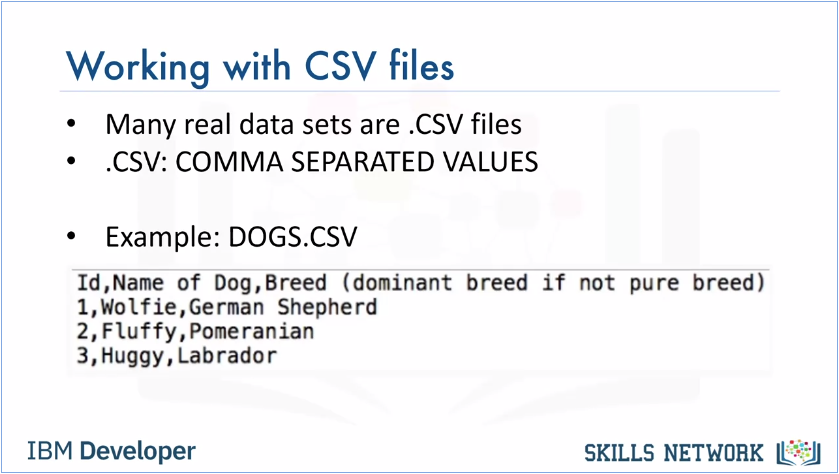
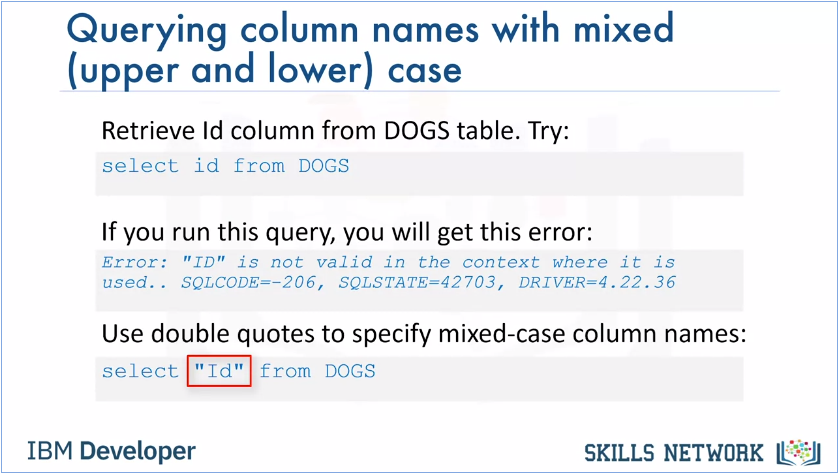
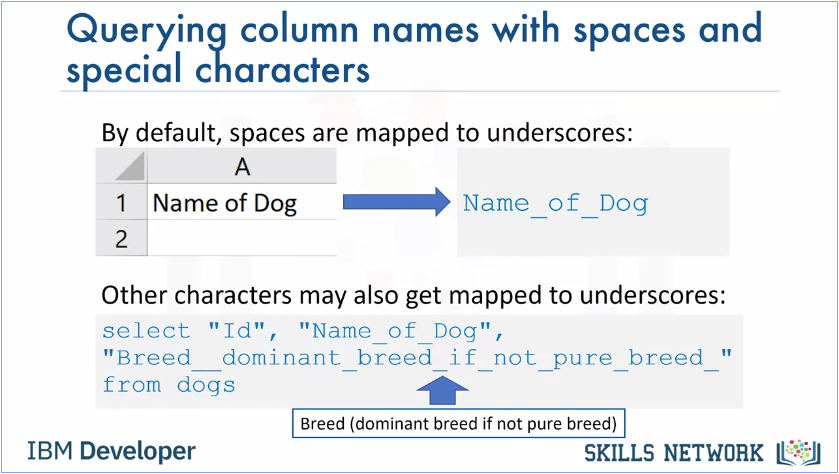
Working with Real-World Data Sets and Built-in SQL Functions

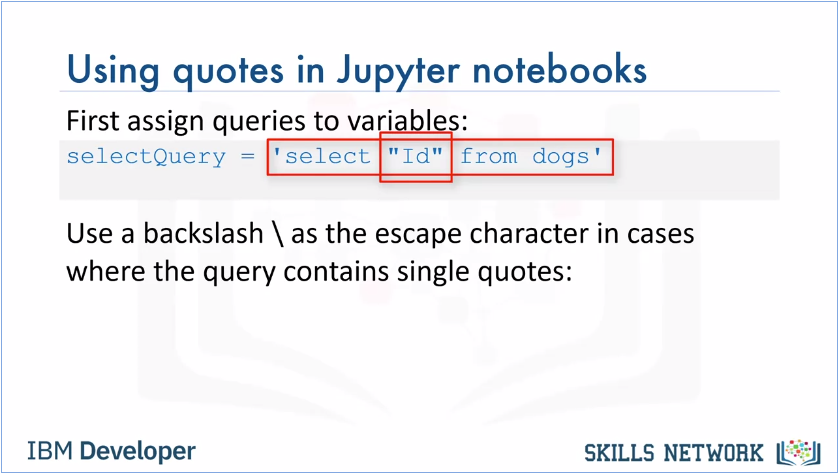




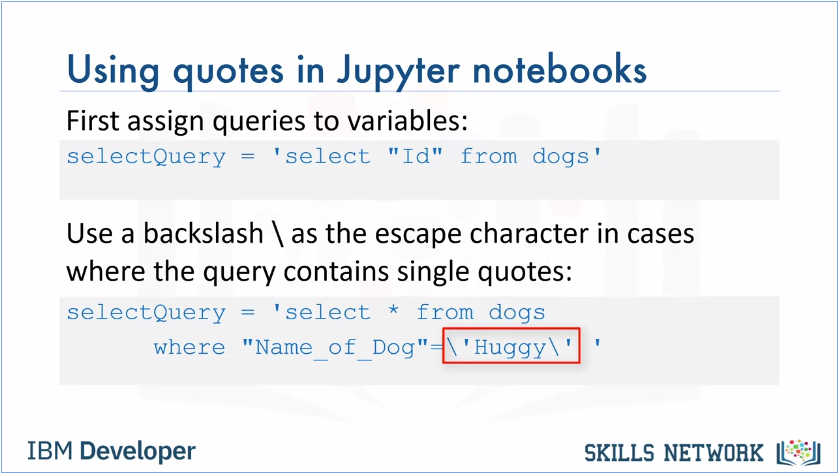
* Let’s talk about querying column names that are lower or mixed case, that is, a combination of upper and lowercase. Let's assume we loaded the DOGS.CSV file using the default column names from the CSV. If we try to retrieve the contents of the Id column using the query { select id from DOGS }, we'll get an error as shown indicating the id is not valid. This is because the database parser assumes uppercase names by default. Whereas when we loaded the CSV file into the database, it had the Id column name in mixed case, i.e. an uppercase I and a lowercase d (Id). In this case, to select data from a column with a mixed case name, we need to specify the column name in its correct case within double quotes as follows: { Select \* "Id" from DOGS }. Ensure we use double quotes around the column name and not single quotes.



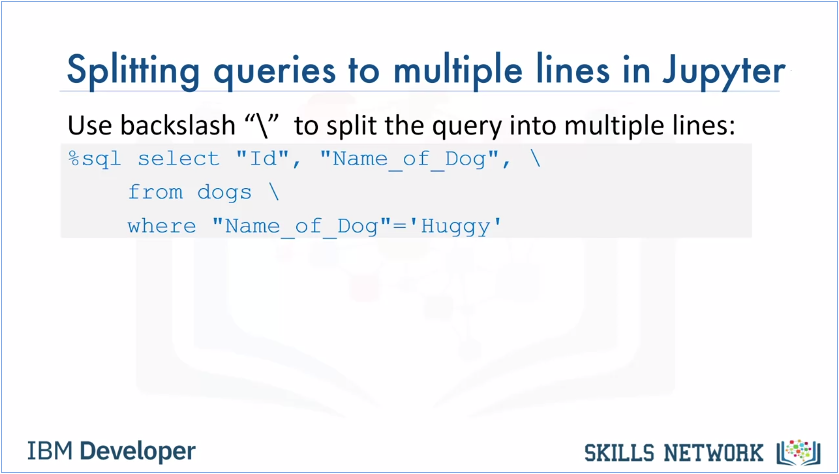
* Next, we'll cover querying column names that have spaces and other characters. In a CSV file, if the name of the column contains spaces, by default the database may map them to underscores. For example, in the name of dog column, there are spaces in between the three words. The database may change it to Name\_of\_Dog.
* Other special characters like parentheses or brackets may also get mapped to underscores. Therefore, when we write a query, ensure we use proper case formatting within quotes and substitute special characters to underscores as shown in this example. {Select "Id," "Name\_of\_Dog," "Breed\_\_dominant\_breed\_if\_not\_pure\_breed\_"from dogs }.
* Please note the underscores separating the words within double quotes.
* Also note the double underscore between breed and dominant as shown.
* Finally, it's also important to note the trailing underscore after the word breed near the end of the query.This is used in place of the closing bracket.



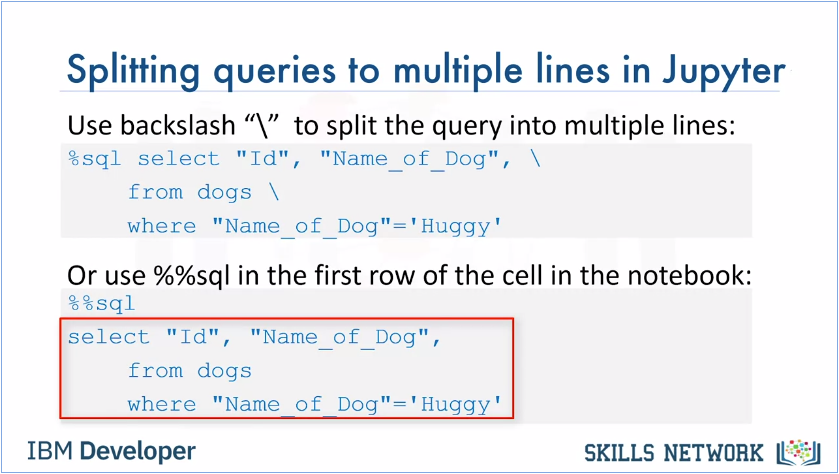
* When using quotes in Jupyter notebooks, we may be issuing queries in a notebook by first assigning them to Python variables. In such cases, if wer query contains double quotes for example, to specify a mixed case column name, we could differentiate the quotes by using single quotes for the Python variable to enclose this SQL query and double quotes for the column names. For example, selectQuery ='select "Id" from dogs.'



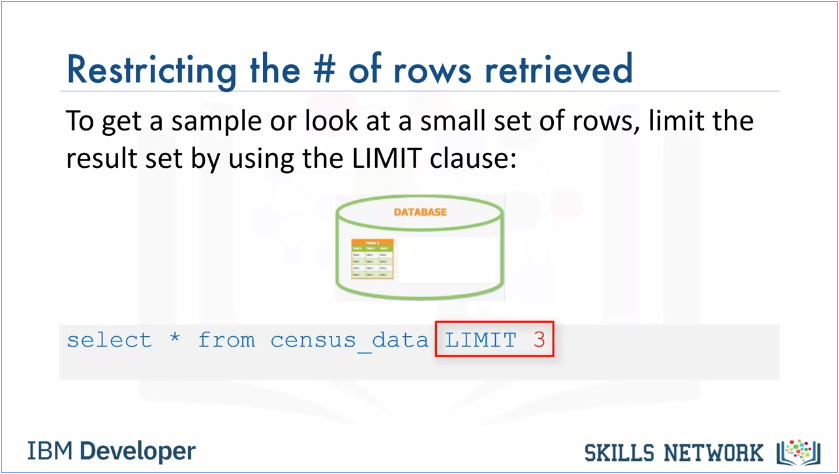
Now, what if we need to specify single quotes within the query, for example, to specify a value in the where clause? In this case we can use backslash as the escape character as follows, select Query = 'select \* from dogs where "Name\_of\_Dog"=\'Huggy\' '.



* If we have very long queries such as join queries or nested queries, it may be useful to split the query into multiple lines for improved readability. In Python notebooks, we can use the backslash character to indicate continuation to the next row as shown in this example. %sql select "Id," Name\_of\_Dog," \ from dogs \ where"Name\_of\_Dog" = 'Huggy.' It would be helpful at this point to take a moment to review the special characters as shown. Please keep in mind that we might get an error if we split the query into multiple lines in a Python notebook without the backslash.

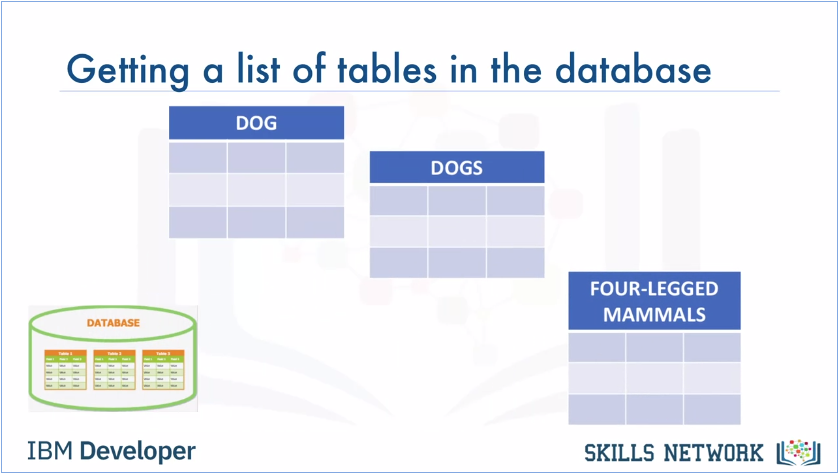


When using SQL magic, we can use the double percent SQL in the first line of the cell in Jupyter Notebooks. It implies that the rest of the content of the cell is to be interpreted by SQL magic. For example %% sql new row select "Id", "Name\_of\_dog," new row, from dogs, new row, where "Name\_of\_dog = 'Huggy.' Again, please note the special characters as shown. When using %%sql, the backslash is not needed at the end of each line.

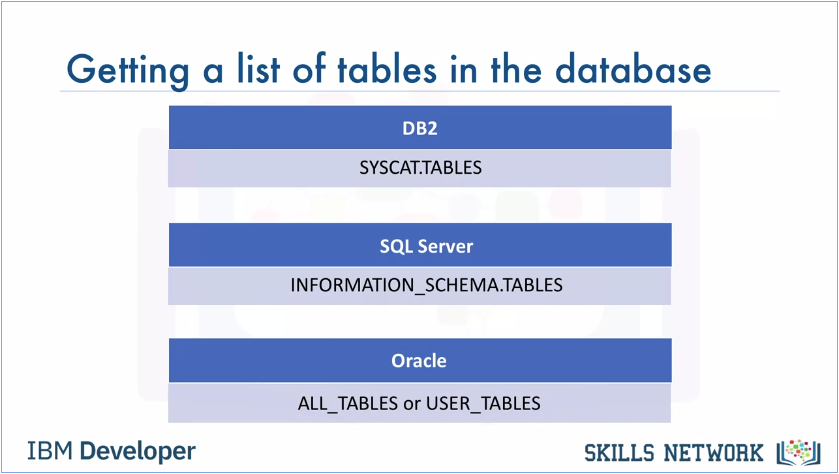


* At this point we might be asking, how would we restrict the number of rows retrieved? It's a good question because a table may contain thousands or even millions of rows, and we may only want to see some sample data or look at just a few rows to see what kind of data the table contains. We may be tempted to just do select \* from table name to retrieve the results in a Pandas data frame and do a head function on it. But, doing so may take a long time for a query to run. Instead, we can restrict the results set by using the limit clause. For example, use the following query to retrieve just the first three rows in a table called census data. Select \* from census\_data limit 3.

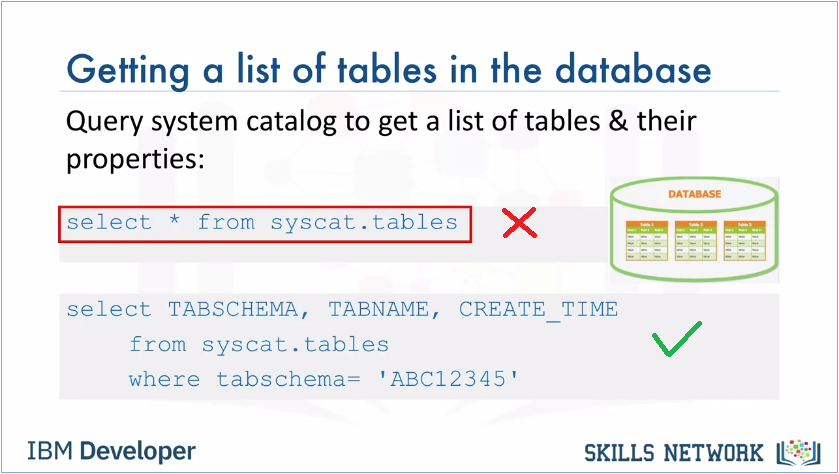
**Getting Table and Column Details**



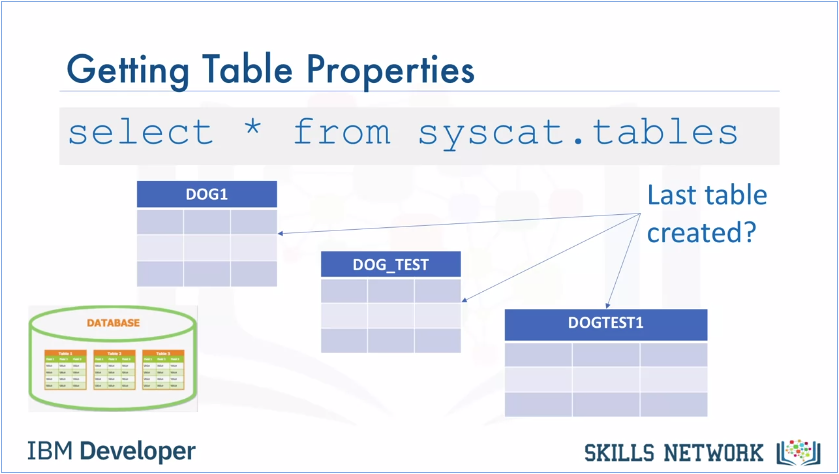
* How would we get a list of tables in the database? Sometimes wer database may contain several tables, and we may not remember the correct name. For example, we may wonder whether the table is called dog, dogs or four legged mammals…?



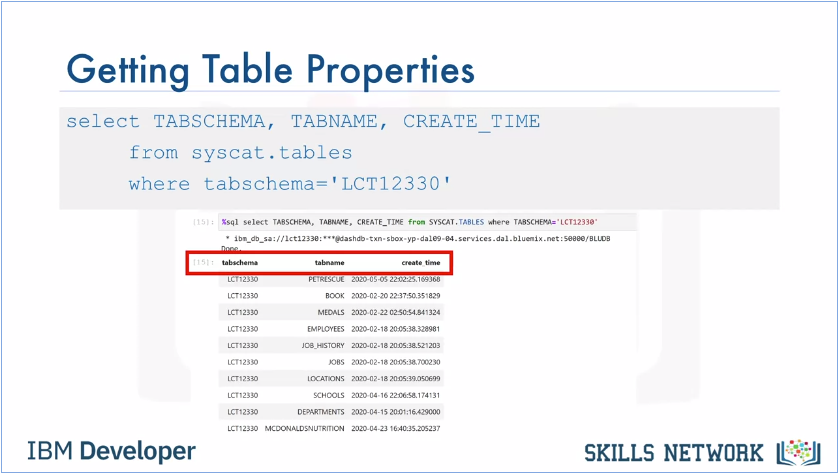
* Database systems typically contain system or catalog tables, from where we can query the list of tables and get their properties. In DB2 this catalog is called syscat tables. In SQL Server, it's information schema tables, and in Oracle it's all tables or user tables.



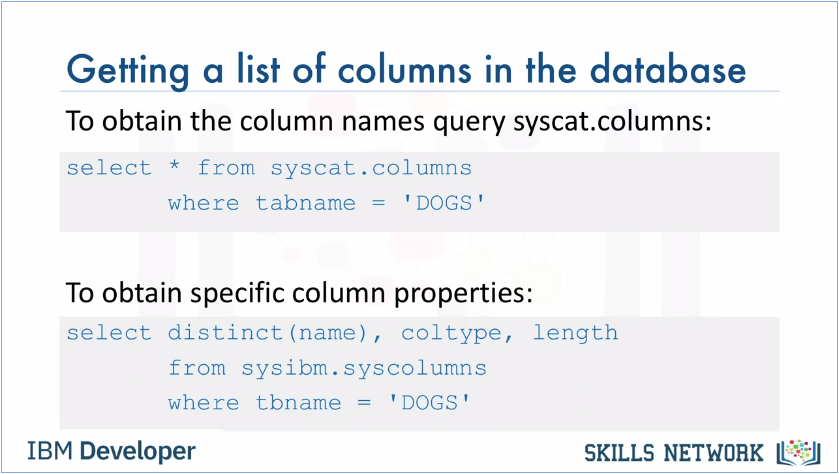
* To get a list of tables in a DB2 database, we can run the following query. Select star from syscat tables. This select statement will return too many tables including system tables, so it's better to filter the result as shown here. Select tabschema, tabname, create underscore time from syscat tables, where tabschema equals ABC12345. Please ensure that we replace ABC12345 with our own DB2 username.



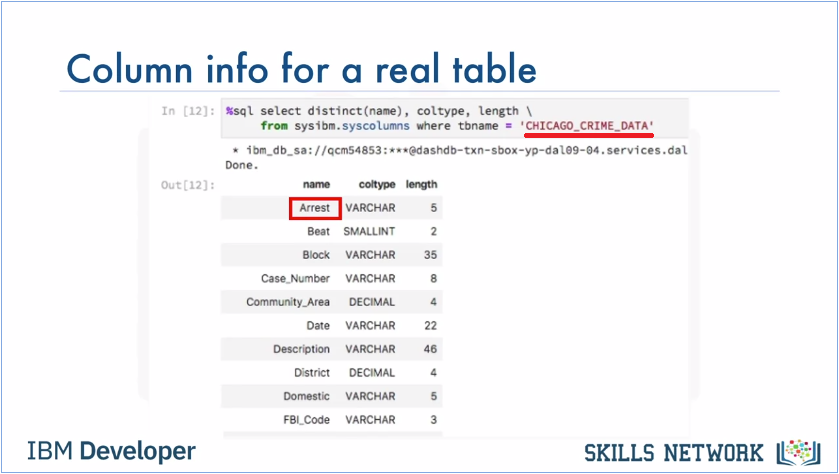
When we do a select star from syscat tables, we get all the properties of the tables. Sometimes we're interested in specific properties such as creation time. Let's say we've created several tables with similar names. For example, dog one, dog underscore test, dog test one and so on. But, we want to check which of these tables was the last one we created.



* To do so, we can issue a query like select tabschema, tabname, create underscore time from syscat tables: Where a tabschema equals LCT12330. The output will contain the schema name, table name, and creation time for all tables in our schema.



* Next, let's talk about how to get a list of columns in a table. If we can't recall the exact name of a column for example, whether it had any lowercase characters or an underscore in its name, in DB2 we can issue a query like the one shown here. Select star from syscat columns where tab name equals dogs. For our information, in my SQL, we can simply run the command show columns from dogs, or we may want to know specific properties like the datatype and length of the datatype. In DB2, we can issue a statement like, select distinct name, coltype, length from sysibm, syscolumns where tbname equals dogs.



* Here we look at the results of retrieving column properties, for a real table called Chicago Crime Data from a Jupyter notebook. Notice in the output, we can tell certain column names show different cases. For example, the column titled arrest has an uppercase A, and the rest of the characters are lowercase. So, keep in mind that when we refer to this column in wer query, not only must we enclose the word arrest within double quotes, we must also preserve the correct case inside the quotes.