 4: Execute a query
    System.out.println("Inserting records into the table...");
    stmt = conn.createStatement();

    String sql = "INSERT INTO Registration VALUES (100, 'Zara', 'Ali', 18)";
    stmt.executeUpdate(sql);
    sql = "INSERT INTO Registration VALUES (101, 'Mahnaz', 'Fatma', 25)";
    stmt.executeUpdate(sql);
    sql = "INSERT INTO Registration VALUES (102, 'Zaid', 'Khan', 30)";
    stmt.executeUpdate(sql);
    sql = "INSERT INTO Registration VALUES(103, 'Sumit', 'Mittal', 28)";
    stmt.executeUpdate(sql);
    System.out.println("Inserted records into the table...");

}catch(SQLException se){
```

691

JDBC – Update Records Example

```

try{
    //STEP 2: Register JDBC driver
    Class.forName("com.mysql.jdbc.Driver");

    //STEP 3: Open a connection
    System.out.println("Connecting to a selected database...");
    conn = DriverManager.getConnection(DB_URL, USER, PASS);
    System.out.println("Connected database successfully...");

    //STEP 4: Execute a query
    System.out.println("Creating statement...");
    stmt = conn.createStatement();
    String sql = "UPDATE Registration "
                + "SET age = 30 WHERE id in (100, 101)";
    stmt.executeUpdate(sql);

    // Now you can extract all the records
    // to see the updated records
    sql = "SELECT id, first, last, age FROM Registration";
    ResultSet rs = stmt.executeQuery(sql);
}

```

692

```

while(rs.next()){
    //Retrieve by column name
    int id   = rs.getInt("id");
    int age  = rs.getInt("age");
    String first = rs.getString("first");
    String last = rs.getString("last");

    //Display values
    System.out.print("ID: " + id);
    System.out.print(", Age: " + age);
    System.out.print(", First: " + first);
    System.out.println(", Last: " + last);
}
rs.close();
}catch(SQLException se){
}

```

693

```

Connecting to database...
Creating statement...
ID: 100, Age: 18, First: Zara, Last: Ali
ID: 101, Age: 25, First: Mahnaz, Last: Fatma
ID: 102, Age: 30, First: Zaid, Last: Khan
ID: 103, Age: 28, First: Sumit, Last: Mittal

```



```

Connecting to a selected database...
Connected database successfully...
Creating statement...
ID: 100, Age: 30, First: Zara, Last: Ali
ID: 101, Age: 30, First: Mahnaz, Last: Fatma
ID: 102, Age: 30, First: Zaid, Last: Khan
ID: 103, Age: 28, First: Sumit, Last: Mittal

```

694

JDBC – Delete Records Example

```

try{
    //STEP 2: Register JDBC driver
    Class.forName("com.mysql.jdbc.Driver");

    //STEP 3: Open a connection
    System.out.println("Connecting to a selected database...");
    conn = DriverManager.getConnection(DB_URL, USER, PASS);
    System.out.println("Connected database successfully...");

    //STEP 4: Execute a query
    System.out.println("Creating statement...");
    stmt = conn.createStatement();
    String sql = "DELETE FROM Registration " +
                "WHERE id = 101";
    stmt.executeUpdate(sql);

    // Now you can extract all the records
    // to see the remaining records
    sql = "SELECT id, first, last, age FROM Registration";
    ResultSet rs = stmt.executeQuery(sql);
    // ...
}

```

695

```

Connecting to a selected database...
Connected database successfully...
Creating statement...
ID: 100, Age: 30, First: Zara, Last: Ali
ID: 101, Age: 30, First: Mahnaz, Last: Fatma
ID: 102, Age: 30, First: Zaid, Last: Khan
ID: 103, Age: 28, First: Sumit, Last: Mittal

```



```

Connecting to a selected database...
Connected database successfully...
Creating statement...
ID: 100, Age: 30, First: Zara, Last: Ali
ID: 102, Age: 30, First: Zaid, Last: Khan
ID: 103, Age: 28, First: Sumit, Last: Mittal

```

696

JDBC – Transactions

If your JDBC Connection is in *auto-commit* mode, which it is by default, then every SQL statement is committed to the database upon its completion.

```

try{
    //Assume a valid connection object conn
    conn.setAutoCommit(false);
    Statement stmt = conn.createStatement();

    String SQL = "INSERT INTO Employees  " +
                 "VALUES (106, 20, 'Rita', 'Tez')";
    stmt.executeUpdate(SQL);
    //Submit a malformed SQL statement that breaks
    String SQL = "INSERTED IN Employees  " +
                 "VALUES (107, 22, 'Sita', 'Singh')";
    stmt.executeUpdate(SQL);
    // If there is no error.
    conn.commit();
}catch(SQLException se){
    // If there is any error.
    conn.rollback();
}

```

697

```

try{
    //Assume a valid connection object conn
    conn.setAutoCommit(false);
    Statement stmt = conn.createStatement();

    //set a Savepoint
    Savepoint savepoint1 = conn.setSavepoint("Savepoint1");
    String SQL = "INSERT INTO Employees " +
                 "VALUES (106, 20, 'Rita', 'Tez')";
    stmt.executeUpdate(SQL);
    //Submit a malformed SQL statement that breaks
    String SQL = "INSERTED IN Employees " +
                 "VALUES (107, 22, 'Sita', 'Tez')";
    stmt.executeUpdate(SQL);
    // If there is no error, commit the changes.
    conn.commit();

}catch(SQLException se){
    // If there is any error.
    conn.rollback(savepoint1);
}

```

698

The PreparedStatement Objects

This statement gives you the flexibility of supplying arguments dynamically.

```

PreparedStatement pstmt = null;
try {
    String SQL = "Update Employees SET age = ? WHERE id = ?";
    pstmt = conn.prepareStatement(SQL);
    ...
}
catch (SQLException e) {
    ...
}
finally {
    ...
}

```

699

- All parameters in JDBC are represented by the ? symbol, which is known as the parameter marker. You must supply values for every parameter before executing the SQL statement.
- The **setXXX()** methods bind values to the parameters, where XXX represents the Java data type of the value you wish to bind to the input parameter. If you forget to supply the values, you will receive an SQLException.
- Each parameter marker is referred by its ordinal position. The first marker represents position 1, the next position 2, and so forth. This method differs from that of Java array indices, which starts at 0.
- All of the **Statement object's** methods for interacting with the database (a) execute(), (b) executeQuery(), and (c) executeUpdate() also work with the PreparedStatement object. However, the methods are modified to use SQL statements that can input the parameters.

700

The CallableStatement Objects

Just as a Connection object creates the Statement and PreparedStatement objects, it also creates the CallableStatement object, which would be used to execute a call to a database stored procedure.

```
CallableStatement cstmt = null;
try {
    String SQL = "{call getEmpName (?, ?)}";
    cstmt = conn.prepareCall (SQL);
    . .
}
catch (SQLException e) {
    . .
}
finally {
    . .
}
```

701

- The String variable SQL, represents the stored procedure, with parameter placeholders.
- Using the CallableStatement objects is much like using the PreparedStatement objects. You must bind values to all the parameters before executing the statement, or you will receive an SQLException.
- If you have IN parameters, just follow the same rules and techniques that apply to a PreparedStatement object; use the setXXX() method that corresponds to the Java data type you are binding.
- When you use OUT and INOUT parameters you must employ an additional CallableStatement method, registerOutParameter(). The registerOutParameter() method binds the JDBC data type, to the data type that the stored procedure is expected to return.
- Once you call your stored procedure, you retrieve the value from the OUT parameter with the appropriate getXXX() method. This method casts the retrieved value of SQL type to a Java data type.

702

```

try{
    //STEP 2: Register JDBC driver
    Class.forName("com.mysql.jdbc.Driver");

    //STEP 3: Open a connection
    System.out.println("Connecting to database...");
    conn = DriverManager.getConnection(DB_URL,USER,PASS);

    //STEP 4: Execute a query
    System.out.println("Creating statement...");
    String sql = "{call getEmpName (?, ?)}";
    stmt = conn.prepareCall(sql);

    //Bind IN parameter first, then bind OUT parameter
    int empID = 102;
    stmt.setInt(1, empID); // This would set ID as 102
    // Because second parameter is OUT so register it
    stmt.registerOutParameter(2, java.sql.Types.VARCHAR);
}

```

703

```

//Use execute method to run stored procedure.
System.out.println("Executing stored procedure..." );
stmt.execute();

//Retrieve employee name with getXXX method
String empName = stmt.getString(2);
System.out.println("Emp Name with ID:" +
empID + " is " + empName);
stmt.close();
conn.close();
}catch(SQLException se){

```

704

Servlets are Java classes which service HTTP requests and implement the **javax.servlet.Servlet** interface. Web application developers typically write servlets that extend javax.servlet.http.HttpServlet, an abstract class that implements the Servlet interface and is specially designed to handle HTTP requests.

```

1 // Import required java libraries
2 import java.io.*;
3 import javax.servlet.*;
4 import javax.servlet.http.*;
5
6 // Extend HttpServlet class
7 public class HelloWorld extends HttpServlet {
8
9     private String message;
10
11    public void init() throws ServletException {
12        // Do required initialization
13        message = "Hello World";
14    }

```

705

```

16 public void doGet(HttpServletRequest request, HttpServletResponse response)
17     throws ServletException, IOException {
18
19     // Set response content type
20     response.setContentType("text/html");
21
22     // Actual logic goes here.
23     PrintWriter out = response.getWriter();
24     out.println("<h1>" + message + "</h1>");
25 }
26
27 public void destroy() {
28     // do nothing.
29 }
30 }
```

706

Here is a simple example which passes two values using HTML FORM and submit button. We are going to use Servlet HelloForm to handle this input.

```

1 <html>
2   <body>
3     <form action = "HelloForm" method = "GET">
4       First Name: <input type = "text" name = "first_name">
5       <br />
6       Last Name: <input type = "text" name = "last_name" />
7       <input type = "submit" value = "Submit" />
8     </form>
9   </body>
10 </html>
```

First Name: Last Name:

707

```

1 // Method to handle GET method request.
2   public void doGet(HttpServletRequest request, HttpServletResponse response)
3     throws ServletException, IOException {
4
5     // Set response content type
6     response.setContentType("text/html");
7
8     PrintWriter out = response.getWriter();
9     String title = "Using GET Method to Read Form Data";
10    String docType =
11        "<!doctype html public \"-//w3c//dtd html 4.0 \" +
12        "transitional//en\\>\\n";

```

708

```

14      out.println(docType +
15          "<html>\\n" +
16          "  <head><title>" + title + "</title></head>\\n" +
17          "  <body bgcolor = \"#f0f0f0\\>\\n" +
18          "    <h1 align = \"center\\>" + title + "</h1>\\n" +
19          "    <ul>\\n" +
20          "      <li><b>First Name</b>: " +
21          "        + request.getParameter("first_name") + "\\n" +
22          "      <li><b>Last Name</b>: " +
23          "        + request.getParameter("last_name") + "\\n" +
24          "    </ul>\\n" +
25          "  </body>" +
26          "</html>" +
27      );
28  }
29
30 // Method to handle POST method request.
31   public void doPost(HttpServletRequest request, HttpServletResponse response)
32     throws ServletException, IOException {
33
34     doGet(request, response);
35  }

```

709

Using GET Method to Read Form Data

- **First Name:** ZARA
- **Last Name:** ALI

710

PYTHON DATABASE APPLICATION

711

PyMySQL is an interface for connecting to a MySQL database server from Python. It implements the Python Database API v2.0 and contains a pure-Python MySQL client library. The goal of PyMySQL is to be a drop-in replacement for MySQLdb.

Before proceeding further, you make sure you have PyMySQL installed on your machine. Just type the following in your Python script and execute it:

```
import pymysql
```

The last stable release is available on PyPI and can be installed with pip:

```
pip install pymysql
```

712

Following is an example of connecting with MySQL database "TESTDB"

```
1 import pymysql
2
3 # Open database connection
4 db = pymysql.connect("localhost","testuser","test123","TESTDB" )
5
6 # prepare a cursor object using cursor() method
7 cursor = db.cursor()
8
9 # execute SQL query using execute() method.
10 cursor.execute("SELECT VERSION()")
11
12 # Fetch a single row using fetchone() method.
13 data = cursor.fetchone()
14 print ("Database version : %s" % data)
15
16 # disconnect from server
17 db.close()
```

713

Once a database connection is established, we are ready to create tables or records into the database tables using **execute** method of the created cursor.

```

1 import pymysql
2
3 # Open database connection
4 db = pymysql.connect("localhost","testuser","test123","TESTDB" )
5
6 # prepare a cursor object using cursor() method
7 cursor = db.cursor()
8
9 # Drop table if it already exist using execute() method.
10 cursor.execute("DROP TABLE IF EXISTS EMPLOYEE")
11
12 # Create table as per requirement
13 sql = """CREATE TABLE EMPLOYEE (
14     FIRST_NAME  CHAR(20) NOT NULL,
15     LAST_NAME   CHAR(20),
16     AGE INT,
17     SEX CHAR(1),
18     INCOME FLOAT )"""
19
20 cursor.execute(sql)
21
22 # disconnect from server
23 db.close()
```

714

The INSERT Operation is required when you want to create your records into a database table.

```

1 import pymysql
2
3 # Open database connection
4 db = pymysql.connect("localhost","testuser","test123","TESTDB" )
5
6 # prepare a cursor object using cursor() method
7 cursor = db.cursor()
8
9 # Prepare SQL query to INSERT a record into the database.
10 sql = """INSERT INTO EMPLOYEE(FIRST_NAME,
11     LAST_NAME, AGE, SEX, INCOME)
12     VALUES ('Mac', 'Mohan', 20, 'M', 2000)"""
13 try:
14     # Execute the SQL command
15     cursor.execute(sql)
16     # Commit your changes in the database
17     db.commit()
18 except:
19     # Rollback in case there is any error
20     db.rollback()
21
22 # disconnect from server
23 db.close()
```

715

```

1 import pymysql
2
3 # Open database connection
4 db = pymysql.connect("localhost","testuser","test123","TESTDB" )
5
6 # prepare a cursor object using cursor() method
7 cursor = db.cursor()
8
9 # Prepare SQL query to INSERT a record into the database.
10 sql = "INSERT INTO EMPLOYEE(FIRST_NAME, \
11     LAST_NAME, AGE, SEX, INCOME) \
12     VALUES ('%s', '%s', '%d', '%c', '%d' )% \
13     ('Mac', 'Mohan', 20, 'M', 2000)
14 try:
15     # Execute the SQL command
16     cursor.execute(sql)
17     # Commit your changes in the database
18     db.commit()
19 except:
20     # Rollback in case there is any error
21     db.rollback()
22
23 # disconnect from server
24 db.close()

```

716

Once the database connection is established, you are ready to make a query into this database. You can use either **fetchone()** method to fetch a single record or **fetchall()** method to fetch multiple values from a database table.

- **fetchone()** – It fetches the next row of a query result set. A result set is an object that is returned when a cursor object is used to query a table.
- **fetchall()** – It fetches all the rows in a result set. If some rows have already been extracted from the result set, then it retrieves the remaining rows from the result set.
- **rowcount** – This is a read-only attribute and returns the number of rows that were affected by an execute() method.

717

```

9 # Prepare SQL query to INSERT a record into the database.
10 sql = "SELECT * FROM EMPLOYEE \
11      WHERE INCOME > '%d'" % (1000)
12 try:
13     # Execute the SQL command
14     cursor.execute(sql)
15     # Fetch all the rows in a list of lists.
16     results = cursor.fetchall()
17     for row in results:
18         fname = row[0]
19         lname = row[1]
20         age = row[2]
21         sex = row[3]
22         income = row[4]
23         # Now print fetched result
24         print ("fname = %s,lname = %s,age = %d,sex = %s,income = %d" % \
25               (fname, lname, age, sex, income ))
26 except:
27     print ("Error: unable to fetch data")
28
29 # disconnect from server
30 db.close()

```

718

UPDATE Operation on any database means to update one or more records, which are already available in the database.

```

1 import pymysql
2
3 # Open database connection
4 db = pymysql.connect("localhost","testuser","test123","TESTDB" )
5
6 # prepare a cursor object using cursor() method
7 cursor = db.cursor()
8
9 # Prepare SQL query to UPDATE required records
10 sql = "UPDATE EMPLOYEE SET AGE = AGE + 1
11          WHERE SEX = '%c'" % ('M')
12 try:
13     # Execute the SQL command
14     cursor.execute(sql)
15     # Commit your changes in the database
16     db.commit()
17 except:
18     # Rollback in case there is any error
19     db.rollback()
20
21 # disconnect from server
22 db.close()

```

719

DELETE operation is required when you want to delete some records from your database.

```

1 import pymysql
2
3 # Open database connection
4 db = pymysql.connect("localhost","testuser","test123","TESTDB" )
5
6 # prepare a cursor object using cursor() method
7 cursor = db.cursor()
8
9 # Prepare SQL query to DELETE required records
10 sql = "DELETE FROM EMPLOYEE WHERE AGE > '%d'" % (20)
11 try:
12     # Execute the SQL command
13     cursor.execute(sql)
14     # Commit your changes in the database
15     db.commit()
16 except:
17     # Rollback in case there is any error
18     db.rollback()
19
20 # disconnect from server
21 db.close()
```

720

Performing Transactions

```

# Prepare SQL query to DELETE required records
sql = "DELETE FROM EMPLOYEE WHERE AGE > '%d'" % (20)
try:
    # Execute the SQL command
    cursor.execute(sql)
    # Commit your changes in the database
    db.commit()
except:
    # Rollback in case there is any error
    db.rollback()
```

721

Python provides various options for developing graphical user interfaces (GUIs). The most important features are listed below:

- **Tkinter** – Tkinter is the Python interface to the Tk GUI toolkit shipped with Python. We would look this option in this chapter.
- **wxPython** – This is an open-source Python interface for wxWidgets GUI toolkit. You can find a complete tutorial on WxPython [here](#).
- **PyQt** – This is also a Python interface for a popular cross-platform Qt GUI library. TutorialsPoint has a very good tutorial on PyQt [here](#).
- **JPython** – JPython is a Python port for Java, which gives Python scripts seamless access to the Java class libraries on the local machine <http://www.jython.org>.

722

Tkinter is the standard GUI library for Python. Python when combined with Tkinter provides a fast and easy way to create GUI applications. Tkinter provides a powerful object-oriented interface to the Tk GUI toolkit.

- Creating a GUI application using Tkinter is an easy task. All you need to do is perform the following steps –
- Import the *Tkinter* module.
- Create the GUI application main window.
- Add one or more of the above-mentioned widgets to the GUI application.
- Enter the main event loop to take action against each event triggered by the user.

723

```
import tkinter # note that module name has changed from Tkinter in Python 2 to tkinter in Python 3
top = tkinter.Tk()
# Code to add widgets will go here...
top.mainloop()
```



724

The Common Gateway Interface, or CGI, is a set of standards that define how information is exchanged between the web server and a custom script. The CGI specs are currently maintained by the NCSA.

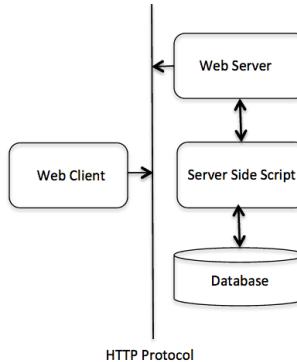
- The Common Gateway Interface, or CGI, is a standard for external gateway programs to interface with information servers such as HTTP servers.
- The current version is CGI/1.1 and CGI/1.2 is under progress.

To understand the concept of CGI, let us see what happens when we click a hyper link to browse a particular web page or URL:

- Your browser contacts the HTTP web server and demands for the URL, i.e., filename.
- Web Server parses the URL and looks for the filename. If it finds that file then sends it back to the browser, otherwise sends an error message indicating that you requested a wrong file.
- Web browser takes response from web server and displays either the received file or error message.

725

However, it is possible to set up the HTTP server so that whenever a file in a certain directory is requested that file is not sent back; instead it is executed as a program, and whatever that program outputs is sent back for your browser to display. This function is called the Common Gateway Interface or CGI and the programs are called CGI scripts. These CGI programs can be a Python Script, PERL Script, Shell Script, C or C++ program, etc.



726

Simple FORM Example

```

1 <form action = "/cgi-bin/hello_get.py" method = "get">
2   First Name: <input type = "text" name = "first_name"> <br />
3
4   Last Name: <input type = "text" name = "last_name" />
5   <input type = "submit" value = "Submit" />
6 </form>
  
```

First Name:	<input type="text"/>
Last Name:	<input type="text"/>
	<input type="button" value="Submit"/>

727

```

1 # Import modules for CGI handling
2 import cgi, cgitb
3
4 # Create instance of FieldStorage
5 form = cgi.FieldStorage()
6
7 # Get data from fields
8 first_name = form.getvalue('first_name')
9 last_name = form.getvalue('last_name')
10
11 print ("Content-type:text/html\r\n\r\n")
12 print ("<html>")
13 print ("<head>")
14 print ("<title>Hello - Second CGI Program</title>")
15 print ("</head>")
16 print ("<body>")
17 print ("<h2>Hello %s %s</h2>" % (first_name, last_name))
18 print ("</body>")
19 print ("</html>")

```

728

For more materials on database application development:

PHP & MySQL Tutorial

https://www.tutorialspoint.com/php_mysql/index.htm

Java JDBC Tutorial

<https://www.tutorialspoint.com/jdbc/index.htm>

Java Servlets Tutorial

<https://www.tutorialspoint.com/servlets/index.htm>

Python Database Access

https://www.tutorialspoint.com/python3/python_database_access.htm

Python CGI Programming

https://www.tutorialspoint.com/python3/python_cgi_programming.htm

Python GUI Programming

https://www.tutorialspoint.com/python3/python_gui_programming.htm

729

Extra on MySQL Programming Interfaces

MySQL & PHP

<https://www.mysqltutorial.org/php-mysql/>

MySQL & Node.js

<https://www.mysqltutorial.org/mysql-nodejs/>

MySQL & Java

<https://www.mysqltutorial.org/mysql-jdbc-tutorial/>

MySQL & Python

<https://www.mysqltutorial.org/python-mysql/>

730

Контрольні питання

1. Методи HTTP GET та POST
2. Функція PHP mysqli_connect()
3. Функції PHP mysqli_select_db() та mysql_query()
4. Функція PHP mysql_fetch_array()
5. PHP Data Objects (PDO)
6. Основні класи та інтерфейси JDBC API
7. Управління транзакціями засобами JDBC
8. Об'єкти PreparedStatement та CallableStatement
9. Сервлети у Java
10. Функції pumysql connect() та cursor()
11. Функції pumysql execute() та fetchall()
12. Управління транзакціями засобами pumysql
13. Реалізація інтерфейсу користувача у Python

731

Assessment questions

1. HTTP GET and POST methods
2. PHP function mysqli_connect()
3. PHP functions mysqli_select_db() and mysql_query()
4. PHP function mysql_fetch_array()
5. PHP Data Objects (PDO)
6. Basic JDBC API classes and interfaces
7. Transaction management by means of JDBC
8. PreparedStatement and CallableStatement objects
9. Servlets in Java
10. Pymysql connect() and cursor() functions
11. Pymysql execute() and fetchall() functions
12. Transaction management using pymysql
13. Implementation of the user interface in Python

732

Контрольные вопросы

1. Методы HTTP GET и POST
2. Функция PHP mysqli_connect()
3. Функции PHP mysqli_select_db() и mysql_query()
4. Функция PHP mysql_fetch_array()
5. PHP Data Objects (PDO)
6. Основные классы и интерфейсы JDBC API
7. Управление транзакциями средствами JDBC
8. Объекты PreparedStatement и CallableStatement
9. Сервлеты в Java
10. Функции pymysql connec() и cursor()
11. Функции pymysql execute() и fetchall()
12. Управление транзакциями средствами pymysql
13. Реализация интерфейса в Python