17th Apr 2018 | 9 minutes read

Essential SQL Terms to Know for Beginners and Pros

Aldo Zelen

Extras How To In Sql SQL Basics

Working with databases can seem daunting to a non-technical person. Right away, you're bombarded with new terms that make your head spin. Database, database instance, table, SQL and others are some of the basic terms that you need to understand just to have a normal conversation with your technical colleagues. In this article, we'll explore some basic SQL database terminology you need to know to succeed.

LearnSQL.com provides a one-stop-shop for all things SQL, covering basic to advanced concepts in one single platform. LearnSQL.com is specifically geared towards SQL. It offers over 60 interactive SQL courses that range in difficulty from beginner to advanced and monthly SQL challenges to practice your SQL skills.

Imagine you're attending a meeting with the development staff and suddenly feel as if you're listening to white noise. You understand what a "file" is, but all these other terms your colleagues are throwing at you are incomprehensible. Don't fret—we've got you covered.

To start things off, visualize a large office building teeming with activity. Let's say it's owned by a fictitious company called SQL Inc. This company's building has standard architecture with doors, windows, elevators, etc. The building, with all its corporate life, will serve as our analogy for a **database** to simplify explaining SQL terminology.

Database Basics

Database Management Systems

Let's start with basic SQL terminology. SQL Inc. has many buildings (databases) of the same type scattered around the globe. And when you see one of these buildings, you know it is managed by SQL Inc. The same goes for **database management systems** (DBMSs). As the name suggests, these systems manage databases! There are plenty of them, and they range from big market players like Oracle, Microsoft SQL Server, and IBM DB2 to others like MySQL, PostgreSQL, and so on. The largest DBMSs are mostly used by large corporations that need the support and features that enterprise-grade tools offer. The others are used in smaller firms and open-source projects.

Database Transactions

There are many SQL terms connected with databases. Another one is a database transaction. Simply put, a database's job is to create, read, update, and delete data. In our SQL Inc. building, this is analogous to managing corporate papers that you fill out, read, correct, and discard. Just like there is exactly one **relevant** version of a certain document in the organization, there is only one **relevant** and **truthful** version of data in a database (e.g., that there is only one version of an employee manifesto that is replicated many times within an SQL Inc. building). Databases ensure that there is only one relevant version of data via **transactions**. When the power goes out in our building, all of the work is saved in paper form. When the power goes out in a working database, the **transactional** model of the database ensures that, when it is powered up, the database can **roll back** to a previous stable state.

Database Files

At the end of a busy work day, everybody goes home, and our building remains empty. But the actual work that the staff generated during the day is not lost—it is archived in different places throughout the building. The same is true for our database. If we take our database **offline**, a term we use for the regular *powering off* of the system, the data are still secured in our **database files**.

Database Instance

Our staff is busy at work during the day, changing and creating papers. Their workflow is fluid and active; many employees complete different tasks throughout the day. This workflow is only "alive" during work hours, and it's the *only* process that can change the documents stored in our building. Similarly, the collection of processes that manage the files in a database, and therefore the **database itself**, forms a database instance. In more technical terms, a database instance is a collection of *processes* and *temporary memory structures* that allow the database system to work.

Database Server and Client

One SQL Inc. building consolidates its files and sends them to be viewed by people outside the building organization. Various outside parties, companies, and other entities receive paper files from our building. In this way, our building interacts with the outside business world. Similarly, a database has a **server** that communicates with the outside **client**. We usually have one **database server** (an SQL Inc. building), at some location, and many **database clients**, the outside parties, that interact with the database in a number of ways. The client can, and usually does, manipulate data in the database by communicating with the server. You must remember that SQL Inc. has many buildings around the world, and by analogy, one database vendor also has many databases (with servers and clients).

To understand SQL database terminology better, read more about database server and database client **here**.

Structured Query Language, the Language of Databases

Let's move on to the core of the SQL terminology. Corporate communication needs to be direct and unambiguous. It has rules of its own—someone might even call it a language! The corporate division communicates with the outside world in this business language. The language that we use to communicate with a database is

called **SQL**, which stands for **Structured Query Language**. This language is much simpler than English. The database knows *exactly* what it needs to do with the data when it gets a command in SQL, much like our division knows what it needs to do when it gets its orders from the central management.

Keywords

The business language of an organization follows certain common rules or standards. Likewise, SQL has a set of reserved words. These words must be used in a certain way; they cannot be used to name objects in the database. We call these words **keywords**. Examples of SQL keywords are `CREATE`, `INSERT`, `GRANT`, `UPDATE`, `DELETE`, `SELECT`, etc.

Data Definition Language and Data Manipulation Language

When the central division needs to implement a new type of file as a template, it sends this template to all corporate divisions. Similarly, when we need to define the structure of new database objects (such as a new table), we issue a **DDL** (data definition language) statement. To populate the templates with real corporate data, the central division sends a new set of orders. This process of changing the data populated in spreadsheets is a subset of SQL called **DML** (data manipulation language).

Our database data are represented in tabular form. We see our **database** data as tables with **rows** and **columns**. The tables represent certain entities, like `sales`, in our building. These `sales` have certain attributes, such as date of sale, amount, and sale number. These attributes represent the columns of our tables, while the rows represent different sales. There can be many different tables in a database, and they **relate** to each other, like `sales` and `items_sold`. This relation is the reason why traditional databases are called **relational** databases.

Let's look at an example. With a DDL statement, we create a **sales** table:

Code

```
CREATE TABLE sales (
    id NUMBER,
    amount NUMBER,
    sales_date DATE
);
```

We populate the table with some **sales** using DML:

Code

```
INSERT INTO TABLE sales (id, amount, sales_date) VALUES (1,100,'2018-01-10);
INSERT INTO TABLE sales (id, amount, sales_date) VALUES (2,500,'2018-02-10);
INSERT INTO TABLE sales (id, amount, sales_date) VALUES (3,250,'2018-03-10);
```

To retrieve this data from the table, we issue a `SELECT` query (part of DML):

Code

```
SELECT id, amount, sales_date;
```

Statements and queries

More SQL terms to know? You're probably wondering why some instructions to databases are called **statements** and some **queries**. It's simple. A **query** asks a database to return data in some form. This querying of data is usually the most important work that a database does for a user. On the other hand, a **statement** is a generic instruction to the database. This includes creating and dropping tables, inserting and deleting data, etc. So we can conclude that all queries are statements and not all statements are queries.

Clauses, Predicates, and Expressions

Let's look at our example of the `SELECT` statements above. If we would like to see only the sales from January 10th, we would expand our query like so:

Code

```
SELECT id, amount, date_time WHERE date_time = '2018-01-10';
```

The `WHERE` part of our statement represents an optional SQL clause. SQL statements are constructed from both optional and mandatory **clauses**, just like the sentences of the English language. Clauses are **keywords** in SQL that allow you to request a certain *action* based on some *condition* that must be satisfied. In our `WHERE` clause, these conditions are questions that we ask the database. For example, is the date of sale January 10th ? This question is either true or false. We call this question (condition) a **predicate**. The type of question we ask is an **expression**. In our example, the predicate is of the equality type because we are asking if the date equals a certain value. If instead we asked for all dates after January 10th, we would have a *comparison* expression as the predicate of the `WHERE` clause in the query.

Functions and Procedures

The last essential SQL terms to know on our list are functions and procedures. Let's say the management of SQL Inc. assign some repetitive task for the staff of our building to complete. For example:

1. take this report;
2. file it with another building; and
3. tell me how long it took you.

It would be more efficient if the management simply told the division to execute this process with a single command. In databases, processes that can be repeated as many times as necessary are called **functions** or **procedures**. Functions always return some result of their execution. In our example, the function `file_report` returns the time it took it to perform this task. In contrast, procedures do not return any value—they simply do what they are told.

Conclusion

This article presented a brief overview of the most commonly used SQL terms in the database world. We touched on the basics of databases architecture files, database instances, transactions, and client-server communication. SQL is also a very powerful and comprehensive language that we use to communicate with databases. We also covered SQL terms to know related to topics like DML, DDL, statements, keywords, queries, and clauses.

Of course, knowing SQL database terminology is only half the battle. The other half is applying them. As a programming language of the fourth generation, SQL is incredibly easy and fun to learn. Whether you're just starting out or have some introductory knowledge, an excellent way to learn SQL is through our fully interactive **SQL Basics course** on the LearnSQL.com platform. Give it a shot!

Tags: Extras How To In Sql SQL Basics

You may also like