SQL essentials

Mastering database queries: a comprehensive guide to SQL

Learning session 01
Introduction

Instructor
Péter Fülöp
peteristvan.fulop@hcl.com

Welcome to the “SQL essentials” course

COURSE LEVEL	BEGINNER It covers fundamental concepts while also touching on a selection of **intermediate topics** to provide a comprehensive introduction to the subject matter.
COURSE FORMAT	10 SESSIONS 8 of these are instructional modules 2 mentor-guided learning sessions
PREREQUISITES	No previous experience with databases or SQL is necessary.
FORM OF KNOWLEDGE TEST	8 x assignments (homework) 1 x capstone project

Course overview

DATABASE PLATFORMS

Hands-on practices will involve the utilization of **PostgreSQL** version 11 or later.

IN-PERSON ATTENDANCE

While it is not mandatory, in-person attendance **it is highly encouraged** for optimal learning experience. **The capstone project presentation necessitates in-person participation**.

RECOMMENDED STUDY TIME

8 instructional session x 2 hours
8 homework x 60 minutes
1 capstone project x 4-6 hours

LANGUAGE

The course language will be either English or Hungarian, depending on the preferences of participants.

Requirements for Course Completion

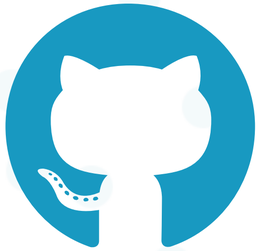
Each educational session is accompanied by homework assignments.
To earn **1 point**, your submitted homework must have correct answers for at least 3/4 of the questions.
Partially solved homework tasks can earn a fraction of a point. Moreover, for creative, outstanding, and innovative solutions, you have the opportunity to earn an additional point for each assignment.

8 x 1 point

To qualify for a final project assignment, you must accumulate a minimum of 6 points through the completion of homework assignments.

If someone prepares a presentation for the final project and delivers it, providing correct answers to at least half of the questions asked, they will receive a certificate confirming the successful completion of the "SQL essentials" course.

GitHub repository for the learning materials and practice files



A dedicated GitHub repository for the course is set up to house the course and practice exercise files at **https://github.com/ipeterfulop/sql-essentials-course**. Content will be progressively added to this repository as we proceed along our learning path.

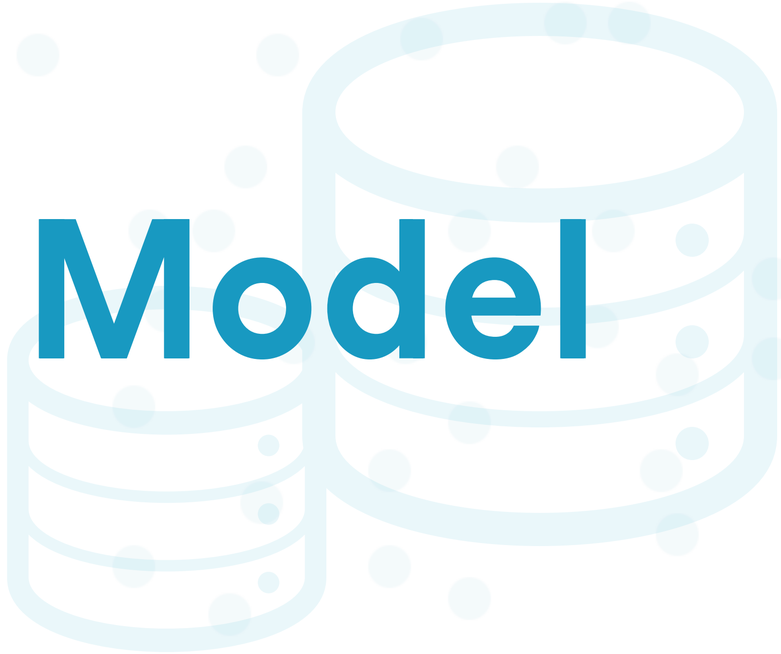
This repository will house the presentations, practice exercises, and homework assignments.

Focusing on Hands-On Exercises

This course places a significant emphasis on practical application, and ideally, we will dedicate the majority of our time to writing queries.

However, there are a few fundamental theoretical concepts that we must address.

The Relational Model



Origins of the Relational Model

Taking a leap back to the **early years of databases**, we find that in their infancy, each application stored data in its own unique format.

Developers, when building applications to utilize this data, needed a deep understanding of the specific data structure to access the required information.

These data structures were not only inefficient and challenging to maintain but also difficult to optimize for robust application performance.

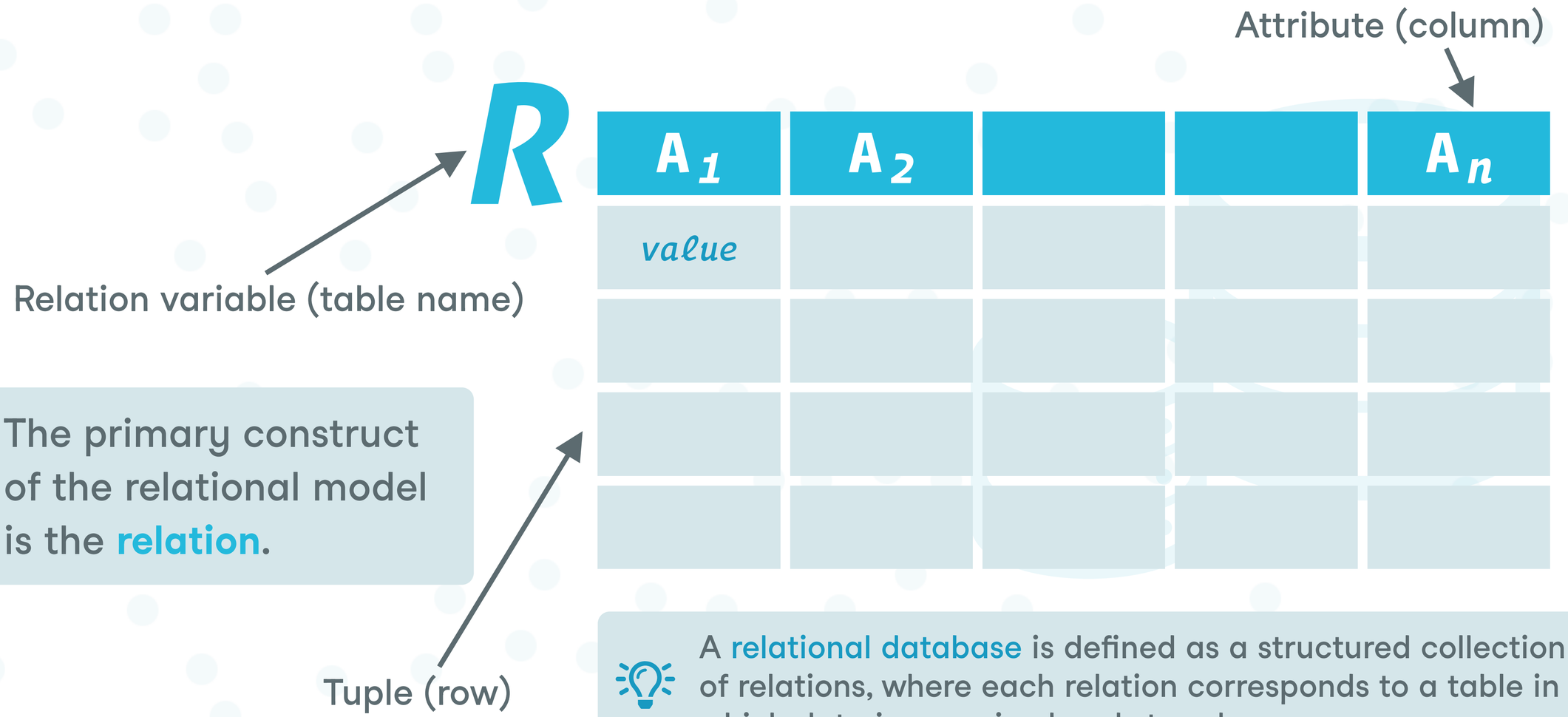
The relational database model was designed to solve the problem of multiple arbitrary data structures.

Definition of the Relational Model

The **relational model** (RM) is an approach to managing data using a **structure and language** consistent with **first-order predicate logic**, first described in 1969 by English computer scientist Edgar F. Codd, where all data is represented in terms of **tuples**, grouped into **relations**. (*https://en.wikipedia.org/wiki/Relational_model*)

A database organized in terms of the relational model is a **relational database**.

Elements of the Relational Data Model



The primary construct of the relational model is the **relation**.

A **relational database** is defined as a structured collection of relations, where each relation corresponds to a table in which data is organized and stored.

Elements of the Relational Data Model

ATTRIBUTES

Attributes, also known as **columns** or **fields**, are the properties which define a relation (table). Each attribute has a **name** and a **data type**, and it defines what kind of information can be stored in that column. **Example:** In a "Student" relation, attributes might include “id”,” “first_name," “last_name," and “date_of_birth."

ATTRIBUTE TYPES

Attribute types, also referred to as data types, specify the kind of data that can be stored in an attribute. Common attribute types include **integers, strings, dates, and floating-point numbers**. Attribute types ensure that data is stored and processed consistently. **Example:** The attribute type of “id” might be defined as an integer, while “first_name" and “last_name" could be defined as strings.

Elements of the Relational Data Model

ROWS(TUPLES)

Rows, also known as tuples or **records**, represent individual data entries in a relation (table). Each row consists of values corresponding to the **attributes** defined for that relation. In essence, a row represents a single record or entity within the table.

students

id	first_name	last_name	date_of_birth
3	Emily	Johnson	1999-10-05
7	Michael	Davis	2005-07-25

KEY ATTRIBUTES

A **key** is an attribute or a set of attributes within a relation (table) that helps to uniquely identify a tuple (record) in that relation. Example For the `students` relation above `id` is a key.

Querying the Relational Data Model

The **relational model** aims to offer a declarative method for **defining data and queries**, allowing users to explicitly specify the information contained in the database and the information they seek from it, while the database management system software manages the data structures for data storage and retrieval procedures for query responses.

Additionally, the relational model can be queried using **Relational Algebra** or the **Standard Query Language** (SQL), providing a structured and efficient means to interact with the data.

Other formal languages associated with the relational model:

→ Tuple relational calculus (*https://www.geeksforgeeks.org/tuple-relational-calculus-trc-in-dbms/*)

→ Domain relational calculus (*https://www.scaler.com/topics/dbms/relational-calculus-in-dbms/*)

Note: For more information on the relational algebra visit *https://www.scaler.com/topics/dbms/relational-algebra-in-dbms/*

Operations of the relational algebra

Relational algebra is a high level formal language associated with the relational model. Is a basis for database query languages.

Operations	basic operations		set operations		derived operations	
	σ	selection	∪	union	∩	intersection
	π	projection	⟍	difference	⋈	join
	ρ	rename	×	cartesian product	÷	division

Examples: operations of the relational algebra (1)

employees

id	first_name	last_name	date_of_birth	department	salary
31	Emily	Johnson	1999-10-05	Consulting	3500
47	Michael	Davis	2005-07-25	Managed services	3200
55	Lisa	King	2006-08-12	Managed services	2900

Obtain the first name, last name, and department for each employee.

$$\pi_{\text{first\_name, lastname, department}}(\text{employees}) =$$

first_name	last_name	department
Emily	Johnson	Consulting
Michael	Davis	Managed services
Lisa	King	Managed services

Examples: operations of the relational algebra (2)

Retrieve the department name and salary of employees whose salary exceeds 3000.

$$\pi_{\text{department, salary}}\left(\sigma_{\text{salary} > 3000}(\text{employees})\right) =$$

department	salary
Consulting	3500
Managed services	3200

Retrieve the names of departments where there is at least one employee born on or after 2000-01-01.

$$\pi_{\text{department}}\left(\sigma_{\text{date\_of\_birth} >= 2000\text{-}01\text{-}01}(\text{employees})\right) =$$

department
Managed services

Note: The projection operation is removing the DUPLICATES!

The Structured Query Language

Querying the Relational Data Model

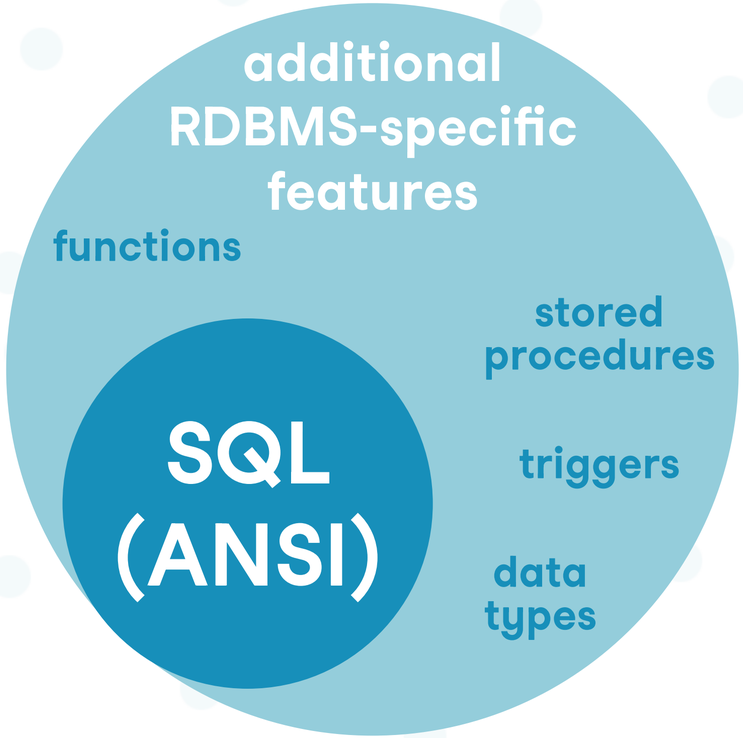
Structured Query Language (**SQL**) is a domain-specific language used in programming and **designed for managing data held in a relational database management system** (RDBMS), or for stream processing in a relational data stream management system (RDSMS). It is particularly useful in handling structured data, i.e., data incorporating relations among entities and variables.

1986 ANSI SQL was first standardized by the American National Standards Institute (ANSI) in 1986. This initial version is sometimes referred to as SQL-86 or SQL/86

1992 **SQL-92** released in 1992 (also known as SQL/92), is one of the best-known and most **widely implemented versions**.

later Further versions were released in later years, each introducing new features and enhancements to the SQL language.

RDBMS-specific implementations of SQL



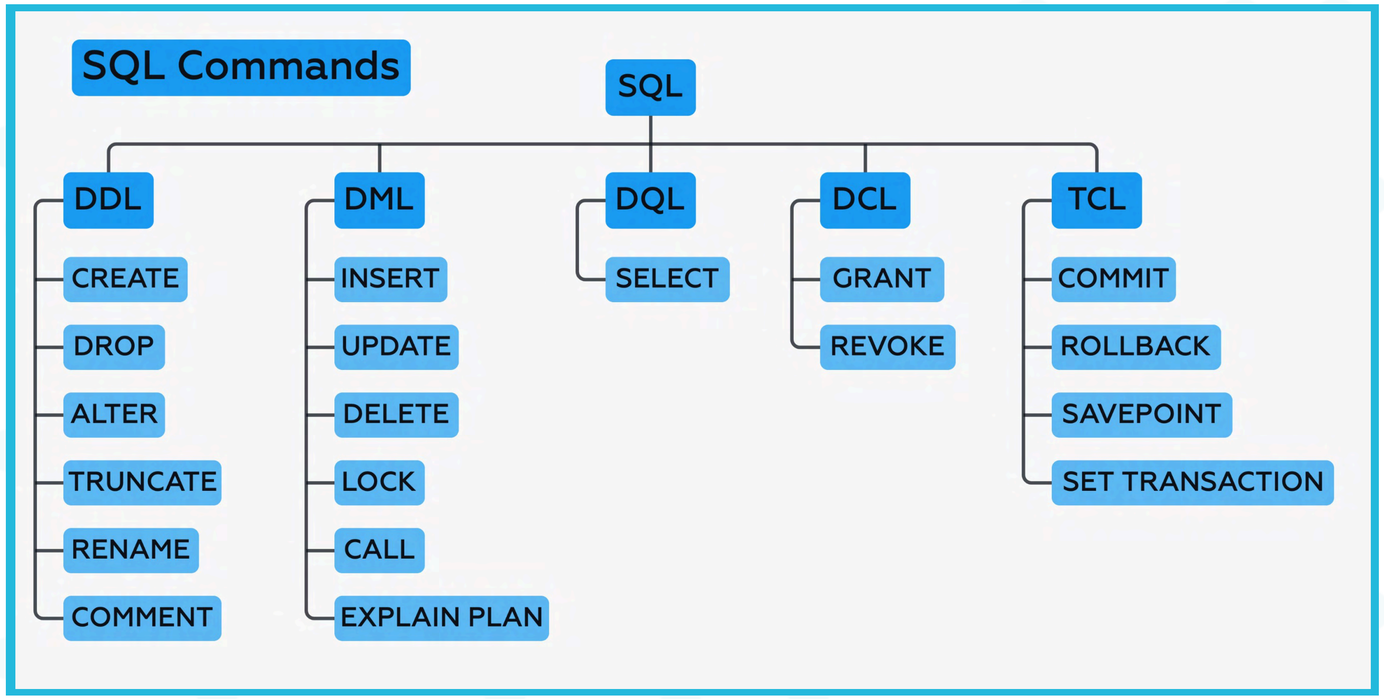
Various **Relational Database Management Systems** (RDBMS) like PostgreSQL, Oracle PL-SQL, and MSSQL T-SQL **adhere to the ANSI SQL standard** to a large extent but also **introduce their own extensions** and enhance functionality and optimize performance.

Each RDBMS **introduces additional language elements** and features, such as stored procedures, triggers, and functions, which are not standardized and can vary significantly between systems.

We could assert that the implementation of ANSI SQL is the commonality among RDBMS.

Types of SQL commands

SQL commands can be organized into various categories based on the functionality and purpose of the language elements.



DDL - Data definition language

DML - Data manipulation language

DQL - Data query language

DCL - Data control language

TCL - Transaction control language

Let's connect to the database dedicated to the course!

Before we dive into the SQL commands, let's ensure we have connectivity to the relevant **practice database and schemas**.

Host=**92.119.120.193**
Database=**{username}_db**
Port=**39023**
User=**{username}**
Password=**{sent in Slack DM}**

Show me all the records from the messages table!

SQL

```sql
SELECT content FROM from user7_db.practice_01.messages;
```

Note: replace user7_db in the query above with your own {user}_db.

Let's connect to the database dedicated to the course!

To avoid referencing your tables as `database_name.schema_name.table_name`, set the `search_path` to the current working schema.

SQL

```
\c user7_db; -- remove this line if the client does not support it
SET search_path TO practice_01;
SELECT content FROM messages;
```

Note: replace user7_db in the query above with your own {user}_db.

Show me the tables!

Before exploring the **SELECT** SQL command, let's identify the other tables prepared for our initial practice session.

SQL

```sql
SELECT
    table_name
FROM
    information_schema.tables
WHERE
    table_schema = 'practice_01'
    AND table_type = 'BASE TABLE'
    AND table_catalog = current_database();
```

Show me the table structure!

Let's examine the columns of the `employees` tables identified by the preceding query.

SQL

```
SELECT column_name, data_type, character_maximum_length, is_nullable
FROM information_schema.columns
WHERE (table_schema = 'practice_01') AND table_name = 'employees';
```

Let's examine the results:

19 rows

	column_name	data_type	character_maximum_length	is_null...
1	id	integer	<null>	NO
2	first_name	character varying	25	NO
3	middle_name	character varying	25	YES
4	last_name	character varying	25	NO
5	hire_date	date	<null>	NO

Current schema and the search_path

In PostgreSQL, **"public" is the default schema**, automatically created with new databases.
It is used when no schema is specified for object referencing and is included in the default search path.
Users can alter the current schema using `SET search_path TO {schema1_name}, {schema2_name};`

Let use dicover the current schema and set the search path after.

SQL

```sql
SELECT current_schema;
SET search_path TO practice_01, public;
SHOW SEARCH_PATH;
```

Case-sensitive, case-insensitive

In PostgreSQL, **identifiers** (including column and table names) **are case-insensitive** by default.

However, when double-quoted, identifiers become case-sensitive.

SQL keywords are not case-sensitive. String literals are case-sensitive (`SELECT 'Apple'<>'apple';`)

SQL

```sql
SELECT columnName FROM tableName;
```

Is equivalent to:

SQL

```sql
SELECT columnname FROM tablename;
```

If you use double quotes, then the identifier is treated as case-sensitive.

SQL

```sql
SELECT "ColumnName" FROM "Tablename";
```