Talent Acceleration Program

Workbook

tap.kiitos.tech

Week 2

“Time is the scarcest resource; and unless it is managed, nothing else can be managed.”

Welcome to **week 2** of the Talent Acceleration Program!

On the *Technical Skills* side,

- The Frontend specialization will go deeper into the core concepts of Single Page Applications (SPA): component-based thinking, the rendering lifecycle and state are found in any of the popular SPA frameworks. As it’s the most popular at the moment, we’ll use React.js as the one to illustrate these concepts with!
- The Backend specialization will learn more about the key roles the backend fulfills: Application Programming Interfaces (API), the architectural pattern Representational State Transfer (REST) and lastly, database management (and its most important operations: CRUD).

On the *Soft Skills* side you will get familiar with another fundamental topic: time management. More importantly, how to take ownership of your time so that you can be more productive.

Good luck!

The TAP Team

Part 1: Technical Skills

Jump ahead to Frontend or Backend

Week	Topics
1	Introduction to Specialization
2	Core Concept Specialization
Frontend	State Management
Backend	Core API Features
3	Agile Fundamentals
4	Project Management Fundamentals

Frontend Track

Learning goals:

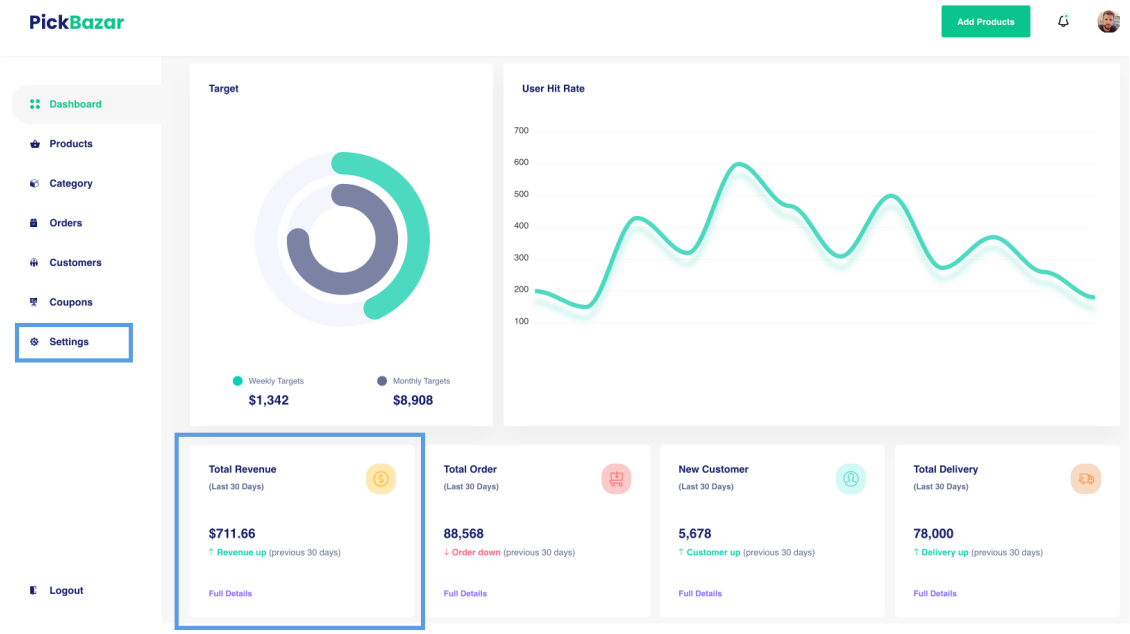
- ❏ Gain awareness of a core concept of modern frontend development: component-based thinking
- ❏ Understand the lifecycle of page/component rendering
- ❏ Learn about the distinction between local and global state
- ❏ Differentiate between the use cases for various state management tools

CORE CONCEPT: COMPONENT-BASED THINKING

One of the most important concepts in modern frontend development is the

idea of components.

Simply put, a component is *a reusable part of a user interface*. In the following example, you'll see 2 components highlighted that are reused in the page: the *menu item* and a *results count container*.



Knowing how to dissect any webpage in this component-based way of thinking is crucial to learning how to work with any type of frontend framework.

Learning Materials

- Component Hierarchy

CORE CONCEPT: COMPONENT LIFECYCLE

If you have no background with any frontend framework and start with React, you might've come across the concept of component lifecycle. The basic idea

is that the rendering process of every component goes through roughly 3 stages: mounting, updating and unmounting.

On the surface this seems like a radical idea: rendering a page in various stages. Usually when you deal with regular HTML you just load a page into the browser and it's done, right? Well, actually no. A regular HTML page actually also has a lifecycle!

Learning Materials

- Page: DOMContentLoaded

So the idea of a lifecycle isn't actually new. React just took the concept and reduced its scale from a complete page to a component of a page.

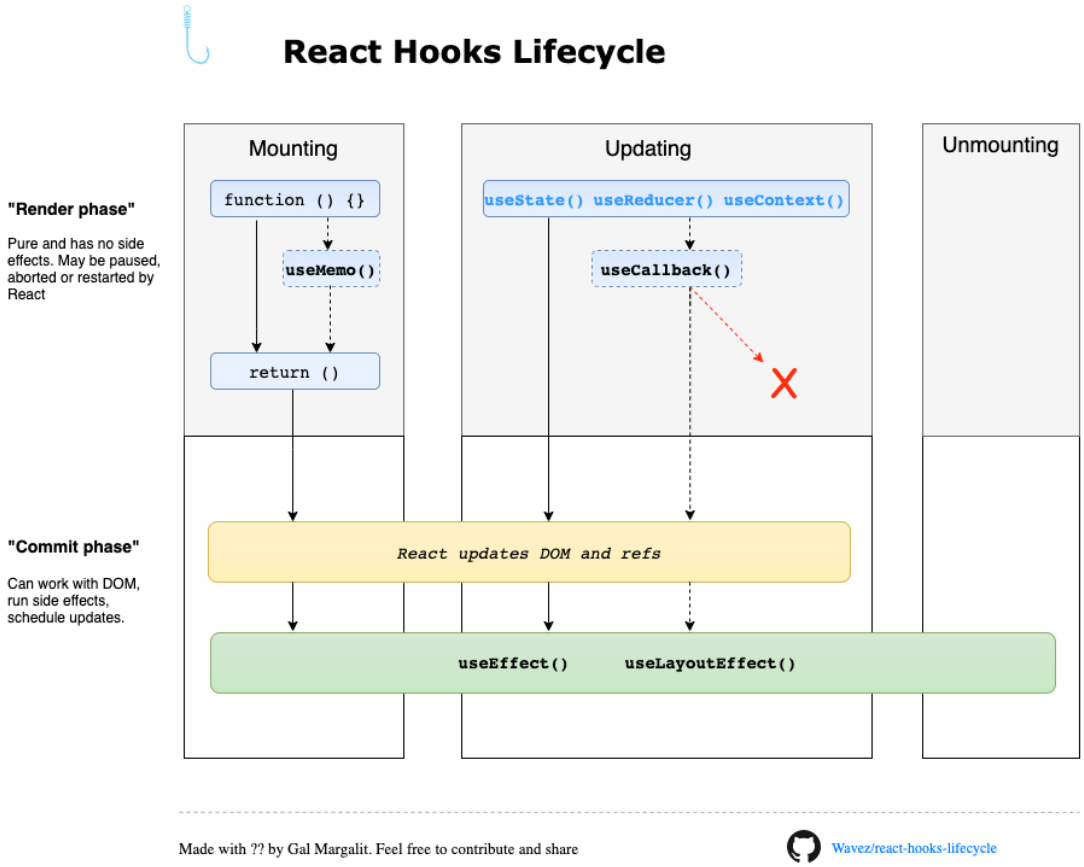
Learning Materials

- The React Lifecycle of a Functional Component

CORE CONCEPT: LIFECYCLE HOOKS

The concept of a React hook is misleading. Isn't it just a function that deals with state management in some way? Yes it is. However, the term "hook" is actually short for lifecycle hook. And what does it "hook into"? *It hooks into one of the stages in the rendering process of a component.*

In other words, it's a special function that should be executed in a certain stage (Mounting, Updating or Unmounting) in order to (1) render the right data at the right time or (2) destroy that data after it has served its purpose so that browser computing power can be freed for rendering something else.



The diagram above shows which specific hooks are best used in which stage. Effectively, you only need to use useState (to create and manage local state) and useEffect (to deal with side effects) and the others are just there for optimization!

Learning Materials

- 10 React Hooks Explained

- The useEffect Hook and its Lifecycle

CORE CONCEPT: STATE

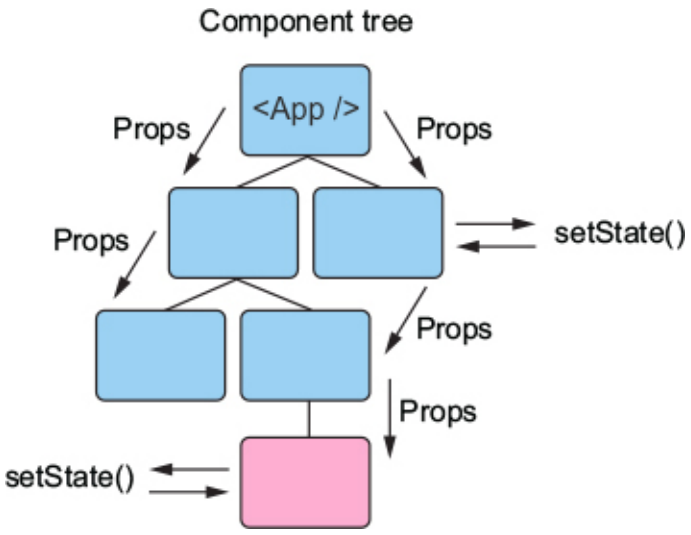
One of the most expensive and complex operations that happens in the frontend is updating the UI state. In this context the term “state” refers to all the (dynamic) data contained with the application.

Learning Materials

- What is "State" in Programming?
- The deepest reason why modern JavaScript frameworks exist

React is one solution that’s created in order to simplify this process. All you have to do is:

- Build up the page by creating custom components
- Nest components inside components, to create a tree-like structure
- Pass data in a unidirectional (from top to bottom) way



Learning Materials

- React State vs. Props

- Understanding unidirectional data flow in React

When it was first released, React didn't come with an efficient way to pass data far down the component tree. If you had some kind of shared state you would like to use anywhere else in your application, you had to pass it down from one level to another. In large scale React applications with 100's of components, you can imagine this is very hard to maintain.

This pattern of passing down data many levels downwards is also referred to as prop drilling. This only works in small scale applications with several components. However, as software developers we should also think about building our software in a scalable way: *would this solution still work if our application was 10X its size?*

Learning Materials

- What is Prop Drilling?

ADVANCED STATE MANAGEMENT

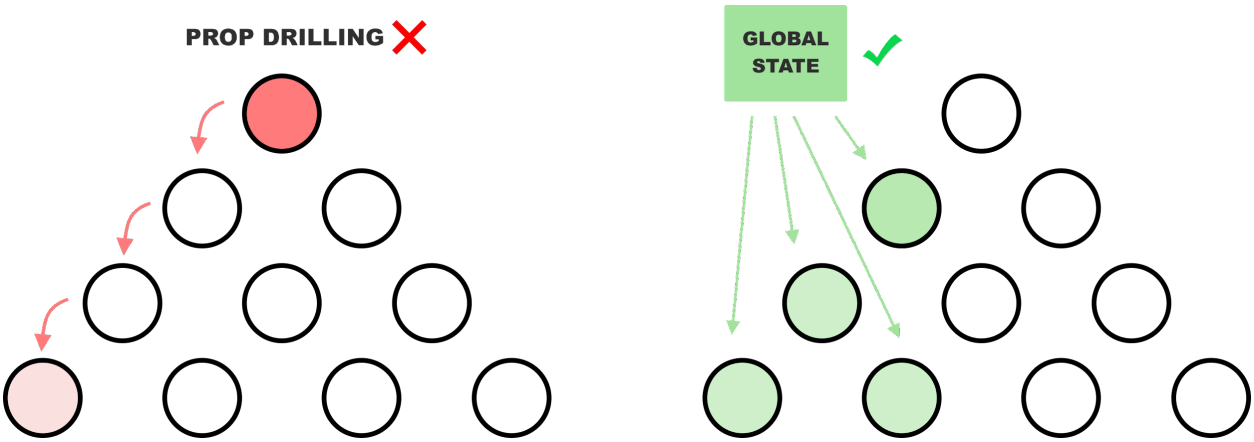
So far we've established that we can manage it *locally* by passing down data as props, custom attributes given to components, in order to access them within its structure.

```
<SomeComponent propName={propData} />
```

However, if we have a large-scale application this will not work. There will be

too many levels of components to pass through in order to get the state to the right place.

So this is where we have to start thinking about a concept called global state: the idea of moving all of our data into a globally accessible object and then connecting that object directly to the particular component that needs to grab data from it.



This is where state management tools come in: libraries that have been developed to solve the problem of prop drilling. Whichever way you go doesn't matter that much.

The various state management tools that exist ultimately all solve the same basic problem of prop drilling, by allowing us to (1) create a central place that holds all of the state (which we call global state, or store) and (2) provide a way to connect that state directly to the component.

Learning Materials

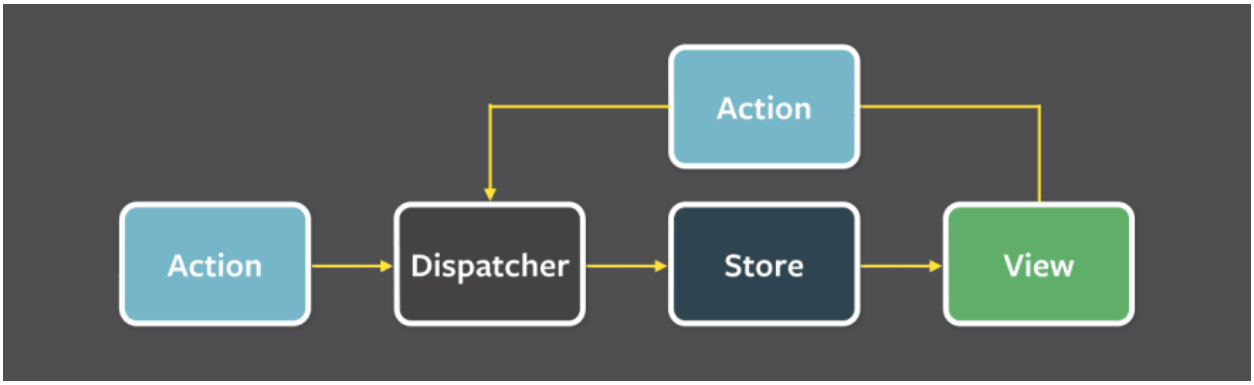
- How I Manage State in React

ARCHITECTURAL PATTERN: FLUX

This idea of "global state" came from the need to simplify state changes within a large-scale application. The dominant way to implement this comes from an architectural pattern known as Flux.

Flux consists of 4 parts:

1. **Action**. These are customly defined browser events (i.e. "Add items to shopping cart" or "Log in user") that contains data which will change the application's state.
2. **Dispatcher**. This is the function that will take the data from the action and puts it in the store
3. **Store** (also known as "global state"). The central place where we store all of our application's state).
4. **View**. The user interface, composed of the component hierarchy, will receive state in various places.



The diagram above shows the way data flows in a Flux architecture. As you may notice, it goes in 1 direction only (this is also known as unidirectional data flow). This means that a SPA must follow this very specific procedure, namely

creating functions and variables in certain places, in order to update its state. It's not allowed to update the state anywhere else.

Learning Materials

- Flux Architectural Pattern
- Flux Explained in 3 Minutes

STATE MANAGEMENT TOOLS

When you're building modern frontends, the way you do state management becomes essential. How do you decide on which approach to take? There are 3 basic questions that need to be answered:

1. Size of application

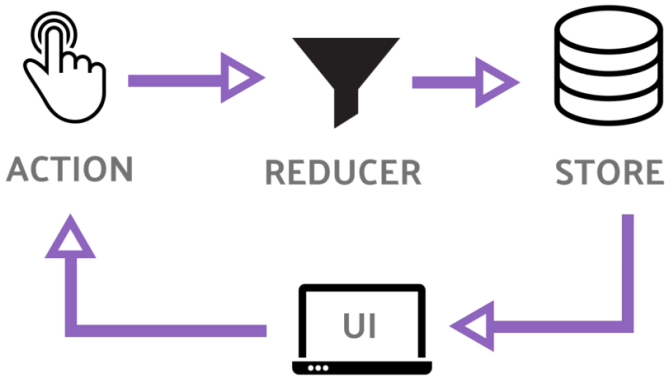
   The bigger the size (i.e. amount of components), the more layers the application state has to pass through.
2. Frontend framework

   Each framework has communities that have built tools in order to make state management easier.
3. Team expertise

   The strategy to use should match with the skill level of the development team that builds the application, as well as those that will maintain it in the future.

This Flux architectural pattern has been the inspiration for many libraries that aim to bring an implementation to our modern frontend application. When it comes to tools the industry standard has been Redux, which is an independent library available that can be used with React, Angular & Vue.

To the right you'll find the way Redux implements the Flux architectural pattern. Notice how it's very much similar. The only exception being the reducer, which is actually just a more descriptive term to describe what the dispatcher would be doing: *modify the state in order to update the store*.



Nowadays the modern frameworks have evolved to now have their own built-in state management solution (being heavily inspired by Redux). React has the Context API, Angular has NgRx and Vue has Vuex. Different implementations that work more closely with the framework, but that ultimately solves the same problem.

Learning Materials

- Top 5 React State Management Libraries
- Is Redux Dead?

Backend Track

Week	Topics
1	Introduction to Specialization
2	Core Concept Specialization
Frontend	State Management
Backend	Fundamentals of Databases
3	Agile Fundamentals
4	Project Management Fundamentals

Learning goals:

- ❏ Get a better idea of the complexity of data and data storage
- ❏ Understand the basics of relational databases
- ❏ Learn essential SQL syntax in order to do common database operations

WHAT IS DATA

Data is information that adds knowledge. It tells you something you didn't know before.

For example: *"At 9:30 on Sunday November 10th 2019 Reza Abadi was walking in Nablus, Palestine."*

This is data because it adds knowledge: you now know where Reza was at a particular point in time.

For a computer this sentence is difficult to understand, however, because there's no meaning assigned to each individual part. It's much easier if you structure it this way:

- City: Nablus
- Country: Palestine
- Who: Reza Abadi
- When: 2019-11-10T09:30:00+02:00 (+02:00 refers to the Palestinian timezone)
- Activity: Walking

A database, also known as an *information system*, will allow you to store structured data and at a later point in time retrieve that data again. You can ask the database: "Who was walking in Nablus on November 10th 2019?" and the database will tell us: "Reza Abadi".

Learning Materials

- What is an Information System

WHY DO WE NEED A DATABASE?

The evolution of data storage goes through many stages. There used to be a time where everyone would invent their own data formats and operations in order to access that data. While this worked in terms of being able to keep data safely stored, it was hard for different systems to communicate.

Learning Materials

- The Data Storage Timeline

However, as businesses started to use more technology in order to automate their process, system requirements became more complex. As a result, there arose a need for a database system that could store and restore data extremely fast. More importantly, there arose a need for a standardized way of doing it: this is how relational databases and SQL (Structured Query Language) were born.

From a technical perspective, the primary reason why you would want to use a database is that you can store information in a reliable and structured way.

The database will ensure your information is stored safely, with a high degree of reliability whenever we want to store or retrieve data.

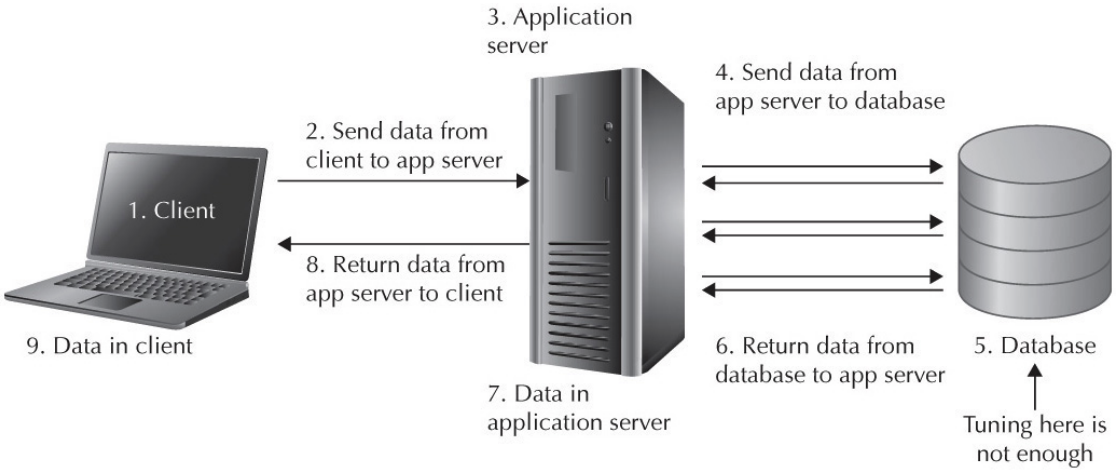
However, keep in mind that technology is always used to support business goals. This makes the ultimate goal of a database just storage of data, but the reliable extraction of information to support decision making by key people and groups in the organization!

Learning Materials

- Intro to Databases

ROLE OF A DATABASE IN A WEB APPLICATION

A database involves two components, a *server* and a *client*. The server is the actual database management system (DBMS) and runs as a process on a machine either on your computer or on another computer in a data center somewhere. The client is an application server (or web server) that communicates directly with the DBMS server.



Unlike the web applications that you are used to, DBMSes usually do not come with a frontend (also known as a user interface). The database server application can only be given commands using the *Command Line Interface* (CLI) or using a *web server* that provides instructions to the DBMS.

That said, it is possible to give your database a graphical user interface (GUI). Examples are MySQL Workbench, PgAdmin or DBeaver. These GUIs allow you to easily show the structure and contents of your database and run your own queries.

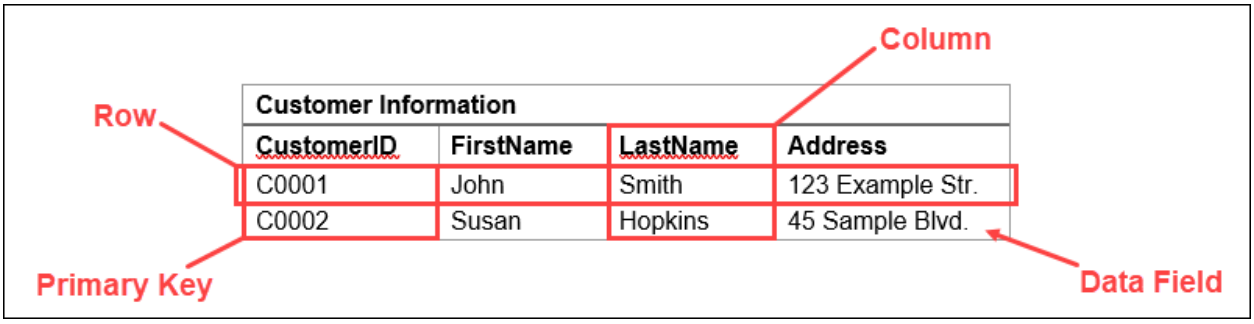
WHAT IS A RELATIONAL DATABASE?

A *relational database* (also known as a Relational Database Management System, RDMS for short) is a type of database that stores and provides access to data that relates to each other. This is done by use of tables. In the image above you'll find the most common examples of relational databases.

A table consists of columns and rows. When a row is filled with data, we call this a record.

The columns of the table hold attributes of the data, and each record usually

has a value for each attribute, making it easy to establish the relationships among data points.



Learning Materials

- Relational Database Concepts
- Relational Model in DBMS
- A beginner's guide to database table relationships

SQL

SQL (pronounced "ess-que-el") stands for Structured Query Language. SQL is a standardized language used to communicate with a relational database. SQL statements are used to perform tasks such as *UPDATE data* on a database, or *retrieve data* from a database.

Common relational database management systems that use SQL are: Oracle, Sybase, Microsoft SQL Server, MySQL or PostgreSQL. Although most database systems mainly use SQL in order to query a database (a way of saying “to

communicate” with a database), most of them also have their own additional proprietary extensions that are usually only used on their system.

The standard SQL commands such as "SELECT", "INSERT", "UPDATE", "DELETE", "CREATE", and "DROP" can be used to accomplish almost everything that one needs to do with a database.

In the following sections we’ll take a look at each more deeply.

Learning Materials

- What is the SQL Language?

SELECT

The **SELECT** statement is used to query the database and retrieve selected data that match the criteria that you specify. Here is the format of a simple SELECT statement:

```
SELECT "column1"
  [,"column2",etc]
  from "tablename"
  [where "condition"];
    [] = optional
```

The column names that follow the **SELECT** keyword determine which columns will be returned in the results. You can SELECT as many column names that you'd like, or you can use a "*" to SELECT all columns.

The table name that follows the keyword **FROM** specifies the table that will be queried to retrieve the desired results.

The **WHERE** keyword (optional) specifies which data values or rows will be returned or displayed, based on the criteria described after the keyword **WHERE**.

Conditional operators used in the **WHERE** keyword:

=	Equal
>	Greater than
<	Less than
>=	Greater than or equal
<=	Less than or equal
<>	Not equal to

CREATE

The CREATE statement is used to CREATE a new table. Here is the format of a simple CREATE table statement:

```
CREATE table "tablename"
("column1" "data type",
 "column2" "data type",
   "column3" "data type");
```

Format of CREATE table if you were to use optional constraints:

```
CREATE table "tablename"
("column1" "data type"
         [constraint],
 "column2" "data type"
         [constraint],
 "column3" "data type"
        [constraint]);
   [ ] = optional
```

Note: You may have as many columns as you'd like, and the constraints are optional.

Example:

```
CREATE table employee
(first varchar(15),
 last varchar(20),
 age number(3),
 address varchar(30),
 city varchar(20),
   state varchar(20));
```

To CREATE a new table, enter the keywords CREATE table followed by the table name, followed by an open parenthesis, followed by the first column name, followed by the data type for that column, followed by any optional constraints, and followed by a closing parenthesis.

It is important to make sure you use an open parenthesis before the beginning table, and a closing parenthesis after the end of the last column definition. Make sure you separate each column definition with a comma. All SQL statements should end with a ";".

The table and column names must start with a letter and can be followed by letters, numbers, or underscores - not to exceed a total of 30 characters in length. Do not use any SQL reserved keywords as names for tables or column names (such as "SELECT", "CREATE", "INSERT", etc).

Data types specify what the type of data can be for that particular column. If a column called "Last_Name"is to be used to hold names, then that particular column should have a "varchar" (variable-length character) data type.

Here are the most common data types:

`char(size)`	Fixed-length character string. Size is specified in parenthesis. Max 255 bytes.
`varchar(size)`	Variable-length character string. Max size is specified in parenthesis.
`number(size)`	Number value with a max number of column digits specified in parenthesis.
`date`	Date value
`number(size,d)`	Number value with a maximum number of digits of "size" total, with a maximum number of "d" digits to the right of the decimal.

What are constraints? They're rules associated with a column that the data entered into that column must follow. When tables are created, it is common for one or more columns to have constraints associated with them.

For example, a "unique" constraint specifies that no two records can have the same value in a particular column. They must all be unique. The other two most popular constraints are "not NULL" which specifies that a column can't be left blank, and "primary key". A "primary key" constraint defines a unique identification of each record (or row) in a table.

INSERT

The **INSERT** statement is used to INSERT (or add) a row of data into the table.

To INSERT records into a table, enter the key words **INSERT into** followed by the table name, followed by an open parenthesis, followed by a list of column names separated by commas, followed by a closing parenthesis, followed by the keyword **values**, followed by the list of values enclosed in parenthesis.

The values that you enter will be held in the rows and they will match up with the column names that you specify. Strings should be enclosed in single quotes, and numbers should not.

```
INSERT into "tablename"
(first_column,...last_column)
 values (first_value,...last_value);
```

In the example below, the column name `first` will match up with the value `'Luke'`, and the column name `state` will match up with the value `'Georgia'`.

```
INSERT into employee
 (first, last, age, address, city, state)
 values ('Luke', 'Duke', 45, '2130 Boars Nest',
         'Hazard Co', 'Georgia');
```

Note: All strings should be enclosed between **single** quotes: `'string'`

UPDATE

The **UPDATE** statement is used to UPDATE or change records that match a specified criteria. This is accomplished by carefully constructing a WHERE keyword.

```
UPDATE "tablename"
set "columnname" = "newvalue"
[,"nextcolumn" = "newvalue2"...]
where "columnname"
 OPERATOR "value"
[and|or "column"
 OPERATOR "value"];
[] = optional
```

Examples:

```
UPDATE phone_book
 set area_code = 623
 where prefix = 979;

UPDATE phone_book
 set last_name = 'Smith', prefix=555, suffix=9292
 where last_name = 'Jones';

UPDATE employee
 set age = age+1
 where first_name='Mary' and last_name='Williams';
```

DELETE

The **DELETE** statement is used to DELETE records (also known as rows) from the table.

```
DELETE from "tablename"
```

```
where "columnname"
 OPERATOR "value"
[and|or "column"
 OPERATOR "value"];

[ ] = optional

DELETE from employee;
```

If you leave off the **WHERE** keyword, <u>all records will be deleted!</u>

```
DELETE from employee
 where lastname = 'May';

DELETE from employee
 where firstname = 'Mike' or firstname = 'haneen';
```

To DELETE an entire record/row from a table, enter "`DELETE from`" followed by the table name, followed by the where `keyword` which contains the conditions to DELETE. If you leave off the `where` keyword, all records will be DELETEd.

Learning Materials

- SQL Tutorial - Full Database Course for Beginners