Prince Sefa

JUSTIT SKILLS BOOTCAMP |

**Assignment 2: Databases & MySQL**

Table of Contents

[Task 1: List the different types of relationships in relational databases and provide examples. 3](#_Toc149487632)

[Task 2: What is Normalization and why is it important to database development? 5](#_Toc149487633)

[Task 3: Using count, get the number of cities in the USA 7](#_Toc149487634)

[Task 4: Population and life expectancy for people in Argentina (ARG) 7](#_Toc149487635)

[Task 5: Using ORDER BY, LIMIT, what country has the highest life expectancy? 7](#_Toc149487636)

[Task 6: Select 25 cities around the world that start with the letter 'F' in a single SQL query 7](#_Toc149487637)

[Task 7: Create a SQL statement to display columns Id, Name, Population from the city table and limit results to the first 10 rows only. 8](#_Toc149487638)

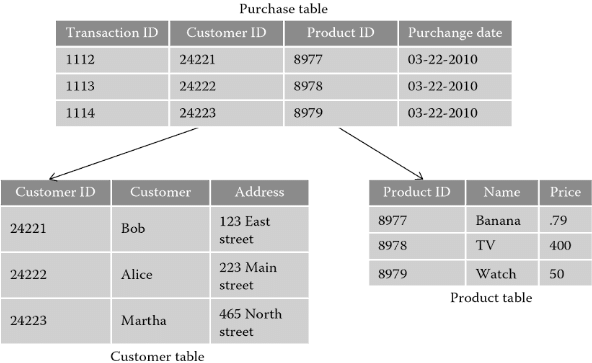
[Task 8: Create a SQL statement to find only those cities from city table whose population is larger than 2000000 8](#_Toc149487639)

[Task 9: Create a SQL statement to find all city names from city table whose name begins with “Be” prefix. 8](#_Toc149487640)

[Task 10: Create a SQL statement to find only those cities from city table whose population is between 500000-and 1000000. 8](#_Toc149487641)

# Task 1: List the different types of relationships in relational databases and provide examples.

Relational databases organize data into rows and columns that come together to form tables. Data is typically structured in multiple tables that can be linked using “primary keys” or “foreign keys.” These “keys” are unique identifiers that indicate different relationships that exist between tables, and these relationships are typically represented by different types of data models. The relationship between database tables is described with the term “relation”. Analysts use Structured Query Language (SQL) queries to combine various data points and summarize business performance by being able to extract information and trends rapidly and directly.

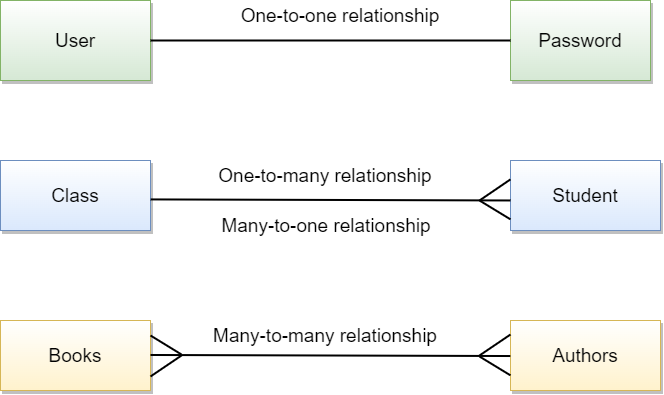
* Fig.1.*

*Diagram showing an example of a relational database where the attributes from the “Purchase table” (CustomerID and ProductID) can be extracted and split into 2 separate tables which have their own headings or entities as the Primary Keys*

From this businesses can gain insights, optimize workflows, and identify new opportunities. For example, imagine a company that maintains a customer information database table that contains account-level company data. There may also be another table that describes all the individual transactions associated with this account. Combined, these tables provide information about the different industries that purchase a particular software product. Columns (or fields) in the Customer table can include customer ID, company name, company address, industry type, and so on. Columns in a transaction table can include transaction date, customer ID, transaction amount, payment method, and more. Tables can be connected using a common "Customer ID" field. Therefore, you can query the table and create valuable reports such as sales reports- by industry or company. The resulting report can serve as a source of information for sending messages to potential customers.

The data presented in relational databases must conform to certain parameters: The data must be based on a relational model; each row in the table must have its own unique Identification “key”; and the columns of the table hold the attributes of data. With this, there are 3 main types of relationships which can be described in a relational DB:

* one-to-one
* one-to-many
* many-to-many.

A “many-to-one” relationship is a special case of a one-to-many relationship and self-referencing relationship which occurs when only one table is involved.

A **one-to-many relationship** is the most commonly type of relationship. A one-to-many relationship occurs when one record in Table 1 is related to one or more records in Table 2. However, one record in Table 2 cannot be related to more than one record in Table 1. Examples of this include: pages and the book they belong to, pupils and their class, orders and the customer who placed them, etc.

A **one-to-one** **relationship** in a database occurs when each row in Table 1 has only one related row in Table 2. For example, a department may have only one head manager, a husband — only one wife, an employee — one company car, etc.

A **many-to-many** relationship occurs when multiple records in one table are related to multiple records in another table. An example of this is with products and suppliers: one supplier may deliver one or many products and at the same time, the company may order one product from one or many suppliers.

# Task 2: What is Normalization and why is it important to database development?

As businesses collect, store, and analyse data, they create and use databases to process data. The data that’s collected via businesses is usually funnelled in from multiple systems and endpoints which all have varying structures and forms which can be difficult to extract insights, analyse and present.

Data normalization is the process of reorganizing data in a database so that it is available to users for further querying and analysis. In other words, it is the process of developing clean data. This includes eliminating “redundant” unstructured data and ensuring that data looks the same across all records and fields. Data normalization is critical to business operations, and a thorough understanding of data normalization can give any business a real advantage in leveraging big data to continue to grow. This depends on how well the data is arranged inside a database, with the goal being to ensure that data is similar across all records.

Normalization requires creating tables and joining these tables according to principles aimed at protecting data and making the database more adaptable by eliminating duplicates and inconsistent dependencies. Redundant data wastes disk space and creates maintenance issues. If you need to change data that already exists in multiple locations, you must update it the same way everywhere. Changing a customer's address is much easier if the information is stored only in the Customers table and not elsewhere in the database. At its most basic, normalization is simply creating a standard format for all data throughout a company. Examples of this being:

“Miss EMILY will be written in Ms. Emily”

“8023097864” will be written “802-309-7864”

“24 canillas RD” will be written “24 Canillas Road”

“GoogleBiz” will be written “Google Biz, Inc.”

“VP marketing” will be written “Vice President of Marketing”

There are a number of reasons why database normalisation is integral in the development of efficient and accurate databases. It is a necessary process for maintaining data integrity and creating a single source of truth. Furthermore, data normalization aims to eliminate data redundancy, which occurs when there are multiple fields with duplicate information. Eliminating redundancy can make a database more flexible which can subsequently allow databases to grow and scale.

As well as this, the impact of normalised data on effective communication, decision-making and overall information usability cannot be understated as messy and unnormalised data poses challenges that can hinder human and machine comprehension. Unorganised data within a database negatively impacts the functionality of features and functions. For example, searching for "Google Chrome" may not yield the same results as searching for "Chrome" due to inconsistencies in data representation. This undermines the accuracy and efficiency of searching and querying processes, leading to less precise results. So, normalization makes searching for specific terms or entities more efficient and accurate. Connections between related data elements are strengthened, enabling improved information retrieval and analysis.

Additionally, this practice has far-reaching implications for database functionality and performance. Normalized data is easier to sort, filter, and analyse, making data exploration and pattern recognition easier. With fewer columns and improved organization, users can now see more records on a single page, improving visualization, understanding, and pattern recognition. In terms of user experience, processes are simplified, allowing users to easily access and modify information while ensuring consistency, accuracy, and lack of duplication or redundancy. This means that multiple users can safely read, interpret, and trust the presented dataset. Data normalization is necessary for online transaction processing (OLTP), where data quality and discoverability are top priorities.

In summary, the main benefits normalisation brings are:

* Data is easier to sort through - Normalized data is easy to handle, facilitating your agents’ work.
* More storage space available - When you’re dealing with terabytes and petabytes, data normalization can significantly optimize the storage space.
* Easier to work with data analysis tools - A normalized database can be smoothly connected to data processing tools to visualize and analyse it. Without standardization, these solutions don’t have accurate information to work with and can produce incorrect outputs.
* Better outputs - Cleaner and standardized data produces better results. And you need to be able to rely on that information to make good decisions.

In the real world, there may be instances where businesses that use big data may not always need to normalize. At times it makes sense to do the opposite and add redundancy to a database. Adding redundant data to a database or “denormalization”, by adding redundant data can sometimes improve application performance and data integrity. Denormalization can also help you achieve faster querying. By adding extra redundancy, you can sometimes find information faster. However, in a position to sacrifice quality for speed, it’s generally better to use denormalized data. It’s common to use denormalization in online analytical processing (OLAP) systems, where the goal is to streamline search and analysis.  Consequently, denormalization increases disk space because it allows for duplicate data. This, in turn, uses more memory, driving up operating costs.

Task 3: Using count, get the number of cities in the USA

*SELECT COUNT(\*) AS CityCount*

*FROM City*

*WHERE CountryCode = 'USA';*

=274

Task 4: Population and life expectancy for people in Argentina (ARG)

*Argentina (ARG), Population:* 37032000

*Argentina (ARG), Life expectancy:* 75.1

(Did not need to execute a specific script to find that out)

Task 5: Using ORDER BY, LIMIT, what country has the highest life expectancy?

*SELECT Name, LifeExpectancy*

*FROM Country*

*ORDER BY LifeExpectancy DESC*

*LIMIT 1;*

=Andorra: 83.5

Task 6: Select 25 cities around the world that start with the letter 'F' in a single SQL query

*SELECT Name*

*FROM City*

*WHERE Name start='F%'*

*LIMIT 25;*

Task 7: Create a SQL statement to display columns Id, Name, Population from the city table and limit results to the first 10 rows only.

*SELECT Id, Name, Population*

*FROM City*

*LIMIT 10;*

Task 8: Create a SQL statement to find only those cities from city table whose population is larger than 2000000

*SELECT \* FROM City*

*WHERE Population> 2000000*

Task 9: Create a SQL statement to find all city names from city table whose name begins with “Be” prefix.

*SELECT Name*

*FROM City*

*WHERE Name LIKE 'Be%';*

Task 10: Create a SQL statement to find only those cities from city table whose population is between 500000-and 1000000.

*SELECT Name*

*From City*

*WHERE population BETWEEN 500000 AND 1000000*

Task 11: Create a SQL statement to find a city with the lowest population in the city table.

*SELECT Name*

*FROM City*

*ORDER BY Population*

*LIMIT 1;*

Task 12: Create a SQL statement to show the population of Switzerland and all the languages spoken there.

*SELECT Name, Population*

*FROM Country*

*WHERE Country Name= Switzerland;*