SQL

BEGINNERS

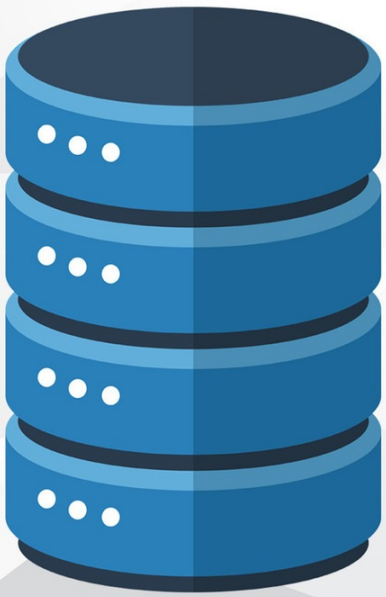
The Ultimate Beginner's Guide to Learn SQL Programming and Database Management Step-by-Step, Including MySQL, Microsoft SQL Server, Oracle and Access

JOHN RUSSEL

SQL

BEGINNERS

The Ultimate Beginner's Guide to Learn SQL Programming and Database Management Step-by-Step, Including MySQL, Microsoft SQL Server, Oracle and Access



JOHN RUSSEL

SQL

-

The Ultimate Beginner's Guide to Learn SQL Programming and Database Management Step-by-Step, Including MySQL, Microsoft SQL Server, Oracle and Access

Table of Contents

Preface

What is SQL?

The SQL (**the Structured Query Language** , **Structured Query Language**) is a special language used to define data, provide access to data and their processing. **The SQL language** refers to nonprocedural languages - it only describes the necessary components (for example, tables) and the desired results, without specifying how these results should be obtained. Each **SQL** implementation is an add-on on the database engine, which interprets **SQL statements** and determines the order of accessing the database structures for the correct and effective formation of the desired result.

SQL to work with databases?

To process the request, the database server translates **SQL** commands into internal procedures. Due to the fact that **SQL** hides the details of data processing, it is easy to use.

You can use SQL to help out in the following ways:

- **SQL** helps when you want to create tables based on the data you have.

- **SQL** can store the data that you collect.
- **SQL** can look at your database and retrieves the information on there.
- **SQL** allows you to modify data.
- **SQL** can take some of the structures in your database and change them up.
- **SQL** allows you to combine data.
- **SQL** allows you to perform calculations.
- **SQL** allows data protection.

Client and server technology?

When using client-server technology, the application is divided into two parts. The client part provides a convenient graphical interface and is located on the user's computer. The server part provides data management, information sharing, administration and ensures the security of information. The client application generates requests to the database server on which the corresponding commands are executed. Query results are sent to the client.

When developing distributed information systems in the organization of interaction between the client and server parts, the following important tasks in a practical sense are distinguished:

Transfer of a personal database to a server for its subsequent collective use as a corporate database;

Organization of requests from one end of the client over to the company’s database will help make sure that the client will receive the right results.

Development of a client application for remote access to a corporate database from a client computer.

The task of transferring a personal database to a server may arise in situations when it is necessary to provide collective access to a database developed using a personal DBMS (FoxPro, Access). To solve this problem, these personal DBMSs have the appropriate tools designed to convert databases to SQL format.

The preparation of queries to the database on the server (in SQL) from the client side can be performed using a specially designed utility. To provide the user with great opportunities and convenience in preparing and executing requests, client applications are created.

To organize queries to a server database in SQL or using a client application, various methods of interaction are possible that significantly affect efficiency. The main ways of such interaction include:

Interface DB-LIB (database libraries);

ODBC technologies (open database compatibility);

OLE DB interface (linking and embedding database objects);

DAO technologies (data access objects);

ADO technologies (data objects).

The DB-LIB interface is an application programming interface specifically designed for SQL. Therefore, it is the least mobile among those considered in the sense of the possibility of transferring applications to another environment. In terms of performance, this method allows the fastest access to information. The reason for this is that it represents an optimized application programming interface and directly uses the SQL system query language.

ODBC technologies are designed to provide the possibility of interconnection between different DBMSs and to receive requests from the application to retrieve information, translate them into the core language of the addressable database to access the information stored in it.

The main purpose of ODBC is to abstract the application from the features of the core of the server database with which it interacts, so the server database becomes as if transparent to any client application.

The advantage of this technology is the simplicity of application development, due to the high level of abstractness of the data access interface of almost any existing DBMS types. Using this technology, it is possible to create client-server applications, and it is advisable to develop the client part of the application using personal DBMS tools, and the server part using SQL tools.

The main disadvantage of ODBC technology is the need to translate queries, which reduces the speed of data access. On client-server systems, this drawback is eliminated by moving the request from the client computer to the server computer. This eliminates the intermediate links, which are the main reason for reducing the speed of information processing using the tools of this technology.

When using ODBC tools in a client application, a specific data source is accessed, and through it, to the DBMS that it represents. When installing ODBC tools, the common ODBC subsystem is installed and the “driver-database” pairs are defined, which are the names used to establish the connection with. database. Corresponding pairs are called named data sources.

Each named data source describes the actual data source and information about access to this data. The data can be databases, spreadsheets and text files. Access information, for example, to a database server, usually includes server location information, database name, account ID and password, as well as various driver parameters that describe how you should establish the right kinds of connections to your source of data.

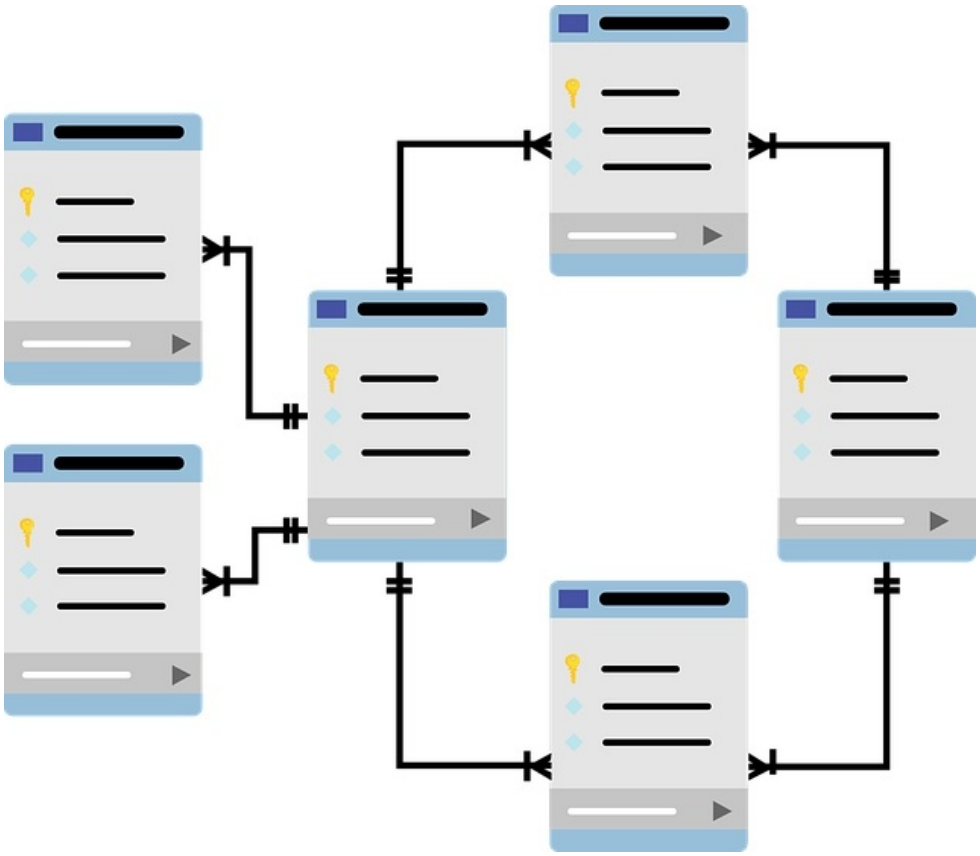
When processing data on a server using ODBC technology and using a client application, two main stages are distinguished: setting a data source - creating and configuring a connection, as well as actually processing data using queries.

It is usually best if you are able to work with the OLE DB interface to create tools and utilities, or system-level developments that require high performance or access to SQL properties that are not available using ADO technology. Key features of the OLE DB specification provide full data access functionality. In SQL, the server database processor uses this interface for communication: between internal components, such as the storage processor and the relationship processor; Between SQL installations using remote stored procedures as an interface to other data sources for distributed queries.

When using OJSC technology, work with databases and tables is carried out using collections of objects. This provides great convenience in working with database objects.

At present, the technology of JSC is gradually superseded by ADO technology, which allows you to develop Web applications for working with databases. In general, ADO technology can be described as the most advanced application development technology for working with distributed client-server technology databases.

Chapter 1: A Look At the Basics of SQL



Many companies have to spend some time finding ways to store all of the data that they want to work with. This data is such an important part to how they will run their business, and the amount of success that they are going to find, that they know it is one of the best ways to help them get ahead.

Thanks to the modern world of technology we are currently in, there are a ton of resources out there to provide us with some of the data that we are looking for. This data can be used by companies in so many ways. It can help them to learn how to reach their customers better, how to make better decisions, how to reduce some of the waste that they are seeing on a regular basis in their business, and how to beat out the competition.

Gathering up the data that is needed is not going to be the hardest part. In fact, often the hardest part of all of this is figuring out the best way to store it, and then figuring out how you are able to take that information and actually sort through it, and make it work for your needs. One of the methods that can help with this is a database.

These databases are able to hold onto large amounts of information and can sort through it as well. But when you have millions of potential points of data that you want to sort, compare, and more, it is going to seem like a huge undertaking. And this is where the SQL language is going to come into play as well. Let's dive into what this language is all about, and why it is so important to helping us to achieve our goals with all of that data.

A Look at SQL

The first thing that you need to ask is what is SQL. SQL is going to stand for Structured Query Language and it is a pretty basic language that you can use in order to interact with the different databases that are on your system. The original version did come out in the 70s, but it has really started to see some changes when IBM released a new prototype that released SQL to the world.

This first particular tool was called ORACLE and it was so successful that the part of the company that worked with ORACLE was able to break off from IBM and started off on their own. ORACLE is still one of the leaders in the programming language field because it works with SQL and continuously makes it easier for people to learn how to work with the database.

Working with the Database

When we decide to spend some of our time looking at these databases and working with some of the neat things that are available with the SQL language, you will find that the database is simple, and will just hold onto groups of information. Sometimes we are going to think of these as mechanisms that will only hold onto the information that the user is able to access wen they would like but other times it is going to come into play and will help a business to get ahold of the information that they need, without having to worry about some other potential issues showing up as well.

There are going to be some times when you would wan to bring out the database, and working with SQL, even if you didn't realize it at the time, can help you to work through this process. For example, if you have gone through and looked for a product on one of your favorite websites, then you are used to some of he ideas that come with this SQL and all that it is able to do to make things easier for you. It will help to sort through the perhaps millions of items on that website, and will make it easier for you to really see some of the results that you would like.

With this in mind, we do need to take a look at some of the other information that we can use, and the different parts that are going to come with the databases that we are able to use when it comes to the SQL language.

Relational database

The first type of database that we are going to talk about is a relational database. These are ones that are going to be segregated into tables or logical units. These tables can be interconnected inside of the database so that they make sense based on what you are working on at the time. The database is going to also make it easier to break up the data into some smaller units so that you are able to manage them easier and they will be optimized for making things easier on you.

It is important to know about the relational database because it is going to help to keep all your information together, but it does help to split it up so that the pieces are small enough to read through easier. The server will be able to go through all of these parts to see what you need because the smaller pieces are easier to go through compared to the bigger pieces. Because of the optimization and the efficiency that is found in this kind of system, it is common to see a lot of businesses going with this option instead of another one.

Client and server technology

For quite a while, a lot of the computers that were being used in the business industry were considered mainframe computers. This means that the machines were holding a large system that was great for processing as well as storing information. The user was able to use these computers in order to interact with the mainframe using what was done with a dumber terminal, or one that doesn't think on their own. In order to get all the right functions to perform, the dumb terminals are going to rely on the memory, storage, and processor that are inside of the computer.

While these systems worked and there isn't anything wrong with this setup (many companies even still use them to help get things done in their business), there is a better solution that will get the job done faster and more efficiently than you will find with the mainframe option.

The client/server systems will use a slightly different process in order to get the results that you would like. The main computer, also known as the server, can be accessed by the user that is on the network; for the most part they are going to have a WAN or a LAN network that will help them to access the network. The user will be able to access this server with a desktop computer, as well as another server, rather than having to use the old dumb terminals. Every computer, which will be known as the client inside of this system, will be able to access this system, which can make it easier to have interaction between any clients and the server to get things done.

Internet based database systems

While the client to server kind of technology is widely popular and has worked well for a lot of people to use over time, you will also find that there are some programmers who have decided to add in some databases that are going to have more integration with the internet. This kind of system is going to allow the user to access the database when they go online, so they will be able to use their own personal web browser in order to check this out when they would like.

Along with this, the customer can go and check out the data if they would like, and they can even go online and make some changes to the account, check on the transactions that are there, check out the inventories, and purchase items. They can even make payments online if this is all set up in the right manner. thanks to act that we are able to base some of our database online, we will find that it is possible to go online and let the customer access this database, even from their own home.

To help us work with these databases, we will just need to pick out the kind of web browser that we would like to go with, and then head on over to the website of the company that we go with. At times, you may need to have some credentials to get onto the account, but that is going to depend on the requirements of the company at the time. Then you can work with the search function in order to find some of the information hat you would like on the database.

You may find that a lot of these online databases are going to require us to have a log in to have any kind of access, especially if there is some kind of payment requirement that goes with them. This may seem like a pain to work with, but it does add in some of that extra security that you are looking for to make sure your personal and financial information are always safe.

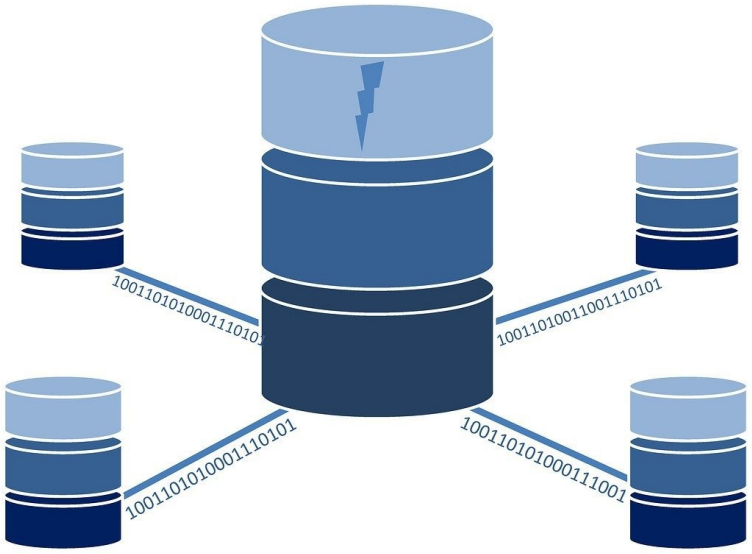
Of course, we have to remember that while this may seem like a simple and easy to handle kind of database, there are actually quite a few things that are showing up behind the scenes in order to make sure that the database is going to work. We have to make sure the database is working properly and that the right commands are being sent out at the right times, to ensure we can reach our accounts, find the products we want, look at out order history and more.

For example, we may see that the web browser that we want to work with at any given time is able to use and also execute the SQL in order to make sure that it is showing any of the data and information that the user would like to see. SQL is going to be used to help reach the database that the customer is looking for, such as clothing or food that they are interested in seeing, and then SQL is going to

send this information right back to the website before showing it to the user in that browser.

This sounds pretty complicated in the beginning, but you will find that it is actually going to be really easy to work with, and can get done in just a few minutes if the program is set up in the proper manner. all of the things that we have been able to discuss with SQL so far will ensure that we are able to make a simple search or something else similar happen in no time at all. SQL can really make it that simple.

The Benefits of Using SQL



And finally, we need to spend some time taking a look at some of the advantages that we are going to see when it comes to using the SQL language to help us sort through all of the information that may be found in our database. There are actually a few other options that we can use when we would like to sort through a database that we want to work with. But SQL is going to often be the one that is chosen because it is easy to use, works well with databases in particular, and so much more.

There are a ton of benefits that you will be able to enjoy when it comes to working with SQL for some of your database programing needs. And using this is going to ensure that you are able to make sure that your database is set up and ready to go. Some of the many benefits that you will be able to notice when it comes to working with this kind of language will include:

- High speed—if you are looking for something that is high speed, the SQL option is one of the best. The SQL queries are able to retrieve a lot of information from the database in just a little bit of time and it is one of the most efficient options that you are able to use on the market to get this done.
- Well defined standards—the SQL databases have been around for some time and they have some good standards that will help to make it easier to keep the database nice and strong. Some of the other databases don't have these clear standards and it can make it difficult to store the information that you need.
- No coding needed—you are able to use the SQL system in order to store some of the information for your business without knowing a lot of coding ahead of time. You will learn a few parts of coding inside of this guidebook to do the right commands, but you don't need to have extensive knowledge of coding before getting started.
- Emergence of object oriented DBMS—the earlier SQL databases were based on the relational databases and while this is not a bad thing, there are some better and faster options. Now they are moving on to some of the newer options to help get the work done, including with object oriented DBMS to help out.

Now with this in mind, we also have to remember that even though there are a ton of benefits that we are able to see when we are working with this kind of coding languages in our database, there are a few negatives that are going to show up and a few reasons why people may not want to work with this kind of language even on some of their database needs.

The first issue is that sometimes the SQL interface can be difficult to work with. Even though you do not need to know a ton of coding rules in order to start off with this language, it is still hard in some cases to interface with the database when you use SQL. This database is going to be a bit more complex in terms of its interface compared to just working with a few lines of code that we may find with some of the other options out there, so you at least need to take this issue into consideration.

Then there is also the issue that there are some features that come with this that are going to need third-party extensions on them. There are going to be some features that are added to SQL that will have to come with a third-party part in order to get them to work the way that you would like. If you do not want to take

all of this effort to make it happen, then you may want to work with another language along the way instead.

SQL is going to be one of the best tools that you are able to use when it comes to handling some of the work that needs to be done with your database. It is a good method to use to store things to the database, make the tables that you want, bring in new information compare parts, bring other parts out, and so much more. When you are ready to work with your database and actually see some of the neat information hat is inside, make sure to take a look at some of the different things that you are able to do with the SQL language.

Chapter 2: Some of the Basic Commands We Need to Know

Now, before we are able to get too far into some of the coding that we are able to do with this kind of language, one of the first things that we need to learn a bit more about is some of the basic commands that come with this language, and how each of them is going to work. You will find that when you know some of the commands that come with any language, but especially with the SQL language, it will ensure that everything within the database is going to work the way that you would like.

As we go through this, you will find that the commands in SQL, just like the commands in any other language, are going to vary. Some are going to be easier to work with and some are going to be more of a challenge. But all of them are going to come into use when you would like to create some of your own queries and more in this language as well so it is worth our time to learn how this works.

When it comes to learning some of the basic commands that are available in SQL, you will be able to divide them into six categories and these are all going to be based on what you will be able to use them for within the system. Below are the six different categories of commands that you can use inside of SQL and they include:

Data Definition Language

The data definition language, or DDL, is an aspect inside of SQL that will allow you to generate objects in the database before arranging them the way that you would like. For example, you will be able to use this aspect of the system in order to add or delete objects in the database table. Some of the commands that you will be able to use with the DDL category include:

- Drop table
- Create table
- Alter table
- Create index
- Alter index
- Drop index
- Drop view

Data Manipulation Language

The idea of a DML, or data manipulation language, is one of the aspects of SQL that you will be bale to use to help modify a bit of the information that is out there about objects that are inside of your database. This is going to make it so much easier to delete the objects, update the objects, or even to allow for something new to be inserted inside of the database that you are working with. You will find that this is one of the best ways to make sure that you add in some freedom to the work that you are doing, and will ensure that you are able to change up the information that is already there rather than adding to something new.

Data Query Language

Along the same kinds of lines and thoughts here is the DQL or the data query language. This one is going to be kind of fun to work with because it is going to be one of the most powerful of the aspects that you are able to do with the SQL language you have. This is going to be even more true when you work with a modern database to help you get the work done.

When we work with this one, we will find that there is only really one command that we are able to choose from, and this is going to be the SELECT command. You are able to use this command to make sure that all of your queries are ran in the right way within your relational database. But if you want to ensure that you are getting results that are more detailed, it is possible to go through and add in some options or a special clause along with the SELECT command to make this easier.

Data control language

The DCL or the data control language is going to be a command that you are able to use when you would like to ensure you are maintaining some of the control that you need over the database, and when you would like to limit who is allowed to access that particular database, or parts of the database, at a given time. You will also find that the DCL idea is going to be used in a few situations to help generate the objects of the database related to who is going to have the necessary access to see the information that is found on that database.

This could include those who will have the right to distribute the necessary privileges of access when it comes to this data. This can be a good thing in order to use your business is dealing wit a lot of sensitive information and you only want a few people to get ahold of it all the time. Some of the different commands that you may find useful to use when working with the DCL commands are going to include:

1. Revoke
2. Create synonym
3. Alter password
4. Grand

Data Administration Commands

When you choose to work with these commands, you will be able to analyze and also audit the operation that is in the database. In some instances, you will be able to assess the overall performance with the help of these commands. This is what makes these good commands to choose when you want to fix some of the bugs that are on the system and you want to get rid of them so that the database

will continue to work properly. Some of the most common commands that are used for this type includes:

Start audit

Stop audit

One thing to keep in mind with the database administration and the data administration are basically different things when you are on SQL. The database administration is going to be in charge of managing all of the database, including the commands that you set out in SQL. This one is also a bit more specific to implementing SQL as well.

Transactional Control Commands

The final type of command that we are going to take a look at is going to be the transactional control commands. These are going to be some good commands that you are able to work with in SQL if you would like to have the ability to keep track of as well as manage some of the different transactions that are going to show up in the database that you are working with.

If you sell some products online for your website, for example, you will need to work with the transactional control commands to help keep track of the different options that the user is going to look for, to keep track of the profits, and to help you to manage this kind of website so that you know what is going on with it all of he time. there are a few options that you are able to work with when it comes to these transactional control commands, and a few of the most important ones that we need to spend our time on will include:

- Commit—this one is going to save all the information that you have about the transactions inside the database.

- Savepoint—this is going to generate different points inside the groups of transactions. You should use this along with the Rollback command.
- Rollback—this is the command that you will use if you want to go through the database and undo one or more of the transactions.
- Set transaction—this is the command that will assign names to the transactions in your database. You can use it to help add in some organization to your database system.

All of the commands that we have spent some time discussing in this chapter are going to be important to some of the work that we want to get done and will

ensure that we are going to find the specific results that we need out of our database. We will be able to spend some time looking through them in this book, but this can be a good introduction to show us what they mean, and how we will be able to use them for some of our needs later on.

Chapter 3: Creating Your SQL Tables

We have spent some time here looking at how to create a table in SQL. These tables are going to be there to hold onto the various types of data that we want to work with along the way. These tables are going to be able to make SQL work as efficiently as possible. What this means is that you are going to be in charge of creating the data that will go into these tables and SQL can make this easier than ever before.

There are a lot of times when we need to work with creating some of our own tables. We are not going to get very far with some of the databases that we want to create if we do not first work to create a few SQL tables These tables are going to help us to keep the information organized in the manner that we need, and will make it easier for some of the searches that we would like to accomplish to happen.

You are able to create any of the tables that you would like, and these can hold onto all sorts of information. Maybe you have a table that is responsible for holding onto your products for sale, and then one that holds onto all of the customer information, and one that holds onto some of the information about the payments and transactions that will happen in your business.

You can then go through and bring up the table that you would like and then

You can then go through and bring up the table that you would like and then search for the information that comes in that table. If you want to look at adding a new product to your system, for example, you would be able to go to the product table, and make the adjustments that are needed.

Depending on the kind of business that you are trying to run, and what you hope to get out of the process, you may find that there could be just a handful of tables to get the work done, and other times you will need to have quite a few of these tables to see the work be accomplished. You may need to go through ahead of time and figure out how many of these you are going to need before starting. This will help you to stay on track, and will ensure that you are going to be able to get all of the ones set up that you want so nothing is going to get missing in the process.

You will not be able to get things to work as well as you would like if you do not add in these tables along he way. this is the best way to ensure that your information and your data is going to stay as organized as possible and that you are going to be able to get some of the results that you are looking for as well when you do a query and more.

The first step that we have to take here is to work with a new list, such as like a shopping list, with all of the items that you want to work with found inside one of the tables in the database. We are going to start this of with a simple list to see how this is going to work, including three items, and the number of each item that you would like to work with as well. This will give us a good start to what we would like to see get done on this kind of process, and will ensure that we are going to get the results that we want out of it.

The three items that we are going to work with to help us get the table created here is a blouse, underwear, and pants. We will have to go through and make a new command for the table that will then be able to store this particular type of list. So the sample list that you are going to see inside of the SQL that you are working with will look like the following:

/**Shopping list:

Blouse (4)

Pants (1)

Underwear (2)

**/

CREATE TABLE shopping_list(name TEXT):

If you place this syntax into the program, you are going to see that the table will have on right side column. You only listed that you wanted to have the text inside of the database, and nothing about the amount of the items, so the database at this point is just going to have the list of the things that you want to purchase (the blouse, pants, and underwear). Now it is time to add in the amount of each that you would like to have in the database. We need to add in the integer for this data type with the help of this new code:

```
/**Shopping list:

Blouse (4)

Pants (1)

Underwear (2)

**/

CREATE TABLE shopping_list(name TEXT, quantity INTEGER);
```

Now you will notice here is that when you look at the table, there is going to be a new column that will be listed out in the database that you worked with before. The first column is going to have all of the different items that you would like to purchase, and then he second one is going to have the amount of each of these items that you would like to purchase as well. This is going to be a nice way to create the table that you want to use because it will ensure that you are able to get the unique identifier that you would like to have for these rows, and can help you when it is time to delete or update the rows later on. If you are reliant on just using the rows for your identification purposes, you are going to find that this causes a mess because these values are things that can change for you later on.

After you have had some time to name the rows that you would like to have in the table, it is time to fill all of this in. You will find at this time that the table is not going to have any of the information inside that you would like, and you will need to use the command of INSERT INFO to help get this started. you will be able to set up the first column that you would like to use and then label it as 1, and then the second one is going to be blouse, and then the third one is going to be how many are inside, which is four.

You are able to continue on with this process until you make sure that all of the information that you would like to use is found in the database as well. To make sure that all of these steps are going to be done in the proper manner you will

need to spend some time creating a good syntax. This will ensure that the table is going to look the way that you would like. The syntax, or an example of a code, that you are able to use to help with that shopping list that we made a bit before, and will ensure that everything is in order in the code, will include the following.

```
/**Shopping list:
Blouse (4)
Pants (1)
Underwear (2)
**/
CREATE TABLE shopping_list(id INTEGER PRIMARY KEY, name TEXT, quantity INTEGER);

INSERT INTO shopping_list VALUES (1, “Blouse”, 4);
INSERT INTO shopping_list VALUES (2, “Pants”, 1);
INSERT INTO shopping_list VALUES (3, “Underwear”, 2);
```

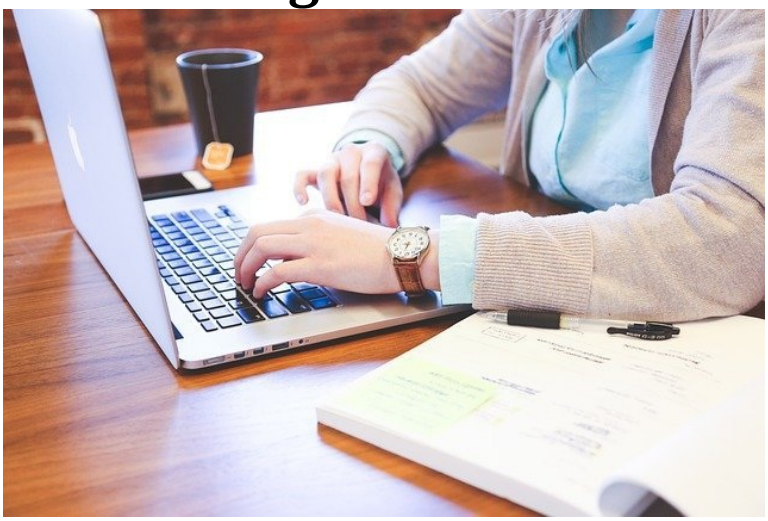
Once you have placed this information into your new table, you can take a look back at it any time during this process to see how it is going, make sure that the numbers are still the right way that you want. You can even change the numbers, delete some of the things that you are working on, and even add in some new items if you would like to expand the list.

There is so much that you are able to do when it comes to creating your own tables inside of this program. While we just started out with a simple formula that has three items inside and a few items of each, you will find that most businesses are going to need tables that are much larger than this. You will still need to go through the same process in order get this to work. Whether you just need five items or five hundred items inside of your table in order to keep all of your stuff inside the database, you will find that these same syntaxes are going to work, you will just need to use more of them.

As we can see with some of this, creating a table to work inside of the SQL program is going to be something that is fairly easy for us to work with. It is going to bring in a few different parts, and you do have the freedom of adding in the number of columns and rows that you would like in order to make sure that

the number of columns and rows that you would like in order to make sure that this database is going to work in the manner that you would like. You do have some freedom later on to make some of the necessary changes to your database, such as adding something new in, taking things out and more. There is a lot of freedom that is going to happen when you work with your own table in SQL, and it is one of the best ways to make sure that your data is safe and secure along the way, and that you will be able to pull it out and use it in the manner that you would like.

Chapter 4: Learning Phase One - The Basics



Now that we have been able to get this far, it is time for us to take a look at a few more of the basics that we need when it comes to learning with this language. We have looked at these a bit already, but now it is time for us to take a closer look at how to make it work, and some of the commands that are going to be important in this phase as well. Some of the different things that we need to explore when it comes to working with the SQL basics will include

Extract Data - SELECT statement

The first statement that we need to look at is the SELECT statement. This one is going to help us retrieve the items that we want out of the database. We can ask the language to bring back anything that we would like to view in the database, and the command that we need for this includes

```
SELECT * FROM PC;
```

With this one, we will be able to gather up all of the records that are in that database and match up with our object that are named PC. However, the columns and rows of the result set are not ordered. To help us order the fields in the result set, they should be listed with a comma in the desired order that we would like to see after the command of SELECT:

```
SELECT price, speed, hd, ram, cd, model, code
FROM Pc;
```

The following is the result of executing this query.

price	speed	hd	ram	cd	model	code
600.0	500	5	64	12x	1232	one
850.0	750	fourteen	128	40x	1121	2
600.0	500	5	64	12x	1233	3
850.0	600	Fourteen	128	40x	1121	four
850.0	600	8	128	40x	1121	5
950.0	750	Twenty	128	50x	1233	6
400.0	500	10	32	12x	1232	7
350.0	450	8	64	24x	1232	8
350.0	450	10	32	24x	1232	9
350.0	500	10	32	12x	1260	10
980.0	900	40	128	40x	1233	eleven

When we take a look at the vertical projection of the PC table, we will find that we are also able to obtain this kind of listing when we use only the fields that are the most necessary to it. A good example of this is when we would like to be able to get the information out about the speed, and nothing else, of our processor, or the amount of RAM that our computer holds. We would then be able to work with the following code to make this happen.

SELECT speed, ram FROM PC;

which will return the following data:

speed	ram
500	64
750	128
500	64
600	128
600	128
750	128
500	32

500	32
450	64
450	32
500	32
900	128

Elimination of duplicates - DISTINCT clause

One thing that we are going to notice here is that the vertical selection that we would like to work with could contain duplicate rows on occasion, if it does not contain a potential key ahead of time that will help us to identify the unique record that we want. In the PC table that we work with, the potential key is the code field, which is selected as the main key that we need for this table. Since this field will not be part of the query, the result that we get above has duplicate rows (for example, rows 1 and 3). If you want to get unique lines (say, we are only interested in various combinations when it comes to the processor speed, and memory size, and it will not necessarily be the characteristics of all the computers that are available and that you can use, then you would want to work with the following code:

```
SELECT DISTINCT speed, ram FROM Pc;
```

When we run this code, we are going to get the following output as a result

speed	ram
450	32
450	64
500	32
500	64
600	128
750	128
900	128

In addition to DISTINCT, the **ALL keyword** (all lines) can also be used, which is the default.

The ORDER BY Command

To sort the rows of the result set, you can sort by any specific amount of fields that you would like that were originally specified with the SELECT command or clause. To do this, use **the ORDER BY clause** . At the same time, *the field list* can contain both field names and their ordinal positions in the SELECT clause list.

This is going to be a good command that you are able to use when you want to make sure that you are not just getting some random order of information back at the time. instead, you want to put it in alphabetical or by the age of the data, to help you sort through that information and keep it as organized and easy to work with as possible. With this command, you can tell the database, and the SQL language how you would like things to play out when you are all done.

Sample - WHERE clause

The horizontal selection is implemented **by the WHERE clause** , which is written after the FROM clause. In this case, only those rows from the record source will fall into the result set, for each of which the *predicate* value is TRUE. That is, the predicate is checked for each record.

Boolean operators AND, OR, NOT and three-valued logic - Predicates

Predicates are expressions that take on a truth value. They can be either a single expression or any combination of an unlimited number of expressions constructed using the Boolean operators **AND** , **OR** or **NOT** . In addition, the **IS** SQL statement can be used in these combinations, as well as parentheses to specify the order in which operations are performed.

A SQL predicate can take one of three values: **TRUE** (true), **FALSE** (false), or **UNKNOWN** (unknown). The following predicates are an exception: **NULL** (no value), **EXISTS** (existence), **UNIQUE** (uniqueness), and **MATCH** (match), which cannot take the value UNKNOWN.

This can be a good way that we would want to go through and make sure that we are getting the right information out of the database that we are using. You would be able to use this to make sure that the data is only coming back as you would like.

We can take a look at the example of a search engine for this one. You do not

We can take a look at the example of a search engine for this one. You do not want to have a bunch of items showing up that the customer does not want, especially if you have hundreds of items. You can use the Boolean operators to help you determine what items on the database match up to the query from the customer, and which ones do not, providing them with a seamless experience that will help them to get the most out of what you are providing in the website.

Comparison Queries

There are times when you will want to do a comparison between what is found on your database, and the query that you are completing. And maybe sometimes you would even like to go through and compare more than one part of the database to see what is all there. You could try to do this by hand or manually, but it will not take long before you find that, with a potential for millions of data points in the database, this is going to take a long time and be too hard to handle.

Instead, we can work with some of the comparison operators that are present in the SQL language. These will allow us to figure out if the different parts are equal, not equal, or something else. There are a number of these found inside of the SQL language, and this can really help us to find out what is there and whether it is going to work for our needs or not.

A Look at Queries

While we have spent a little bit of time taking a look at some of the commands and queries that we are able to use when it comes to working in the SQL language, it is time for us to go more in depth about these queries and what they are able to do for some of our needs along the way as well.

When we are working on our own business database and it is all set up the way that we would like, it is going to be possible that at one point or another you will want to do a search in order to make sure you are able to find the perfect information inside of all that. This is going to make it easier for us to find the easier information and results that we want. But we do have to make sure that the database is set up in the right manner so that we can use the right commands, and see that it is fast and accurate in the process.

Think of this like when someone comes to your website, searching for that particular product that they would like to purchase. Do you want them to get stuck on the website that is slow, and have them see results that have nothing to do with the item they wanted? Or would you like a fast search that was helpful and will encourage the person to make that purchase? Of course, for the success of your business you are more likely to want the second option, and we can take a look at how to set up your database and work with queries in order to make this happen.

Working with the Queries

When you do set up the query that you would like to use, you will find that you

When you do set up the query that you would like to use, you will find that you are basically sending out an inquiry to the database that you already set up. You will find that there are a few methods to do this, but the SELECT command is going to be one of the best options to make this happen, and can instantly bring back the information that we need from there, based on our search.

For example, if you are working with a table that is going to hold onto all of he products that you offer for sale, then you would be able to use the command of SELECT in order to find the best selling products, or ones that will meet another criteria that you have at that time. The request is going to be good on any of the information of the product that is stored in the database, and you will see that this is done pretty normally when we are talking about work in a relational database.

Working with the SELECT Command

Any time that you have a plan to go through and query your database, you will find that the command of SELECT is going to be the best option to make this happen. This command is important because it is going to be in charge of starting and then executing he queries that you would like to send out. in many cases, you will have to add something to the statement as just sending out SELECT is not going to help us to get some of the results that you want. You can choose the product that you would like to find along with the command, or even work with some of the features that show up as well.

Whenever you will work with the SELECT command on one of your databases inside of the SQL language, you will find that there are four main keywords that we are able to focus on. These are going to be known as the four classes that we need to have present in order to make sure that we are able to complete the command that we want and see some good results. These four commands are going to include:

- SELECT—this command will be combined with the FROM command in order to obtain the necessary data in a format that is readable and organized. You will use this to help determine the data that is going to show up. The SELECT clause is going to introduce the columns that you would like to see out of the search results and then you can use the FROM in order to find the exact point that you need.

- FROM—the SELECT and the FROM commands often go together. It

is mandatory because it takes your search from everything in the database, down to just the things that you would like. You will need to have at least one FROM clause for this to work. A good syntax that would use both the SELECT and the FROM properly includes:

- SELEC [* | ALL | DISTINCT COLUMN1, COLUMN2]
- FROM TABLE1 [, TABLE2];

- WHERE—this is what you will use when there are multiple conditions within the clause. For example, it is the element in the query that will display the selective data after the user puts in the information that they want to find. If you are using this feature, the right conditions to have along with it are the AND and OR operators. The syntax that you should use for the WHERE command includes:
  - SELEC [* | ALL | DISTINCT COLUMN1, COLUMN2]
  - FROM TABLE1 [, TABLE2];
  - WHERE [CONDITION1 | EXPRESSION 1]
  - [AND CONDITION2 | EXPRESSION 2]
- ORDER BY—you are able to use this clause in order to arrange the output of your query. The server will be able to decide the order and the format that the different information comes up for the user after they do their basic query. The default for this query is going to be organizing the output going from A to Z, but you can make changes that you would like. The syntax that you can use for this will be the same as the one above, but add in the following line at the end:
  - ORDER BY COLUMN 1 | INTEGER [ASC/DESC]

You will quickly see that all of these are helpful an dyou can easily use them instead of the SELECT comand if you would like. They can sometimes pull out the informaiton that you need from the database you are working with in a more efficient manner thatn you will see with just the SELECT command. But there are going to be many times when you will find that the SELECT command will be plenty to help you get things done when it is time to search your database as well.

A Look at Case Sensitivity

Unlike some of the othr coding languages that are out there and that you may be tempted to use on your database searches, you may find that the case sensitivity in SQL is not going to be as important as it is in some of those other ones. You

are able to use uppercase or lowercase words as you would like, and you can use either typing of the word and still get the part that you need out of the database. It is even posisble for us to go thrugh and enter in some clauses and statements in uppercase or lowercase, without having to worry too much about how these commands are going to work for our needs.

However, there are a few exceptions to this which means there are going to be times when we need to worry about the case sensitivity that is going to show up in this language a bit more than we may want to. One of the main times for this is when we are looking at the data objects. For the most part, the data that you are storing should be done with uppercase letters. This is going to be helpful because it ensures that there is some consistency in the work that you are doing and can make it eaiser for us to get the results that we want.

For example, you could run into some issues down the road if one of he users is going through the database and typing in JOHN, but then the next person is typing in John, and then the third person is going through and typing in john to get the results. If you make sure that there is some consistency present, you will find that it is eaiser for all of the users to get the information that they want, and then you can make sure that you are able to provide the relevant informaiton back when it is all done.

In this case, working with letters in uppercase is often one of the easiest ways to work with this because it is going to make it easier and the user is going to see that this is the normal in order options as well. If you choose to not go with uppercase in this, then you should try to find some other method that is going to keep the consistency that you are looking for during the whole thing. This allows the user a chance to figure out what you are doing, and will help them to find what they ened with what is inside of their queries.

As you go through this, you may notice that these queries and transactions that you are able to create for the database that you aer working on are going to be improtant to the whole system and ensuring that it is actually going to work in the manner that you would like. In the beginning, this may feel like a lot of buy work to keep up with and that it is not worth your time or energy to do, or that it is not that big of a deal. We may assume that with a good query from the user, they will be able to find all of the informaiton that they need in no time.

However, it will not take long working with the SQL language and more to figure out that this is going to make everyone a bit frustrated. It is always better to go througha nd set up a good search query wih SQL and make sure that your data is organized and ready to go, so that everyone is bale to find what they

would like out of the database each time. This is going to be one of the best ways to ensure that you are going to provide the user with a good experience on your website, so take some extra care when you are working with this.

There are then going to be some times when the user is going to be able to do their searching and look up some information just by doing a seach and finding the informatio that they want to use inside of that particular database. This is when they will want to ensure that the database is set up in a manner so that they are going to find the task or the item that they want right away, rather than having to continue to search and not having to put in many keywords in order to hopefully find the thing that they want along he way.

No matter which mehod you choose to use, or what seems to work best for the work that you are trying to accomplish with all of this, you will find that it is very improtant along the way that your commands fall in the right place each time and that they will lead you to the place where you want to be in your database. When you make sure that the table is going to be set up in the right manner so that you, as well as any potential users, are able to find what they want quickly and efficienlty, you will find that it is so much easier to enhance the experience and grow your business.

This chapter is improtant because it shows us how to get all of this done. It helped us to see what was going on in the database nad provided us with the right commands in order to bring out that informaiton and make it work for some of our needs as well. While the SELECT command is often the one that is used to help contrl the database and make it work well, and you can choose the one that you would like to use. Setting up the database in a good and storng manner, and being careful with the commands that you are using is one of the best ways to ensure that you are going to get the best results with this along the way, while providing your users and customers with the shopping and transaction experience that they want to enjoy it and that will get them to come back again.

Chapter 5: A Bit About Subqueries

The next topic that we are going to spend some time talking about is how to work with the subqueries in this language. We will take a look at some of the queries in another chapter in this guidebook, but getting a good look at what these subqueries is all about, and how we are able to add them into some of the codes that we are doing is going to be very important as well. Some of the things that we need to know when it comes to the subqueries in SQL will include:

More about subqueries

Note that in general, a query returns a **plurality of** values. Therefore, using a subquery when you are working with a **WHEREclause** without **EXISTS** , **IN** , **ALL,** and **ANY** statements that give a **Boolean** value can result in a query runtime error.

Example. Find PC models and prices, the cost of which exceeds the minimum cost of PC notebooks:

```
SELECT DISTINCT model, price
FROM PC
WHERE price
(SELECT MIN(price)
FROM Laptop);
```

You will find that this is the right way to work with this kind of request because the scalar value of the price is going to be compared back to the subquery that returns just one value to you.

The result that you will get with this is going to depend on what you are doing, but in this option, we will find that we will get the results of three models of PC's because that is how we worked on the code. You can input this into your SQL language and see what results you get.

On the other hand, a subquery that returns multiple rows and contains multiple columns can naturally be used in **the FROM clause** . This allows you to limit the set of columns and / or rows when performing the join operation of tables.

Example. Display the manufacturer, type, model and frequency of the processor for PC notebooks whose processor frequency exceeds 600 MHz.

This request can be formulated, for example, as follows:

```
SELECT prod.maker, lap.*
FROM (SELECT 'Laptop' AS type, model, speed
FROM Laptop
WHERE speed 600) AS lap INNER JOIN
(SELECT maker, model
FROM Product) AS prod ON lap.model = prod.model;
```

You can take a moment to go through all of this and see how we are able to make it work for some of our own needs. See what the output gives to you and if this provides you with the results that you are looking for in the process.

In addition, you will find that some of these subqueries are going to be found in the SELECT cause. This is going to make it possible to formulate a query very compactly.

We can take some time to look at how this is done. For this example, we want to be able to find the difference between the average price o a notebook PC, and the PC that we want. Or, we would want to figure out, on average, whether the notebook is going to be more expensive than we will see with the regular PC. A good code that we can use for this will be the following:

```
SELECT (SELECT AVG(price)
FROM Laptop) -
(SELECT AVG(price)
FROM PC) AS dif_price;
```

Type conversion

In implementations of the SQL language, implicit type conversion can be performed. So, for example, in T-SQL, when comparing or combining values of the **smallint** and **int** types , data of the **smallint** type **is** implicitly converted to the **int** type . Details on explicit and implicit type conversion in MS SQL Server can be found in BOL.

Example. Display the average price of notebook PCs with the preceding text "average price =".

Attempt to execute request

```
SELECT 'average price = ' + AVG(price) FROM laptop;
```

will result in an error message. This message is going to tell us that we are trying to do what is known as an implicit conversion from the varchar data type over to money, and this is something that we are not allowed to do. This is because the system is not able to move one data type, especially this kind, over to another type, or a money type in this situation.

In these kinds of situations, the process of using an explicit type of conversion is going to help. If you end up in a situation with that error message from before, you can then choose to work with the CONVERT function instead. But we have to remember that even with this, the function you are working with will not be standardized. Therefore, for portability purposes, it is recommended that you use the standard **CAST** expression.

We can go through and make this a little bit easier to see. If we go through and rewrite the request that we want to use, working with the form below:

```
SELECT 'average price = ' + CAST(AVG(price) AS CHAR(15)) FROM laptop;
```

We will find that it is a lot easier to go through and get the results that we would like. The answer that you get here is going to be that the average price is $1410.44, but this number can change based on the input that we get, or what

you are hoping to do with this kind of code.

We used an explicit **CAST** type conversion expression to cast the average price to a string representation. The syntax of the **CAST** expression is very simple:

CAST (AS)

It should be borne in mind, firstly, that not any type conversions are possible (the standard contains a table of valid data type conversions). Secondly, the result of the **CAST** function for an expression value of NULL will also be NULL.

We can also take some time to look at another example of how this is going to work to our advantage as well. In this one, we are going to use the same idea in order to figure out the average year that certain ships were launched, working with the Ships table that we are going to work with here. The command that we want to use in order to make this happen includes:

SELECT AVG(launched) FROM ships;

This one is going to give us the result of 1926. In principle, everything is correct, because as a result we received what we asked - YEAR. However, if you take a look at this, the actual mean of all the years does not quite end up with the same year as above. There are a few decimal points behind it. It should be something that we note that the aggregate functions we are working with, outside of the COUNT function that will provide us with an integer all of time), will inherit the type of data that is in the possessed values.

Since we went through this and launched a field that was an integer, we are going to end up with an average value. And since the integer is usually not going to have a fraction or a decimal with it, we are going to discard this part of the year to keep things as simple as possible.

And if we are interested in the result with a given accuracy, say, up to two decimal places? Applying the **CAST** expression to the average will not do anything for the above reason. Really, this will:

SELECT CAST(AVG(launched) AS NUMERIC(6,2)) FROM ships;

will return the value 1926.00. Therefore, **CAST** needs to be something that we are going to apply to the argument that we will get with one of these aggregate

functions. The coding that we can use for this one will be below:

SELECT AVG(CAST(launched AS NUMERIC(6,2))) FROM ships;

The result is 1926.238095. Not that again. The reason is that when calculating the average, an implicit type conversion was performed. Let's take one more step:

SELECT CAST(AVG(CAST(launched AS NUMERIC(6,2))) AS NUMERIC(6,2)) FROM ships;

As a result, we get what we need - 1926.24. You can guess from here though that we don't want to figure out exactly how long that .24 is supposed to stand for, and it can be kind of a pain, so we want to make sure that make it easier. This is where we are going to work with more of an implicit type of conversion. The commands that we are able to use to make this one happen includes:

SELECT CAST(AVG(launched*1.0) AS NUMERIC(6,2)) FROM ships;

In the option that we did above, you will see that we were relying on the implicit type of conversion to go with our integer argument to the exact type of nuuber, and then we went through and multiplied it by a real unit. And finally, we were able to apply it to the explicit cast of the result type that would become our aggregate function here.

Similar type conversions can be performed using the **CONVERT** function:

SELECT CONVERT(NUMERIC(6,2),AVG(launched*1.0)) FROM ships;

This will help us to finish up the conversion that we would like to see with some of the work, and will ensure that we are going to really make sure that all of this works in the manner that it should.

The **CONVERT** function has the following syntax:

CONVERT ([()], [,])

The main difference between the **CONVERT** function and the **CAST** function is that the first allows you to format data (for example, temporal data of the **datetime** type) when converting it to a character type and specify the format when reversing. The different integer values of the optional *style*

argument correspond to specific formats. Consider the following *example.*

```
SELECT CONVERT(char(25),CONVERT(datetime,'20030722'));
```

Here we convert the string representation of the date to the **datetime** type, and then perform the inverse transformation to demonstrate the formatting result. Because the style argument is not specified, the default value (0 or 100) is used. As a result, we get

```
Jul 22 2003 12:00
AM
```

Below are some other values of the *style* argument and the result obtained in the above example. Note that *style* values greater than 100 result in a four-digit display of the year.

style	format
one	07/22/03
eleven	07/03/22
3	07/22/03
121	2003-07-22 00: 00: 00.000

A list of all possible values of the *style* argument can be found in BOL.

CASE statement

Suppose you want to list all PC models with their prices. Moreover, if the model is not available for sale (not in the table PC), then instead of the price display the text: "Not available".

A list of all PC models with prices can be obtained using the request:

```
SELECT DISTINCT product.model, price FROM product LEFT JOIN pc c
ON product.model=c.model
WHERE product.type='pc';
```

In the result set, the missing price will be replaced by a NULL value:

model	price
1121	850
1232	350
1232	400
1232	600
1233	600
1233	950
1233	980
1260	350
2111	Null
2112	Null

To replace NULL values with the desired text, you can use the **CASE** statement :

```
SELECT DISTINCT product.model,
CASE WHEN price IS NULL THEN 'Нет в наличии' ELSE CAST(price AS CHAR(20)) END price
FROM product LEFT JOIN pc c ON product.model=c.model
WHERE product.type='pc'
```

The **CASE** statement, depending on the specified conditions, returns one of the many possible values. In our example, the condition is a check for NULL. If this condition is met, the text "Out of stock" is returned; otherwise (**ELSE**), the price value is returned. There is one fundamental point. Since the result of the **SELECT statement is** always a table, all values of any column must have the same data type (taking into account the implicit type conversion). Therefore, we cannot, along with price (numerical type), derive a symbolic constant. This is why type conversion is applied to the price field to bring its values to a symbolic representation. As a result, we get:

model	price
1121	850
1232	350

1232	400
1232	600
1233	600
1233	950
1233	980
1260	350
2111	Not available
2112	Not available

The **CASE** statement can be used in one of two syntactic forms of notation:

1st form

CASE
WHEN
THEN
...
WHEN
THEN
[ELSE]
End

2nd form

CASE
THEN
WHEN
...
THEN
WHEN
[ELSE]

End

All **WHEN** clauses must have the same syntactic form, i.e. You cannot mix the first and second forms. Using the first syntactic form, the **WHEN** clause is satisfied as soon as the value of *the expression* being *tested* becomes equal to the value of the expression specified in the **WHEN** clause. When using the second syntactic form, the **WHEN** clause is satisfied as soon as the *predicate* takes the value TRUE. When the condition is satisfied, the **CASE** statement returns the value specified in the corresponding **THEN** clause. If none of the **WHEN** conditions if it fails, then the value specified in the **ELSE clause** will be used. If there is no **ELSE** , a NULL value will be returned. If several conditions are satisfied, the value of the **THEN** clause of the first of them will be returned.

In the above example, the second form of the **CASE** statement was used.

Note that to check for NULL, the standard offers a shorter form of the operator - **COALESCE** . This operator has an arbitrary number of parameters and returns the value of the first non-NULL. For two parameters, the COALESCE (A, B) statement is equivalent to the following **CASE** statement:

CASE WHEN A IS NOT NULL THEN A ELSE B END

The solution to the above example when using the **COALESCE** operator can be rewritten as follows:

```
SELECT DISTINCT product.model,
COALESCE(CAST(price as CHAR(20)),'not available') price
FROM product LEFT JOIN pc c ON product.model=c.model
WHERE product.type='pc';
```

The use of the first syntactic form of the **CASE** operator can be demonstrated by the following *example* : List all available PC models with price. Mark the most expensive and cheapest models.

```
SELECT DISTINCT model, price,
CASE price WHEN (SELECT MAX(price) FROM pc) THEN 'most expensive'
WHEN (SELECT MIN(price) FROM pc) THEN 'most cheap'
```

```
ELSE 'average price' END comment
FROM pc ORDER BY price;
```

As a result of the query, we get

model	price	comment
1232	350	The cheapest
1260	350	The cheapest
1232	400	average price
1233	400	average price
1233	600	average price
1121	850	average price
1233	950	average price
1233	980	The most expensive

Chapter 6: Transact-SQL Functions for Date / Time Processing

The SQL-92 standard only specifies functions that return a system date / time. For example, the CURRENT_TIMESTAMP function returns both date and time at once. Plus there are functions that return one thing.

Naturally, due to such limitations, language implementations expand the standard by adding functions that facilitate the work of users with data of this type. Here we look at date / time processing functions in T-SQL.

DATEADD Function

Syntax

DATEADD (*datepart* , *number* , *date*)

This function returns a value of type **datetime** , which is obtained by adding to

the date the number of intervals of type *datepart* equal to *number* . For example, we can add any number of years, days, hours, minutes, etc. to a given date. Valid values for the *datepart* argument *are* given below and are taken from BOL.

Datepart	Permissible abbreviations
Year - year	yy, yyyy
Quarter - quarter	qq, q
Month - month	mm, m
Dayofyear - day of the year	your, the
Day - day	dd d
Week is a week	wk, ww
Hour - time	hh
Minutes - minutes	my n
Second	ss, s
Millisecond - milliseconds	ms

Let use 01/01/2004, and we want to find out what day will be in a week. We can write

SELECT DATEADD(day, 7, current_timestamp)

but we can

SELECT DATEADD(ww, 1, current_timestamp)

As a result, we get the same thing; something like 2004-01-07 19: 40: 58.923.

However, we cannot write in this case

SELECT DATEADD(mm, 1/4, current_timestamp)

because the fractional part of the value of the *datepart* argument *is* discarded,

and we get 0 instead of one fourth and, as a result, the current day.

In addition, we can use the T-SQL **GETDATE ()** function instead of **CURRENT_TIMESTAMP** with the same effect. The presence of two identical functions is supported, apparently, pending the subsequent development of the standard.

Example . Determine what will be the day one week after the last flight.

SELECT DATEADD(day, 7, (SELECT MAX(date) max_date FROM pass_in_trip))

Using a subquery as an argument is acceptable because this subquery returns the ONLY **datetime** value.

DATEDIFF Function

Syntax

DATEDIFF (*datepart* , *startdate* , *enddate*)

The function returns the interval of time elapsed between two timestamps - *startdate* (start mark) and *enddate* (end mark). This interval can be measured in different units. The options are determined by the *datepart* argument and are listed above for the **DATEADD** function.

Example . Determine the number of days elapsed between the first and last completed flights.

SELECT DATEDIFF(dd, (SELECT MIN(date) FROM pass_in_trip), (SELECT MAX(date) FROM pass_in_trip))

Example . Determine flight duration 1123 in minutes.

It should be noted here that the departure time (*time_out*) and arrival time (*time_in*) are stored in fields of the **datetime** type of the Trip table. Note that, up to version 2000, SQL Server does not have separate temporal data types for the date and time that are expected to appear in the next version

(Yukon). Therefore, when only time is inserted in the **datetime** field (for example, UPDATE trip SET time_out = '17: 24: 00 'WHERE trip_no = 1123), the time will be supplemented by the default date value (' 1900-01-01 ').

Asking decision:

SELECT DATEDIFF(mi, time_out, time_in) dur FROM trip WHERE trip_no=1123,

(which gives -760) will be incorrect for two reasons.

Firstly, for flights that depart on one day and arrive on the next, the value calculated in this way will be incorrect. Secondly, it is unreliable to make any assumptions regarding the day, which is present only because of the need to conform to the **datetime** type.

But how to determine that the plane landed the next day? This helps the description of the subject area, which says that the flight cannot last more than a day. So, if the arrival time is not more than the departure time, then this fact takes place. Now the second question: how to calculate only the time, no matter what day it is?

The T-SQL **DATEPART** function can help **here** .

Datepart function

Syntax

DATEPART (*datepart* , *date*)

This function returns an integer that represents a specified argument *datepart* part specified by the second argument date (*date*).

The list of valid values for the *datepart* argument described earlier in this section is complemented by another value

Datepart	Permissible abbreviations

Weekday - day of the week	dw

Note that the value returned by the **DATEPART** function (day of the week number) in this case depends on the settings that can be changed using the **SET DATEFIRST statement** , which sets the first day of the week. For some, Monday is a tough day, and for some, Sunday. By the way, the last value is accepted by default.

But back to our example. Assuming that the departure / arrival time is a multiple of a minute, we can define it as the sum of hours and minutes. Since the date / time functions work with integer values, we reduce the result to the smallest interval - minutes. So, the departure time of flight 1123 in minutes

```
SELECT DATEPART(hh, time_out)*60 + DATEPART(mi, time_out) FROM trip WHERE trip_no=1123
```

and arrival time

```
SELECT DATEPART(hh, time_in)*60 + DATEPART(mi, time_in) FROM trip WHERE trip_no=1123
```

Now we must compare whether the arrival time exceeds the departure time. If so, subtract the second from the first to get the duration of the flight. Otherwise, you need to add one day to the difference (24 * 60 = 1440 minutes).

```
SELECT CASE WHEN time_dep=time_arr THEN time_arr-time_dep+1440 ELSE time_arr-time_dep END dur FROM

(SELECT DATEPART(hh, time_out)*60 + DATEPART(mi, time_out) time_dep, DATEPART(hh, time_in)*60 + DATEPART(mi, time_in) time_arr FROM trip WHERE trip_no=1123) tm
```

Here, in order not to repeat long constructions in the CASE statement, a subquery is used. Of course, the result turned out to be rather cumbersome, but absolutely correct in the light of the comments made to this problem.

Example . Determine the date and time of departure of flight 1123.

The table of completed flights Pass_in_trip contains only the date of the flight, but not the time, because in accordance with the subject area, each flight can be operated only once a day. To solve this problem, you need to add the time from the Trip table to the date stored in the Pass_in_trip table

```
SELECT pt.trip_no, DATEADD(mi, DATEPART(hh,time_out)*60 +
DATEPART(mi,time_out), date) [time]
FROM pass_in_trip pt JOIN trip t ON pt.trip_no=t.trip_no WHERE
t.trip_no=1123
```

Having completed the request, we obtain the following result:

Trip_no	Time
1123	2003-04-05 16: 20: 00.000
1123	2003-04-08 16: 20: 00.000

DISTINCT is necessary here to exclude possible duplicates, since the flight number and date are duplicated in this table for each passenger of this flight.

Datename function

Syntax

DATENAME (*datepart* , *date*)

This function returns the symbolic representation of the component (*datepart*) of the specified date (*date*). The argument specifying the date component can take one of the values listed in the table above.

This gives us a simple opportunity to concatenate date components, getting any desired presentation format. For example, construction:

```
SELECT DATENAME ( weekday , '2003-12-31' )+', '+DATENAME ( day ,
'2003-12-31' )+' '+ DATENAME ( month , '2003-12-31' )+' '+DATENAME (
year , '2003-12-31' )
```

will give us the following result

Wednesday, 31 December
2003

It should be noted that this function detects difference values of *day* and *dayofyear* argument *datepart* . The first gives a symbolic representation of the day of the specified date, while the second gives a symbolic representation of the day from the beginning of the year.

SELECT DATENAME (day , '2003-12-31')

will give us 31, and

SELECT DATENAME (dayofyear , '2003-12-31')

- 365.

In some cases, the **DATEPART** function can be replaced with simpler functions. Here they are:

DAY (*date*) - An integer representation of the day of the specified date. This function is equivalent to the **DATEPART** (*dd* , *date*) function.

MONTH (*date*) - An integer representation of the month of the specified date. This function is equivalent to the **DATEPART** (*mm* , *date*) function.

YEAR (*date*) - An integer representation of the year of the specified date. This function is equivalent to the **DATEPART** (*yy* , *date*) function.

Chapter 7: Data Modification Operators

The Data Manipulation Language (DML), in addition to the **SELECT statement** , which extracts information from the database, includes statements that change the state of the data. These operators are:

INSERT - Adding records (rows) to the database table

UPDATE - Updating data in a column of a database table

DELETE - Delete records from a database table

INSERT statement

The **INSERT statement** inserts new rows into the table. In this case, the column values may be literal constants or may be the result of a subquery. In the first case, a separate **INSERT statement is** used to insert each row; in the second case, as many rows will be inserted as the number returned by the subquery.

Operator Syntax

INSERT INTO < *table name* [(< *column name* , ...)]

{VALUES (< *column value* , ...)}

| < *query expression*
| {DEFAULT VALUES};

As you can see from the syntax presented, a list of columns is optional. If it is absent, the list of inserted values must be complete, i.e. provide values for all columns of the table. In this case, the order of values must correspond to the column order specified by the **CREATE TABLE** statement for the table into which the rows are inserted. In addition, each of these values must be of the same type (or cast to it) as the type defined for the corresponding column in the **CREATE TABLE** statement. As an example, consider inserting a row into a *Product* table created by the following **CREATE TABLE** statement:

```
CREATE TABLE [dbo].[product] (
[maker] [char] (1) NOT NULL ,
[model] [varchar] (4) NOT NULL ,
[type] [varchar] (7) NOT NULL )
```

Suppose you want to add the PC model 1157 of manufacturer B to this table. This can be done with the following operator:

```
INSERT INTO Product VALUES ('B', 1157, 'PC');
```

If you specify a list of columns, you can change the "natural" order of their sequence:

```
INSERT INTO Product (type, model, maker) VALUES ('PC', 1157, 'B');
```

It would seem that this is a completely redundant feature, which makes the design only more cumbersome. However, it becomes advantageous if the columns have default values. Consider the following table structure:

```
CREATE TABLE [product_D] (
[maker] [char] (1) NULL ,
[model] [varchar] (4) NULL ,
[type] [varchar] (7) NOT NULL DEFAULT 'PC' )
```

Note that here the values of all columns have default values (the first two are NULL, and the last column is *type* - 'PC'). Now we could write:

```
INSERT INTO Product_D (model, maker) VALUES (1157, 'B');
```

In this case, the missing value when inserting a row will be replaced by the default value - 'PC'. Note that if a default value is not specified for a column in the **CREATE TABLE** statement and a **NOT NULL** constraint is specified that prohibits the use of NULL in this column of the table, then the default value is NULL.

The question arises: is it possible not to specify a list of columns and, nevertheless, use the default values? The answer is yes. To do this, instead of explicitly specifying a value, use the reserved word **DEFAULT** :

```
INSERT INTO Product_D VALUES ('B', 1158, DEFAULT);
```

Since all columns have default values, to insert a row with default values, one could write:

```
INSERT INTO Product_D VALUES (DEFAULT, DEFAULT, DEFAULT);
```

However, for this case, the special **DEFAULT VALUES** construct is designed (see the syntax of the operator), with which the above operator can be rewritten as:

```
INSERT INTO Product_D DEFAULT VALUES;
```

Note that when inserting a row into a table, all restrictions imposed on this table are checked. These may be primary key or unique index constraints, **CHECK** type constraints, referential integrity constraints. If any restriction is violated, the row insertion will be rejected.

Consider now the use of a subquery. Suppose we need to insert into the *Product_D* table all the rows from the *Product* table related to personal computer models (*type* = 'PC'). Since the values we need are already in a certain table, the formation of inserted rows manually, firstly, is inefficient, and, secondly, it may allow input errors. Using a subquery solves these problems:

```
INSERT INTO Product_D SELECT * FROM Product WHERE type = 'PC';
```

The use of the symbol "*" in a subquery is justified in this case, since the order of the columns is the same for both tables. If this were not so, then a column list should be used either in the **INSERT statement** , or in a subquery, or in both

places, which would bring the order of the columns in correspondence:

```
INSERT INTO Product_D(maker, model, type)
SELECT * FROM Product WHERE type = 'PC';
```

or

```
INSERT INTO Product_D
SELECT maker, model, type FROM Product WHERE type = 'PC';
```

or

```
INSERT INTO Product_D(maker, model, type)
SELECT maker, model, type FROM Product WHERE type = 'PC';
```

Here, as before, you can specify not all columns if you want to use the available default values, for example:

```
INSERT INTO Product_D (maker, model)
SELECT maker, model FROM Product WHERE type = 'PC';
```

In this case, the default type 'PC' for all inserted rows will be substituted into the *type* column of the *Product_D* table.

Note that when using a subquery containing a predicate, only those rows will be inserted for which the predicate value is TRUE (not UNKNOWN!). In other words, if the *type* column in the *Product* table allowed a NULL value, and this value was present in a number of rows, then these rows would not be inserted in the *Product_D* table.

To overcome the restriction on inserting a single line in the **INSERT statement** when using **VALUES, the** artificial use of the subquery forming the line with the **UNION ALL** clause allows. So if we need to insert multiple lines using a single **INSERT statement** , we can write:

```
INSERT INTO Product_D
SELECT 'B' AS maker, 1158 AS model, 'PC' AS type
UNION ALL
SELECT 'C', 2190, 'Laptop'
UNION ALL
```

SELECT 'D', 3219, 'Printer';

Using **UNION ALL is** preferable to **UNION,** even if there are no duplicate rows, as in this case, a check will not be performed to eliminate duplicates.

Insert rows into a table containing an auto-increment field

Many commercial products allow the use of auto-incrementing columns in tables, i.e. fields whose value is generated automatically when adding new records. Such columns are widely used as primary keys of the table, because they automatically provide uniqueness. A typical example of a column of this type is a sequential counter, which when inserting a row generates a value one greater than the previous value (the value obtained by inserting the previous row).

The following is an example of creating a table with an auto-incrementing column (*code*) in MS SQL Server.

```
CREATE TABLE [Printer_Inc] (
[code] [int] IDENTITY(1,1) PRIMARY KEY ,
[model] [varchar] (4) NOT NULL ,
[color] [char] (1) NOT NULL ,
[type] [varchar] (6) NOT NULL ,
[price] [float] NOT NULL )
```

An auto-incrementing field is defined using the **IDENTITY (1, 1)** construct. In this case, the first parameter of the **IDENTITY (1)** property determines which value starts the count, and the second - which step will be used to increment the value. Thus, in our example, the first inserted record will have a value of 1 in the *code* column, the second - 2, etc.

Since the value is generated automatically in the *code* field, the operator

```
INSERT INTO Printer_Inc VALUES (15, 3111, 'y', 'laser', 2599);
```

will lead to an error even if there is no row in the table with a value in the *code* field equal to 15. Therefore, to insert a row into the table, we simply will not indicate this field in the same way as in the case of using the default value, i.e.

```
INSERT INTO Printer_Inc (model, color, type, price)
VALUES (3111, 'y', 'laser', 2599);
```

VALUES (3111, 'y', 'laser', 2599);

As a result of this statement, information about the model 3111 color laser printer, the cost of which is $ 2599, will be inserted into the *Printer_Inc* table. In the **code** field there will be a value that can only happen to be equal to 15. In most cases, this is enough, because the value of the auto-incrementing field, as a rule, does not carry any information; the main thing is that it is unique.

However, there are times when you need to substitute a very specific value in the auto-incrementing field. For example, you need to transfer existing data to a newly created structure; however, this data is involved in a one-to-many relationship from the one side. Thus, we cannot allow arbitrariness here. On the other hand, we do not want to abandon the auto-incrementing field, because it will simplify data processing during the subsequent operation of the database.

Since the standard of the SQL language does not imply the presence of auto-incrementing fields, there is accordingly no single approach. Here we show how this is implemented in MS SQL Server.

Operator disables (ON value) or enables (OFF) the use of auto-increment. Therefore, to insert a line with the value 15 in the *code* field, you need to write

```
SET IDENTITY_INSERT Printer_Inc ON;
INSERT INTO Printer_Inc(code, model, color, type, price)
VALUES (15, 3111, 'y', 'laser', 2599);
```

Please note that the list of columns in this case is required, i.e. we cannot write like this:

```
SET IDENTITY_INSERT Printer_Inc ON;
INSERT INTO Printer_Inc
VALUES (15, 3111, 'y', 'laser', 2599);
```

or more so

```
SET IDENTITY_INSERT Printer_Inc ON;
INSERT INTO Printer_Inc(model, color, type, price)
VALUES (3111, 'y', 'laser', 2599);
```

In the latter case, the value cannot be substituted into the skipped column *code* , because auto increment is disabled.

It is important to note that if the value 15 turns out to be the maximum in the *code* column, then the numbering will continue from the value 16. Naturally, if you enable auto-increment: **SET IDENTITY_INSERT Printer_Inc OFF** .

Finally, consider the example of inserting data from the *Product* table into the *Product_Inc* table, saving the values in the *code* field:

```
SET IDENTITY_INSERT Printer_Inc ON;
INSERT INTO Printer_Inc(code, model,color,type,price)
SELECT * FROM Printer;
```

Regarding auto-incrementing columns, the following should also be said. Let the last value in the *code* field be 16, after which the line with this value has been deleted. What will be the value in this column after inserting a new row? Right, 17 because the last value of the counter is preserved, despite the deletion of the line containing it. Therefore, the numbering of values as a result of deleting and adding rows will not be sequential. This is another reason for inserting a row with a given (missing) value in an auto-incrementing column.

UPDATE statement

The **UPDATE statement** modifies the data in the table. The command has the following syntax

```
UPDATE
SET {column name = {expression to calculate the column value
| Null
| DEFAULT}, ...}
[{WHERE}];
```

Using one operator, values can be set for any number of columns. However, in the same **UPDATE statement,** you can make changes to each column of the specified table only once. If there is no **WHERE clause,** all rows in the table will be updated.

If a column allows a NULL value, then it can be specified explicitly. In addition, you can replace the existing value with the default value (**DEFAULT**) for this column.

A reference to an “expression” may refer to the current values in the table being edited. For example, we can reduce all prices of PC notebooks by 10 percent using the following operator:

```
UPDATE Laptop SET price=price*0.9
```

It is also allowed to assign values of some columns to other columns. Suppose, for example, you want to replace hard drives of less than 10 GB in PC notebooks. In this case, the capacity of new disks should be half the amount of RAM available in these devices. This problem can be solved as follows:

```
UPDATE Laptop SET hd=ram/2 WHERE hd<10
```

Naturally, the data types of the *hd* and *ram* columns must be compatible. **CAST** can be used to cast types.

If you want to change the data depending on the contents of a certain column, you can use the **CASE** expression. If, say, you need to put 20 GB hard drives on PC notebooks with memory less than 128 MB and 40 gigabyte ones on other PC notebooks, then you can write this request:

```
UPDATE Laptop
SET hd = CASE WHEN ram<128 THEN 20 ELSE 40 END
```

You can also use subqueries to calculate column values. For example, you need to equip all PC notebooks with the fastest processors available. Then you can write:

```
UPDATE Laptop
SET speed = (SELECT MAX(speed) FROM Laptop)
```

A few words need to be said about auto-incrementing columns. If the *code* column in the *Laptop* table is defined as **IDENTITY (1,1)** , then the following statement

```
UPDATE Laptop SET code=5 WHERE code=4
```

will not be executed because the auto-incrementing field does not allow updates, and we will receive the corresponding error message. To accomplish this task nevertheless, one can proceed as follows. First insert the desired row using **SET IDENTITY_INSERT** , then delete the old row:

```
SET IDENTITY_INSERT Laptop ON
INSERT INTO Laptop_ID(code, model, speed, ram, hd, price, screen)
SELECT 5, model, speed, ram, hd, price, screen
FROM Laptop_ID WHERE code=4
DELETE FROM Laptop_ID WHERE code=4
```

Of course, there should not be another row with *code* = 5 in the table.

In Transact-SQL, the **UPDATE** statement ng> extends the standard by using the optional FROM clause. This proposal specifies a table that provides the criteria for the update operation. Additional flexibility is provided by the use of table join operations.

An example . Let it be required to indicate “No PC” in the *type* column for those PC models from the *Product* table for which there are no corresponding rows in the *PC* table. A solution by joining tables can be written as follows:

```
UPDATE Product
SET type='No PC'
FROM Product pr LEFT JOIN PC ON pr.model=pc.model
WHERE type='pc' AND pc.model IS NULL
```

An external join is used here, as a result of which the *pc.model* column for PC models that are not in the *PC* table will contain a NULL value, which is used to identify the rows to be updated. Naturally, this problem has a solution in the "standard" version:

```
UPDATE Product
SET type='No PC'
WHERE type='pc' and model NOT IN (SELECT model FROM PC)
```

DELETE statement

The **DELETE statement** deletes rows from temporary or permanent base tables, views, or cursors, and in the last two cases, the operator's action extends

to those base tables from which data was extracted to these views or cursors. The delete operator has a simple syntax:

DELETE FROM [WHERE];

If the **WHERE** clause is missing, all rows from the table or view are deleted (the view must be updatable). You can also perform this operation (deleting all rows from a table) in Transact-SQL more quickly using the command

TRUNCATE TABLE

However, there are a number of differences in the implementation of the **TRUNCATE TABLE** command compared to using the **DELETE statement** , which should be kept in mind:

1. It is not journalized to delete individual rows of a table. Only the release of pages that were busy with table data is written to the log.

2. Do not work out triggers. As a result, this command is not applicable if there is a foreign key reference to this table.

3. The counter value (IDENTITY) is reset to the initial value.

An example. You want to remove all notebooks with a screen size less than 12 inches from the *Laptop* table.

```
DELETE FROM Laptop
WHERE screen<12
```

All notebooks can be deleted using the operator.

```
DELETE FROM Laptop
```

or

```
TRUNCATE TABLE Laptop
```

Transact-SQL extends the syntax of a DELETE statement by introducing an optional **FROM clause**

FROM

Using a *table type source,* you can specify the data that is deleted from the table in the first FROM clause.

Using this clause, you can join tables, which logically replaces the use of subqueries in the WHERE clause to identify deleted rows.

Let us explain what was said by an example. Suppose you want to remove those PC models from the *Product* table for which there are no corresponding rows in the *PC* table.

Using standard syntax, this problem can be solved by the following query:

```
DELETE FROM Product
WHERE type='pc' AND model NOT IN (SELECT model FROM PC)
```

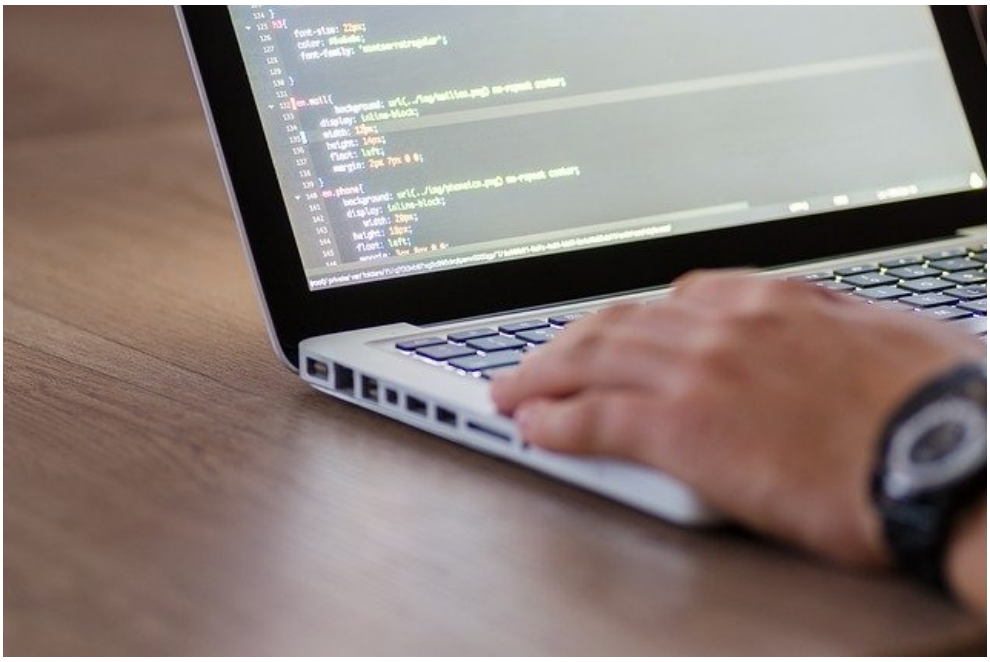
Note that the predicate *type* = 'pc' is needed here so that printers and notebook PC models are not deleted either.

The same problem can be solved using the additional FROM clause as follows:

```
DELETE FROM Product
FROM Product pr LEFT JOIN PC ON pr.model=pc.model
WHERE type='pc' AND pc.model IS NULL
```

An external join is used here, as a result of which the *pc.model* column for PC models that are not in the *PC* table will contain a NULL value, which is used to identify the rows to be deleted.

Chapter 8: Some More Learning Phases to Work With



We need to take some time now to look at some of more of the things that we are going to be able to do when it is time to handle the SQL that we want to work with. We need to take a look at the PL and SQL and how these are able to work with each other, and even how they are supposed to be different in some cases as well.

PL / SQL - “Procedural Language extensions to the Structured Query Language”, which translates as “Procedural Language Extensions for SQL”. Almost every enterprise-level DBMS has a programming language designed to expand the capabilities of SQL:

PL / SQL - in Oracle Database Server
Transact-SQL - in Microsoft SQL Server;
SQL PL - in IBM DB2;
PL / pgSQL - in PostgreSQL.

In these languages, programs are created that are stored directly in the databases and are executed by the DBMS, therefore they are called stored procedure languages. The languages of stored procedures have similar syntax and

semantics, therefore, after mastering the PL / SQL language, it will later become quite easy to switch, for example, to Transact-SQL or PL / pgSQL.

Function in Oracle PL / SQL
Function in PL / pgSQL PostgreSQL

```
CREATE FUNCTION F1 RETURN INT AS
BEGIN
FOR r IN (SELECT * FROM tab1) LOOP
UPDATE tab2 SET at3 = r.at2;
END LOOP;
RETURN 1;
END
CREATE FUNCTION F1 () RETURNS int AS '
Decare
r RECORD;
BEGIN
FOR r IN SELECT * FROM tab1 LOOP
UPDATE tab2 SET at3 = r.at2;
END LOOP;
RETURN 1;
END
'LANGUAGE plpgsql;
```

Tasks solved by PL / SQL
PL / SQL, unlike Java, Python or C ++, is not used to develop mathematical applications, games, etc. It is a specific third generation programming language designed to work with Oracle databases directly in the core of the Oracle server. In fact, PL / SQL programs are wrappers around SQL statements.

PL / SQL is used to solve the following tasks:

* implementation of server-side business logic in the form of stored programs;

* automation of Oracle database administration tasks;

* web application development;

* development of client applications in the Oracle Developer environment.

We will not dwell on automating database administration tasks and developing

We will not dwell on automating database administration tasks and developing client applications, but focus on the main direction of using PL / SQL - implementing server-side business logic in the form of stored programs.

Use Case for PL / SQL Programs

Let there be an Oracle database in the corporate network on a Linux server with information about the organization's clients. Connect to the Oracle server from the laptop over the network using the SQL * Plus utility. Starting PL / SQL calc_clients_debt from SQL * Plus to calculate customer debt may look something like this:

```
SQL> BEGIN
2 calc_clients_debt (p_account_from => 100001, p_account_to => 200000);
3 END;
four /

PL / SQL procedure successfully completed.
```

Only four lines to start the calc_clients_debt procedure will be transferred from the laptop to the Linux server, where the Oracle database server, having received these lines, will execute the PL / SQL procedure. Only information about the successful completion of the procedure will be returned to the laptop - one line. The gigabytes of financial data required for the calculations for a given range of 100,000 personal accounts will not be transferred to the laptop via the network - all client data will be sampled using SQL from PL / SQL and all calculations will be performed in PL / SQL by the Oracle database engine on a powerful Linux server. On the same server, in the same Oracle database, calc_clients_debt will also save the calculation results.

This is how a debt calculation might look if it were run by a technical specialist who knows the database device and prefers to work with it in SQL * Plus. It is clear that accounting or client department employees do not work with the database in SQL * Plus. For them, a client program in C #, Java or another programming language with screen forms and reports should be developed and installed. In this program, on the screen, the user sets the range of processed personal accounts and clicks the button "Calculate debt".

The client program, through the appropriate programming interfaces that are found in most modern programming languages, launches the calc_clients_debt stored procedure in Oracle and starts showing the user an hourglass or a running

bar (progress bar) to the user. The program itself does not carry out data processing, which at this time is on a remote Linux server. As soon as the stored procedure successfully completes and the Oracle server reports this to the client program, it will give the user the message "Debt has been successfully calculated."

This is a typical PL / SQL usage scenario: implementing business logic (in this example, calculating customer debt) as a PL / SQL procedure stored in a database and running it from a client program that connects to an Oracle server via a network. Typically, PL / SQL programs run “under the hood” and are not visible from the outside.

Advantages and disadvantages of stored programs

When implementing business logic, it is quite possible to do without using stored programs. So, the problem of calculating customer debt can be solved in two ways:

to develop one or several (frontend, backend) applications in Java, JavaScript, C ++, Python, etc., that implement only the user interface, and implement the business logic of the debt calculation itself in the form of a stored program that the applications call when the calculation process starts;

* to develop one or several (frontend, backend) applications in Java, JavaScript, C ++, Python, etc. that implement both the user interface and the business logic of debt calculation.

For the second method, the database is used only for storing data. All the necessary data for each client is retrieved by the application from the database, calculated by the application and the received debt information is saved back to the database. An application that reviews data is often placed on the same server as the database — so that the network does not become a system bottleneck.

The choice of the method used to solve the problem is the responsibility of the system architect, and many factors should be taken into account that are formed in each case based on the known advantages and disadvantages of using stored programs.

Advantages of stored programs:

* portability of stored programs with the database;

* increased processing performance due to the lack of data transfer outside the database server;

database server;

* close integration with the SQL execution subsystem (SQL statements in stored programs are executed without the use of additional interfaces and drivers);

* control of access to data based on stored programs (access is not granted to database tables for reading and writing data to them, but to executing stored programs - thereby isolating data from application programs);

* implementation of dynamic integrity constraints and the concept of active databases using the trigger mechanism.

Disadvantages of stored programs:
* "Smearing" the logic of the system according to several programs written in different languages;

* the need, along with Java, Python, C ++ programmers, to have a database programmer in the team;

* the scarcity of the expressive capabilities of the languages of stored procedures against the background of modern languages Java, Python, C ++;

* intolerance of stored programs between different DBMSs;

* possible problems with scaling.

The most significant drawback of stored programs is their binding to a specific DBMS. For example, when switching from Oracle to PostgreSQL within the framework of the current topic of import substitution, all stored programs will have to be rewritten from PL / SQL to PL / pgSQL, and this will lead to significant costs for reengineering PL / SQL code, which can amount to hundreds of thousands of lines.

As for the problems of scaling, the processing of data directly in the database using the DBMS itself is the advantage of stored programs until the required level of performance is provided. Otherwise, this circumstance will interfere with scaling, since the installation of additional servers will require a large amount of work. We will have to install a DBMS on each new server, create our own database with stored programs and solve the problem of distributing data across several databases. Of the virtues of stored programs, integrating data storage and processing can thus be a drawback. With separate applications that

implement server-side business logic without stored programs, there is usually no scaling problem - adding new servers only for computing applications is usually quite easy.

Portability of PL / SQL Programs
Portability of PL / SQL programs along with the database is provided by the PL / SQL language architecture, similar to the Java language architecture.

When programming in C / C ++, Pascal, as a result of the compiler, an executable file is obtained. This file contains machine instructions for a specific hardware platform and is designed to work on a specific operating system. Therefore, if you copy the executable file for Windows to a computer with the Linux operating system, then it will not start there. If the executable file for one hardware platform is transferred to another platform (a computer with other machine instructions), then it will not start there either. As a result, if you need to provide cross-platform, then for the same C ++ program, you have to have a version for Windows and a version for Linux, a version for x86 (32-bit) and a version for x64 (64-bit) and so on.

The situation is different with Java programming. As a result of the Java compiler, the result is not an executable file with machine instructions, but a bytecode file - a low-level machine-independent code executed by a bytecode interpreter. This bytecode interpreter is called the Java Virtual Machine (JVM). When the Java program starts, a file with its bytecode is fed to the input of the Java virtual machine, which converts the bytecode instructions into machine codes of a specific platform. Thus, in order to run a program in Java in a particular environment, it is enough to have JVM for this environment. For example, in the core of the Oracle server there is a Aurora JVM virtual machine designed to run Java programs stored in Oracle databases.

As a result of the PL / SQL compiler, the result is not an executable file, but a bytecode called p-code. This PL byte code is interpreted by the PL / SQL virtual machine (PL / SQL Vitual Machine, PVM) located in the Oracle database engine. There is a PL / SQL virtual machine in all versions of the Oracle DBMS for any operating system and for any hardware platform; therefore, PL / SQL programs remain operational when transferring Oracle databases from one computing system to another.

Evaluating PL / SQL
As a programming language, PL / SQL has the following advantages:

* static typing;

* availability of error handling tools and user exceptions;

* the presence of concise and convenient language constructs for executing sentences of the SQL language.

It is believed that efficient high-performance code for working with an Oracle database is easier to write in PL / SQL than in any other procedural programming language. In particular, in PL / SQL there are special means of bulk data processing (bulk processing), allowing to increase productivity by an order or more.

Here is a fairly large quote from the book “Oracle for Professionals” written by Tom Kyte, vice president of Oracle Corporation [18, p. 48]:

“When developing database software, I adhere to a fairly simple philosophy that has remained unchanged for many years:

Everything that is possible should be done in **one** SQL statement. Believe it or not, but it is almost always possible. Over time, this statement becomes even more true. SQL is a language with a lot of power.

If something cannot be done in one SQL statement, then it must be implemented in PL / SQL using as short code as possible. Follow the principle of “more code = more errors, less code = less errors”.

If the problem cannot be solved using PL / SQL, we may find that working with the procedure stored as Java is the best to work with. However, after the release of Oracle 9i and subsequent versions, the need for this is very rare. PL / SQL is a full-fledged and popular third-generation programming language.

If the problem cannot be solved in Java, try writing an external procedure C. This is the approach most often used when you need to ensure high speed of the application, or use an API from independent developers implemented in C.

If you cannot solve the problem using the external procedure C, seriously think about whether there is a need for it.

... We will work with PL / SQL and its types of objects to help us work with our problems that are impossible or inefficient in SQL. PL / SQL is actually

something that has been around for a number of years - it took more than 27 years to develop it (by 2015); in fact, back to Oracle 10g, the PL / SQL compiler on its own was rewritten a few times in order to make sure that it is as optimized for the work that it needs as possible.

No other language is so closely related to SQL and is not so optimized for interacting with SQL. Working with SQL in this kind of format is actually going to feel pretty natural, while it would be something that is hard in other kinds of languages that you would want to work with, and can be quite burdensome. ”

The New Procedural Option - PL / SQL appeared in Oracle 6.0 in 1988. Since then, PL / SQL has written millions of lines of server-side business logic code and developed thousands of client forms and reports in the Oracle Developer environment. For many years, Oracle Corporation has demonstrated its commitment to PL / SQL, and with the release of each new version of Oracle Database Server, PL / SQL introduces new enhancements. PL / SQL is an integral part of Oracle technologies and the corporation plans to develop and support it in the future.

First PL / SQL program

According to a long tradition dating back to C, textbooks on programming languages begin with the program “Hello, World!” And a description of the data types of the language being studied. We will not break this tradition, but make one change to it. Since the PL / SQL language is designed to work with Oracle databases, we will not set the string “Hello, World!” Statically in the source code, but take it from the database table.

```
CREATE TABLE hello_world_table (message VARCHAR2 (30));
INSERT INTO hello_world_table VALUES ('Hello, World!');
```

To take this a bit further, we are able to work with the following code, making sure that we do it in SQL Plus, to see what is going to happen

```
SQL> SET SERVEROUTPUT ON
SQL> DECLARE
2 l_message VARCHAR2 (30);
3 BEGIN
4 SELECT message INTO l_message FROM hello_world_table;
5 DBMS_OUTPUT.PUT_LINE (l_message);
6 END;
```

6 END;
7 /
Hello World!

PL / SQL procedure successfully completed.

In the PL / SQL program presented above, using the SELECT INTO command, the value of the message column of the hello_world_table table row is read from the database and assigned to the local variable l_message, the value of which is then displayed in the SQL * Plus window. The l_message variable is previously declared in the declarations section after the DECLARE keyword.

Screen output in PL / SQL is performed by the PUT_LINE procedure of the built-in DBMS_OUTPUT package, which is available in all Oracle databases. We can assume that the DBMS_OUTPUT.PUT_LINE procedure in PL / SQL is an analogue of the printf procedure in C.

Recall two things that are important when working with the SQL * Plus utility:

To run the PL / SQL program in SQL * Plus for execution, you need to type the / character on a new line and press the Enter key on the keyboard;

in SQL * Plus, the screen output of PL / SQL programs is enabled with the SET SERVEROUTPUT ON command (by default, screen output is turned off).

If you do not execute the SET SERVEROUTPUT ON command, then nothing will be printed in the SQL * Plus console. In the popular Quest SQL Navigator GUI client, the PL / SQL screen output is also turned off by default and is turned on by the special button “Turn the server output”, which after pressing should remain in the “pressed” position.

PL / SQL Data Types

Recall that a data type (data tour) is a named set of data values of a given structure that satisfy the specified integrity constraints and allow the execution of a certain set of operations associated with this set. For example, numbers and dates can be added and subtracted, but strings and logical values cannot.

PL / SQL refers to languages with static typing. Static typing is called data type checking during program compilation. Static typed programming languages include Pascal, Java, C / C ++ / C #. Dynamic typing languages (JavaScript,

Python, Ruby) perform most type checks at run time. Static typing can detect errors during compilation, which increases the reliability of programs.

Types of Data Types
Since PL / SQL is a procedural extension of SQL, PL / SQL has all the data types available in the Oracle SQL dialect with some minor differences. In addition to them, PL / SQL also has data types that are not available in Oracle SQL.

PL / SQL has scalar and composite data types:
* data of scalar types consist of one indivisible (atomic) value (logical values, numbers, dates, strings);

* composite type data consists of several values (records and collections).

Scalar PL / SQL Data Types
PL / SQL data type declarations are in the STANDARD package, located in the SYS user schema:

```
package STANDARD AUTHID CURRENT_USER is
type BOOLEAN is (FALSE, TRUE);
type DATE is DATE_BASE;
type NUMBER is NUMBER_BASE;
subtype FLOAT is NUMBER; - NUMBER (126)
subtype REAL is FLOAT; - FLOAT (63)
subtype INTEGER is NUMBER (38.0);
subtype INT is INTEGER;
subtype DEC is DECIMAL;
...
subtype BINARY_INTEGER is INTEGER range '-2147483647' .. 2147483647;
subtype NATURAL is BINARY_INTEGER range 0..2147483647;
...
type VARCHAR2 is NEW CHAR_BASE;
subtype VARCHAR is VARCHAR2;
subtype STRING is VARCHAR2;
...
```

It can be seen that the PL / SQL data types declared in the STANDARD package either correspond to Oracle SQL data types (_BASE types) or are entered as their subtypes.

Note the presence of the BOOLEAN data type, which is not available in Oracle SQL. Values of type BOOLEAN can, for example, be used in code of the following form:

```
l_amount_negative_flag BOOLEAN: = amount <0;
IF l_amount_negative_flag THEN ... END IF;
```

A significant difference between the PL / SQL and Oracle SQL data types is *the* greater maximum length of the values of the CHAR and VARCHAR2 types, designed to represent fixed and variable length strings:

* for VARCHAR2 in PL / SQL, the maximum length of values is in the range from 1 to 32,767 bytes (in Oracle SQL up to version Oracle 12c, the maximum length of VARCHAR2 was up to 4,000 bytes, in Oracle 12c it was also increased to 32,767 bytes);

* for CHAR in PL / SQL, the maximum length of values is in the range from 1 to 32,767 bytes (in Oracle SQL up to version Oracle 12c, the maximum length of CHAR was up to 2,000 bytes, in Oracle 12c it was also increased to 32,767 bytes).

PL / SQL Records

PL / SQL records are composite data types and are defined as sets of attributes associated with specific relationships. Record attributes can be either scalar data types or other composite types — other records and collections.

A PL / SQL record is declared as a user data type using the RECORD keyword, in general, working with PL / SQL records is similar to working with Pascal records or structures in C:

```
Decare
TYPE t_person IS RECORD
(name VARCHAR2 (100),
secname VARCHAR2 (100),
surname VARCHAR2 (100),
born DATE);
l_person t_person;
BEGIN
l_person.surname: = 'Ilyin';
l_person.name: = 'Victor';
l_person.secname: = 'Semenovich';
```

```
l_person.secname: = 'Semenovich';
l_person.born: = TO_DATE ('07 .08.1969 ',' dd.mm.yyyy ');
print (l_person);
END
```

Purpose of PL / SQL records:

* reading in the PL / SQL records the rows of the resulting SQL query samples (when declaring PL / SQL records based on tables and cursors using the% ROWTYPE attribute);

* combining several parameters of procedures and functions into one structure (instead of a large number of parameters of scalar types it is more convenient to transfer one parameter of a composite type to procedures and functions).

Compactness and extensibility of the source code is the main advantage of using PL / SQL records. Compare the two options for calling the procedure for printing information about a person - with one PL / SQL record parameter and with several parameters of scalar data types:

print (l_person) and print (l_name, l_secname, l_surname, l_born)

The first call option looks more compact. In addition, if it becomes necessary to process new information about a person, for example, TIN and SNILS, then for the second option, you will need to add two new parameters to all calls to the print procedure throughout the code. If you pass the description of the person in the form of a PL / SQL record, you only need to add new attributes to the declaration of type t_person. It is not necessary to make changes to the header of the print function and to its calls by the source code. By using PL / SQL records, this extends the extensibility of the source code.
Here are the basic rules for working with PL / SQL records:

* the definition of record attributes can specify NOT NULL constraints and set attribute values by default;

* assigning a record NULL assigns NULL to all its attributes;

* to compare two entries for equality or inequality, you need to consistently compare the values of all attributes in pairs.

Since PL / SQL records are similar to table rows, the advantages of using them

for compactness and extensibility of the code are especially pronounced when executing SQL statements in PL / SQL. One row of the table - one PL / SQL record. Table rows “live” in the database, PL / SQL records “live” in programs. A row of a table can be considered as a single command in a PL / SQL record, a PL / SQL record can be inserted as a row in a single command, that is, data can be moved in both directions. PL / SQL also has special language constructs that allow you to move between the database and the PL / SQL program not individual PL / SQL records and table rows, but many of them. And all this is done very compactly - with one or two lines of PL / SQL code.

Bind Variable Declarations

Since the PL / SQL language is designed to process the data that is in the tables of the Oracle database, it provides the ability to declare variables with reference to the schemes of these tables. For example, if some variable is used to read the values of the surname column of the person table into it, then it would be logical to indicate when declaring this variable a data type that matches the data type of the column.

There are two types of variable binding:
* scalar binding (using the% TYPE attribute, the variable is declared with the data type of the specified table column);

* binding to a record (using the% ROWTYPE attribute, a PL / SQL record variable is declared with attributes by the number of columns of the specified table or cursor).

Consider an example. Let the database have a table tab1 with at1 columns of type DATE and at2 of type VARCHAR2 (20). Then in the PL / SQL code, you can declare variables as follows:

```
l_tab1 tab1% ROWTYPE;
l_at1 tab1.at1% TYPE;
```

The variable l_tab1 will be a PL / SQL record with two attributes at1, at2, the data types of which will be the same as the data types of the columns at1, at2 of the table tab1, that is, DATE and VARCHAR2 (20), respectively. The l_at1 variable will have a data type that is the same as the at1 column, that is, date.

Benefits of declaring bind variables:

* synchronization with table schemas is automatically performed;

* compact extensible code for reading rows of result sets of SQL queries without listing columns.

Automatic synchronization of variable declarations in PL / SQL programs and database table schemas makes PL / SQL programs resistant to possible future changes, such as adding, deleting or renaming table columns, changing their data types. In practice, such table schema changes occur quite often.

We give a concrete example. There was a clients table in the CRM system database, in which there was an inn column. At the time of the development of the system, clients could only be legal entities with a TIN of 10 characters. Over time, the company began to serve individuals who have a TIN of 12 characters. The database administrator changed the data type of the inn column of the clients table from VARCHAR2 (10) to VARCHAR2 (12) and rows with long TINs began to appear in the table. Since in the PL / SQL code all the variables for working with the TIN were declared as VARCHAR2 (10), errors occurred when reading individuals from the TIN database in PL / SQL programs. If the variables for the TIN were declared in due time with reference to the inn column using the% TYPE attribute, then they would automatically "expand" themselves and there would be no errors at the execution stage.

Without listing the columns of the resulting SQL query samples, a very compact code is written like:

```
l_person person% ROWTYPE;
SELECT * INTO l_person FROM person WHERE id = 13243297;
print (l_person);
```

The SQL query selects all the columns of the person table, and the variable l_person declared with% ROWTYPE will have exactly the same attributes as the columns of the person table, with the same names and data types, in the same order. The values of all columns of the table row being read are assigned to the corresponding attributes of the PL / SQL record. If a new column appears in the person table in the future, it will automatically be picked up by both the SQL query (SELECT *) and the declaration of the l_person variable in the PL / SQL program. No changes to the code will be required; the program will automatically recompile the first time it is accessed.

Variables can be declared as PL / SQL records using the% ROWTYPE attribute not only on the basis of a single table, but also on the basis of columns of the

resulting samples of arbitrary SQL queries. For this, PL / SQL records are declared based on explicit cursors.

Chapter 9: PL / SQL Program Structure

Now that we have had some time to take a closer look at the PL and SQL kind of structure from the last chapter, it is time to dive into this a bit more and see what else we are able to do with it. In particular, this chapter is going to explore some of the structure that comes with the PL and SQL program structure. Let's dive right in and see what it is all about.

Block structure

In PL / SQL, as in most procedural programming languages, the smallest unit of source grouping is block. It is a piece of code that defines the boundaries of code execution and the scope for ads. Blocks can be nested into each other.

PL / SQL Block Sections

A PL / SQL block consists of four sections:
* heading section;
* announcements section;
* executable section;
* exception handling section.

The heading sections, declarations, and exception handling may not be in the block; only the executable section is mandatory.

The syntax of a PL / SQL block is as follows:
heading section
Decare

ads section
BEGIN
executable section
EXCEPTION
exception handling section
END

The heading section indicates:
block type (procedure, function);
block name (name of procedure, function);
parameter names, their data types and value transfer modes.

The declaration section declares custom data types, variables, and constants, which are then used in the executable section and the exception handling section. The executable section implements the actual program logic. In a degenerate case, there can be only one “empty” NULL command.

The BEGIN and END keywords in PL / SQL are operator brackets similar to the {and} characters in other programming languages and mark the beginning of an executable section and the end of a block. Each command in PL / SQL must end with a semicolon (symbol;).

Types of PL / SQL Blocks
There are two kinds of blocks in PL / SQL:
* named blocks (with header section);
* anonymous blocks (no header section).

Named blocks, in turn, also come in two forms:
* named blocks of programs stored in the database (procedures, functions, packages, and triggers);

* named blocks in the declaration sections of other blocks (anonymous or named).

Stored programs are objects of the Oracle database and are created by the CREATE DDL command, after which the named PL / SQL block is written. The block name indicated in the header section will be the name of the database object.

Anonymous blocks do not have a header section. If a block does not have a

header section, then it does not have a name that is indicated in this section; therefore, such blocks are called anonymous.

Anonymous blocks are either embedded in other blocks or stored as text script files. In the latter case, anonymous blocks are usually used to call stored programs or to automate database administration tasks.

Anonymous block script file1.sql
- Anonymous nested blocks in a named block of a stored program

```
Decare
i INTEGER;
- named block of proc1 procedure
- in the anonymous block section
PROCEDURE proc1 IS
BEGIN
NULL
END
BEGIN
- proc1 procedure call
proc1;
END
- named block of proc2 procedure
```

Chapter 10: Other Things We Can Work On In SQL

After we go through and do some of the initial loading of some of the spatial columns that are there, it is time for us to go through and make a request, and then analyze them. MySQL provides a set of functions to perform a lot of the operations that we need when it comes to this kind of analysis. We are able to take these functions and then group them into four new categories based on what they are able to perform for us. These are going to include the following:

1. The first type is going to be the type of function that is able to convert a configuration between more than one type of format.
2. Then we have the type that is going to help provide us with the access that we need in order to work with quantitative or qualitative details that come with geometry.
3. The next type is going to be any of the functions that are going to help us get a description of a relationship between the configurations that

you have.
4. Then the last kind is going to be the functions that are going to be able to create a new configuration from one that is already existing.

All of these are going to be important to some of the things that we are going to be able to do when it comes to spatial analysis. This kind of analysis can be used in a variety of contexts including:

1. Any kind of program in SQL that is interactive, including the SQL query browser.
2. Applications that are done in any other kind of coding language that you would like that are also able to support the client of MySQL API Geometry Conversion Functions.

Geometry Format Conversion Functions

MySQL supports the following functions for converting geometry values between the internal format and the WKT or WKB format:

AsBinary (g)

Converts an internal geometry value to a WKB representation and returns a binary result. SELECT AsBinary (g) FROM geom;

Astext (g)

Converts a value in the internal geometry format to the WKT representation and returns a string result.mysql> SET @g = 'LineString (1 1,2 2,3 3)';

mysql> SELECT AsText (GeomFromText (@g));

+ ------------------------ +

| AsText (GeomFromText (@g)) |

+ ------------------------ +

| LINESTRING (1 1,2 2,3 3) |

+ ------------------------ +

GeomFromText (wkt [, srid])

Converts a string value from a WKT representation to an internal geometry format and returns the result. A number of type-specific functions are also provided, such as PointFromText () and LineFromText ().

GeomFromWKB (wkb [, srid])

Converts a binary value from a WKB representation to an internal geometry format and returns the result. A number of type-specific functions are also provided, such as PointFromWKB () and LineFromWKB ().

Geometry functions

Each of the functions that we are hoping to use in this and that will belong in a group will take the value of that geometry as a parameter, and then will be able to return some quantitative or qualitative property of the geometry that we work with. Some of the functions that you would want to work with along the way will end up limiting the types of parameters that you are able to work with. And these are going to provide us with a return known as NULL if the parameter does not provide us with the right type.

Property Limitations

While we are here, we need to spend some time taking a look at what are known as the property limitations when we work with the SQL language. This will ensure that we are going to be able to take advantage of what is available in some of our coding, and will make it easier for us to use with the database. Let’s take a look at how we can make this work for our needs.

Views Limitations

View processing is not optimized: Unable to create index on view.

Indexes can be used for processed views using a combining algorithm. However, the view, which is processed by the temptable algorithm, is not able to take advantage of indexes on the main tables (although indexes can be used during the generation of temporary tables).

Subqueries cannot be used in the FROM view clause. This restriction will be removed in the future.

There is a general principle that you cannot modify a table and select from the same table in a subquery.

The same principle also applies if you select from a view that selects from a table, if the selection of view from a table in a subquery and view are evaluated using a combining algorithm. Example:

```
CREATE VIEW v1 AS SELECT * FROM t2 WHERE EXISTS (SELECT 1
FROM t1 WHERE
t1.a = t2.a);
UPDATE t1, v2 SET t1.a = 1 WHERE t1.b = v2.b;
```

If we find that the view is something that we are going to evaluate using a able that is temporary, we are then able to select form the table in view with the help of a subquery and then change around that table with the help of a query that is external in the process.

If we end up doing this, then we are going to be able to create a temporary table to work with and then store our view in there for now. And we can really use this to make sure that we are not selecting from the table in the subquery and try to modify this table when we do not want to.

Along with this idea, we are able to force MySQL to do a few things here, including use the algorithm that we want. The right one to work with here is going to be the temptable one. This is done with the code of ALGORITHM = TEMPTABLE when we are in the definition of view. From there, we are going to choose to either drop or alter the table based on whether we would like to modify or delete. But keep in mind when we do this one, we are not going to get a warning at all.

The error that is going to show up when we try to do this will not appear until later when you actually use the view. The view definition is going to be frozen based on the instructions that are there. if the instructions were prepared with the command of PREPARE and it goes back and refers to the view, then the contents of this particular view are going to be the same any time that the execution happens.

This is going to be true even if you find that the definition of view is going to be changed once the statement is prepared, but this does have to be done before we execute that statement or it is not going to work. A good example of how we can do this is below:

```
CREATE VIEW v AS SELECT 1;
PREPARE s FROM 'SELECT * FROM v';
```

```
ALTER VIEW v AS SELECT 2;
EXECUTE s;
```

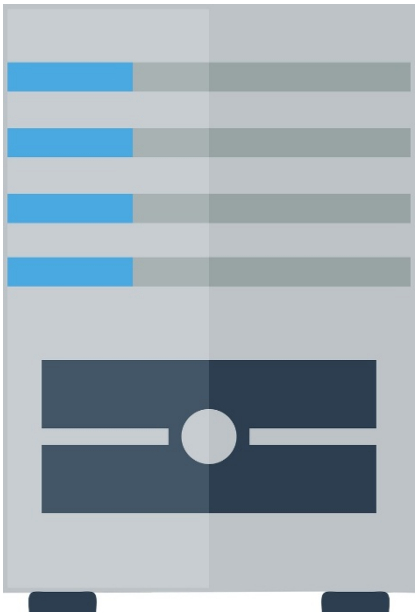
The result returned by the EXECUTE statement is 1, not 2.

If you see an instruction that is stored in what is a routine is able to access the view, then the content of the view is going to end up being exactly same as what we found the first time we executed the instructions. For example, we can see that this will mean that if the statement is going to be done with the help of a loop, the further iterations are going to be in the same view content, even if you see that you will need to change up the view of the definition later on when you are doing the loop. A good example of how this is going to work will be the following:

```
CREATE VIEW v AS SELECT 1;
delimiter //
CREATE PROCEDURE p ()
BEGIN
DECLARE i INT DEFAULT 0;
WHILE i <5 DO
SELECT * FROM v;
SET i = i + 1;
ALTER VIEW v AS SELECT 2;
END WHILE;
END
//
delimiter;
CALL p ();
```

Regarding updatable views: the overall goal for a view is that if any view is theoretically updatable, it should be updatable and practical. This includes views that have UNION in their definition. Currently, not all views that are theoretically updated are those in practice (may be modified). The initial implementation of the view was intentionally written in this way to become usable; updated views in MySQL will be made fast as we are able to. When we

have a lot of updated views on the database, it is possible for us to modify them and make changes, but we have to make sure that we know there are still some limitations that will exist on this.



If we are working with some of the subqueries that are present in our WHERE clause, then we know that we are working with ones that are updatable. But if we find that we are working with the subqueries that are found more in the SELECT list, this doesn't mean that we can't update those. Keep in mind here that we are not able to use the UPDATE command in order to work with more than one of these main tables at a time. We also can't go through and use the DELETE command in order to modify a view that has been previously defined as a union.

Along this same idea, if we have granted a user the basic privileges already in order to create a view in the database, then this means that the user is not going to have enough authority along he way in order to do the CREATE, SHOW, VIEW on that object unless you have given them that right along he way as well. This could lead to some problems when you try to copy the database. When you do not provide he user with the right authority to get it done, it is going to fail.

There is a workaround that we are able to use with this to get it to behave the way that we would like. For example, the administrator to manually grant these privileges if they would like to the users who have the authority to do CREATE VIEW.

In this though, we have to remember that there are going to be a few restrictions that we need to be careful about. The maximum tables that we are able to name

in a single join will only be 61. This is going to also come up with the number of tables that can be named when we work with the view of definition.

Chapter 11: Data Security - What Could Go Wrong?

When it comes to the databases that you are using, you want to make sure that you are working with a format that is going to be as safe and secure as possible. Most of the databases that are out there are going to contain valuable information for a big company. They will hold onto the data that a company needs in order to get ahead, to help them keep track of their customers and more.

For example, it is not that uncommon for a company to go through and decide to work with a database to hold onto information about customers, and what they purchased at their store. This means that the company would have a lot of valuable information, including information on the individuals name, their address, their credit card information, and more. For a hacker or someone who would like to steal these identities, that is some really valuable information.

This is why, if you plan to have your own database along the way, you need to make sure that you actually take care of the data that you are working with, rather than taking advantage and assuming that it is all going to be safe. This is a wealth of information in your database, and certainly a lot of other people outside of you would like to be able to get ahold of this information and use it for their own needs as well. If you are not careful with that information, then it is likely that they will, and this can cause you a loss of reputation and money, and

really harm your customers

Now, there are going to be a few different types of security attacks that a hacker is able to do against your system, and you need to make sure that you take care of these along the way, and do your best to keep the network safe and secure as well. We are going to spend some time in this guidebook looking at some of the basics that come with this process, and explore how you can work to keep the database as safe and secure as possible.

SQL injection (also known as "Violation of the integrity of the structure of the SQL-query") are one of the vulnerabilities that we need to really take a look at when it comes to the security of our database is the SQL injections. These are so dangerous because they will allow the hacker to open up a back door into your network and system, and allow unlimited access: for example, delete tables, modify the database, and even gain access to the internal corporate network. SQL injection is a purely software bug, and has nothing to do with the host provider. So, you were looking for safe JSP hosting, PHP hosting, or any other, you should know that only developers, not the host provider, are responsible for the prevention of SQL injections.

Why do SQL injections happen?

SQL injections are a very common problem, but ironically, they are also easy to prevent. SQL injections are so widespread, since there are so many places where the vulnerability can be present, and if the injection is successful, the hacker can get a good reward (for example, full access to the database data).

The risk of SQL injection occurs whenever a programmer creates a dynamic query to the database containing user input. This means that there are two ways to prevent SQL injection:

- Do not use dynamic database queries.
- Do not use user data in queries.

Everything seems to be simple, but these are theories; in practice, it is impossible to refuse dynamic queries, as well as to exclude user data input. But this does not mean that it is impossible to avoid injections. There are some tricks and technical features of programming languages that will help prevent SQL injections.

How to Prevent an SQL Injection?

Although the decision is highly dependent on the particular programming language, the general principles for preventing SQL injection are similar. Here are some examples of how this can be done:

1.) Don’t Bring Out the SQL Queries All the Time

There are times when the dynamic query is going to be necessary. But for most cases, you will find that we are able to replace these with some stored procedures, parameterized queries, and even prepared statements. This can actually clean up some of the code that we are working with and will make it look a little bit nicer overall.

For example, rather than working with the dynamic SQL like you may feel needs to be done, you can use the PreparedStatement() in Java with some bound parameters to get it done. When working with the .NET platform, you are able to work with some of the parameterized queries, like SqlCommand or the OleDB Command() and then add in some of your own bound parametesr to make it work.

Along with some of the prepared statements that we just talked about, it is also possible to work with stored procedures. Unlike the prepared procedures, these stored ones are going to be the type that you keep handy and stored up in the databases. But in both of these situations, the query from SQL is going to first help us to determine which parameters we should work with in it.

2.) Validation of entered data in queries

Validating data entry is less efficient than parameterized queries and stored procedures, but if it is not possible to use parameterized queries and stored procedures, then it is better to validate the entered data - this is better than nothing. The exact syntax for using input validation is highly dependent on the database; read the docks on your specific database.

3.) Do not rely on Magic Quotes

Enabling the magic_quotes_gpc parameter can help us to sometimes prevent injections of SQL> The Magic Quotes is not the last defense and should not be treated as such, especially since they could be turned off without you knowing, or do not have the ability to turn it on. That is why it is necessary to use code that will escape quotation marks. Here's a piece of code suggested:

• Regular and timely installation of patches. Even when your code does not have

vulnerabilities, there is a database server, server operating system, or developer utilities that may have vulnerabilities. That is why always install patches immediately after they appear, especially if it is a fix for SQL injections.

• Remove all functionality that you are not using.

The database server is a complex creation and has much more functionality than you require. And as far as security is concerned, here the principle “the more the better” does not work. For example, the extended system procedure xp_cmdshell in MS SQL gives access to the operating system, and this is just a dream for a hacker. That is why this function needs to be turned off, like any others, which make it easy to abuse the functionality.

4.) Using automated tools for finding SQL injections

Even if the developers followed all the above rules to avoid dynamic queries with the substitution of unverified user data, you still need to confirm this with tests and checks. There are automated testing tools for detecting SQL injections, and there is no excuse for those who do not use these tools to verify procedures and queries.

One of the simple tools (and one of the more or less reliable) for detecting SQL injection is an extension for Firefox called SQL Inject ME. After installing this extension, the tool is available by right-clicking in the context menu, or from the menu Tools → Options.

You can choose which test to run, and with what parameters. At the end of the test, you will see a report on the test results.

As you can see, there are many solutions (and above all, all simple ones) that you can take to clear the code of potential vulnerabilities to SQL injections. Do not neglect these simple things, as you endanger not only your security, but all the sites hosted on your host provider.

SQL Commands

SQL commands are divided into the following groups:

- Data Definition Language Commands - DDL (Data Definition Language). These **SQL** commands can be used to create, modify, and delete various database objects.
- Data Management Language Commands - DCL (Data Control Language). Using these **SQL** commands, you can control user access to the database and use specific data (tables, views, etc.).
- Commands of the transaction management language are TCL (Transaction Control Language). These **SQL** commands allow you to determine the outcome of a transaction.
- Data Manipulation Language Commands - DML (Data Manipulation Language). These **SQL** commands allow the user to move data to and from the database.

This guide, as you know now, is devoted to the study of Structured Query Language (SQL) and is complemented by a large number of examples that clearly demonstrate the capabilities of SQL. SQL is an ANSI standard, but there are a large number of versions of this query language. Whichever you choose, I wish you fun and success in your endeavor!

Conclusion

Thank you for making it through to the end of *SQL* , let’s hope it was informative and able to provide you with all of the tools you need to achieve your goals whatever they may be.

When you are ready to learn a little bit more about the SQL and how you would be able to benefit from learning how to make this language your own and what all you are able to do with it, then check out my advanced guide on SQL by Stephen Fletcher.

Finally, if you found this book useful in any way, a review on Amazon is always appreciated!