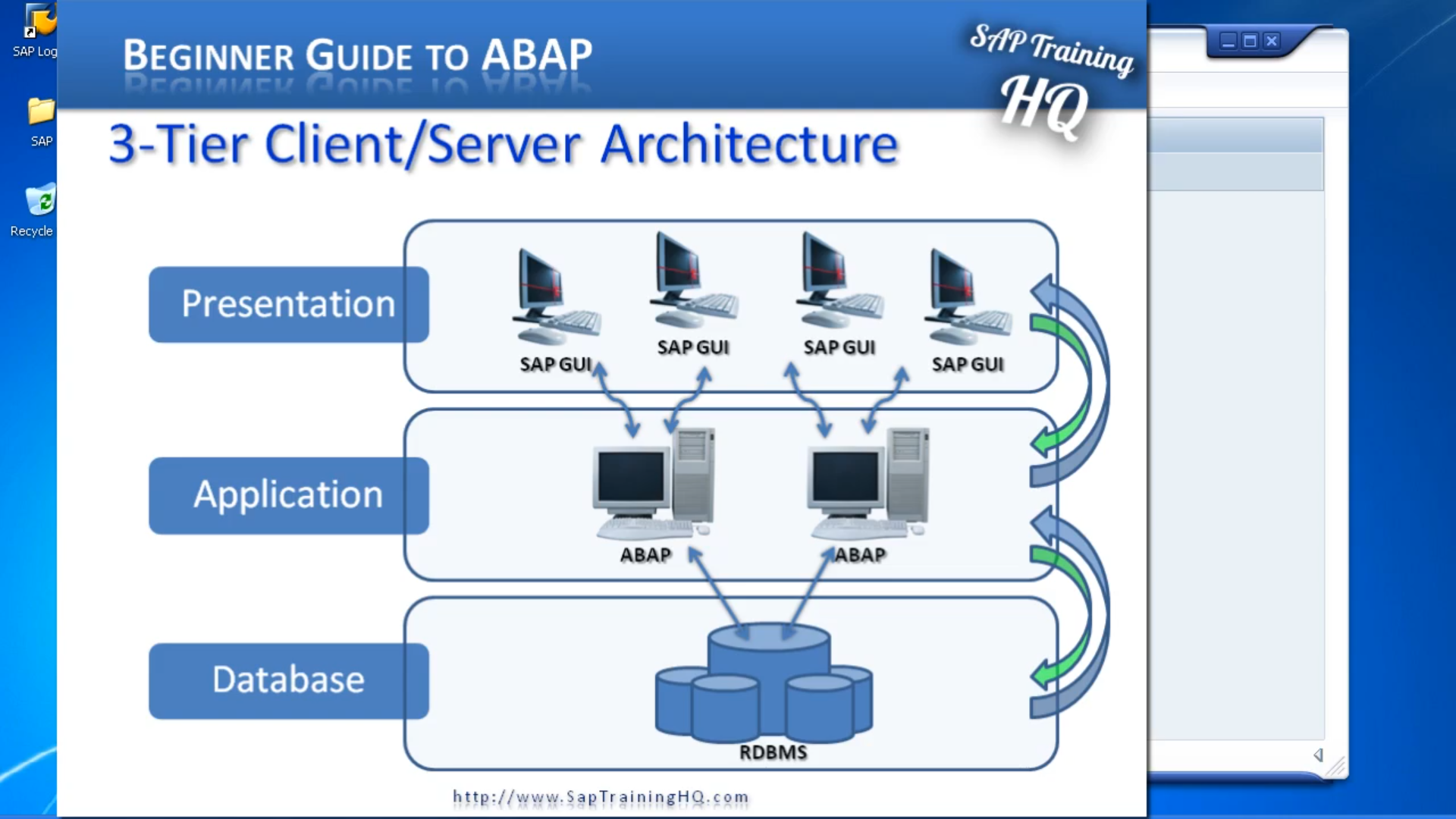
now time to take a look at how we can modify and delete data using ABAP.



There are some important concepts to keep in mind when going through this module including the architecture of our system. Now as a reminder, we have a three-tier architecture, we have the client who uses the tool such as the SAP GUI to request data from the application server, our second tier, which in turn communicates with the database server, our third tier.

Now when we create our programs, we have to keep in mind that there could be tens,100, maybe 1000 of users all accessing the same data at the same time, and we need to ensure the programs that we create do not cause any problems with the rest of the system, and ensure we access the most recent version of the data. Held on the database when our program runs.

Because think about it. If you've got 1000 of users, you've got records probably being updated every single minute of every single working day, and you need to make sure your programs can read, add, and update data that is current in the system. Now fortunately for us an SAP system takes care of a lot of this work for us. And the ABAP programmer doesn't have to worry about the underlying technology of how records are locked and being used by other programs at the same time. And one of the key tools that we'll use in our programs is what we call open SQL. And no doubt many of you have heard of SQL. It's standard query language. SAP, I've introduced what's called open SQL which acts like an interface between our program and the database. By using Open SQL, we can read the modified data, and also buffer data, on the application server, which in turn reduces the number of database accesses our system has to perform. And it's the database interface which is also responsible for synchronizing the buffers with the database tables at predetermined intervals. And when we are creating programs, we've got to keep in mind that if data is buffered and we are reading this buffered data it may not always be up to date. So, when we create tables, we need to create them in such a way that we tell the system that buffering can be used or it can't be used. Or maybe it can only be used in certain situations. And when we created our example table in the system in the earlier module. We touched on this, and we told the system that for our table do not use buffering and by using this setting that meant every time we read data from our table, we will always read the most up to date records. We will not be accessing data that was read from the table say five seconds, ten seconds or even a minute, five minutes ago. Now, buffering can be great for tables that hold master data and configuration settings because data in those type of tables don't get updated on a regular basis, but when we're working transactional data, it’s a whole nother ball game, because if you've got 1000 of users telling the application server to process transactions on transactional data. But when we read that data, we want it to be as up-to-date as we can, so we don't want buffering set on transactional data. And if you are dealing with transactional data, whether tables are configured to use buffering, you have to ensure your program can take this into account. And make sure the buffer gets updated with brand new data at the time you need it. Now back to open SQL. Remember, I said that open SQL acts as an interface between our programs and the database. When we use open SQL statements in our program, we are only allowed to access tables through the ABAP dictionary. This is our interface. We don't actually access the database tables directly. Through our programs now there's no need to worry about how we go about doing this because, in our programs, we use open SQL statements just as though we were accessing the database tables. Directly. We don't have to think about it. Open SQL as an interface manages the database access all by itself. All we need to do is code our statements just as though we were accessing the tables. But we will have the underlying knowledge that by using open SQL we are accessing the data through the ABAP dictionary, which has a built-in level of safety, to ensure our ABAP code does not have a direct effect on the SAP database system itself.