In queries that use **UNION** (and in other types of queries), you will sometimes need to use *explicit type conversion* (also called *explicit casting*) to convert a column (or a scalar value) from one data type to another. In most SQL dialects, this is done using the **cast** function. However, you need to be careful about a couple of things when using **cast**.

Review: Explicit Type Conversion

As you might recall, you can cast any numeric column to a character string column, like this:

SELECT cast(list_price AS STRING) FROM games;

If you have a character string column whose values represent numbers, then you can cast it to a numeric column, like this:

SELECT cast(year AS INT) FROM games;

Refer back to the video "Data Type Conversion" in Week 2 of this course if you need more of a refresher on the basics of data type conversion.

Type Conversion Can Return Missing Values

Under some circumstances, the **cast** function will return missing (**NULL**) values. A common situation in which this happens is when you have a character string column whose values do *not* represent numbers, and you try to convert it to a numeric column.

For example, this query attempts to convert the character string values in the **name** column (values like **Monopoly** and **Scrabble**) to integer values:

SELECT cast(name AS INT) FROM games;

When you run this query with Hive or Impala, it returns a column of **NULL** values, because there is no way to cast these character string values to known integer values.

However, some other SQL engines have different ways of handling situations like this. If you use MySQL to cast a character string column as a numeric column, it returns *zeros* for the values that do not represent numbers (not **NULL**s like Hive and Impala). And PostgreSQL throws an error if you attempt to cast a character string that does not represent a number as a number. Also note that different SQL engines have different data types, so the data type name you use after the **AS** keyword in the **cast** function varies depending on the engine. For example, in MySQL you should use **SIGNED INT** instead of **INT**, and in MySQL and PostgreSQL you should use **CHAR** instead of **STRING**.

Type Conversion Can Return Truncated Values

Under some circumstances, the **cast** function will return truncated (cut off) values. A common situation in which this happens is when you convert decimal number values to integer values.

For example, this query converts the decimal numbers in the **list_price** column (which have two digits after the decimal) to integer values:

SELECT cast(list_price AS INT) FROM games;

When you run this query with Impala or Hive, it truncates (cuts off) the decimal point and the two digits after it. For example: **19.99** becomes **19**.

However, in this situation, some other SQL engines *round* instead of truncating. When you run a query like this with MySQL or PostgreSQL, it rounds each decimal number value to the nearest integer value. For example: **19.99** becomes **20**.

✓ 完成 转到下一项