Learner Guide

595783184 / Wright Studio / shutterstock.com

ICTPRG431

Apply query language in relational databases

tafe.qld.edu.au

MAKE GREAT HAPPEN

tafe Queensland

Unit version	Document version	Release date	Comments/actions
ICTPRG431	V2	May 2022	

Introduction

This Learner Guide has been developed to support you as a resource for your study program. It contains key information relating to your studies including all the skills and knowledge required to achieve competence.

What will I learn?

This unit describes the skills and knowledge required to retrieve and manipulate information stored in information systems using a query language.

It applies to those who are involved in a range of work environments, who are required to extract information from a relational database by creating and running queries.

No licensing, legislative or certification requirements apply to this unit at the time of publication.

Are there any special requirements?

Students will be required to be working in an Information and Communications Technology environment.

To complete this unit you will need access to:

- a lab computer
- internet access, access to Connect (LMS)
- database server with administration privileges – e. g., MariaDB server, DBMS software – e.g. MySQL, MariaDB, Microsoft SQL Server or similar relational database
- word processing software, such as Microsoft Word
- special purpose tools, equipment and materials to complete the assessment.

Students are required to provide their own storage device. The recommendation for this qualification is an external SSD drive with at least 256 GB capacity, if you need to store a copy of the Virtual Machine (VM). For assessment files only, a 64 GB thumb drive will be sufficient.

Where can I get more information?

For further information on your qualification, accredited course or Unit/s of Competency, please go to: https://training.gov.au/Training/Details/ICTPRG431

TAFE Queensland student rules

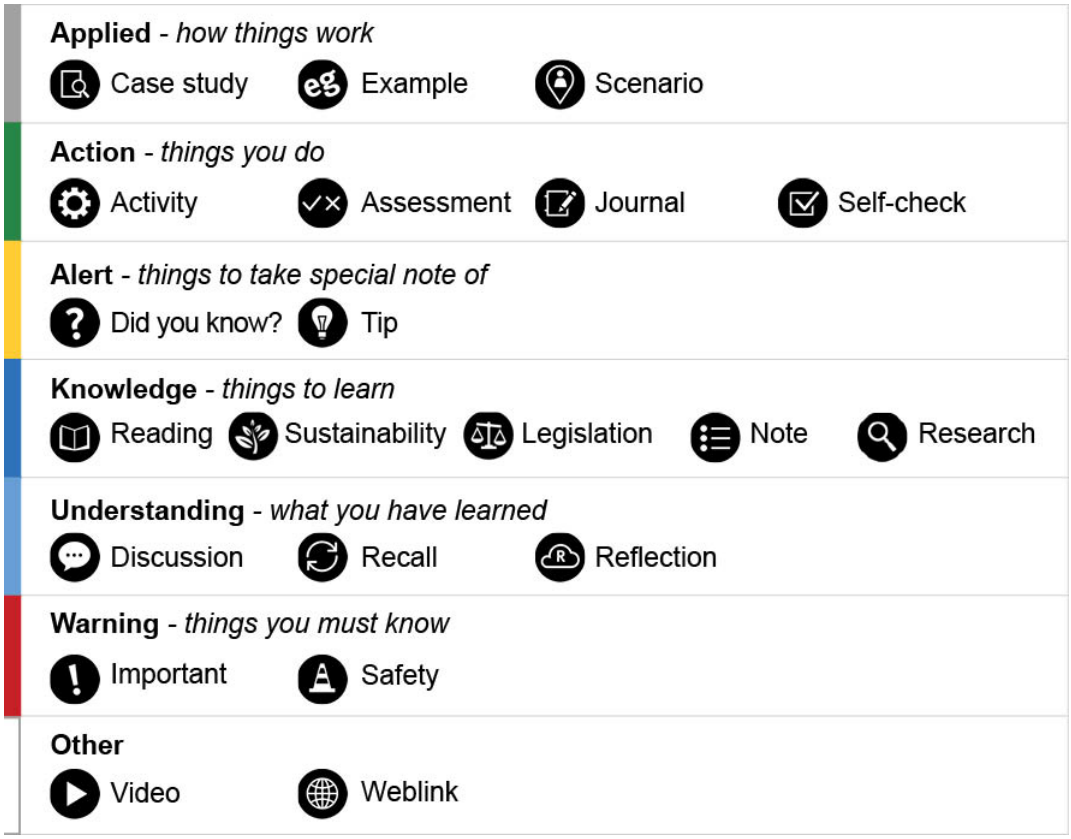
TAFE Queensland student rules are designed to ensure that learners are aware of their rights as well as their responsibilities. All learners are encouraged to familiarise themselves with the TAFE Queensland student rules, specifically as they relate to progress of study and assessment guidelines. A full guide to student rules can be found at Student rules[1].

Information to support your learning and assessment

There's always someone to help you. Undertaking further study can bring both excitement and challenges. Our Student Services, Learning Support and Library staff can help you make the most of your time at TAFE.

Callout panels

A number of panels have been designed to help guide you to important information and actions throughout this Learner Guide. The full choice of panels you are likely to encounter to support you in your studies are included below. NB: not all the panels will be used in every learner guide.



[1] http://tafeqld.edu.au/current-students/student-rules/

Contents

Welcome

Welcome to the *ICTPRG431 – Apply query language in relational databases*.

By completing this unit you will be able to:

- Determine requirements of developing queries
- Write queries to extract required information
- Perform calculations and use expressions in queries
- Create and manipulate tables
- Create and use views and stored procedures

Relational databases and structured query language (SQL)

Introduction

The purpose of this topic is to provide a general introduction to relational databases and structured query language (SQL). You will learn the brief history of relational databases and will also learn about common web software 'stacks' that make up the web development environment. Database structures are explored, as are various SQL variants such as MySQL and Microsoft SQL Server.

Objectives

By completing this unit, you will be able to do the following:

- identify terminologies applicable to query language
- identify and build queries using required tools and environment
- appreciate an historical perspective of databases
- understand how a web development environment is set up using software stacks
- understand the structures that make up a relational database
- appreciate different variations of SQL.

699634498 / REDPIXEL.PL / shutterstock.com

Database history

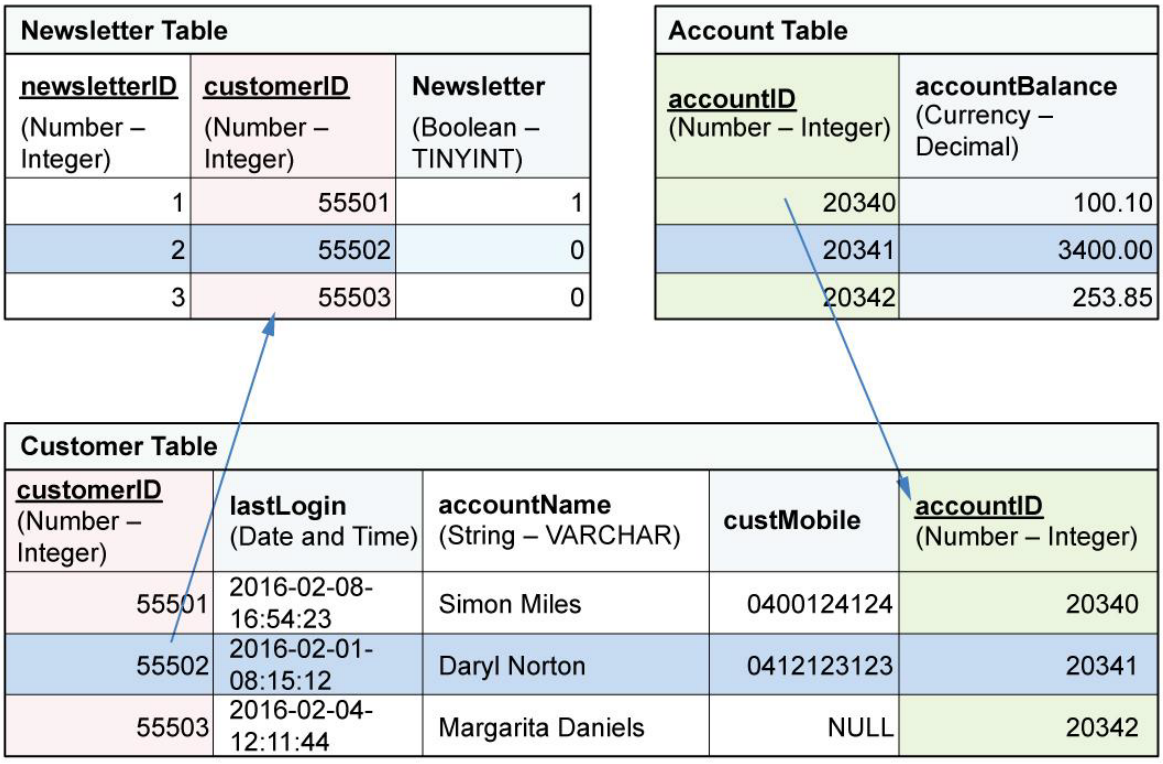
The relational database model was developed by an IBM research scientist, Dr Edgar F. Codd in the late 1960s. Codd was dissatisfied with existing database models, which are discussed further below (these are the hierarchical and network models).

As a mathematician, he developed an alternative database model that would be based on data sharing relations. The revised model was published in June 1970 in an academic paper titled: **A Relational Model of Data for Large Shared Banks.**

In this new model relations are represented as tables, which are created, manipulated and managed by database administrators. Each table contains a number of 'fields' (or columns). Rows of data are stored in a database table where each field for a given row holds a specific value.

Think of a database as a container. The container holds several tables. Data is stored in a table where each row in the table represents a single record. Manipulating a database consists of being able to access, update and retrieve the desired information from the database.

The following illustration explains this concept further by showing how multiple tables are related using the relational model.



Newsletter Table

newsletterID (Number – Integer)	customerID (Number – Integer)	Newsletter (Boolean – TINYINT)
1	55501	1
2	55502	0
3	55503	0

Account Table

accountID (Number – Integer)	accountBalance (Currency – Decimal)
20340	100.10
20341	3400.00
20342	253.85

Customer Table

customerID (Number – Integer)	lastLogin (Date and Time)	accountName (String – VARCHAR)	custMobile	accountID (Number – Integer)
55501	2016-02-08-16:54:23	Simon Miles	0400124124	20340
55502	2016-02-01-08:15:12	Daryl Norton	0412123123	20341
55503	2016-02-04-12:11:44	Margarita Daniels	NULL	20342

In the above example of three relational database tables, notice that all tables have been designed for a specific purpose and therefore contain columns that have been grouped together for that purpose.

The Newsletter table contains a Boolean value to indicate whether a customer has subscribed to the company newsletter.

Looking at the **Newsletter table**, you can see the same customer has subscribed to the newsletter (indicated by a 1). The unique customer ID (55501) links the Customer and Newsletter tables, establishing a relationship between the two tables.

The blue shading represents a row (or record). A ‘field’ is where the column and row insert (the light blue cell) and each ‘field’ holds a data value.

The **Account table** contains unique account IDs and account balances and the Customer table holds information specifically related to the customer.

The account ID column (shown with green shading) is defined in both the Account and Customer tables and is used to establish a relationship between both tables.

By looking up the corresponding account ID (20340), you can see that Simon Miles has an account balance of $100.10. Both tables contain unique IDs (numbers) that uniquely identify and link the record (or row) to the customer.

Note

Notice the 'NULL' value in the custMobile table field – this means that a value has not yet been set.

Tip

Because you can identify each record using unique IDs, you do not necessarily need to know the order in which the records are stored. Using these IDs, you can easily retrieve related information from multiple tables and present this information in many ways.

What is a relational database?

To understand what relational databases are, it is helpful to compare them with traditional flat file databases.

38800492 / mitya73 / shutterstock.com

What is a flat file database?

A common example of a flat file database is a spreadsheet.

There are several key differences between a traditional flat file database and a relational database:

Relational databases

Use special key fields, often referred to as primary and foreign keys. These define relationships between the tables in the database which enable us to retrieve corresponding data from multiple tables and then use this data in meaningful ways.

Relational databases avoid duplication of data by a process called normalisation where data is efficiently organised by implementing a set of rules referred to as normal forms.

Flat file databases

Store data as a single table while relational databases store data across multiple tables. Flat file databases often contain a lot of duplication and repetition of the same data, which can lead to search inefficiencies and corruption of data. Any changes to the data can lead to errors as some values may be missed during the update.

Several models were developed to improve upon the flat file model that existed before the relational database model. The **hierarchical database model** consisted of an upside-down tree structure with a root table at the top that branched out to other tables. The top-level table was also called the parent and the branch level tables are called child tables. These are defined in a one-to-many relationship, meaning that there is one parent table (e.g. Customer) that has many child tables associated with it (e.g. Account), and these child tables could also have a child table (e.g. Sub-Account).

To get to a lower-level table the user would need to know the structure of the hierarchy and work their way down the structure until they reached the records they were searching for.

The next step toward the relational database model was an extension of the hierarchical database model – the **network model**. The network model allowed the user to define child tables that have multiple parents (this was not possible with the hierarchical model). This resulted in the ability to support many-to-many relationships (a parent table could have many child tables, and a child table could have many parent tables). This model required a team of programmers to maintain (users of the database were dependent on the programmers to manage the database). To overcome this problem, the **relational database model** was developed.

Over the past several years, **NoSQL database interfaces** have increased in popularity. Where SQL databases store information in predefined data structures, called tables, NoSQL databases store information in documents that have no predefined schema (data structure).

Weblink

Visit the following websites for a comparison of the major differences between SQL and NoSQL databases:

Differences between SQL and NoSQL[2]

5 Critical Differences between SQL vs NoSQL[3]

When to Use, How to Choose the appropriate database[4]

[2] https://www.mongodb.com/nosql-explained/nosql-vs-sql

[3] https://www.integrate.io/blog/the-sql-vs-nosql-difference/

[4] https://towardsdatascience.com/datastore-choices-sql-vs-nosql-database-ebec24d56106

Data concepts and database systems

Information system

The purpose of an information system is to create a dynamic structure that allows data to be gathered, stored, retrieved, and managed to provide support for products and/or services.

Hardware

Software

Communication technologies

Databases and data warehouses

Human resources

Policies and procedures

730871755 / MicroOne, 1038711322 / elenabsl, 454439383 / VectorShow, 709156837 / hanss, 1318923254 / ChingizDs, 194378240 / madpixblue / shutterstock.com. Modified by TAFE Queensland.

In general, information systems require several sub-systems to organise and support the infrastructure, and although they may change from organisation to organisation, they may include a base system comprising the supply chain or technical background and the customer group, a knowledge support system, and a management system.

Data sources

Data can be initially classified as statistical and non-statistical data. The sources of data could be **internal** data collected by the organisation as part of their day-to-day operations or external data that is relevant to the organisation but that originates at external sources e.g. customer surveys, market research, competitors' analysis and so on. We can further classify data as primary or secondary data, which refers to the way data was collected.

Primary data is often collected for the first time, and it is expected to be original and reliable e.g. census data.

Secondary data is data that has been previously collected by other entities and is then passed on or published for further use.

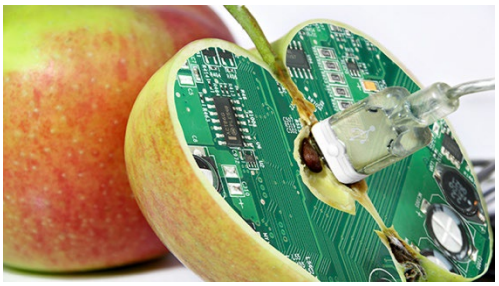
Data gathering methods

Data can be generated and gathered or collected in organisations from their daily transactions, operations and interactions or can be acquired from external sources.

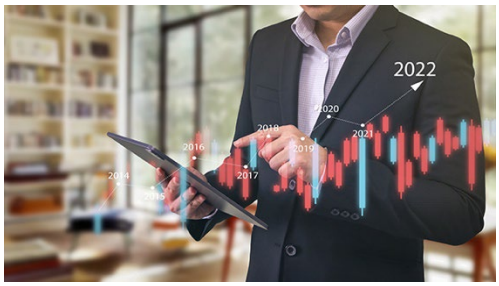
There are four main formal methods for data collection:

Observational Observing behaviour or activities, e.g., useability testing groups

Experimental Gathered by researcher to determine cause and effect



Simulations Simulating real world operations, e.g., weather prediction, economic models

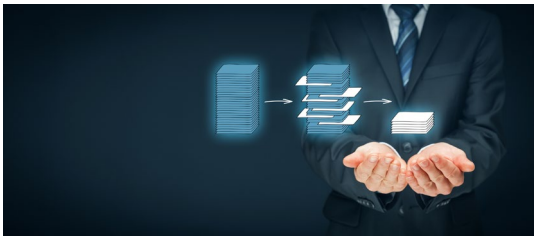


Derived from other sources Transformations of existing data

1936499986 / Gorodenkoff, 93233221 / YueStock, 2089943491 / TimmyTimTim, 1228926949 / Natali _ Mis / shutterstock.com

Data mining

Data mining has been carried out for a long time with the technology available at the time. Before the term data mining became popular in the 1990s it was known as Knowledge Discovery in Data (KDD).



593162990 / Jirsak / shutterstock.com.

Data mining is rooted in three scientific disciplines:

- Statistics
- Artificial intelligence
- Machine learning algorithms

It is the evolution of the last two disciplines that has contributed to make data mining an indispensable tool for businesses to analyse vast volumes of data to find anomalies, correlations and patterns that can be used to predict from market trends to consumer expectations and behaviour. However, data mining's main purpose is to find patterns in the data. Techniques used in data mining analysis include:

- regression
- association rule discovery
- classification
- clustering.

Weblink

Visit the following websites to learn more about Data Mining, particularly what Data Mining is and why it is important:

SAS Insights – What Data Mining is and why it matters[5]

Galvanize – 4 Data Mining techniques for businesses[6]

[5] https://www.sas.com/en_au/insights/analytics/data-mining.html

[6] https://blog.galvanize.com/four-data-mining-techniques-for-businesses-that-everyone-should-know/

Database development environment

In Database Management Systems (DBMS), the principal data source is a database, which is composed of several tables to store the data. You are going to concentrate on creating and working with a relational database management system (RDBMS) to use structured query language (SQL) for this competency unit.

501236956 / Profit_Image / shutterstock.com.

SQL standards

MySQL has added some extensions to current SQL standards, this means that if you use the extensions your code is not portable to other SQL servers.

Weblink

By visiting the following website, you can learn more about MySQL: Extensions to Standard SQL[7]

SQL conforms to the following standards:

ANSI/ISO/IEC 9075:2003	"Database Language SQL", Parts 1 ("SQL/Framework"), 2 ("SQL/Foundation"), 3 ("SQL/CLI"), 4 ("SQL/PSM"), 9 ("SQL/MED"), 10 ("SQL/OLB"), 11 ("SQL/Schemata"), 13 ("SQL/JRT") and 14 ("SQL/XML")
ISO/IEC 9075:2003	"Database Language SQL", Parts 1 ("SQL/Framework"), 2 ("SQL/Foundation"), 3 ("SQL/CLI"), 4 ("SQL/PSM"), 9 ("SQL/MED"), 10 ("SQL/OLB"), 11 ("SQL/Schemata"), 13 ("SQL/JRT") and 14 ("SQL/XML")

[7] https://dev.mysql.com/doc/refman/8.0/en/extensions-to-ansi.html

Which DBMS to use?

A popular solution is to use a free open-source solution for a DBMS. MariaDB (developed from MySQL) will be used in conjunction with **phpMyAdmin** as the DBMS will demonstrate the use of structured query language SQL within a relational database approach.

514021990 / LeoWolfert / shutterstock.com

Other open-source alternatives to phpMyAdmin are:

- MySQL Workbench
- Dbeaver
- AdMiner (formerly phpMinAdmin)
- Navicat.

Weblink

MariaDB is a replacement for MySQL. Oracle acquired MySQL in 2010 and MariaDB was launched to keep it open source.

Check the OpenSource Forum[8] website to gain an insight into MariaDB as well as MySQL.

The examples presented in this guide will use the phpMyAdmin administrative tool shipped with the XAMPP stack. The stack includes an easy to install self-configuring Apache distribution containing MariaDB, PHP, and Perl.

The latest stable version of XAMPP (at the time of writing this is version 8.011 – 8 Oct 2021).

Connecting to a DBMS

For this unit, you will learn how to work directly on the DBMS using phpMyAdmin (or similar administration tool) but in a different environment the counteraction with the database server might be via a HTML page. In all instances you would require, as a minimum the following:

- a host address or name
- an authentication method
- a user and password
- a database name to connect to.

[8] https://www.opensourceforu.com/2018/04/why-mariadb-scores-over-mysql/

Activity

For this activity students are encouraged to install XAMPP on their home computer/laptop so they can practice what is covered in class.

Once installed, access the XAMPP Control Panel and start the Apache and MySQL/MariaDB services.

To start Apache, select Apache > Start (the tab turns to green colour).

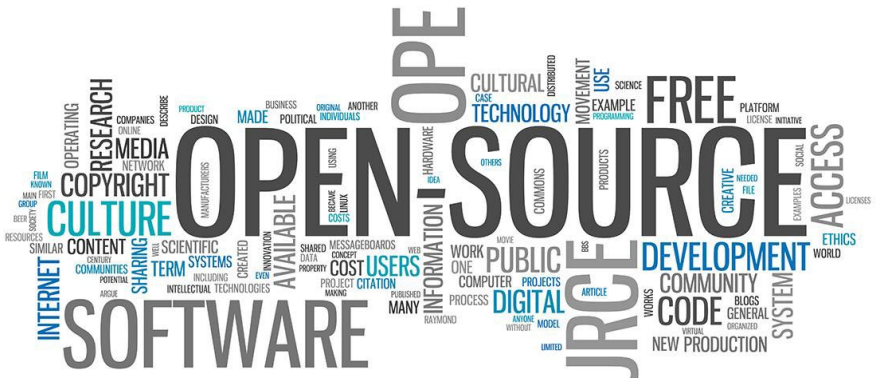
1. To start the MySQL/MariaDB service, select MySQL > Start > Admin.
2. By default the username is "root" and the password is blank.
3. The homepage for phpMyAdmin will open.
4. If the services are active (Apache and MySQL), you can access phpMyAdmin through the browser bar by typing **localhost/phpMyAdmin**.

Weblink

For more information, you can also visit the following websites:

XAMPP Windows installation[9]

XAMPP for Windows installation – Wikihow.com[10]



217859110 / mindscanner / shutterstock.com. Modified by TAFE Queensland.

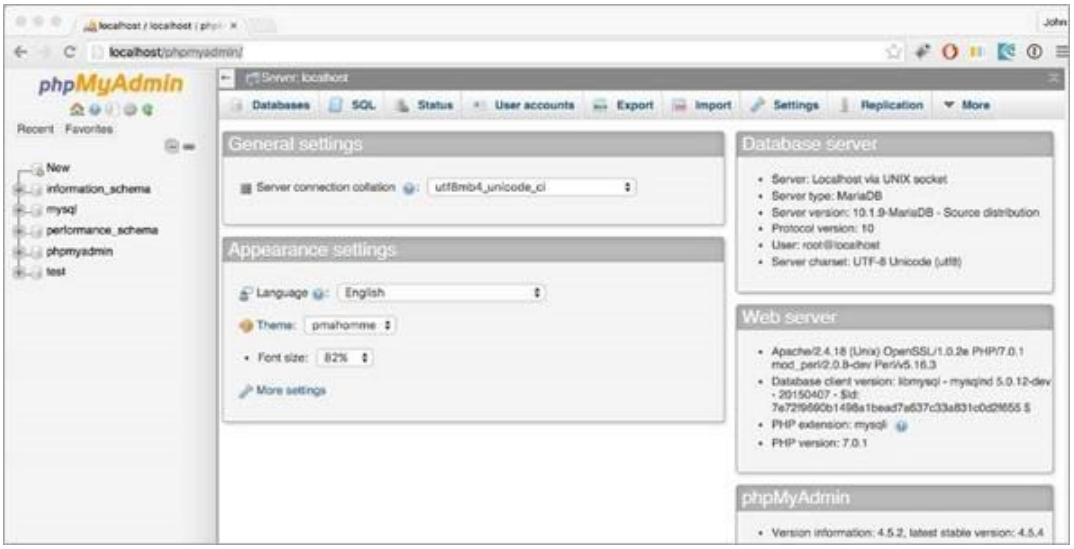
[9] https://www.apachefriends.org/faq_windows.html

[10] https://www.wikihow.com/Install-XAMPP-for-Windows

Using phpMyAdmin

You can access the database administration tool from the localhost by typing the following into a browser address bar:

- localhost/phpMyAdmin



Screenshot of phpMyAdmin software, 16/12/21.GNU General Public License, version 2

By accessing phpMyAdmin, you/the developer will be provided with administrator to the server access whereby you will have full privileges to all the administrative functions.

Using this feature-rich tool phpMyAdmin you can:

- Create and drop databases.
- Perform SQL queries to retrieve, change, add, or remove records (using SELECT, UPDATE, INSERT, DELETE respectively).
- Setup relationships between tables (using primary and foreign keys) to maintain referential integrity.
- Manage user accounts and set database privileges.
- Setup indexes on certain columns to optimise table performance during record retrieval.
- Export and import database.
- Add new database tables and define the columns within those tables, while ensuring data integrity by applying constraints to edit existing tables (for example, remove existing or add new columns).

Overview of client/server architecture

In the web development environment you are learning from here at TAFE, there is access to a local web server called localhost from which you run phpMyAdmin. The **localhost** is technically a web server locally hosted.

In a workplace environment within industry, you will find a distinct separation of client and server. For instance, you are highly likely to find that there are several servers setup specifically for the purposes of development and testing. Furthermore, there will typically be one or more production (or 'live') servers, which will be what end users access when interacting with a web-based application or website that is attached to the database.

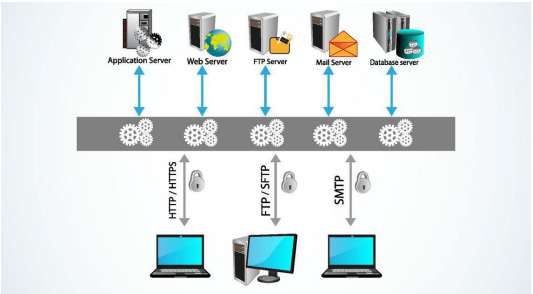
A client can therefore represent several types of organisational profiles:

- a database administrator maintaining the database server
- a computer programmer coding on a development server
- a website designer testing functionality of a new page on a testing server
- an end-user interacting with the live website on a production server
- a manager generating executive reports on a production server (back-end).

A client device could be a smart phone, tablet, laptop, or desktop computer. Depending on the organisational business requirements, server configuration can differ between organisations.

In the illustration above, servers are setup by function, handling the sending and receiving of data to and from the client devices they are interacting with. Clients interact with servers using several protocols:

- HTTP/HTTPS, Hypertext Transfer Protocol (Secure) - a protocol used to handle web requests. Using this protocol, a web server serves web pages.
- (S)FTP, (Secure) File Transfer Protocol - a protocol used to send and receive files to and from a client. Using this protocol, a file server serves files.
- SMTP, Simple Mail Transfer Protocol - a protocol used to send and receive emails.



202575325 / TechnoVectors / Shutterstock.com

Now that you have a basic understanding of client/server architecture, you can continue to look at database concepts.

Topic summary and review

In this topic you have learnt about the history and background of relational databases, data and databases concepts and have been introduced to several open-source DBMS including common RDBMS like MySQL.

You have also been provided with information about different web development software stacks and about how to install the XAMPP stack.

To gain a deeper understanding of the concepts and systems presented so far, complete this self-check quiz below:

Self-check – Relational databases

Please complete the following questions

Relational databases contain what type of keys to identify relationships between tables?

- ☐ Primary and secondary
- ☐ Primary and tertiary
- ☐ Primary and major
- ☐ Primary and foreign

In what decade did Dr Edgar F. Codd introduce relational databases?

- ☐ 1980s
- ☐ 1960s
- ☐ 1990s
- ☐ 1950s

Please select the three scientific disciplines used in data mining?

- ☐ Statistics
- ☐ Machine learning algorithms
- ☐ Counter Intelligence
- ☐ Artificial Intelligence

Data mining was previously known by which of the following:

- ☐ Data knowledge collation
- ☐ Data search and pattern finding
- ☐ Knowledge discovery in data
- ☐ knowledge-based data search

A business infrastructure is comprised of which of the following?

- ☐ software
- ☐ policies and procedures
- ☐ people and their skills / human resources
- ☐ hardware
- ☐ all of the above

Check your answers at the end of this Learner Guide

Structured query language (SQL)

Introduction

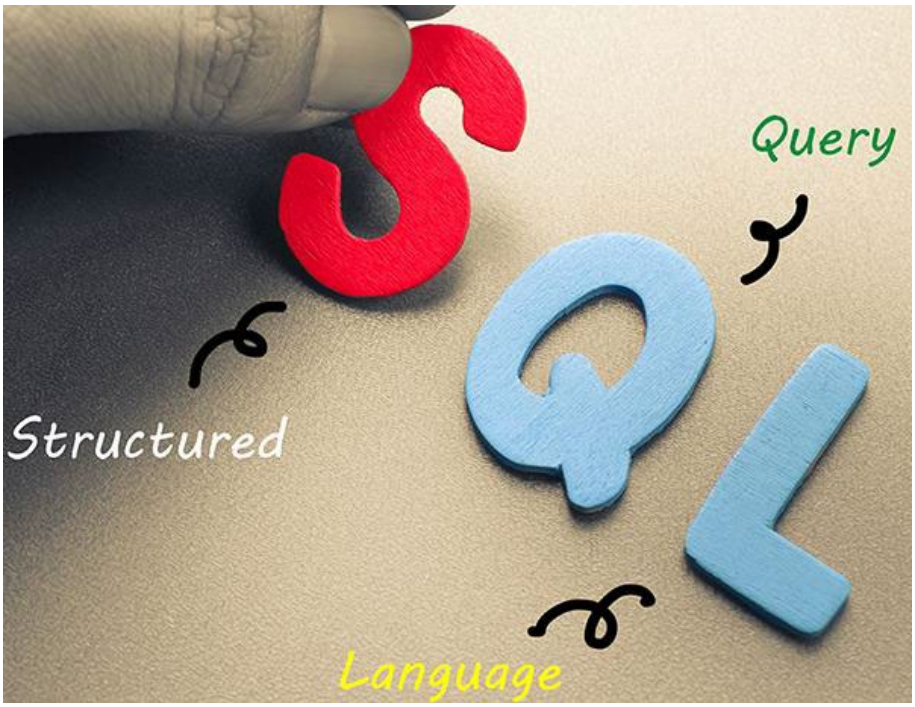
The purpose of this topic is to provide a general introduction to structured query language (SQL) and the data manipulation language (DML) commands. It will start by presenting several underlying SQL related syntax concepts and keywords, such as data types, constraints, and naming conventions.

The next step is to demonstrate the syntax to insert, select, update, and delete data from the database using basic SQL writing and SQL queries and statements to retrieve data from a single table.

Objectives

By successfully completing this topic you will be able to do the following:

- understand relational database concepts including key constraints, naming conventions, and data types
- understand SQL language components and their associated functions
- understand and apply basic SQL commands to SELECT, INSERT, UPDATE, and DELETE data
- understand and apply comparison operator to expressions
- understand table operations and apply SQL commands to create and alter table structure.



278070971 / patpitchaya / shutterstock.com. Modified by TAFE Queensland.

Relational database concepts

Key constraints

Here you are going to look at basic key constraints you may encounter in relational database tables. The purpose of these constraints is to ensure that only data that follows specified rules is entered in the tables.

Relational databases are strict in their rules to ensure that there is no loss of data integrity or redundancy. SQL basic key constraints.

For more information on each of the table constraints you may encounter in relational databases see the information below:

Primary Key The primary key uniquely identifies a row in a table, for example, a customer number is unique and so are an employee and student numbers e.g., CustomerNumber, EmployeeNumber and StudentNumber.

Foreign Key The foreign key (in the child table) references the primary key in another table (the parent table). It must follow the referential integrity constraints:

- the foreign key column must have the same declaration as the primary key
- a value in the foreign key column must be equal to a value in the primary key column or set to NULL

Null/Not Null This constraint identifies if a column can have NULL values or not.

A NULL value is used to identify that the table cell can be empty.

Unique A unique constraint forbids two rows to store identical values. This is not a primary key. For example, there is a column named EmailAddress and the primary key of the table is EmployeeID. The unique constraint will stop two employees from using the same email address.

Default Adds customisable default data to columns.

Check The check constraint restricts the data that can be inserted into a table, for example, in the PurchasedQTY column we may add a CHECK constraint to restrict the quantity to 10 e.g., PurchasedQTY <= 10

Weblink

Check the link to learn more information about the Key Constraints in DBMS[11]

[11] https://prepinsta.com/dbms/key-constraints/

Naming conventions

A naming convention is a set of agreed rules to name objects or identifiers such as databases, tables, columns and foreign keys. It is not a standard and for that reason you may encounter different naming conventions for different organisations within the same industry.

The links below provide two sources of information for database naming conventions. The important thing is to identify a naming convention and apply it consistently.

1496612570 / FrankHH / shutterstock.com

Weblink

Refer to the following web pages for more detailed information on identifier rules and naming conventions used by MariaDB or proposed by other sources.

MariaDB Identifier names[12]

SQL naming conventions[13]

Learn more about the Dos and Don'ts of general rules for naming identifiers in databases

- Use one single word, if the name is a combination of two or more words, decide casing – camel case or underscores to join multiple words? E.g. CustomerName (camel case) or customer_name (use of underscore)
- Do not use spaces
- Do not start name with numbers
- Database name, table names, general column names - as per general rules
- For table names, it is recommended to use the singular form of the word, for example customer instead of customers

[12] https://mariadb.com/kb/en/identifier-names/

[13] https://www.red-gate.com/simple-talk/blogs/sql-naming-conventions/

MariaDB data types

Think of a database as a container of one or more tables. Within each table you have several predefined columns which you have created. The values stored in each column are of a different type. The column CustomerName is expected to include characters while the column BeforeTaxIncome should contain numbers and specifically decimals. In the table below are some of the numerous MySQL/MariaDB data types.

MariaDB Data Type	Comments
BOOL, BOOLEAN or TINYINT (1)	A non-zero value = true. Zero = false
INT or INTEGER	A whole number (no decimal point) that is within the range -2,147,483,648 and 2,147,483,647
DECIMAL	Ideal for storing monetary values. E.g. DECIMAL(5,2) – specifying 5 would store up to a five-digit number, and the 2 means it would hold a maximum of two-decimal places. Maximum value of 99999.99.
VARCHAR	Stands for 'character varying'. A variable-length string. E.g. VARCHAR(50) would define a column capable of storing a string value of up to 50 characters. If the string value stored is 'David', the allocated number of bytes would be the equivalent of 5 characters not 50.
TEXT	TEXT can store large amounts of textual data and is ideal to store information collected in multi-line textboxes.
CHAR	A fixed-width string used when the length of a string is known (for example, a column for an Australian postcode).
DATETIME	Stored value as YYYY-MM-DD HH:MM:SS ideally used to record a date stamp when a particular event occurred (such as a user login).
AUTO_INCREMENT	When the first column in a table has been set as a primary key (INT) it is recommended that this attribute be set. When a new record is inserted into the table, the primary key does not need to be specified. The database will automatically increment the ID and assign it to the new record, ensuring that each record can be identified uniquely using the primary key ID.

Weblink

Check the link for more detailed information on all data types available[14].

[14] https://mariadb.com/kb/en/data-types/

SQL language components and SQL syntax

SQL language components

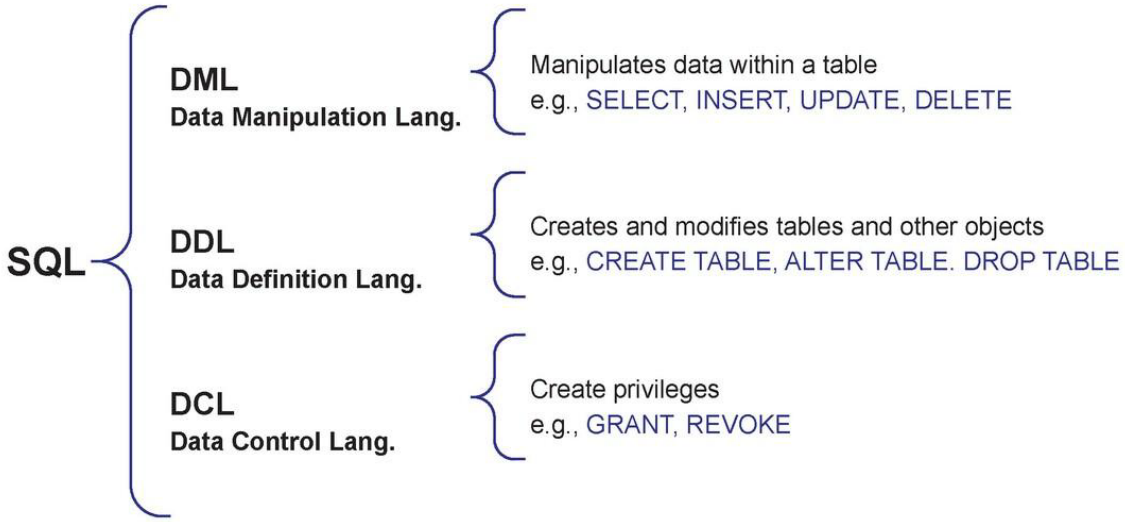
Three main components of the SQL language are:

Learn more about the three letter acronyms.

DML Data Manipulation Language (DML) allows the database user to manipulate the data stored in the database by selecting several operations such as select, insert, update and delete.

DDL Data Definition Language (DDL) provides the necessary commands to create the database structure. It allows the database administrator to perform functions such as create a database structure (schema) and create, alter, and drop tables.

DCL Data Control Language (DCL) deals mainly with user access and privileges such as grant and revoke.

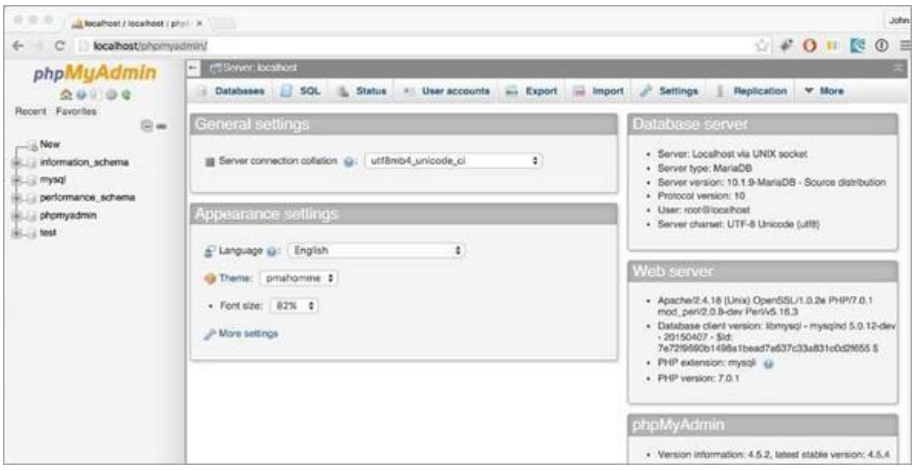


SQL Language Components

SQL syntax

In this section you will continue to explore the possibilities with some of the functions you can perform using phpMyAdmin, a database administration tool. You will more specifically look at SQL queries – how you can manipulate the records stored in our database tables.

For access to the SQL tab review the screen snippet below and click on the SQL tab in phpMyAdmin:



Screenshot of phpMyAdmin software , 16/12/21.GNU General Public License, version 2

The SQL commands used for querying and updating the database can be described using the CRUD acronym:

C Create	(INSERT INTO) – inserts new data into a database
R Read	(SELECT) extracts data from a database
U Update	updates data in a database
D Delete	deletes data from a database

These commands will be discussed in further detail in the following sections.

Data manipulation language (DML) commands

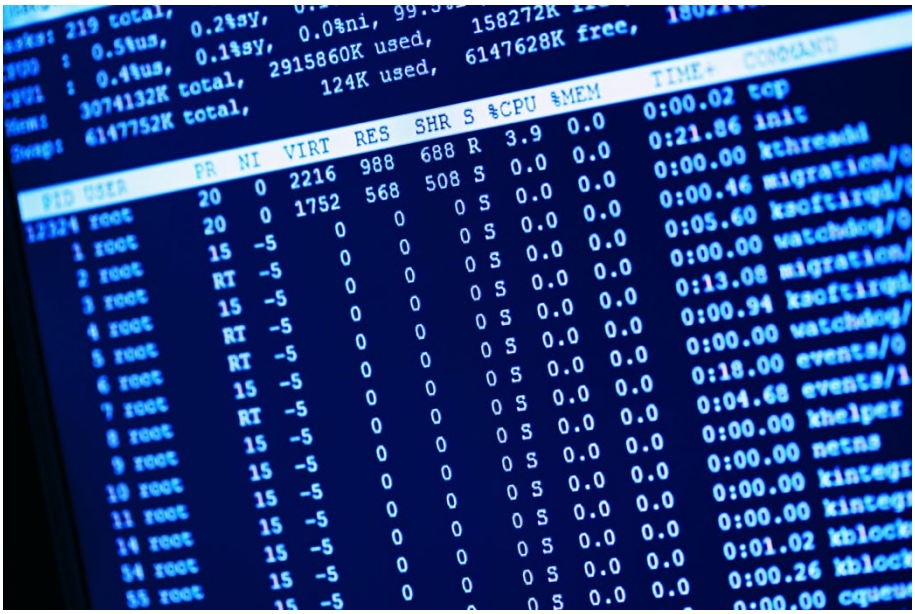
Introduction

This section will introduce you to the syntax of SQL and specifically to the DML commands.

Objectives

By successfully completing this topic you will be able to do the following commands:

- INSERT
- SELECT
- UPDATE
- DELETE.



124757866 / isak55 / shutterstock.com

DML commands

Use the link below to access an overview of the MariaDB SQL commands.

Weblink

MariaDB SQL statements tutorials: MariaDB.com -SQL Statements[15] and Tech-on-the Net MariaDB tutorial[16]

Because of the similarities between MariaDB and MySQL you can also use these tutorials: W3Schools – SQL Tutorial[17] and TutorialsPoint - SQL[18]

Note

MariaDB was developed from MySQL and except for a few differences uses the same syntax.

Learn more regarding the functionality of INSERT INTO and SELECT commands:

INSERT INTO

Allows you to insert a new record into the database. You write this statement as follows (this is the structure of the query which represents the **syntax**).

Syntax structure:

```
INSERT INTO [table_name]
([column1], [column2]…)
VALUES ([value1], [value2]…)
```

- [table_name] corresponds to the name of the table that you are inserting a new record into
- [column1], [column2] are column names that correspond to the columns in the table. Multiple columns can be specified
- Each column specified has a corresponding value – [value1], [value2]

[15] https://mariadb.com/kb/en/sql-statements/

[16] https://www.techonthenet.com/mariadb/index.php

[17] https://www.w3schools.com/MySQL/default.asp

[18] https://www.tutorialspoint.com/sql/index.htm

Sample query:

```
INSERT INTO Customer(CustomerName, CustomerSurname, Address)
VALUEs('John', 'Jones', '18 Sunshine Boulevard');
```

Important

Text (e.g. character, string) and date values must be quoted

e.g. name = "Alan" or dateCreated = "2021-01-12".

You can use single or double quotes in most databases and version.

SELECT

The example below shows how you can SELECT (or read) records from a database table.

SELECT statements have the following syntax:

```
SELECT [column1], [column2]…
FROM [table_name] WHERE [condition]
ORDER BY [column]...
```

- [column1], [column2] are column names that correspond to the actual column names in the database. We can specify multiple columns here (you are not limited to two). You can also use the * wildcard to return all columns
- [table_name] corresponds to the name of the table you are querying which is one of the tables in the database
- [condition] is optional. You can apply a filter to our SELECT statement so that only specific records are returned that match the criteria specified in the condition
- ORDER BY [column] – you can sort the record results by a particular column (for example, alphabetically or numerically)

You can learn more about these concepts in the following pages.

Text and numeric values

The following query demonstrated the use of a text value example. This is correct:

```
SELECT CustomerName, City
FROM Customer
WHERE City = "London";
```

This is not correct – London needs to be quoted:

```
SELECT CustomerName, City
FROM Customer
WHERE City = London;
```

Numeric values do not require quotes example. This is correct:

```
SELECT ContactName
FROM Customer
WHERE CustomerID = 16;
```


WHERE clause

The WHERE clause is used with the SELECT statement to filter records that match a specified condition.

Referring to the previous example, if you only want to see all customers who live in London, you could specify this by using the following:

```
SELECT CustomerName, Country
FROM Customer
WHERE Country = 'Canada';
```

Another example:

```
SELECT * FROM Customer
WHERE CustomerID <= 38;
```

SQL comparison operators

The queries above use the equal operator (=). There are other operators that can be used to help with filtering our results. The below table summarises additional operators that can be used.

Comparison Operator	Description
=	Equal
<> !=	Not equal. Note: In some versions of SQL this operator may be written as (!=)
>	Greater than
<	Less than
>=	Greater than or equal
<=	Less than or equal

SQL expressions

What is an expression in SQL? An expression can be thought of as a formula that combines values, operators, and functions that can evaluate into a single value or range.

SQL supports three types of expressions:

- Boolean expressions – evaluate to TRUE or FALSE

```
SELECT CustomerName
FROM Customers
WHERE CustomerID = 35;
```

- Numeric expressions – evaluate to a numerical value or range

```
SELECT CustomerName
FROM Customers
WHERE CustomerID * 3 >= 300;
```

- Date expressions – evaluate the date using various comparisons or functions

Example

```
SELECT CustomerName
FROM Customers
WHERE CustomerDoB > DATE('1986-05-05');
```

Weblink

For more examples using expressions visit: TutorialsPoint - Expressions[19]

[19] https://www.tutorialspoint.com/sql/sql-expressions.htm

DISTINCT statement

The DISTINCT statement will return all records where a particular column value is unique. The SELECT DISTINCT statement shares a similar syntax to the SELECT statement, with the only difference being the DISTINCT keyword:

```
SELECT DISTINCT [column1], [column2] etc. FROM [table_name]
WHERE [condition]
ORDER BY [column]
```

In the example below, the query will return the City names without duplicates even if there are multiple records of customers that reside in the same city, the list of cities names in the example below are only shown once.

```
SELECT DISTINCT City
FROM Customer
ORDER BY CustomerID;
```

ORDER BY statement

The ORDER BY keyword is used to sort the record results, alphabetically or numerically, in ascending or descending order. The default order is ascending (ASC). The keywords ASC or DESC determine the sort order, however because ascending is the default order, ASC does not need to be specified.

```
SELECT [column1], [column2]…
FROM [table_name] WHERE [condition
ORDER BY [column] [ASC, DESC]
```

Multiple columns can be sorted either in ascending or descending order (see the second example below):

```
SELECT DISTINCT City
FROM Customer
ORDER BY City, CustomerID DESC;
```


LIMIT clause, UPDATE and DELETE commands

LIMIT clause

The LIMIT clause can be used to restrict the number of rows returned. It accepts two arguments (or values) which are both numeric.

```
SELECT [column1], [column2]…
FROM [table_name] LIMIT offset, count;
```

For example:

```
SELECT * FROM Customers LIMIT 5, 3;
```

Here you substitute the words **offset** and **count** with actual numerical values. Offset determines the first row to return (this is zero-based so 0 is the first row, not 1). Count determines how many records to return (or the maximum number).

Note

In Microsoft SQL Server, the **SELECT TOP** clause is used instead of **LIMIT**.

UPDATE command

This is used to update values in a particular column for specified records that meet the condition:

```
UPDATE [table_name] SET column1 = value1, column2 = value2, etc.
WHERE [condition]
```

For example:

```
UPDATE Customers SET Country = 'Spain', City = 'Madrid' WHERE
CustomerID = 1;
```

Using this statement, you can update one or more column values based on a condition.

DELETE command

The DELETE keyword is used to delete records from the specified database table. It has the following syntax:

```
DELETE FROM [table_name] WHERE [condition]
```

The example below shows how you can DELETE an existing record from the database.

```
DELETE FROM customer
WHERE CustomerID = 114;
```

Weblink

For examples on using LIMIT, UPDATE and DELETE visit: W3Scools MySQL tutorial[20]

[20] https://www.w3schools.com/MySQL/default.asp

Data definition language (DDL)

Introduction

Databases, tables and indexes

So far you have learnt that a database contains one or more tables. Each table in the database contains a number of columns (fields) and a number of rows (records). This is similar to a spreadsheet. A database can also contain one or more indexes.

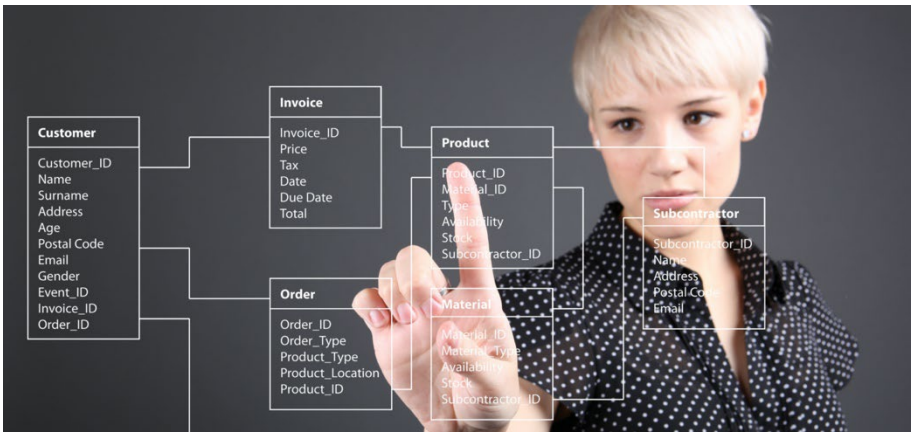
Objectives

By completing this module you will be able to learn about the most important non-CRUD commands in databases, table and indexes:

- Create an index to accelerate searches and how they allow the database to find data fast without reading the whole table.
- Create an index to accelerate searches based on an employee's last name (for instance), which will allow you to quickly find the employee when you use the SELECT statement with a WHERE clause.
- Understand the specific commands that permit databases and tables to be created, altered or deleted (e.g. dropped) and how you can create SQL statements that will achieve these tasks.
- Look at an easier method to perform the above, by using the GUI interface of phpMyAdmin.

Note

When using these non-CRUD commands, the syntax will differ slightly depending on the type of RDBMS you are using.



92126581 / Semisatch / shutterstock.com

CREATE DATABASE command

This SQL command creates a new database in which tables and other database structures (including indexes, stored procedures, views, and triggers) can be stored.

Syntax:

```
CREATE DATABASE [IF NOT EXISTS] database_name
[CHARACTER SET charset_name]
[COLLATE collation_name]
```

```
CREATE DATABASE IF NOT EXISTS classic_books
CHARACTER SET utf8mb4
COLLATE utf8mb4_unicode_ci;
```

Character set and collation are optional and can be set later or use the default.

```
CREATE database classic_books;
```

database_name is the name of the database being created and may follow a particular **naming convention**. Often, there is a naming convention applied to database names, table names, column names and other database structures that follows a standard set by the organisation. CamelCase is a common naming convention where each word in the name is capitalised. For example:

Example

HumanResources. Lower camel-case is the same as camel-case except that the first word in the name is not capitalised, e.g. humanResources. Another convention is the use of singular rather than plural names for tables, for example: customer instead of customers. As mentioned before, once a naming convention has been selected, or you have been directed to use it, be consistent.

Weblink

For an interesting discussion on this, visit the forum on the website Stack Overflow[21].

[21] https://stackoverflow.com/questions/338156/table-naming-dilemma-singular-vs-plural-names

CREATE TABLE command

This SQL command creates a new table. It has the following syntax:

```
CREATE TABLE [table_name]
(
column_name1 data_type(size), column_name2 data_type(size),
column_name3 data_type(size),
…
)
```

Weblink

To learn how to create a table visit MySQL[22] website.

```
DROP TABLE IF EXISTS author
CREATE TABLE author(
AuthorID INT(10) UNSIGNED NOT NULL AUTO_INCREMENT,
AuthorName VARCHAR(30) NOT NULL,
Surname VARCHAR(30) NOT NULL,
Nationality VARCHAR(30) NOT NULL,
BirthYear INT(4) UNSIGNED NOT NULL,
DeathYear INT(4) UNSIGNED DEFAULT NULL,
PRIMARY KEY (AuthorID)
) ENGINE =InnoDB AUTO_INCREMENT =1;
```

[22] https://dev.mysql.com/doc/search/?d=201&p=1&q=Create+table

The **table_name** is the name of the table being created, for example, employee.

The **data_type** denotes the type of data used for the column. For example: integer, or text. Data types were discussed earlier in this topic.

The **size** of the column determines how much data can be stored in the column. For example, VARCHAR(50) will store up to 50 characters of text. Decimal (10, 2) will store a 10-digit number containing two decimal places.

As previously mentioned, when creating a new table it is important to consider applying constraints to columns to ensure data integrity. These are typically set during earlier design phases.

Note

Numerical values can be SIGNED or UNSIGNED. Signed numbers accept positive and negative values e.g. -35 or + 35. Unsigned numbers accept only positive values and do not require the plus (+) or minus (-) symbol. The default is SIGNED.

The AUTO_INCREMENT declaration in the primary key indicates that an incremental numerical value (+1) will be generated for every new row in the table.

Weblink

Learn more about how SQL constraints are used to specify rules for data in a table by visiting this website: SQL Create Constraints[23]

357060374 / Titima Ongkantong / shutterstock.com

[23] https://www.w3schools.com/sql/sql_constraints.asp

CREATE INDEX

An index can be created on one or more columns, which will enable faster searches when the search is based on the indexed columns. A database index is a bit like an index in a book. You can refer to an alphabetical list of words and quickly find relevant information in specific areas within the book. This is a lot faster than scanning through the book page by page.

Syntax:

```
CREATE [UNIQUE] INDEX index_name
ON table_name (column1, column2,…)
```

```
CREATE INDEX idx_surname ON author(Surname);
```

The UNIQUE keyword is optional. This keyword, if included, will prevent any duplicate values from being stored in the index's column.

The **index_name** is the name of the index. The table_name is the name of the table where the index will be setup. Within the parentheses are one or more columns that will be used as the index.

608453894 / ktsdesign / shutterstock.com

ALTER DATABASE and TABLE commands

ALTER DATABASE

This SQL command changes the characteristics of an existing database.

Syntax:

```
ALTER [database_name] [alter_specification]
```

The **alter_specification** has two clauses, allowing you to alter the character set, or collation.

Weblink

For more information on character sets and collations, visit the MySQL[24] website.

ALTER TABLE

This command can be used to add, modify, or remove columns. In MySQL you can also use this command to add and drop primary keys, foreign keys, and indexes.

```
ALTER TABLE [table_name]
[ ADD COLUMN [column_name] [column_definition] ]
[ DROP COLUMN [column_name] ]
```

Some examples of altering a table are:

```
ALTER TABLE author
MODIFY COLUMN Nationality char(20);
ALTER TABLE author
ENGINE InnoDB;
ALTER TABLE book
DROP FOREIGN KEY authorid_fk;
```

The above syntax demonstrates the structure used for adding or dropping columns. For more detail, refer to the online documentation for your RDBMS of choice.

[24] https://dev.mysql.com/doc/refman/8.0/en/charset.html

DROP TABLE and DROP INDEX

DROP TABLE

This command is used to delete an existing database table.

Syntax:

```
DROP TABLE [table_name]
DROP TABLE book;
```

DROP INDEX

This command deletes an index from a database table.

Syntax:

```
DROP INDEX [index_name]
DROP index idx_surname;
```

In MySQL, the DROP INDEX command is used in conjunction with the ALTER TABLE command:

```
ALTER TABLE [table_name] DROP INDEX [index_name]
```

```
ALTER TABLE author DROP INDEX idx_surname;
```

Weblink

To learn more about creating a database, creating an altering table, and creating indexes, refer to the following link and scroll down the menu on the left-hand side of the landing page until you get to MySQL Database MySQL Database[25].

[25] https://www.w3schools.com/MySQL/default.asp

There are two practice activities to complete for executing queries against the database however, before you can undertake those practice activities you will first need to follow the steps 1 – 4 listed below.

1. Referring to the weblink below, download the sample database called Northwind. A sample database called Northwind can be downloaded from this link: MySQL version of Northwind demo database. Download the latest .sql version.
2. Using the sample database provided (Northwind), import the database using phpMyAdmin
3. By referring to previous examples demonstrated, practice using SELECT, UPDATE, INSERT INTO, and DELETE statements and execute these queries against the database.
4. Record the statements you write in a Word document for later use.

Weblink

MySQL version of Northwind demo database[26]

Activity - Executing queries against the database

Practice using the query SELECT by completing the following activities:

Practice activity 1

Using the Northwind database, write a query to SELECT all Orders that are ordered by the order date (with most recent orders appearing at the top of the list).

Practice activity 2

Using the Northwind database, retrieve all Orders for the Employee, Anne Hellung-Larsen.

[26] https://github.com/dalers/mywind

Activity - Create a new database table

In this activity, you will practice creating a new database table.

Please read through the instructions below accordingly.

- Using the Northwind database, create a new database table called 'call_centre_log'. This table will track incoming phone calls received by staff working in a fictitious call centre. It should contain the following columns:
  - logID (primary key, should not be NULL, and should be a unique integer)
  - customerID (can be NULL, and should be of type integer)
  - callDescription (should not be NULL, and should be able to store substantial amounts of text)
  - callDateTime (should record current date/time by default)
- Edit the 'call_centre_log' table (created in part A). Add an additional column to record the employee ID (not NULL, integer). This represents the ID of the employee who took the incoming call.
- INSERT test data into the 'call_centre_log' table to ensure that data is being inserted as expected. The table should contain at least 10 records.

Topic summary and review

In this topic you have learnt how to use DDL commands to create a database and tables and to alter their structures. You have been presented with examples to alter tables, create indexes and drop tables and indexes.

Multiple code examples have been presented to illustrate the syntax and the underlying concepts involved.

Links to external resources have been provided to further your understanding of the concepts presented in this topic.

Self-check – Structured Query Language

Please read each question and make your selection from the multiple-choice options provided.

What does XAMPP stand for?

- ☐ Cross-platform Apache, MySQL, PHP and Perl
- ☐ Cross-platform Apache, MySQL, PHP and Python
- ☐ eXtensible Apache, MySQL, PHP and Perl
- ☐ eXtensible Apache, MySQL, PHP and Python

What type of database model consists of tables in an upside-down tree that defines parent/child relationships?

- ☐ Tiered
- ☐ Logical
- ☐ Hierarchical
- ☐ Pyramid

What is the name of the MySQL data type used to store large text values?

- ☐ Word
- ☐ Text
- ☐ Number
- ☐ Letter

From the options below, which attribute automatically assigns a unique ID value to a new record when inserted?

- ☐ AUTO_INSERT
- ☐ AUTO_ID
- ☐ AUTO_ENTER
- ☐ AUTO_INCREMENT

Which of the following SQL statements will list all customers by their city location (in alphabetical order), and then by their name (in reverse alphabetical order)?

- ☐ SELECT CustomerName, City FROM Customers ORDER BY City DESC CustomerName ASC;
- ☐ SELECT CustomerName, City FROM Customers ORDER BY City ASC CustomerName DESC;
- ☐ SELECT CustomerName, City FROM Customers ORDER BY City DESC CustomerName;
- ☐ SELECT CustomerName, City FROM Customers ORDER BY City DESC CustomerName DESC;

Check your answers at the end of this Learner Guide

Working with logical operators, REGEXP and NULL values.

Introduction

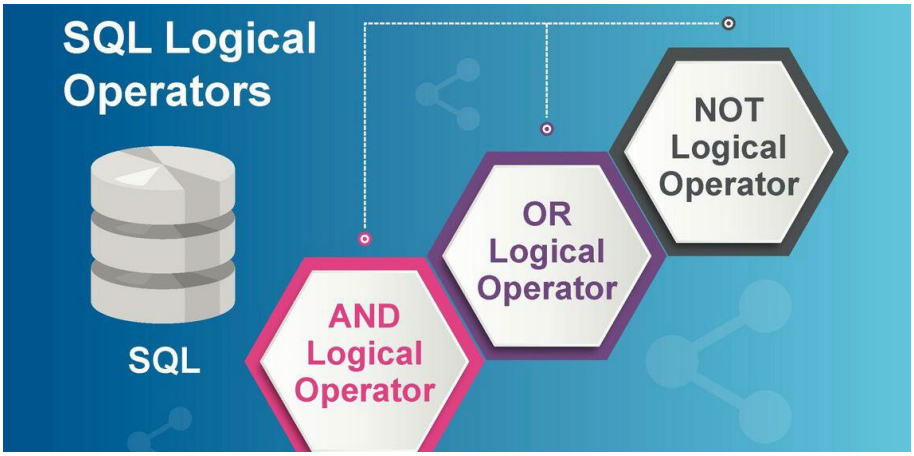
In this module, you will be introduced to additional statements commonly used in structured query language (SQL) and more specifically you will look at advanced methods of using the WHERE clause to filter results.

By using comparison and logical operators, regular expressions, and NULL values, you will examine further ways of limiting the queries' results using different filtering methods.

Objectives

By successfully completing this topic you will be able to do the following:

- understand and use logical operators including AND, OR, BETWEEN, LIKE, and IN
- understand how and when to use regular expressions to help validate data
- understand and correctly use null values.



Logical operators

In the last topic you looked at using the WHERE clause to apply conditions to the record results so that only records matching the condition would be returned. Additional methods to restrict and filter results are discussed below.

SQL Logical Operators

Logical Operator	Description
AND	TRUE if both conditions are TRUE
OR	TRUE if either one or both conditions are TRUE (inclusive OR)
BETWEEN	Displays results between an inclusive range
LIKE	Displays results based on a certain search pattern
IN	Displays results that match specific values for a column from a list of pre-determined values

AND OR operators

The **AND** operator displays a record if both the first and second conditions are true. For example, the book title must be, The Dream of the Red Chamber, and the language must be Chinese.

```
SELECT *
FROM book
WHERE BookTitle = 'The Dream of the Red Chamber'
AND LanguageWritten = 'Chinese';
```

The **OR** operator displays a record if either the first or the second condition is true.

The query will return all Chinese and British records from the author table.

```
SELECT *
FROM author
WHERE Nationality = 'Chinese'
OR Nationality = 'British';
```

The **AND OR** operators can be used to specify multiple conditions after the WHERE clause.

```
SELECT *
FROM book
WHERE BookTitle = 'Don Quixote'
AND MillionsSold > 100;
```

Operator precedence

The statement below contains no parentheses. How does the RDBMS handle the execution of this statement? A logical order is applied - the AND keyword takes precedence over the OR keyword.

```
SELECT *
FROM book
WHERE BookTitle = 'Don Quixote'
OR BookTitle = 'The Hobbit'
AND MillionsSold > 100;
```


The above statement is the equivalent to:

```
SELECT *
FROM book
WHERE BookTitle = 'Don Quixote'
OR
(BookTitle = 'The Hobbit' AND MillionsSold > 100);
```

The first bookTitle and millionsSold fields are evaluated together with the AND keyword, and the second bookTitle field is evaluated separately against the whole AND statement.

This raises an interesting concept called **operator precedence**. When executing an SQL statement, MySQL will always evaluate any AND conditions before any OR conditions, even if the OR condition appears before any AND conditions. This is the default order of precedence. The exception here is that if the OR condition is surrounded with parentheses.

BETWEEN, NOT BETWEEN operators

The **BETWEEN** operator is used to return records where a column falls within a particular range. The BETWEEN operator is used with the SELECT statement, and has the following syntax:

```
SELECT [column1], [column2]…
FROM [table_name]
WHERE [column] BETWEEN value1 AND value2
```

The BETWEEN operator returns values including 50 and 150 – it is an **inclusive** operator. Example using the BETWEEN operator:

```
SELECT ProductName, ProductDescription
FROM product
WHERE Price BETWEEN 50 AND 150
```

When you use the NOT BETWEEN operator the query returns values below 50 and above 150.

```
SELECT ProductName, ProductDescription
FROM product
WHERE Price NOT BETWEEN 50 AND 150
```


LIKE, NOT LIKE and wildcards

The **LIKE** operator is used to search for a particular pattern in the specified column. For example, LIKE can be used to return all records where a customer's last name starts with 'A' or where specifically three characters must be placed in a specific position.

Wildcards are used in conjunction with LIKE when searching for specific data in a database. The wildcard character used is substituted for one or more characters. The most common wildcards are '%' (percentage) which substitutes for zero or more characters, and '_' (underscore) which substitutes for exactly one (1) character. The below SQL statement demonstrates how to use the % (percentage) wildcard.

Examples and explanation for LIKE operator patterns

Example	Explanation
AuthorName LIKE 'P%';	The author's name starts with the letter "P" and can have zero or more characters after the "P".
AuthorName LIKE '%r';	The author's name ends with the letter "r" and can have zero or more characters before the "r".
AuthorName LIKE 'P__e%';	The author's name starts with "P", has exactly two characters after, the letter "e" follows and after that it can have zero or more characters.
AuthorName LIKE '%_e_%';	The author's name can start with zero or more characters, then a single character follows, the letter "e" is next, followed by another single character and finally zero or more characters.
AuthorName LIKE '%ete%';	The letters "ete" (together) are located within the author's name.

Syntax structure using a LIKE operator:

```
SELECT [column1], [column2]… FROM [table_name]
WHERE [column] LIKE [pattern]
```

The query presented below will return all customers where the name includes the letters "te" at any location within the string.

```
SELECT AuthorName
FROM author
WHERE AuthorName LIKE '%te%';
```


The query presented below will return all customers where the name does not start with the letter "M".

```
SELECT AuthorName
FROM author
WHERE AuthorName NOT LIKE 'M%';
```

Weblink

For more examples using LIKE, BETWEEN and IN operators visit:

W3Schools logical operators[27]

W3Schools LIKE Operator[28]

W3Schools BETWEEN Operator[29]

W3Schools IN Operator[30]

Tip

The character list [charlist] wildcard can be used in SQL to return particular records with column values that start with a specified letter. The following query, for example, will return all customer names that start with A, or B, or C:

```
SELECT *
FROM customer
WHERE customer_name LIKE '[ABC]%'
```

This wildcard is not supported in MySQL, however regular expressions can be used instead. See the section on regular expressions later in this topic for more information.

[27] https://www.w3schools.com/sql/sql_operators.asp

[28] https://www.w3schools.com/sql/sql_like.asp

[29] https://www.w3schools.com/sql/sql_between.asp

[30] https://www.w3schools.com/sql/sql_in.asp

IN, NOT IN

The IN operator allows you to setup a condition that is compared against a list of values. It has the following syntax:

```
SELECT [column1], [column2]…
FROM [table_name]
WHERE [column] IN (value1, value2…)
```

Examples using the IN operator:

```
SELECT AuthorName
FROM author
WHERE AuthorName IN('John', 'Alan', 'Peter');
```


Activity - Write an SQL statement that uses logical operators

Take the steps below to take to write an SQL statement.

Step 1	Referring to the Northwind database (downloaded and installed in a previous activity) and open a blank word document which will be used for capturing the statements you write for later use.
Step 2	Write a statement that uses the IN operator to select all products that are in either the ‘Oil’, or ‘Sauces’ category.
Step 3	Write a similar statement to the above that uses the OR operator, rather than the IN operator.
Step 4	Use the NOT operator to display products that belong to any of the categories, except for the ‘Baked Goods & Mixes’ category.
Step 5	Display all records where the product name contains the word ‘pears’.
Step 6	Display all records where the product list price is greater than $40 and less than $60, using the BETWEEN operator.

Regular expressions

Regular expressions are used to identify patterns in data. If you are familiar with programming languages such as PHP or ASP.NET, you may already know about regular expressions and how they are used to identify patterns in data.

HTML5, the hypertext markup language, also uses regular expressions to validate form data that has been entered by a user. The <input> tag has a required pattern attribute that can be used to validate data such as mobile phone numbers and email addresses.

Similarly, regular expressions are used in SQL to return records that contain columns with a matching pattern. In MySQL you use the REGEXP operator.

REGEXP has the following syntax:

```
SELECT [column1], [column2]…
FROM [table_name]
WHERE [column] REGEXP [pattern]
```

In the example below, the regular expression is used to retrieve the author's names where the author's surname starts with either P or Q. The circumflex or hat symbol (^) is used to identify the beginning of the string. The dollar sign ($) is used to identify the end of the string.

```
SELECT AuthorName
FROM author
WHERE AuthorSurname REGEXP '^[Q|P]';
```

REGEXP example using the $ symbol – the author's name must end with either C or M:

```
SELECT AuthorName
FROM author
WHERE AuthorSurname REGEXP '[C|M]$';
```

Weblink

For more information and examples on regular expressions check the MySQL REGEXP – Finding Patterns[31].

[31] https://www.mysqltutorial.org/mysql-regular-expression-regexp.aspx

IS NULL and IS NOT NULL operators

The **IS NULL** is used to check if a column has empty values. The **IS NOT NULL** condition is used to return any records where a column value is not null (that is, it contains a value of some sort). If a column being queried contains a null value, the record will be ignored. This condition can be used in any INSERT, SELECT, UPDATE, or DELETE query, and has the following syntax:

```
SELECT [column1], [column2] …
FROM [table_name]
WHERE [condition] IS NOT NULL
```

Example

```
SELECT AuthorName
FROM author
WHERE DeathYear IS NULL;
```

Important

Correct:

```
SELECT AuthorName
FROM author
WHERE DeathYear IS NULL;
```

Incorrect:

```
SELECT AuthorName
FROM author
WHERE DeathYear = NULL;
```


Activity - Regular expressions (REGEXP)

In this activity you will undertake a series of steps using the REGEXP() function.

Step 1	Continue using the Northwind database (as previously downloaded and installed) plus a blank word document to capture your statements.
Step 2	Using the REGEXP() function return all products that end with ‘Sauce’
Step 3	Using the REGEXP() function return all products that contain the phrase ‘Chocolate’
Step 4	Using the REGEXP() function return all categories that start with the letter ‘A’, ‘B’, or ‘C’
Step 5	Using the REGEXP() function return all categories where the second and third letters in the first word are ‘a’ and ‘n’ respectively

228162115 / BEST-BACKGROUNDS / shutterstock.com. Modified by TAFE Queensland.

Topic summary and review

In this topic you have learnt about using logical operators including AND, OR, BETWEEN, LIKE and IN. You also looked at operator precedence, understanding the importance of using parentheses, and using wildcards and regular expressions to help with searching and filtering records. Lastly, you looked at IS NULL and NOT NULL.

To gain a deeper understanding, visit the recommended sites and complete the activities including this self-check quiz.

Self-check – Structued query language

Please complete the question/s below:

Examine the following syntax and select the correct statement from the options below:

```
SELECT *
FROM Customers
WHERE (state = 'Qld' OR last_name = 'Smith') AND customer_id < 200;
```

- ☐ The query will show all customers where customer ID is less than 200
- ☐ The query will show customers that either live in Qld or have a last name of Smith
- ☐ The query will show customers that either live in Qld or have a last name of Smith, and where the customer ID is less than 200
- ☐ The query will show all customers that live in Qld, including any that have a customer ID of less than 200 and a last name of Smith

If you enter the following query, what results will you see?

```
SELECT *
FROM customers
WHERE customer_name LIKE 'TOM%'
```

- ☐ Return any customer records where the customer's name starts with "TOM"
- ☐ Return any customer records where the customer's name starts with "TOM%"
- ☐ Return any customer records where the customer's name does not start with "TOM"
- ☐ Return any customer records where the customer's name starts with "CUSTOMER_TOM"

Examine the following query. What will be returned when it is executed?

```
SELECT *
FROM customers
WHERE customer_name LIKE '[TOM]%' ;
```

- ☐ Return any customer records where the customer's name contains the phrase ‘TOM’
- ☐ Return any customer records where the customer's name starts with the phrase ‘TOM’
- ☐ Return any customer records where the customer's name contains either a ‘T’, ‘O’, or ‘M’
- ☐ Return any customer records where the customer's name starts with either a ‘T’, ‘O’, or ‘M’

Which of the following wildcards is NOT supported in most relational databases, including MySQL?

- ☐ [charlist]
- ☐ [chargelist]
- ☐ [chartlist]
- ☐ [changelist]

You need to find all 4-digit account numbers in the database but you only have the first 3 digits. Which wildcard is best to use to find all matching account numbers using just the first 3 digits?

- ☐ %
- ☐ _ (underscore)
- ☐ !
- ☐ %!

Check your answers at the end of this Learner Guide

Use SQL functions

Introduction

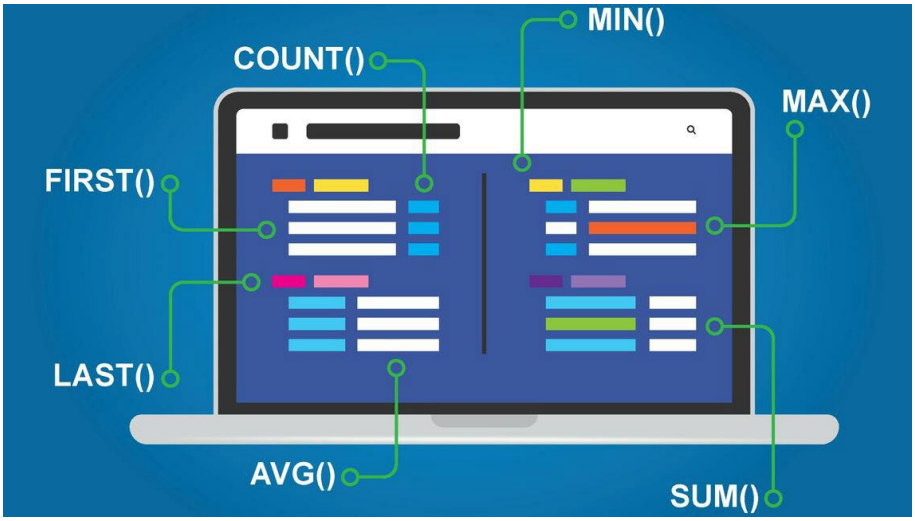
In this topic you will learn about using SQL functions and how they are used to provide aggregation. This is achieved by using a function to calculate a value based on values stored in multiple columns. You will also learn about various functions including: numerical, string, string concatenation, and date functions.

Functions are an important part of SQL especially if you work in data analysis and related fields as they help you search the data and validate the hypothesis through several specialised mathematical functions, string functions such as TRIM and help with data cleansing and the IF function.

Objectives

By successfully completing this topic you will be able to do the following:

- understand the various aggregate functions in SQL and how they are applied to solve different needs
- understand how to use GROUP BY and HAVING statements to filter aggregated results
- understand how to use and apply string-related SQL functions to manipulate strings
- understand how to use and apply functions relating to numbers
- understand how to use date functions.



@ TAFE Queensland

Aggregate functions

So far you have already looked at comparing number values with column values for equality, using the equals (=) operator. The BETWEEN operator was also demonstrated to test for values of columns that are between a specified range.

In this section you will look at SQL aggregate functions used when working with numbers, which return a single value (for example, a total or average amount). The value is a calculation of multiple rows and based on a particular column.

Below is a summary of the aggregate functions, followed by a more in-depth analysis of each.

Aggregate Function	Purpose
SUM()	Returns the sum of the column values
AVG()	Returns the average value of the column
MAX()	Returns the largest value in the column
MIN()	Returns the smallest value in the column
COUNT()	Returns the number of rows
FIRST()	Returns the first value in the column
LAST()	Returns the last value in the column

Select the headings below to learn more about aggregate functions:

SUM() function

The SUM() function returns the sum of all values in an expression. This function uses the following syntax:

```
SELECT SUM([column1]) FROM [table_name]
```

- [column1] is the numeric column to be totalled.
- It can be used with a WHERE clause to add a condition to filter record results.

```
SELECT SUM(MillionsSold)
FROM book;
```

As you will see, other aggregate functions follow a similar format (syntax) and are all used in a similar way to calculate aggregate figures. In the example below the function has two parameters, (Price, 2). In this instance the query will return the price rounded to 2 decimal places.

```
SELECT ROUND(Price, 2)
FROM product
WHERE StockOnHand > 134;
```

Alias names can be used for aggregated and arithmetically calculated columns to help describe the column being returned. If an alias is not used the column name will be the actual function used in the query e.g., ROUND(PRICE, 2). Alias names can also be applied to specific columns to make results easier to read – this is helpful when the column name in the database table is a little ambiguous.

```
SELECT ROUND(Price, 2) AS 'Current Price'
FROM product
WHERE StockOnHand > 200;
```

Note

Depending on the DBMS you are using, the alias (AS) needs to be quoted for multiple words. You can use a single word or you can use camelCase to join multiple words.

AVG() function

The AVG() function calculates the average value of a set of column values based on a condition. This function has the following syntax:

```
SELECT AVG([column1]) FROM [table_name]
```

- [column1] is the numeric column to be averaged
- It can be used with a WHERE clause to add a condition to filter record results

```
SELECT AVG(MillionsSold)
FROM book;
```


MAX() and MIN() functions

The MAX() and MIN() functions return the maximum and minimum values based on a set of column values. Syntax:

```
SELECT MAX([column1]) FROM [table_name]
```

```
SELECT MAX(Price)
FROM product
WHERE StockOnHand > 35;
```

```
SELECT MIN([column1]) FROM [table_name]
```

```
SELECT MIN(Price)
FROM product;
```

- [column1] is the column that the query will use to return the highest value (using MAX), or lowest value (using MIN)
- It can be used with a WHERE clause to add a condition to filter record results

COUNT() function

The **COUNT()** function returns the number of records based on a condition. A column name can be specified when using COUNT. This will return the total number of values that appear in the specified column across all records queried. Syntax:

```
SELECT COUNT (column1) FROM [table_name]
```

Example

```
SELECT COUNT(BookTitles)
FROM book
WHERE YearOfPublication > 1800;
```


Weblink

Check the link to learn more information on how to use SQL Functions[32].

[32] https://www.sqltutorial.org/sql-functions/

Calculations using MATHS functions

MATHS functions are a combination of mathematical operators and mathematical formulas which will allow you to perform any calculation in your queries that you could perform with a different tool.

A combination of mathematical operators and mathematical formulas will allow you to perform any calculation in your queries that you could perform with a different tool.

461093935 / rawf8 / shutterstock.com

Arithmetic Operator	Example
+ Addition	SELECT (income + bonus) FROM salary
- Subtraction	SELECT (amount - penalty) FROM cost
* Multiplication	SELECT (price * quantity) FROM orderDetails
/ Division DIV division (integer)	SELECT (yearBonus / weeks) FROM salary – or DIV (for int)
% or MOD Modulus (division remainder)	SELECT (yearBonus MOD weeks) FROM salary
Arithmetic Operator	Example

Example

Example using calculation - multiplication, percent, and round to 2 decimal places:

```
SELECT (quantityOrdered*priceEach) AS Subtotal,
round((quantityOrdered*priceEach)*0.10,2) AS '10 % Discount',
round((quantityOrdered*priceEach)*0.90,2) AS 'Order Total'
FROM orderDetails;
```


Learn more about using the statements GROUP BY and GROUP BY with HAVING.

GROUP BY Statement

The GROUP BY statement is used with an aggregate function to group the records returned by one or more columns. Syntax:

```
SELECT [column1], [column2],…, [aggregate_function(column)]
FROM [table_name]
WHERE [condition]
GROUP BY [column]
```

Example aggregate function and GROUP BY:

```
SELECT COUNT(*), Nationality
FROM author
ORDER BY COUNT(*) DESC;
```

GROUP BY with HAVING Statement

The **HAVING** statement can be used with **GROUP BY** to filter the distinct groups that are returned when you execute the query.

HAVING is like the **WHERE** clause in that they both use conditions to filter results. The difference is that **HAVING** is used to check a condition **after** the aggregation has been calculated, and **WHERE** checks any conditions **before** the aggregation has been calculated. Syntax:

```
SELECT [column1], [column2],…,
[aggregate_function(column)]
FROM [table_name]
WHERE [before_condition]
GROUP BY [column]
HAVING [aggregate_function(column)] [after_condition]
```


Example

Example aggregate function and GROUP BY and HAVING:

```
SELECT COUNT(*), Nationality
FROM author
HAVING COUNT(*) > 5
ORDER BY COUNT(*) DESC;
```

Weblink

To learn more about GROUP BY and HAVING in SQL visit the Datacamp[33] website.

[33] https://www.datacamp.com/community/tutorials/group-by-having-clause-sql

Calculations using string functions

In a previous topic you used both **LIKE** and **NOT LIKE** to search for records where column values contained certain characters (for example, return all countries that contain the phrase 'land').

In this section you continue to explore ways of working with strings. You will look at several commonly used string functions. The first-string function you will examine is used to make string comparisons and is called STRCMP().

Learn more about string functions:

String functions

The STRCMP() function is used to compare two strings, hence the name String Comparison. The syntax used to compare the strings is:

```
STRCMP(string1, string2)
```

Possible return values of this function are either 0, -1, or 1. The function will return 0 if both strings are the same. It will return -1 if string1 is smaller than string2, and 1 if string2 is smaller than string1.

TRIM function

The TRIM function removes any leading or trailing whitespaces from the data. It is used in data analysis to cleanse the data from whitespaces that may pass unnoticed to the human eye but hides the data from searches.

```
SELECT TRIM(AuthorName), TRIM(AuthorSurname)
FROM author;
```

CONCAT function

The **CONCAT** () string function is used to join several string values to form one string value. The following example (based on the Northwind database) will return a list of products listed with a formatted price, including the dollar sign. The query uses the **FORMAT** () function to round the price to two decimal places.

```
SELECT product_name,
CONCAT('$', format(list_price,2)) AS Price FROM
products
```

The CONCAT() function here is joining the column AuthorName and the column AuthorSurname name. The resulting column has been named with an ALIAS.

```
SELECT CONCAT(AuthorName, " ", AuthorSurname) AS
Author, BirthYear + 18 AS "Just turned 18"
FROM author
WHERE AuthorID = 7;
```

Name and surname are returned in the same column separated by a space. If author ID 7 Name was Cao Xueain the query will output:

Author	**Just turned 18**
Cao Xueain	1733

There is another form of CONCAT called GROUP_CONCAT. This string concatenation returns multiple records in the same row separated by a user defined separator.

```
SELECT MillionsSold,
GROUP_CONCAT(BookTile SEPARATOR '+') AS Books
FROM book
WHERE MillionsSold > 100;
```

The output with GROUP _CONCAT is:

Millions Sold	**Books**
100	And Then There Were None + The Hobbit + The Dream of the Red Chamber

Without the GROUP_CONCAT, the output returns multiple rows:

Millions Sold	Books
100	And Then There Were None
100	The Hobbit
100	The Dream of the Red Chamber

IF() function

The IF() function in SQL allows you to use the conditional IF statements (selection) used in any programming language.

Syntax structure:

IF(expression, value_if_true, value_if_false)

Example

```
SELECT BookTitle, LanguageWritten,
IF(MillionsSold > 200,"Great Success", "Moderate Success")
FROM book;
```

Date and time functions

In this section you will look at several popular functions used when working with dates and times.

Function	Description
CURDATE(), CURRENT_DATE()	Returns the current date
CURTIME(), CURRENT_TIME()	Returns the current time
NOW()	Returns current date and time
CURRENT_TIMESTAMP(), LOCALTIME()	Returns current date and time
DATEDIFF()	Find the difference in days between two dates
DATE()	Extracts the date
YEAR()	Extracts the YEAR part from a date
MONTH()	Extracts the MONTH part from a date
WEEK()	Extracts the WEEK part from a date
DAY()	Extracts the DAY part from a date
INTERVAL (keyword)	Difference between two dates /times
TIMESTAMP()	It is used for values that contain both date and time parts. TIMESTAMP ranges from 1970-01-01 00:00:01UTC to 2038-01-19 03:14:07UTC.
DATE_FORMAT()	Can be used to apply country-specific formatting to a specified date

Weblink

For more information on how to use SQL function visit:

Commonly used SQL functions[34]

Date and Time functions[35]

[34] https://www.analyticsvidhya.com/blog/2020/07/sql-functions-for-data-analysis-tasks/

[35] https://dev.mysql.com/doc/refman/8.0/en/date-and-time-functions.html

Examples of DATE and TIME functions

Simple examples:

```
SELECT now();
SELECT current_date();
SELECT current_time();
SELECT current_timestamp();
```

Formatted OrderDate column from "2024-10 -15" to "Tue 15th Oct 2024":

```
SELECT date_format(OrderDate, '%a %D %b %Y')
AS "Formatted Date"
FROM order;
```

Finds the difference **in days** between the OrderDate and the ShippedDate.

```
SELECT OrderDate, ShippedDate,
DATEDIFF(OrderDate, ShippedDate)
FROM order;
```

Finds the difference in weeks between the OrderDate and the ShippedDate using TIMESTAMP()

```
SELECT OrderDate, ShippedDate,
TIMESTAMPDIFF(OrderDate, ShippedDate)
FROM orders;
```

In MySQL, the INTERVAL keyword can be used to add or subtract a specific date and time. For example:

```
SELECT '2024-10-15' - INTERVAL 1 MONTH; minus 1 month
```

Returns: 2024-09-15

```
SELECT '2024-10-15' + INTERVAL 1 MONTH; plus 1 month
```

Returns: 2024-11-15

```
SELECT NOW() + INTERVAL 1 WEEK; plus 1 week
```

Returns: 2024-10-22 22:35:18 – has return date and time as we used NOW()

Other SQL functions

There are many other aggregate, string, date/time, and other more advanced SQL functions that are beyond the scope of this learner guide. You have been introduced to some of the more commonly used functions, and you are encouraged to explore the other functions available.

Weblink

Visit the following website to learn more about the numerous MySQL functions available. Once on the website you will see a list of categorised functions and you can select a function name to view examples of the chosen function. Spend some time exploring the various functions available at MySQL functions[36]

Activity

1. Using the Northwind database, write an SQL query that lists total orders for each Customer, but only show totals for those Customers who have placed at least two orders.
2. Using the Northwind database, write two separate SQL queries that use the CONCAT() AND STRCMP() string functions.

595783184 / Wright Studio / shutterstock.com

[36] https://www.techonthenet.com/mysql/functions/index.php

Topic summary and review

In this topic you have learnt about SQL functions, including aggregate functions, string functions, and date/time functions. SQL functions provide a powerful way of manipulating and working with the raw data values stored in the database tables.

For a deeper understanding of the syntax and concepts covered in this topic, complete the activities presented and visit the links provided.

Self-check – Working with logical operators

Please complete the following question/s

Which outcome will be returned when executing the following query ?:

```
SELECT STRCMP('Acme Printing Company', 'Creative Dance Academy')
```

- ☐ zero (0)
- ☐ minus 1 (-1)
- ☐ one (1)
- ☐ (NULL

From the query below, when will the condition that checks for the last name of the employee be tested in relation to the aggregated condition?

```
SELECT Employees.LastName,
COUNT(Orders.OrderID) AS NumberOfOrders FROM
(Orders INNER JOIN Employees ON
Orders.EmployeeID=Employees.EmployeeID) WHERE
LastName = 'Buchanan' GROUP BY LastName
HAVING COUNT(Orders.OrderID) > 10;
```

- ☐ It will be tested before the aggregated condition
- ☐ It will be tested after the aggregated condition
- ☐ Both will be tested at the same time
- ☐ It will not be tested as the aggregated condition will override it

‘Address’ in the following query is an example of what SQL feature?

```
SELECT CustomerName, Street +', '+Suburb+', '+Postcode+',
'+Country AS Address FROM Customer;
```

- ☐ Aggregate function
- ☐ String function
- ☐ CONCAT() function
- ☐ Alias name

Of the following options, what outcome will be returned when this query is executed?

```
SELECT COUNT(*) FROM Orders;
```

- ☐ The total number of unique orders stored in the Orders table
- ☐ The total number of orders in the Orders table (including duplicates)
- ☐ The total number of records in the Orders table
- ☐ The grand total of all orders stored in the Orders table

When executing the following query, what will occur from the options below:?

```
SELECT DATE_FORMAT(NOW(), '%d/%m/%y')
```

- ☐ Return the current date and time
- ☐ Return the current date in Australian date format
- ☐ Return the date of last record insert in Australian format
- ☐ Return the current date and time when the last record was inserted

Check your answers at the end of this Learner Guide

Working with multiple tables in SQL

Introduction

In this topic you will be introduced to the concept of working with multiple tables used to display meaningful results based on related data.

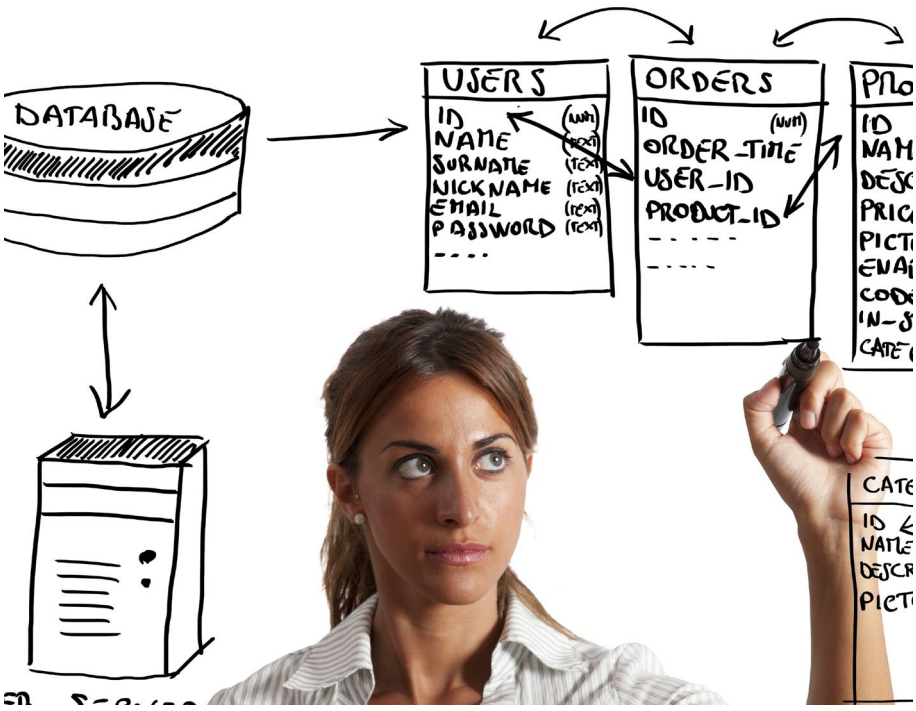
You will begin the module by exploring some examples of using sub-queries where an SQL statement contains a query within a query. A single SQL query can sometimes contain references to multiple tables. Related data can be retrieved from multiple tables based on common column values. To achieve this, the JOIN keyword is used.

Lastly, you will look at the UNION operator to return records from multiple tables that share similar data. The UNION operator is used with two or more SELECT statements.

Objectives

By successfully completing this topic you will be able to do the following:

- understand how to use subqueries and the benefits of using them
- understand the differences between INNER, LEFT, RIGHT and OUTER JOINs, and appreciate how and when to apply them
- understand how to use table aliases to make a query easier to read
- understand the UNION and UNION ALL operators.



116384731 / alphaspirit.it / shutterstock.com. Modified by TAFE Queensland.

Table aliases

A table alias is an alternative name given to a table that is shorter in length, typically just a letter, or two, to make an SQL statement easier to code and read. They are commonly used particularly when the SQL statement includes references to multiple tables.

If you assign an alias to a table, you must use this alias to refer to the table throughout your query. If you refer to the original table name an error will result.

When building JOIN queries, it may make sense to use aliases for some table names and not others. When a table is referenced frequently throughout a query, and the table name is quite lengthy, it would make sense to use an alias as a shorthand reference to the table to save having to repeatedly type the long table name.

Join multiple tables

To return records that contain data from two or more tables you JOIN them together by a column that is common to both, the primary key in one table and the foreign key in the other table. There are several joins available. Depending on the requirement you may use an INNER, LEFT, RIGHT or CROSS JOIN. The UNION operator can also be used to return results from multiple tables. Each of these concepts is explained in greater detail below.

INNER, LEFT, RIGHT or CROSS JOIN

An INNER JOIN, also called a simple join, combines columns from two or more tables into records that are returned based on the conditions that are specified. The INNER JOIN returns the intersection of the two tables, that is, the rows from both tables that match the condition. The LEFT JOIN returns all rows in the left table (Table 1 in the diagram) and the intercepting rows from Table 2 (the right table). The RIGHT JOIN is the opposite of the LEFT JOIN, and the CROSS JOIN returns all records from both tables.

What makes a table left or right? The first table that you declare in the query is referred to as the left table (or table 1) and the second table declared becomes the right table (or table 2).

Weblink

For more information on INNER, LEFT, RIGHT or CROSS JOIN from w3schools, you can visit their website at w3schools.com/MySQL[37]

When using multiple tables, you should use the table name with the column name joined by a full stop and no spaces between them e.g. "orders.id", "orders.customer". This is done to ensure that there is no ambiguity about what column in which table you are referring to. As an alternative you can give your tables an ALIAS or one or two characters to simplify the typing.

The order of the tables is important for LEFT and RIGHT JOINs but not for the INNER JOIN as it will always return the intersection.

INNER JOIN syntax using Northwind database:

```
SELECT orders.id, customers.company
FROM orders INNER JOIN customers
ON orders.customer_id = customers.id;
```

Using aliases:

```
SELECT o.id, c.company
FROM orders o INNER JOIN customers c
ON o.customer_id = c.id;
```

Although tables are typically joined on a relationship between the primary key in one table and a foreign key in the other table, you can also join tables based on relationships not defined in the database. These are called *ad hoc relationships*. *Ad hoc* queries cannot guarantee data integrity as data in the selected columns may be duplicated or missing e.g. NULL values; avoid these types of joins.

The order of the tables is important for LEFT (also referred to as LEFT OUTER) and RIGHT (also referred to as OUTER RIGHT) JOINs but for the INNER JOIN, order is irrelevant as it will always return the intersection. Order is also irrelevant for the CROSS JOIN as it returns all records from both tables.

[37] https://www.w3schools.com/MySQL/mysql_join.asp

The LEFT JOIN syntax using Northwind database:

```
SELECT orders.id, customers.company
FROM orders LEFT JOIN customers
ON orders.customer_id = customers.id;
```

The RIGHT JOIN syntax using Northwind database:

```
SELECT orders.id, customers.company
FROM orders RIGHT JOIN customers
ON orders.customer_id = customers.id;
```

To understand the differences between LEFT and RIGHT look at the Venn diagram, run the queries and check the results.

The CROSS JOIN syntax using Northwind database:

```
SELECT orders.id, customers.company
FROM orders CROSS JOIN customers
ON orders.customer_id = customers.id;
```

Weblink

For more information on how to use JOINs such as joining more than two tables and other types of joins refer to: MySQL.COM Documentation on JOINS[38]

[38] https://dev.mysql.com/doc/refman/8.0/en/join.html

UNION operator

The **UNION** operator is used to combine the records of two or more SELECT statements into one result set.

Each SELECT statement must return the same number of columns, and the corresponding columns in each must have compatible data types. The columns specified in each SELECT statement must also be in the same order.

By default, a UNION eliminates any duplicate records (like using the DISTINCT keyword, duplicate records will only be shown once). The exception is if the UNION ALL operator is used. In this case, all duplicates found across the tables queried in the union will be shown. The syntax for the UNION operator is as follows:

```
SELECT [column1], [column2]… FROM [table1]
```

UNION

```
SELECT [column1], [column2]… FROM [table2]
```

This query will not return duplicates.

```
SELECT City FROM customers
```

UNION

```
SELECT City FROM employees;
```

This query will return duplicates values if they exist in the tables.

```
SELECT City FROM customers
```

UNION ALL

```
SELECT City FROM employees;
```


Subqueries

A subquery is an SQL query that is contained within another query called the outer query. The subquery is called the **inner** query. They are used to retrieve data from one table that is based on data filtered from another table.

Outer query

```
SELECT last_name, first_name FROM employees WHERE id IN
```

Inner query

```
(SELECT employee_id FROM orders WHERE order_date < '2006-02-01')
ORDER BY last_name, first_name
```

This query, based on the Northwind database, will display all employees that have placed orders prior to the 1st of February 2006 (the subquery is used to lookup all relevant employee IDs from the orders table, according to the order date).

They can be embedded not only with SELECT statements (as shown in the above example), but also in INSERT, UPDATE, or DELETE statements. Typically, they are included in the WHERE clause (as the above example demonstrates) but they can also be included in the SELECT list. The above subquery is an example of a **single subquery** and will always be surrounded by parentheses.

You will now look at nested subqueries, which allow you to nest (or place) subqueries inside of other subqueries.

Subqueries general rules

- A subquery must follow an operator in a search condition
- The SELECT clause in a subquery can contain one item only – except if preceded by an EXISTS or IN operator
- A subquery used with an ordinary comparison operator (e.g., =,<, <=) returns a single value, otherwise a SET operator should be used
- The data type of the item returned by the subquery must match that of the item preceding the operator

IN and =ANY operators

When used with a subquery, the word IN is an alias for =ANY. Thus, these two statements are the same.

Example using **IN** :

```
SELECT s1 FROM t1 WHERE s1 IN (SELECT s1 FROM t2);
```

Example using **= ANY** :

```
SELECT s1 FROM t1 WHERE s1 = ANY (SELECT s1 FROM t2);
```

ALL, <>ALL and NOT IN keywords

The keyword **ALL** , which must follow a comparison operator, means "return TRUE if the comparison is TRUE for ALL of the values in the column that the subquery returns." **NOT IN** is an alias for <> **ALL**.

Example

```
SELECT s1 FROM t1 WHERE s1 > ALL (SELECT s1 FROM t2);
```

For **s1 > ALL** expression to be TRUE all values in the subquery must be less than the value of s1 (or s1 is the max value)

Nested subquery

A nested subquery is one that is positioned inside another subquery. The innermost subqueries are executed before the outer subqueries. Like the single subquery, the nested subquery also works with SELECT, INSERT, UPDATE, or DELETE statements. See the below example.

```
SELECT last_name, first_name
FROM employees
WHERE id IN
( SELECT employee_id FROM orders WHERE order_date =
( SELECT order_date FROM orders ORDER BY order_date DESC LIMIT 1 )
)
ORDER BY last_name, first_name;
```

The above query is also based on the Northwind database. It will display all employees that have placed orders on a particular date. This date is determined by a nested main subquery which will return the date of the most recent order in the orders table. Notice that the ORDER BY is part of the main query.

Correlated subquery

A correlated subquery is a subquery that uses a value from the outer query to evaluate which records to return in the inner query. This type of subquery is executed for **each** row of the outer table, they are also known as row-by-row processing. Table aliases are often used.

```
SELECT employee_id, manager_id, first_name, last_name
FROM employees a
WHERE EXISTS
(SELECT employee_id
FROM employees b
WHERE b.manager_id = a.employee_id)
```

The above correlated subquery will display all employees that manage other employees.

Weblink

For a detailed explanation and example of the differences between subqueries and correlated subqueries visit:

SQL World – Difference between subquery and correlated subquery[39]

Also a good tutorial about MySQL subqueries can be found at:

MYSQL Tutorial - Subqueries[40]

Activity

Using the Northwind database, write an SQL statement that shows details of all orders that have been invoiced, including the order ID, customer name, order date, products ordered (product names), date shipped and order status.

[39] https://www.complexsql.com/subqueries-correlated-subquery/

[40] https://www.mysqltutorial.org/mysql-subquery/

Topic summary and review

In this topic you have learnt about how to work with multiple tables using various methods including JOINs, UNION, and subqueries. You learned about the different JOIN methods (including INNER, LEFT, RIGHT and CROSS JOINs) which establish a link (or relationship) between two or more tables, based on a common column (field). Using JOINs, you can retrieve related data from multiple tables, and present this data in meaningful ways.

You also learnt about the benefits of using table aliases to improve the readability of particularly complex queries. To gain a deeper understanding of the syntax covered in this topic visit the links provided and follow the tutorials.

Self-check – Working with multiple tables in SQL

Please read the questions/statements and answer using the multiple-choice answers.

What is contained within this SQL statement?

```
SELECT last_name, first_name FROM employees WHERE id IN (SELECT
employee_id FROM orders WHERE order_date < '2006-02-01') ORDER BY
last_name, first_name;
```

- ☐ A JOIN
- ☐ A UNION
- ☐ A table alias
- ☐ An inner query

Which of the options below are contained within this SQL statement?

```
SELECT s.name AS Student, c.name AS Course
FROM students
INNER JOIN bridge b ON s.id = b.sid
INNER JOIN course c ON b.cid = c.id
ORDER BY s.name;
```

- ☐ At least one column alias
- ☐ At least two joins
- ☐ At least three tables
- ☐ All of the above

Read the query below and from the four possible answers below, select which outcome will be returned?

```
SELECT Customers.CustomerName, Orders.OrderID
FROM Customers
LEFT JOIN Orders
ON Customers.CustomerID=Orders.CustomerID ORDER BY
Customers.CustomerName;
```

- ☐ All orders even if there are no matching customers
- ☐ All orders and all customers
- ☐ All customers who have placed an order
- ☐ All customers including those with no orders

The UNION ALL operator is best used in which of the following circumstances?

- ☐ You want to query multiple tables
- ☐ You prefer not to use a JOIN
- ☐ You want to query multiple tables and show duplicates
- ☐ You want to query multiple tables without showing duplicates

A RIGHT OUTER JOIN returns unmatched rows from which of the following tables?

- ☐ First
- ☐ Second
- ☐ INNER
- ☐ OUTER

Check your answers at the end of this Learner Guide

Advanced SQL: Views, stored procedures, triggers and transactions.

Introduction

In this topic you will explore the use of views, stored procedures, triggers, and transactions.

Views allow you to retrieve data from the database and 'view' the data from different perspectives. By storing SQL statements in the database as a 'view', you can hide the underlying complexity from database users.

Stored procedures allow us to create SQL statements that are stored in the database server. You can call them using a procedure call. These procedures can be executed repeatedly, using different parameters as input (if parameters have been declared).

Triggers are used to validate the data you insert, update, or delete.

Finally, transactions are used to execute multiple statements in one single transaction or action.

Objectives

By successfully completing this topic you will be able to do the following:

- understand, create and manage views
- understand the purpose of stored procedures and learn to create and apply them
- understand the purpose of triggers and learn to apply them to validate data
- understand the purpose of transactions and how to apply them using COMMIT and ROLLBACK procedures.

437747449 / dizain / shutterstock.com. Modified by TAFE Queensland.

What is a view?

A view can be described as a 'virtual table'. It has a name associated with it and it comprises a SQL statement consisting of one or more references to different tables and columns, which, when written, represents a unique 'view' (or perspective) of the data retrieved.

44106241 / cybrain / shutterstock.com.

This gives the impression that all the data is coming from a single table when in fact it is being retrieved from multiple tables.

You use views to store complex SQL statements and this complexity is hidden from the user who is executing the view. The user does not have to remember the sequence of joins to recreate a view, they can simply execute it by referring to the view name.

A view is created using the **CREATE VIEW** statement. Typically, in the workplace, views are created for the purpose of creating reports for management. They allow for easy execution of a query.

The following example has been taken from the w3schools website:

```
CREATE VIEW Current_Product_List AS
SELECT ProductID, ProductName
FROM Products
WHERE Discontinued = No
```

Weblink

Check the link to learn more and view the SQL CREATE VIEW statement[41]

[41] https://www.w3schools.com/sql/sql_view.asp

Here, a view is created that displays the product ID and name of all current products that have not been discontinued. When you execute the SQL statement, the view is created and saved within the database. Once saved, the View is treated like a virtual table that you can query using a SELECT clause:

```
SELECT * FROM Current_Product_List
```

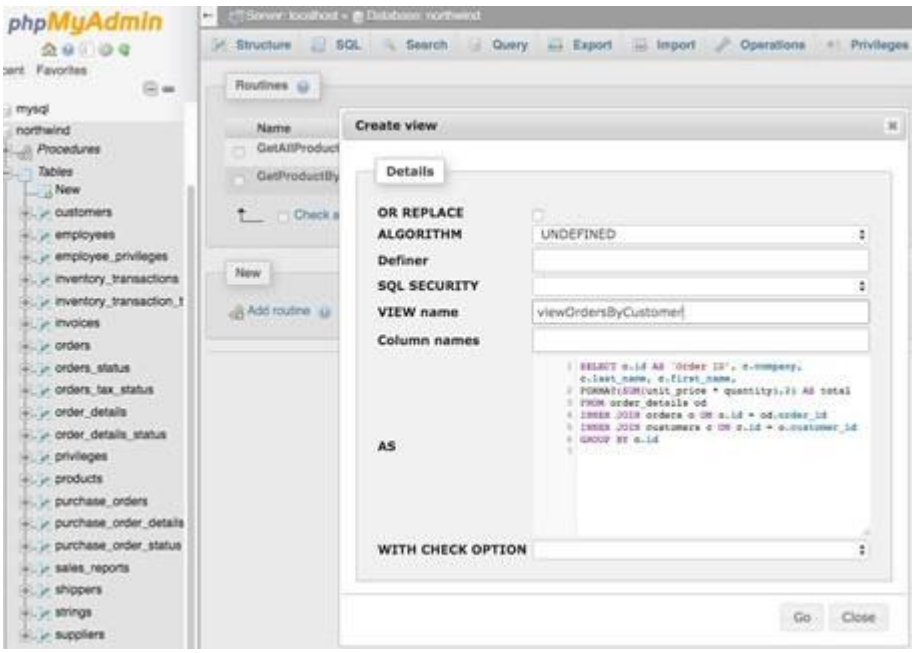
You can also create a View using phpMyAdmin.

The following query, based on the Northwind database, will display a list of orders (grouped by order ID). The customer details are shown along with the order total.

```
SELECT o.id AS 'Order ID', c.company, c.last_name, c.first_name,
FORMAT(SUM(unit_price * quantity),2) AS total
FROM order_details od
INNER JOIN orders o ON o.id = od.order_id
INNER JOIN customers c ON c.id = o.customer_id
GROUP BY o.id
```


Execute the query

Once you have written an SQL query in the SQL tab, you click on 'Go' to execute the query. The results of the query are displayed. Below these results are a number of links, the last of which is 'Create view'. Clicking on this link will display a details window (the SQL statement will automatically be added to the View):



Screenshot of phpMyAdmin software, 16/12/21.GNU General Public License, version 2 .

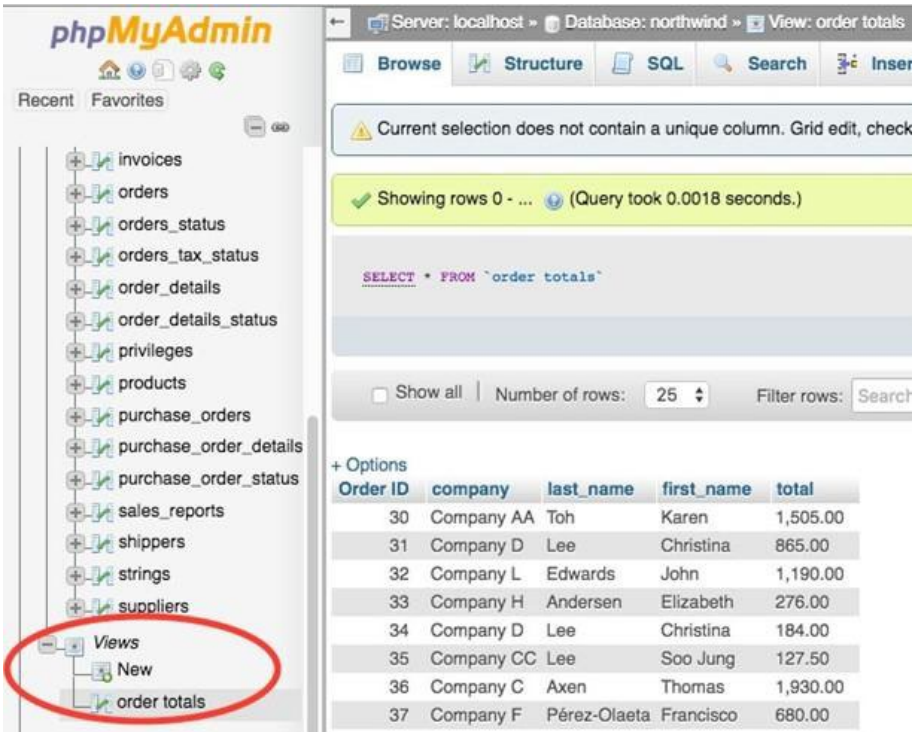
Using phpMyAdmin to create a View called 'viewOrdersByCustomer'.

There are several options here. The 'OR REPLACE' checkbox, if ticked, will allow you to edit and update an existing View. If creating a new View, this should be left unticked. Enter the View name, then click on 'Go'. The new View has been created and can be executed by clicking on the View name in the left-side windowpane.

Important

If you wish to update the SQL statement associated with an existing view, place a tick in the 'OR REPLACE' checkbox. Type the name of the existing view in the 'VIEW name' textbox, enter the new SQL statement that will become the View, then click on 'Go'.

In MySQL once a view is no longer required, it can be removed from the database using the **DROP VIEW** statement. DROP VIEW Current_Product_List;



Executing the view.

Screenshot of phpMyAdmin software,16/12/21.GNU General Public License, version 2

One major advantage in using views is that you can assign a particular database user to a view without giving them access to an entire table. For example, a view may contain an SQL statement that queries tables which may contain sensitive information such as employee salaries. Using a view, you can specify which columns of a table the user can query without giving them access to all columns within a table.

Weblink

For more information and examples on creating and working with views visit:

MySQL Tutorial – Introduction to MySQL Views[42]

MySQL Tutorial – Creating Updatable views[43]

[42] https://www.mysqltutorial.org/mysql-views-tutorial.aspx

[43] https://www.mysqltutorial.org/create-sql-updatable-views.aspx

Activity

Using the Northwind database, create a View that will add 10% GST to the list price (contained in the 'products' table). The view will return a list of products with two prices – including and excluding GST. Execute the View and record results.

Self-check - Transactions

Please complete the question/s below

Which of the following statements best describes a VIEW?

- ☐ A virtual table based on a query
- ☐ A function used to process input
- ☐ An imported table
- ☐ A virtual table based on a transaction

A new view has been added to the database, called ‘Products Above Average Price’. How is this executed? Choose the appropriate option below:

- ☐ CALL [Products Above Average Price]
- ☐ EXECUTE [Products Above Average Price]
- ☐ [Products Above Average Price]
- ☐ SELECT * FROM [Products Above Average Price]

The main advantage in using views is:

- ☐ Encapsulation
- ☐ Security
- ☐ Usability
- ☐ All of the above

A manager has asked you to create a reusable query that will allow her to easily retrieve the latest sales data. You have suggested that the best approach would be to use a view. From the options below, which command would you use to add a new view to the database?

- ☐ ADD VIEW
- ☐ SELECT VIEW
- ☐ CREATE VIEW
- ☐ INSERT VIEW

What clauses or commands cannot be present in the SELECT statement used to create a VIEW if you want to create an updatable view?

- ☐ GROUP BY
- ☐ DISTINCT
- ☐ HAVING
- ☐ All of the above

Check your answers at the end of this Learner Guide

Stored procedures

A stored procedure is a group of one or more SQL statements that are stored in the data dictionary or catalogue. Stored procedures can be called from the command line, other procedures, or remote programs.

The main **advantages** of using stored procedures are as follows:

- efficiencies are gained in executing stored procedures as they are locally stored within the database (therefore reducing network traffic)
- stored procedures are stored as part of the database and therefore are not dependent on specific language syntax (for example, PHP, C#, or Java) to execute one or more SQL statements. This means that they are portable and can be executed on any platform that can run MySQL
- stored procedures can execute multiple SQL statements whereas views can only execute single SQL statements
- stored procedures can be used to modify tables however views cannot
- stored procedures accept parameters (values) however views do not
- security benefits as the procedures are store in the database server.

Disadvantages include:

- maintenance of the procedures as each table may have a number of procedures
- if you use stored procedures as a basic CRUD – you'll need insert, select update, and delete on each table. Multiply this by the number of tables
- no version control
- difficult to backup and export.

Creating a stored procedure

A stored procedure is created using the **CREATE PROCEDURE** statement.

Example

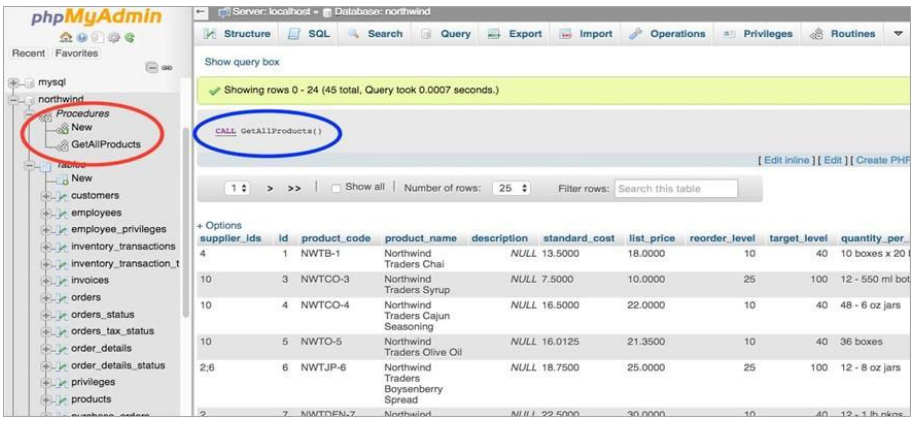
```
DELIMITER //
CREATE PROCEDURE GetAllProducts()
BEGIN
SELECT * FROM products;
END //
DELIMITER ;
```

In the above example, based on the Northwind database, the DELIMITER keyword is used to tell MySQL to substitute the semi-colon (;) for two forward slashes (//). This ensures that the whole procedure is executed as one block rather than separate statements. At the end of the procedure the DELIMITER is changed back to a semicolon.

To execute this in phpMyAdmin you click on the SQL tab and type the CALL statement followed by the name of the Stored Procedure, which, in the example above, is "GetAllProducts();

CALL GetAllProducts();

Selecting the 'Go' button will then execute the query contained within the procedure and display the results in the output pane, as shown below.

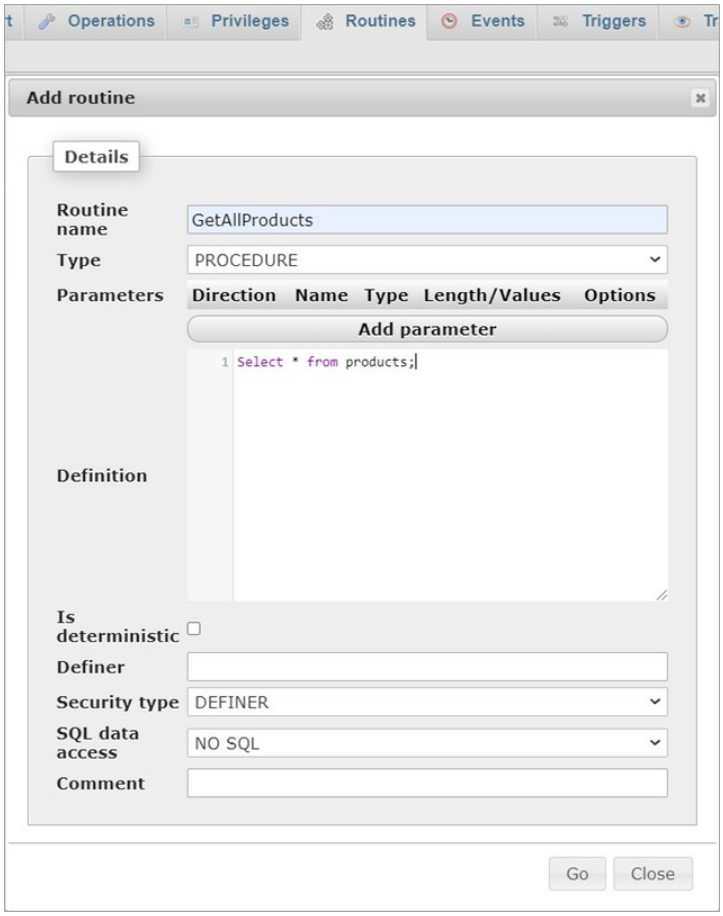


Executing the view.

Screenshot of phpMyAdmin software, 16/12/21.GNU General Public License, version 2

The procedure has now been saved for future use. The red circle in the screen shot above shows that the stored procedure has been saved under a 'Procedures' folder. From here you can click on the procedures name (GetAllProducts) to make changes to it. Alternatively, you can click on the 'New' link (located within the same red circle), which provides an alternative way of creating a new stored procedure using the phpMyAdmin GUI (graphical user interface).

Alternatively, you can use the PhpMyAdmin Routine TAB then 'ADD routine' to create a procedure. Then complete the box displayed below.



Creating a stored procedure using the GUI of phpMyAdmin

Screenshot of phpMyAdmin software, 16/12/21.GNU General Public License, version 2

Create store procedure using parameters

Parameters can be used with stored procedures to make them more dynamic. There are three types of parameters used for different purposes, IN, OUT and INOUT parameters. In this section you are concentrating on the IN parameters. At the end of the section links are provided to further your understanding of stored procedures parameters.

The IN parameter is used to supply a value to a specific condition that is contained within the stored procedure. For instance, you can create a procedure to retrieve a specific product based on the product ID (supplied by the CALL statement).

To create this procedure, you would execute the following:

```
DELIMITER //
CREATE PROCEDURE GetProductByID(IN pID INT(11))
BEGIN
SELECT *
FROM products
WHERE id = pID;
END //
DELIMITER;
```

Now to execute the above procedure you again use the CALL statement and specify the parameter (in this case the product ID you want to look up) within parentheses:

```
CALL GetProductByID(34);
```

This procedure retrieves all records that have a product ID of 34.

Insert and update data

Stored procedures can also be created and used to INSERT new records into a database table. When creating a procedure of this type, it is necessary to specify each of the IN parameters that will be used as values for the INSERT query inside the procedure.

Weblink

More information is illustrated in this example from MySQL Information and Resources[44]

Notice that when you define the stored procedure, you also need to define the data type for each IN parameter. This has advantages in terms of security, as the database will 'know' what data type to expect and thus prevent SQL injection. Another advantage is that if you are building a web-based application, you only need to call the stored procedure by name within the code (you don't need to specify INSERT statement which contains the table and column names).

```
DELIMITER //
CREATE PROCEDURE insertProductLine(
IN pl varchar(50),
IN td varchar(50),
IN hd varchar(255))
BEGIN
INSERT INTO productLines values (pl, td, hd);
END //
DELIMITER;
```

[44] https://dev.mysql.com/doc/refman/8.0/en/insert.html

Stored procedures can also be used to UPDATE existing records in a database table, as shown in the example below:

```
DELIMITER //
CREATE PROCEDURE updateAddress(
IN al1 VARCHAR(255),
IN al2 VARCHAR(255),
IN oc INT(10))
BEGIN
UPDATE offices
SET
addressLine1 = al1,
addressLine2 = al2
WHERE officeCode = oc;
END //
DELIMITER;
```

Remove stored procedures

Stored procedures can be dropped (or deleted) from the database using the DROP PROCEDURE command.

This has the following syntax:

```
DROP PROCEDURE [IF EXISTS] [procedure_name]
DROP PROCEDURE updateAddress;
```

Weblink

Check the link to obtain more information and examples on how to use stored procedures[45] at the SQLShack website.

Activity

1. Using the Northwind database, create a stored procedure that will accept an input parameter (product ID). The stored procedure takes the product ID, looks up the corresponding product, adds 10% GST to the list price of that product, and returns the result. Execute the stored procedure to ensure it performs as expected.
2. Using the Northwind database, create a stored procedure that will insert a new record into the customer's table.
3. Using the Northwind database, create a stored procedure that will update the business phone and fax number for the customer that you created in the previous activity (2).

[45] https://www.sqlshack.com/learn-mysql-the-basics-of-mysql-stored-procedures/

Self-check – Stored procedures

Please complete the question/s below

If you have been given a task to create a stored procedure which can accept two integer parameters and the procedure includes multiplying the two integers together which results in the first parameter. What is the first parameter an example of?

- ☐ IN
- ☐ OUT
- ☐ INOUT
- ☐ None of the above

Which statement best describes a stored procedure?

- ☐ Named pre-compiled collection of SQL statements
- ☐ Random execution of SQL statements
- ☐ Pre-defined list of mathematical functions
- ☐ Pre-defined set of date and time functions

How is a stored procedure executed?

- ☐ By using the CALL statement
- ☐ By using the stored procedure name
- ☐ By using the ACTION statement
- ☐ By specifying the procedure name in a SELECT statement

What is the purpose of an INSERT command inside a stored procedure?

- ☐ Reinstall archived documents
- ☐ Update system events
- ☐ Create new record
- ☐ Update user privileges

Which of the statements below would you use to remove a procedure from the database?

- ☐ DELETE PROCEDURE
- ☐ DROP PROCEDURE
- ☐ REMOVE PROCEDURE
- ☐ DELETE FROM

Check your answers at the end of this Learner Guide

Triggers

Triggers in SQL

A trigger is an instruction which describes what the database should do if DELETE, UPDATE, or INSERT action condition is not met. The operation will stop and an appropriate message will alert the user of the problem.

175199537 / mindscanner / shutterstock.com.

In MySQL, triggers can also be used to maintain **data integrity**. Other relational databases support the **CHECK CONSTRAINT** for this purpose. Support for check table constraints in MYSQL is very recent, it started from MySQL 8.0.16. Prior to that version, if MySQL encountered the corresponding CHECK syntax it did not action it; it did not output an error either as check is part of the SQL syntax.

Triggers are an alternative to check constraints and can be used for:

- before or after an insert operation
- before or after an update operation
- before or after a delete operation.

The example below shows how to create a new trigger that activates when an UPDATE is performed on the Employee table. The trigger is being used here to track changes made to the 'employees' table.

The trigger will insert a new record into the 'employees_audit' table, recording the date and time when the update occurred, along with the employee number and last name.

The **OLD** keyword is used here to refer to the record **before** it is updated. The **NEW** keyword can also be used here, which refers to the record after it has been updated.

The INSERT statement will be executed **FOR EACH ROW** that is being updated.

In this example, the trigger will activate **BEFORE** the **UPDATE**.

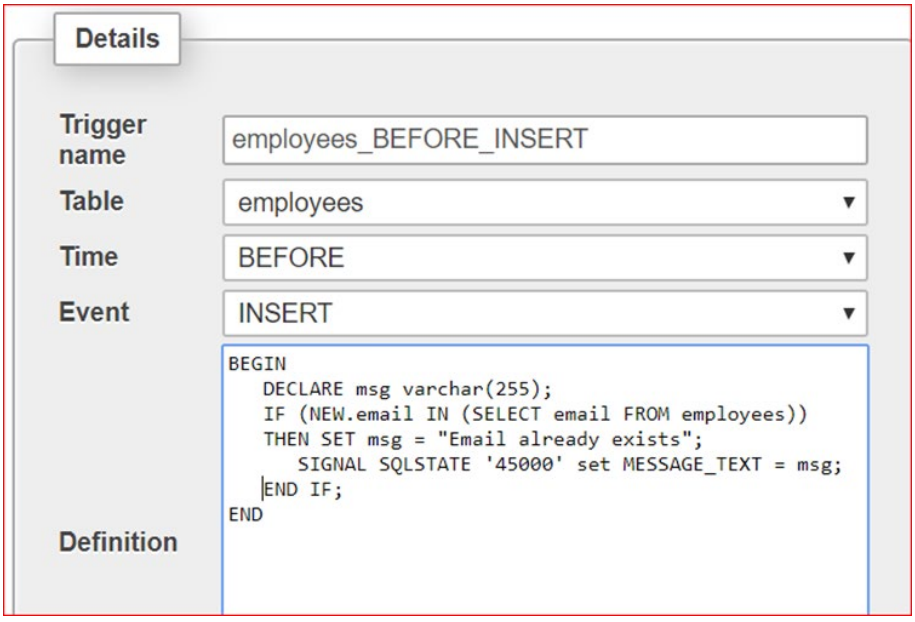
Example

A medical centre needs to be able to track when changes have been made to the 'employees' table in their database. A trigger is created that activates immediately after an employee record has been updated, inserting a new record in an 'employee audit' table with details of the change, and the date and time that the change occurred.


```
DELIMITER $$
CREATE TRIGGER before_employee_update
BEFORE UPDATE ON employees
FOR EACH ROW
BEGIN
INSERT INTO employees_audit
SET action = 'update',
employeeNumber = LD.employeeNumber,
lastname = OLD.lastname,
changedate = NOW();
END$$
DELIMITER;
```

Triggers can also be created in PhpMyAdmin by accessing the TRIGGERS tab and then selecting 'ADD new trigger'.

The example below displays a trigger created to stop recording the same email address for more than one employee. The keyword NEW (NEW.email) represents the email address that the user is attempting to insert into the employee table.



Trigger to check email address

Screenshot ofphpMyAdmin software, 16/12/21.GNU General Public License, version 2

Weblink

For more information and examples on using triggers visit: MySQL.com - Triggers[46]

Self-check - Triggers

Please complete the question/s below

If CHECK constraints are not available, what from the options below can be used to replace CHECK constraints?

- ☐ Tiggers
- ☐ Tridents
- ☐ Trojans
- ☐ Triggers

What can triggers do?

- ☐ Check the query syntax
- ☐ Control input before/after an event
- ☐ Save query results permanently
- ☐ Update the information_schema table

Of the following options, what cannot be created in views?

- ☐ Database alerts and warnings
- ☐ Pin execution notification
- ☐ De-normalisation schemas
- ☐ FOR or AFTER triggers

[46] https://dev.mysql.com/doc/refman/8.0/en/trigger-syntax.html

What type of table can have unlimited triggers for each event type (INSERT, UPDATE, DELETE) of the options below?

- ☐ A single table
- ☐ A double table
- ☐ A triple table
- ☐ A quad table

Referring to what you have learnt and the theme of this quiz, what can be used to maintain data integrity?

.....

.....

.....

Check your answers at the end of this Learner Guide

Working with transactions

MySQL transaction allows you to execute several operations in the same transaction statements, meaning that the database never records partial operations. If one operation fails, the transaction is rolled back to the original state. If each operation included in the transaction is successful, all operations are saved (committed) to the database.

The transaction displayed below:

- inserts one record, into the first table
- assigns the primary key value of the first table to **@lastauthor**
- inserts a record into the second table inserting the **@lastauthor** value into the foreign key column
- assigns the primary key value of the second table to **@lastbook**
- inserts a record into the third table inserting the **@lastbook** value into the foreign key column.

If one of the inserts fails, the transaction will not commit (save) any of the insertions.

The basic syntax structure of a transaction is:

```
START TRANSACTION;
Statements
…
COMMIT;
```

Example of a transaction to insert a record into three tables:

```
START TRANSACTION;
INSERT INTO author
VALUES (AuthorID, 'Agatha', Christie', 'British', '1890', '1976');
SET @lastauthor = LAST_INSERT_ID();
INSERT INTO books
VALUES (BookID, 'And Then There Were None', 'Ten Little Niggers',
'1939', 'Mystery', '100', 'English', last_insert_id(@lastauthor));
SET @lastbook= LAST_INSERT_ID();
INSERT INTO bookrankings
VALUES (RankingID, '6', last_insert_id(@lastbook));
COMMIT;
```

If you rollback the transaction, no insert actions would be recorded. The statement is:

```
/* Rollback before the transaction was started */
ROLLBACK;
```

Transactions can combine different actions, for example, INSERT, SELECT UPDATE, and DELETE. It will depend on the outcome that you need to achieve.

Weblink

For more information and examples about using transactions visit: MySQL tutorial - Transactions[47]

[47] https://www.mysqltutorial.org/mysql-transaction.aspx

Topic summary and review

In this topic you have learnt about subqueries, stored procedures, triggers, and transactions including the advantages of using them, how to create them, how to edit and update them, and how they are activated.

To gain a better understanding of the advanced features learnt in this topic visit the sites provided.

Activity

Using the Northwind database, create a transaction to update an order, specifically, order_id 30 for product_id 34 in the order_details table. The customer has called and has advised that the quantity (quantity ordered) should be changed to 500. Change the quantity and them apply a 2.5% discount to the unit_price for that order only.

Self-check - Transactions

Please complete the following question/s

What is the purpose of the COMMIT statement in a transaction?

- ☐ discard current value
- ☐ save in temporary memory
- ☐ save to database permanently

What is the purpose of the ROLLBACK statement in a transaction?

- ☐ validate the data
- ☐ restore data to original state before transaction
- ☐ use checksum to test data integrity

A transaction terminates when it?

- ☐ has been committed
- ☐ has been rolled back
- ☐ has been committed or rolled back

Which of the statements below can maintain transactions in primary and foreign key relations?

- ☐ UPLOAD
- ☐ INOUT
- ☐ INSERT
- ☐ OUT

Check your answers at the end of this Learner Guide

Self-check answers

Self-check – Relational databases

Please complete the following questions

Relational databases contain what type of keys to identify relationships between tables?

- ☐ Primary and secondary
- ☐ Primary and tertiary
- ☐ Primary and major
- ☑ Primary and foreign

In what decade did Dr Edgar F. Codd introduce relational databases?

- ☐ 1980s
- ☑ 1960s
- ☐ 1990s
- ☐ 1950s

Please select the three scientific disciplines used in data mining?

- ☑ Statistics
- ☑ Machine learning algorithms
- ☐ Counter Intelligence
- ☑ Artificial Intelligence

Data mining was previously known by which of the following:

- ☐ Data knowledge collation
- ☐ Data search and pattern finding
- ☑ Knowledge discovery in data
- ☐ knowledge-based data search

A business infrastructure is comprised of which of the following?

- ☐ software
- ☐ policies and procedures
- ☐ people and their skills / human resources
- ☐ hardware
- ☑ all of the above

Self-check – Structured Query Language

Please read each question and make your selection from the multiple-choice options provided.

What does XAMPP stand for?

- ☑ Cross-platform Apache, MySQL, PHP and Perl
- ☐ Cross-platform Apache, MySQL, PHP and Python
- ☐ eXtensible Apache, MySQL, PHP and Perl
- ☐ eXtensible Apache, MySQL, PHP and Python

What type of database model consists of tables in an upside-down tree that defines parent/child relationships?

- ☐ Tiered
- ☐ Logical
- ☑ Hierarchical
- ☐ Pyramid

What is the name of the MySQL data type used to store large text values?

- ☐ Word
- ☑ Text
- ☐ Number
- ☐ Letter

From the options below, which attribute automatically assigns a unique ID value to a new record when inserted?

- ☐ AUTO_INSERT
- ☐ AUTO_ID
- ☐ AUTO_ENTER
- ☑ AUTO_INCREMENT

Which of the following SQL statements will list all customers by their city location (in alphabetical order), and then by their name (in reverse alphabetical order)?

- ☐ SELECT CustomerName, City FROM Customers ORDER BY City DESC CustomerName ASC;
- ☑ SELECT CustomerName, City FROM Customers ORDER BY City ASC CustomerName DESC;
- ☐ SELECT CustomerName, City FROM Customers ORDER BY City DESC CustomerName;
- ☐ SELECT CustomerName, City FROM Customers ORDER BY City DESC CustomerName DESC;

Self-check – Structured query language

Please complete the question/s below:

Examine the following syntax and select the correct statement from the options below:

```
SELECT *
FROM Customers
WHERE (state = 'Qld' OR last_name = 'Smith') AND customer_id < 200;
```

- ☐ The query will show all customers where customer ID is less than 200
- ☐ The query will show customers that either live in Qld or have a last name of Smith
- ☑ The query will show customers that either live in Qld or have a last name of Smith, and where the customer ID is less than 200
- ☐ The query will show all customers that live in Qld, including any that have a customer ID of less than 200 and a last name of Smith

If you enter the following query, what results will you see?

```
SELECT *
FROM customers
WHERE customer_name LIKE 'TOM%'
```

- ☐ Return any customer records where the customer's name starts with "TOM"
- ☐ Return any customer records where the customer's name starts with "TOM%"
- ☑ Return any customer records where the customer's name does not start with "TOM"
- ☐ Return any customer records where the customer's name starts with "CUSTOMER_TOM"

Examine the following query. What will be returned when it is executed?

```
SELECT *
FROM customers
WHERE customer_name LIKE '[TOM]%' ;
```

- ☐ Return any customer records where the customer's name contains the phrase 'TOM'
- ☐ Return any customer records where the customer's name starts with the phrase 'TOM'
- ☐ Return any customer records where the customer's name contains either a 'T', 'O', or 'M'
- ☑ Return any customer records where the customer's name starts with either a 'T', 'O', or 'M'

Which of the following wildcards is NOT supported in most relational databases, including MySQL?

- ☑ [charlist]
- ☐ [chargelist]
- ☐ [chartlist]
- ☐ [changelist]

You need to find all 4-digit account numbers in the database but you only have the first 3 digits. Which wildcard is best to use to find all matching account numbers using just the first 3 digits?

- ☐ %
- ☑ _ (underscore)
- ☐ !
- ☐ %!

Self-check – Working with logical operators

Please complete the following question/s

Which outcome will be returned when executing the following query ?:

```
SELECT STRCMP('Acme Printing Company', 'Creative Dance Academy')
```

- ☐ zero (0)
- ☑ minus 1 (-1)
- ☐ one (1)
- ☐ (NULL

From the query below, when will the condition that checks for the last name of the employee be tested in relation to the aggregated condition?

```
SELECT Employees.LastName,
COUNT(Orders.OrderID) AS NumberOfOrders FROM
(Orders INNER JOIN Employees ON
Orders.EmployeeID=Employees.EmployeeID) WHERE
LastName = 'Buchanan' GROUP BY LastName
HAVING COUNT(Orders.OrderID) > 10;
```

- ☑ It will be tested before the aggregated condition
- ☐ It will be tested after the aggregated condition
- ☐ Both will be tested at the same time
- ☐ It will not be tested as the aggregated condition will override it

‘Address’ in the following query is an example of what SQL feature?

```
SELECT CustomerName, Street +', '+Suburb+', '+Postcode+',
'+Country AS Address FROM Customer;
```

- ☑ Aggregate function
- ☐ String function
- ☐ CONCAT() function
- ☐ Alias name

Of the following options, what outcome will be returned when this query is executed?

```
SELECT COUNT(*) FROM Orders;
```

- ☐ The total number of unique orders stored in the Orders table
- ☐ The total number of orders in the Orders table (including duplicates)
- ☑ The total number of records in the Orders table
- ☐ The grand total of all orders stored in the Orders table

When executing the following query, what will occur from the options below:?

```
SELECT DATE_FORMAT(NOW(), '%d/%m/%y')
```

- ☐ Return the current date and time
- ☐ Return the current date in Australian date format
- ☐ Return the date of last record insert in Australian format
- ☑ Return the current date and time when the last record was inserted

Self-check – Working with multiple tables inSQL

Please read the questions/statements and answer using the multiple-choice answers.

What is contained within this SQL statement?

```
SELECT last_name, first_name FROM employees WHERE id IN (SELECT
employee_id FROM orders WHERE order_date < '2006-02-01') ORDER BY
last_name, first_name;
```

- ☐ A JOIN
- ☐ A UNION
- ☐ A table alias
- ☑ An inner query

Which of the options below are contained within this SQL statement?

```
SELECT s.name AS Student, c.name AS Course
FROM students
INNER JOIN bridge b ON s.id = b.sid
INNER JOIN course c ON b.cid = c.id
ORDER BY s.name;
```

- ☐ At least one column alias
- ☐ At least two joins
- ☐ At least three tables
- ☑ All of the above

Read the query below and from the four possible answers below, select which outcome will be returned?

```
SELECT Customers.CustomerName, Orders.OrderID
FROM Customers
LEFT JOIN Orders
ON Customers.CustomerID=Orders.CustomerID ORDER BY
Customers.CustomerName;
```

- ☐ All orders even if there are no matching customers
- ☐ All orders and all customers
- ☐ All customers who have placed an order
- ☑ All customers including those with no orders

The UNION ALL operator is best used in which of the following circumstances?

- ☐ You want to query multiple tables
- ☐ You prefer not to use a JOIN
- ☑ You want to query multiple tables and show duplicates
- ☐ You want to query multiple tables without showing duplicates

A RIGHT OUTER JOIN returns unmatched rows from which of the following tables?

- ☐ First
- ☑ Second
- ☐ INNER
- ☐ OUTER

Self-check - Views

Please complete the question/s below

Which of the following statements best describes a VIEW?

- [x] A virtual table based on a query
- [ ] A function used to process input
- [ ] An imported table
- [ ] A virtual table based on a transaction

A new view has been added to the database, called ‘Products Above Average Price’. How is this executed? Choose the appropriate option below:

- [ ] CALL [Products Above Average Price]
- [ ] EXECUTE [Products Above Average Price]
- [ ] [Products Above Average Price]
- [x] SELECT * FROM [Products Above Average Price]

The main advantage in using views is:

- [ ] Encapsulation
- [ ] Security
- [ ] Usability
- [x] All of the above

A manager has asked you to create a reusable query that will allow her to easily retrieve the latest sales data. You have suggested that the best approach would be to use a view. From the options below, which command would you use to add a new view to the database?

- [ ] ADD VIEW
- [ ] SELECT VIEW
- [x] CREATE VIEW
- [ ] INSERT VIEW

What clauses or commands cannot be present in the SELECT statement used to create a VIEW if you want to create an updatable view?

- ☐ GROUP BY
- ☐ DISTINCT
- ☐ HAVING
- ☑ All of the above

Self-check – Stored procedures

Please complete the question/s below

If you have been given a task to create a stored procedure which can accept two integer parameters and the procedure includes multiplying the two integers together which results in the first parameter. What is the first parameter an example of?

- ☐ IN
- ☐ OUT
- ☑ INOUT
- ☐ None of the above

Which statement best describes a stored procedure?

- ☑ Named pre-compiled collection of SQL statements
- ☐ Random execution of SQL statements
- ☐ Pre-defined list of mathematical functions
- ☐ Pre-defined set of date and time functions

How is a stored procedure executed?

- ☑ By using the CALL statement
- ☐ By using the stored procedure name
- ☐ By using the ACTION statement
- ☐ By specifying the procedure name in a SELECT statement

What is the purpose of an INSERT command inside a stored procedure?

- ☐ Reinstall archived documents
- ☐ Update system events
- ☑ Create new record
- ☐ Update user privileges

Which of the statements below would you use to remove a procedure from the database?

- ☐ DELETE PROCEDURE
- ☑ DROP PROCEDURE
- ☐ REMOVE PROCEDURE
- ☐ DELETE FROM

Self-check - Triggers

Please complete the question/s below

If CHECK constraints are not available, what from the options below can be used to replace CHECK constraints?

- ☐ Tiggers
- ☐ Tridents
- ☐ Trojans
- ☑ Triggers

What can triggers do?

- ☐ Check the query syntax
- ☑ Control input before/after an event
- ☐ Save query results permanently
- ☐ Update the information_schema table

Of the following options, what cannot be created in views?

- ☐ Database alerts and warnings
- ☐ Pin execution notification
- ☐ De-normalisation schemas
- ☑ FOR or AFTER triggers

What type of table can have unlimited triggers for each event type (INSERT, UPDATE, DELETE) of the options below?

- ☑ A single table
- ☐ A double table
- ☐ A triple table
- ☐ A quad table

Referring to what you have learnt and the theme of this quiz, what can be used to maintain data integrity?

Triggers

Self-check - Transactions

Please complete the following question/s

What is the purpose of the COMMIT statement in a transaction?

- ☐ discard current value
- ☐ save in temporary memory
- ☑ save to database permanently

What is the purpose of the ROLLBACK statement in a transaction?

- ☐ validate the data
- ☑ restore data to original state before transaction
- ☐ use checksum to test data integrity

A transaction terminates when it?

- ☐ has been committed
- ☐ has been rolled back
- ☑ has been committed or rolled back

Which of the statements below can maintain transactions in primary and foreign key relations?

- ☐ UPLOAD
- ☐ INOUT
- ☑ INSERT
- ☐ OUT

Glossary

Term	Meaning
Aggregate function	An aggregate function will return a single value which is based on a calculation of values stored in multiple rows. For example: SELECT SUM(orderTotal) FROM orders This aggregate function will return a single total value of all orders placed.
Cardinality	In database design, cardinality refers to the type of relationship between two tables. This can be described as 'many-to-many', or 'one-to-many'. An example of a 'one-to-many' relationship: a single customer may have many orders.
ERD	ERD stands for 'entity relationship diagram' and is typically used as a tool to visually conceptualise a database design. It can start as a simple conceptual model, showing various entities and the relationships between them. They can be further developed to show cardinality, tables, columns, data types, and primary and foreign keys.
Foreign key	A foreign key is defined in a second table which relates to the primary key in the first table. The primary and foreign keys share the same values and establish a link between two related tables. Foreign key constraints can be applied to the relationship which can prevent related records from being removed.
Index	An index can be defined on one or more columns, allowing database users to quickly locate records, in a comparable way to using an index in a book. For example, in a customer table, you could index the last name of each customer. Retrieving a customer using their last name can be achieved quickly using an index, compared to not using one. This would be akin to paging through a book starting at the front, to find the information, rather than using the index.
Joins	A JOIN is a way of combining two or more tables together in a query to obtain related data from multiple tables.
Normalisation	The process of normalising a database avoids data redundancy and ensures that all data is logically stored. Data redundancy is where the same piece of data is stored in two (or more) separate places.

Primary key	A primary key can be defined on one or more columns, to uniquely identify a particular row. A primary key defined across multiple columns is called a composite key.
Relational database	An organised and structured set of data held in interrelated tables. Each table typically shares data that is common with other tables, in columns that are defined in a primary/foreign key relationship.
Stored procedures	A stored procedure is a series of one or more SQL statements that are grouped together and stored locally in the database. Stored procedures, or procedures for short, can be executed by calling them by the name used to create them. It is possible to use SELECT, UPDATE, and INSERT queries, and subqueries, in a procedure. A procedure can also accept distinct types of parameter values that can be used in the SQL queries contained within the procedure.
Table	A table contains a series of rows (or records), and columns (or fields). Each column is of a particular data type, for example, text-based, numeric, or date-based. Each row typically holds unique data. For example, a table of customers.
UML	UML (unified modelling language) is a standard way used to visualise the design of a system. There are several UML diagrams that can be used to achieve this goal, including the use case diagram, class diagram, activity diagram, and sequence diagram.
Views	A view is a way of presenting data from a particular perspective. By creating a view, you create a 'virtual table' that can executed by specifying the view name in a SELECT statement. Views are ideally used when security may be of concern, or if you want to encapsulate complex queries.

