Common SQL Commands

The objective of this reading is to teach you how to name and explain the main commands in SQL. SQL is the most widely used database query language. It is designed for retrieving and managing data in a relational database. SQL can be used to perform different types of operations in the database such as accessing data, describing data, manipulating data and setting users roles and privileges (permissions).

Here you will learn about the main commands used in SQL. At a later stage you will explore relevant examples of how to use these commands with a detailed explanation of the SQL syntax for key operations such as to create, insert, update and delete data in the database.

The SQL Commands are grouped into four categories known as DDL, DML, DCL and TCL depending on their functionality, namely the type of operation they're used to perform. Let's explore these commands in greater detail.

Data Definition Language (DDL)

The SQL DDL category provides commands for defining, deleting and modifying tables in a database. Use the following commands in this category.

CREATE Command

Purpose: To create the database or tables inside the database

Syntax to create a table with three columns:

```
CREATE TABLE table_name (column_name1 datatype(size), column_name2 datatype(size), column_name3 datatype(size));
```

DROP Command

Purpose: To delete a database or a table inside the database.

Syntax to drop a table:

```
DROP TABLE table_name;
```

ALTER Command

Purpose: To change the structure of the tables in the database such as changing the name of a table, adding a primary key to a table, or adding or deleting a column in a table.

1. Syntax to add a column into a table:

```
ALTER TABLE table_name ADD (column_name datatype(size));
```

2. Syntax to add a primary key to a table:

```
ALTER TABLE table_name ADD primary key (column_name);
```

TRUNCATE Command

Purpose: To remove all records from a table, which will empty the table but not delete the table itself.

Syntax to truncate a table:

```
TRUNCATE TABLE table_name;
```

COMMENT Command

Purpose: To add comments to explain or document SQL statements by using double dash (`--`) at the start of the line. Any text after the double dash will not be executed as part of the SQL statement. These comments are not there to build the database. They are only for your own use.

Syntax to `COMMENT` a line in SQL:

```
--Retrieve all data from a table
SELECT * FROM table_name;
```

Data Query Language (DQL)

The SQL DQL commands provide the ability to query and retrieve data from the database. Use the following command in this category.

SELECT Command

Purpose: To retrieve data from tables in the database.

Syntax to select data from a table:

```
SELECT * FROM table_name;
```

Data Manipulation Language (DML)

The SQL DML commands provide the ability to query, delete and update data in the database. Use the following commands in this category.

INSERT Command

Purpose: To add records of data into an existing table. Syntax to insert data into three columns in a table:

```
INSERT INTO table_name (column1, column2, column3) VALUES (value1, value2, val
ue3);
```

UPDATE Command

Purpose: To modify or update data contained within a table in the database.

Syntax to update data in two columns:

```
1
UPDATE table_name SET column1 = value1, column2 = value2 WHERE condition;
```

DELETE Command

Purpose: To delete data from a table in the database.

Syntax to delete data:

```
1
DELETE FROM table_name WHERE condition;
```

Data Control Language (DCL)

You use DCL to deal with the rights and permissions of users of a database system. You can execute SQL commands to perform different types of operations such as create and drop tables. To do this, you need to have user rights set up. This is called user privileges. This category deals with advanced functions or operations in the database. Note that this category can have a generic description of the two main commands. Use the following commands in this category:

GRANT Command to provide the user of the database with the privileges required to allow users to access and manipulate the database.

REVOKE Command to remove permissions from any user.

Transaction Control Language (TCL)

The TCL commands are used to manage transactions in the database. These are used to manage the changes made to the data in a table by utilizing the DML commands. It also allows SQL statements to be grouped together into logical transactions. This category deals with advanced functions or operations in a database. Note that this category can have a generic description of the two main commands. Use the following commands in this category:

COMMIT Command to save all the work you have already done in the database.

ROLLBACK Command to restore a database to the last committed state.

Tables overview

In this reading, you'll find out more about tables in a relational database. You've been introduced to the concept of a table within a database by now and you have also reviewed a few examples of tables. And you've gained a basic understanding of what a primary key does in a table.

The main objective of this reading is to examine tables in more depth in terms of the structure of a table, data types, primary and foreign keys and the role they play in a table, as well as table constraints.

To refresh your memory, a table is the most basic type of database object in relational databases. It is responsible for storing data in the database. Like any other table, a database table also consists of rows and columns.

Just like in a spreadsheet of data, there are rows that run horizontally. These rows represent each record. Rows in turn span multiple columns.

Columns run vertically. They are like the definition of each field. Each column has a name that describes the data that is stored in it. Examples of column names could include FirstName, LastName, ProductID, Price and so forth.

Where the row intersects with a column is where a cell is located. A cell is where you store an item of data.

What are data types?

Every column in a table has a data type. These data types are defined by SQL or Structured Query Language. A data type defines the type of value that can be stored in a table column.

For example, here are some of the data types that are available:

- Numeric data types such as INT, TINYINT, BIGINT, FLOAT, REAL.
- Date and time data types such as DATE, TIME, DATETIME.
- Character and string data types such as CHAR, VARCHAR.
- Binary data types such as BINARY, VARBINARY.
- Miscellaneous data types such as:
    - Character Large Object (CLOB) for storing a large block of text in some form of text encoding.
    - Binary Large Object (BLOB) for storing a collection of binary data such as images.

Here's an example of a table. This is the student table that stores data about a student such as:

- student ID,
- first name,
- last name,
- date of birth,
- home address,
- and faculty.

These are the table's columns.

There are also six rows within this table; one for each student. In other words, the table contains the records of six students.

Each cell in a row or record contains a piece of data such as student ID = 1, first name = Emily, last name = Williams and so on.

student_id	first_name	last_name	dob	home_address	school_address	contact_no	faculty
1	Sam	Williams	2010-01-03	Obere Str. 57	25 stafford building	7689876543	Science
2	John	Smith	2010-05-20	Avda. De la constituición 2222	25 stafford building	7623453456	Science
3	Ann	Wilsom	2010-07-15	no 25 Street2	25 stafford building	7623453456	Science
4	Pat	Browns	2010-11-03	Nevida Str.100	25 stafford building	7689876873	Science
5	John	Taylor	2010-08-15	Berguvsvägen 8	25 Tech building	7698765457	Engineering
6	Jane	Thomas	2010-10-10	Mataderos 2312	25 Tech building	7638929876	Engineering

The student ID would probably have a data type of INT, for example. First name and last name would have a data type of VARCHAR and date of birth would have a data type of DATE.

Tables in a relational database

In a relational database there are multiple tables representing the structure of the back end of a software system. For example, in the context of a Student Information System, the tables might include Student, Teacher, Class and Subject.

In relational database terminology a table is also known as a relation. A table row or a record is also known as a tuple. For example, the student relation above has six tuples.

Each table or relation in a database has its own schema. Schema simply means the structure. The structure includes:

- the name of the table or relation,
- its attributes,
- their names
- and data type.

What is a primary key?

In a table, there is a field or column that is known as a key which can uniquely identify a particular tuple (row) in a relation (table). This key is specifically known as a primary key.

For example, in the student table, the student ID allows you to uniquely identify a particular row. The other columns like first name, last name, date of birth and others could contain duplicate or repeating data for multiple students. Therefore, they can't be used to uniquely identify a given student record. So, the student ID is the primary key of the student table.

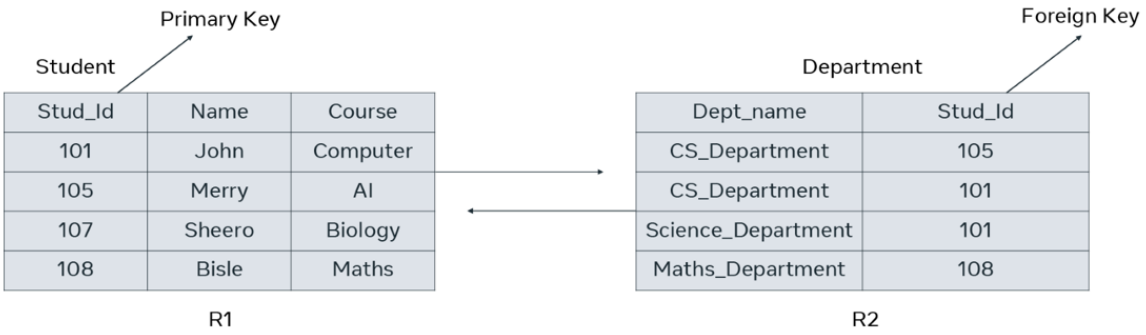
In some cases, the primary key can comprise more than one column or field. This happens when a single column cannot make a record in a table uniquely identifiable. For example, in the table below, the EMP_ID values aren’t unique, so the column is not unique by itself. Thus, this column alone cannot be used as the primary key of this table. However, the EMP_ID and DEPT_ID columns together can make a record unique. Therefore, the primary key of this table is EMP_ID and DEPT_ID. This is also known as a composite primary key.

RESULTS

	EMP_ID	DEPT_ID	EMPNAME	GENDER	SALARY
1	101	1	RAHUL	MALE	22000
2	101	4	SHWETA	FEMALE	22000
3	102	2	RAJ	MALE	25000
4	102	6	VIJAY	MALE	25000
5	103	3	PRIYANKA	FEMALE	25500
6	104	3	SATYA	MALE	23000
7	105	5	VIVEK	MALE	28000

What is a foreign key?

Tables in a database do not stay isolated from each other. They need to have relationships between them. Tables are linked with one another through a key column (the primary key) of one table that's also present in the related table as a foreign key. For example, the student table and the department table are linked via the student ID which is the primary key of the student table that's also present in the Department table as a foreign key.



Integrity constraints

Every table in a database should abide by rules or constraints. These are known as integrity constraints.

There are three main integrity constraints:

1. Key constraints

2. Domain constraints

3. Referential integrity constraints

What are key constraints?

In every table there should be one or more columns or fields that can be used to fetch data from tables. In other words, a primary key. The key constraint specifies that there should be a column,

or columns, in a table that can be used to fetch data for any row. This key attribute or primary key should never be NULL or the same for two different rows of data. For example, in the student table I can use the student ID to fetch data for each of the students. No value of student ID is null, and it is unique for every row, hence it can be the key attribute.

What are domain constraints?

Domain constraints refer to the rules defined for the values that can be stored for a certain column. For instance, you cannot store the home address of a student in the first name column. Similarly, a contact number cannot exceed ten digits.

What are referential integrity constraints?

When a table is related to another table via a foreign key column, then the referenced column value must exist in the other table. This means, according to the student and department examples, that values should exist in the student ID column in the student table because the two tables are related via the student ID column.

In this reading, you learned more about tables in a relational database as you explored the table in terms of its structure, data types, constraints, and the role of primary and foreign keys.

3.2.1 Basic Types

The SQL standard supports a variety of built-in types, including:

- **char**(n): A fixed-length character string with user-specified length n. The full form, **character**, can be used instead.
- **varchar**(n): A variable-length character string with user-specified maximum length n. The full form, **character varying**, is equivalent.
- **int**: An integer (a finite subset of the integers that is machine dependent). The full form, **integer**, is equivalent.
- **smallint**: A small integer (a machine-dependent subset of the integer type).
- **numeric**(p, d): A fixed-point number with user-specified precision. The number consists of p digits (plus a sign), and d of the p digits are to the right of the decimal point. Thus, **numeric**(3,1) allows 44.5 to be stored exactly, but neither 444.5 or 0.32 can be stored exactly in a field of this type.
- **real, double precision**: Floating-point and double-precision floating-point numbers with machine-dependent precision.
- **float**(n): A floating-point number, with precision of at least n digits.

Additional types are covered in Section 4.5.

Each type may include a special value called the null value. A null value indicates an absent value that may exist but be unknown or that may not exist at all. In certain cases, we may wish to prohibit null values from being entered, as we shall see shortly.

The **char** data type stores fixed length strings. Consider, for example, an attribute A of type **char**(10). If we store a string "Avi" in this attribute, 7 spaces are appended to the string to make it 10 characters long. In contrast, if attribute B were of type **varchar**(10), and we store "Avi" in attribute B, no spaces would be added. When comparing two values of type **char**, if they are of different lengths extra spaces are automatically added to the shorter one to make them the same size, before comparison.

When comparing a **char** type with a **varchar** type, one may expect extra spaces to be added to the **varchar** type to make the lengths equal, before comparison; however, this may or may not be done, depending on the database system. As a result, even if the same value "Avi" is stored in the attributes A and B above, a comparison $A=B$ may return false. We recommend you always use the **varchar** type instead of the **char** type to avoid these problems.

SQL supports a number of different integrity constraints. In this section, we discuss only a few of them:

- **primary key** ($A_{j_1}, A_{j_2}, \ldots, A_{j_m}$): The **primary-key** specification says that attributes $A_{j_1}, A_{j_2}, \ldots, A_{j_m}$ form the primary key for the relation. The primary-key attributes are required to be *nonnull* and *unique*; that is, no tuple can have a null value for a primary-key attribute, and no two tuples in the relation can be equal on all the primary-key attributes. Although the primary-key

specification is optional, it is generally a good idea to specify a primary key for each relation.

- **foreign key** ($A_{k_1}, A_{k_2}, \ldots, A_{k_n}$) **references** s: The **foreign key** specification says that the values of attributes ($A_{k_1}, A_{k_2}, \ldots, A_{k_n}$) for any tuple in the relation must correspond to values of the primary key attributes of some tuple in relation s.

  Figure 3.1 presents a partial SQL DDL definition of the university database we use in the text. The definition of the *course* table has a declaration "**foreign key** (*dept_name*) **references** *department*". This foreign-key declaration specifies that for each course tuple, the department name specified in the tuple must exist in the primary key attribute (*dept_name*) of the *department* relation. Without this constraint, it is possible for a course to specify a nonexistent department name. Figure 3.1 also shows foreign key constraints on tables *section*, *instructor* and *teaches*.
- **not null**: The **not null** constraint on an attribute specifies that the null value is not allowed for that attribute; in other words, the constraint excludes the null value from the domain of that attribute. For example, in Figure 3.1, the **not null** constraint on the *name* attribute of the *instructor* relation ensures that the name of an instructor cannot be null.

Database structure overview

In this reading, you'll find out more about the basic database structure. You've been introduced to what a table is in a database, the basic structure of a table, data types, what primary and foreign keys are and the roles they play in a table. You also learned about table constraints.

The main objective of this reading is to cover the basic structure of a database. In other words, you will learn more about tables, fields (or attributes), records, keys and table relationships.

What is database structure?

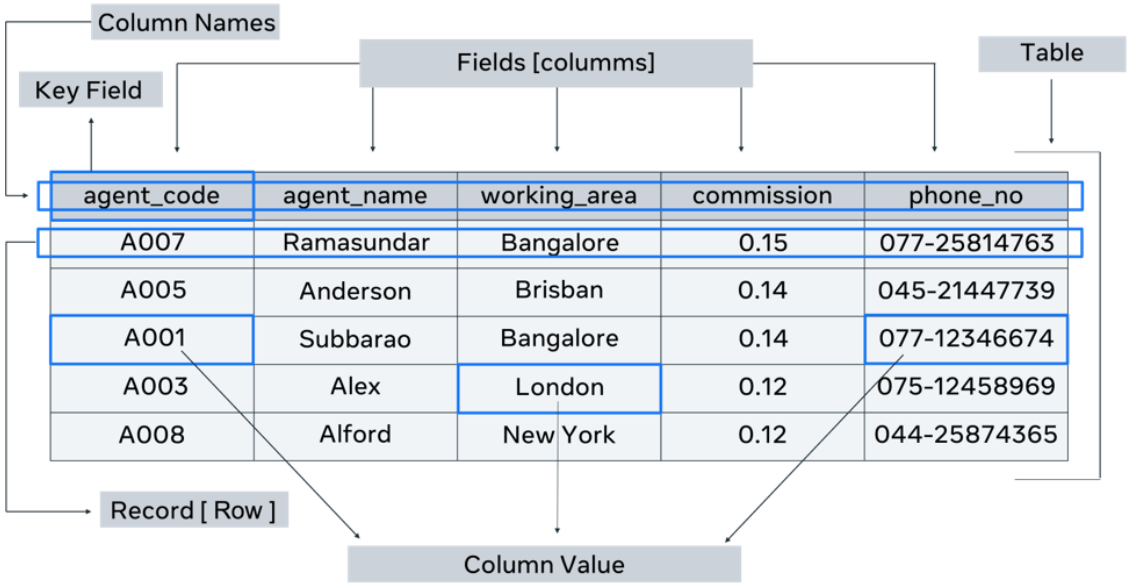
Database structure refers to how data is arranged in a database. Within a database, related data are grouped into tables, each of which consists of rows (also called tuples) and columns, like in a spreadsheet.

The structure of a database consists of a set of key components. These include:

- Tables or entities, where the data is stored.
- Attributes which are details about the table or entity. In other words, attributes describe the table.
- Fields, which are columns used to capture attributes.
- A record, which is one row of details about a table or entity.

- And the primary key, which is a unique value for an entity.

This image shows the basic structural elements of a database table.



Table

A table contains all the fields, attributes and records for a type of entity. A database will most probably contain more than one table.

Fields

Column headings are known as fields. Each field contains a different attribute. For every table, a unit of data is entered into each field. It's also known as a column value. Each column has a data type. For example, the "agent_name" column has a data type of text, and the "commission" column has a numeric data type.

Column value or unit of data

Each individual piece of data entered into a column is a unit of data. These units are also called data elements or column values.

Records

A record consists of a collection of data for each entity. It's also known as a row in the table.

Data types

To keep the data consistent from one record to the next, an appropriate data type is assigned to each column. The data type of a column determines what type of data can be stored in each column.

Data types are also a way of classifying data values or column values. Different kinds of data values or column values require different amounts of memory to store them. Different operations can be performed on those column values based on their datatypes.

Some common data types used in databases are:

- Numeric data types such as INT, TINYINT, BIGINT, FLOAT and REAL.
- Date and time data types such as DATE, TIME and DATETIME.
- Character and string data types such as CHAR and VARCHAR.
- Binary data types such as BINARY and VARBINARY.
- And miscellaneous data types such as:
    - Character Large Object (CLOB), for storing a large block of text in some form of text encoding.
    - and Binary Large Object (BLOB), for storing a collection of binary data such as images.

Logical database structure

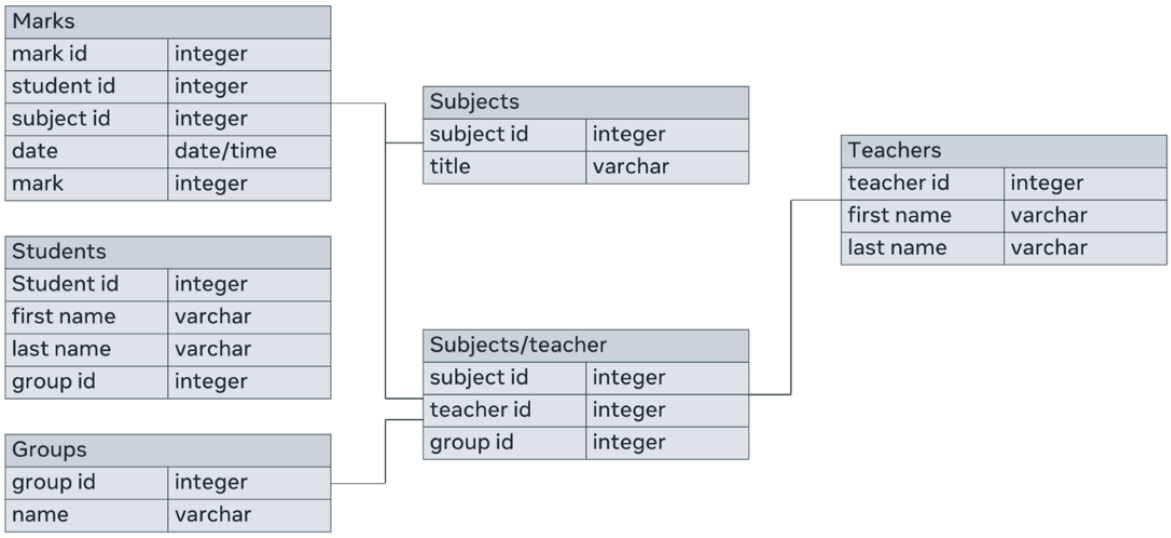
The logical structure of a database is represented using a diagram known as the Entity Relationship Diagram (ERD). It is a visual representation of how the database will be implemented into tables during physical database design, using a Database Management System (DBMS) like MySQL or Oracle, for example.

A part of the logical database structure is how relationships are established between entities. These relationships are established between the instances of the entities. Accordingly, there can be three ways in which entity instances can be related to each other:

- One-to-one relationships
- One-to-many relationships
- Many-to-many relationships

This is also known as cardinality of relationships. The logical database structure which is represented using an ERD also depicts these relationships.

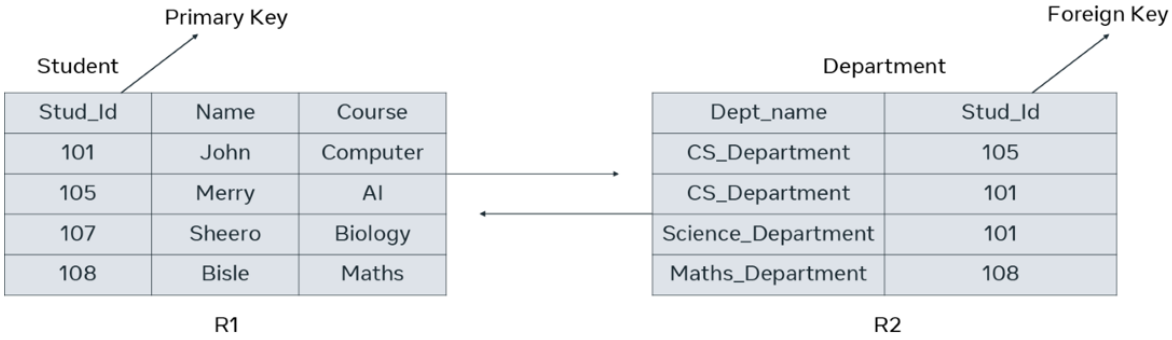
Here's an example of an ERD that has all these elements.



Physical database structure

In the physical database structure, where entities are implemented as tables, the relationships are established using a field known as a foreign key. A foreign key is a field in one table that refers to a common field in another table (usually the primary key).

Let's take the example of a database that contains two tables: student and department. The student table has a primary key of "Stud_id", which is also present in the Department table as a foreign key. Therefore, the two tables are related to each other via the "Stud_id" field.



In this reading, you learned more about the basic database structure including tables, fields or attributes, records, keys and relationships between tables.