An Introduction to Databases

Ted Laderas

R-Bootcamp

Module 5

Database Tables

Mouse	
MouseID	**VARCHAR(8)**
Strain	VARCHAR(12)
Weight	FLOAT(2)

Experiment	
ExperimentID	**VARCHAR(8)**
BalanceTime	FLOAT(2)
Date	DATE

Assignment	
LabtechID	**VARCHAR(8)**
ExperimentID	**VARCHAR(8)**
MouseID	**VARCHAR(8)**

At their simplest, Databases are related sets of tables with some additional software that governs how we can update them. The database system we are using is called SQLite, which is a very lightweight database system. SQLite is also used as the basis of a file format for a large number of programs, including Photoshop.

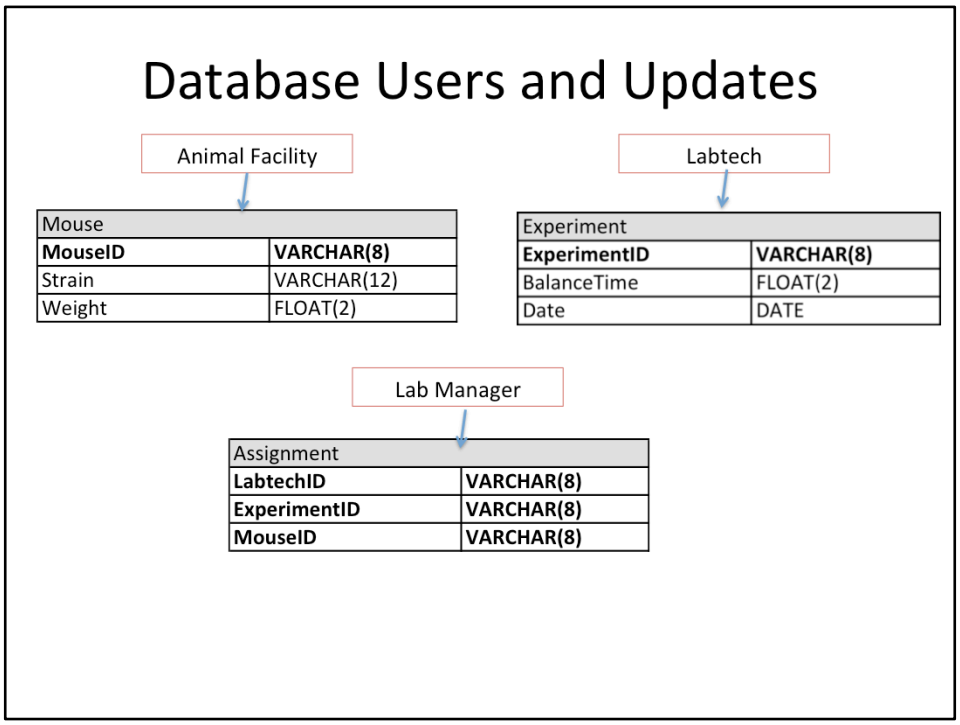
In this slide is a simple database that stores information about 1) mice (Mouse table), 2) experimental results (Experiment table), and 3) which tech is responsible for those results (assignment table).

Each table contains a number of fields, which each have a datatype and size assigned to them. For example, the Experiment table has ExperimentID, which is defined as a VARCHAR (or variable character) field that is 8 characters long. BalanceTime is a FLOAT datatype with 2 decimal places of precision.

Part of the reason we assign fields a size is because of memory management. Databases can become extremely large and unwieldy if we don't place limits on field size. As a database can often have millions of rows in a single table, you can see that it is important to manage the amount of memory for a field in that table.

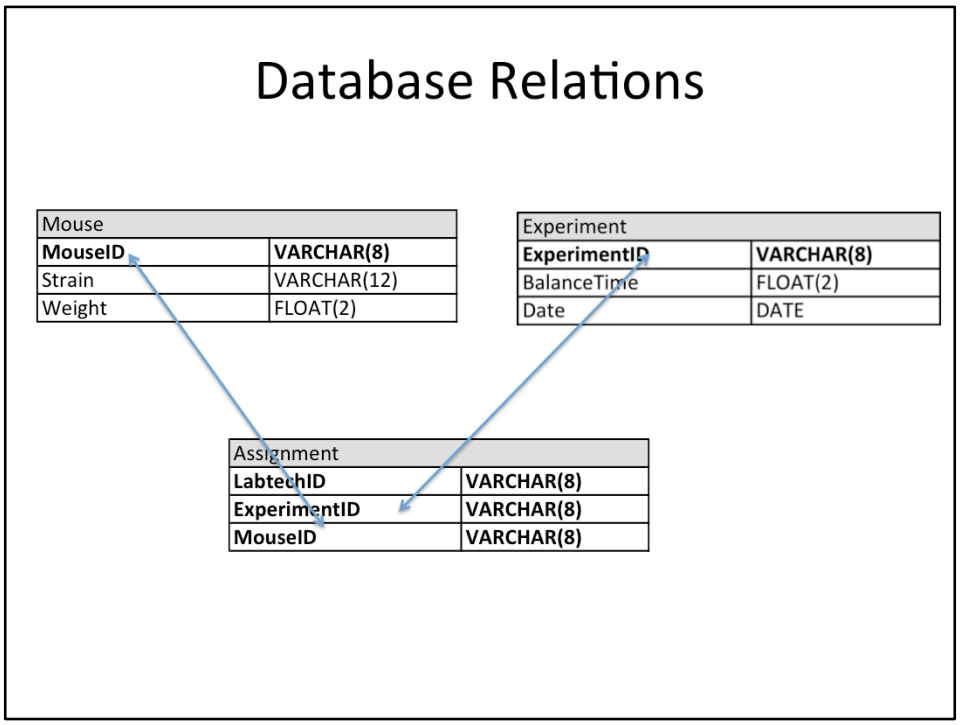
Why not use Excel?

- 1) Scalability – Excel has memory limits
- 2) Concurrency – multiple people can update different parts of the database at the same time
  - "Rollback" – can bring database back up if catastrophic changes are made
- 3) Security – Can set access to users
- 4) Queries – Joining tables using Structured Query Language (SQL)
  - Much like merging, but over multiple tables
  - Enables reporting at multiple levels and use cases



Why not have everything as a single table? Well, one of the strengths of having separate tables is that multiple people can alter the database at the same time.

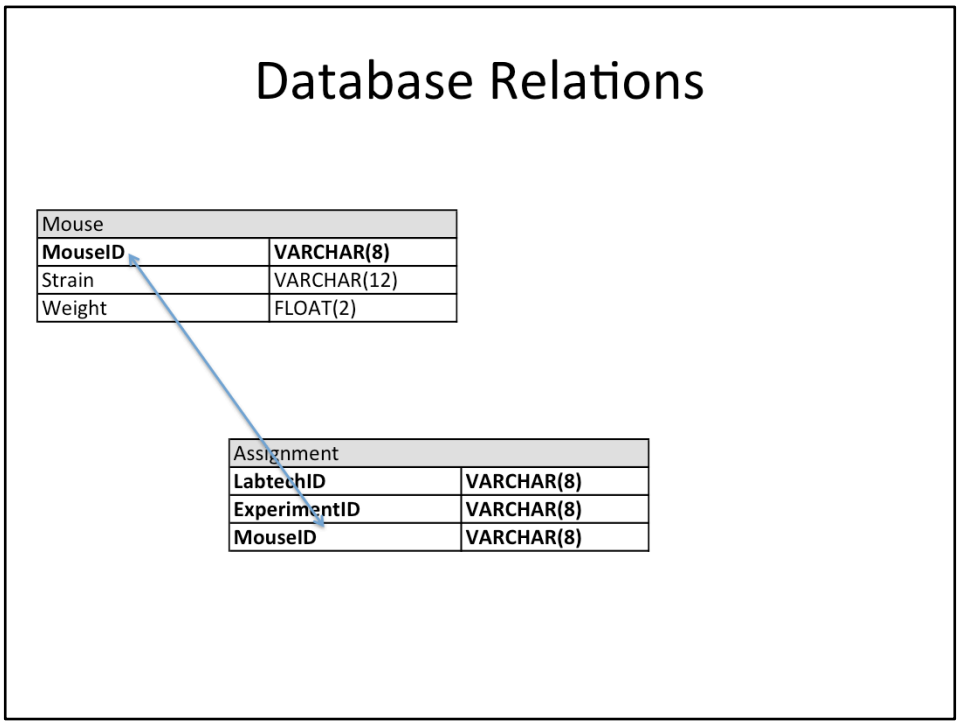
For example, the Mouse table might be updated by people at the Animal Facility, while the experiment table is updated by the lab's labtechs, and the assignment table by the lab manager.



Note that I mentioned the tables are related. Well, tables can contain two special kinds of fields. The first is known as a primary key, and it has the special property that it must be unique to each row in the dataset. This allows us a unique way to identify a row, which is important, because we will be doing the equivalent of subsetting and merge operations on these tables.

The second type of key is called a foreign key, and it refers to a primary key field in another table. Essentially, you can think of foreign keys as columns that we can merge our tables on.

The primary keys for each table are highlighted in bold. Note that for the Assignment table, we count the combination of LabtechID, ExperimentID and MouseID as our primary key, because the combination of each is unique. However, ExperimentID and MouseID are considered foreign keys, because they refer to unique rows in the other two tables.

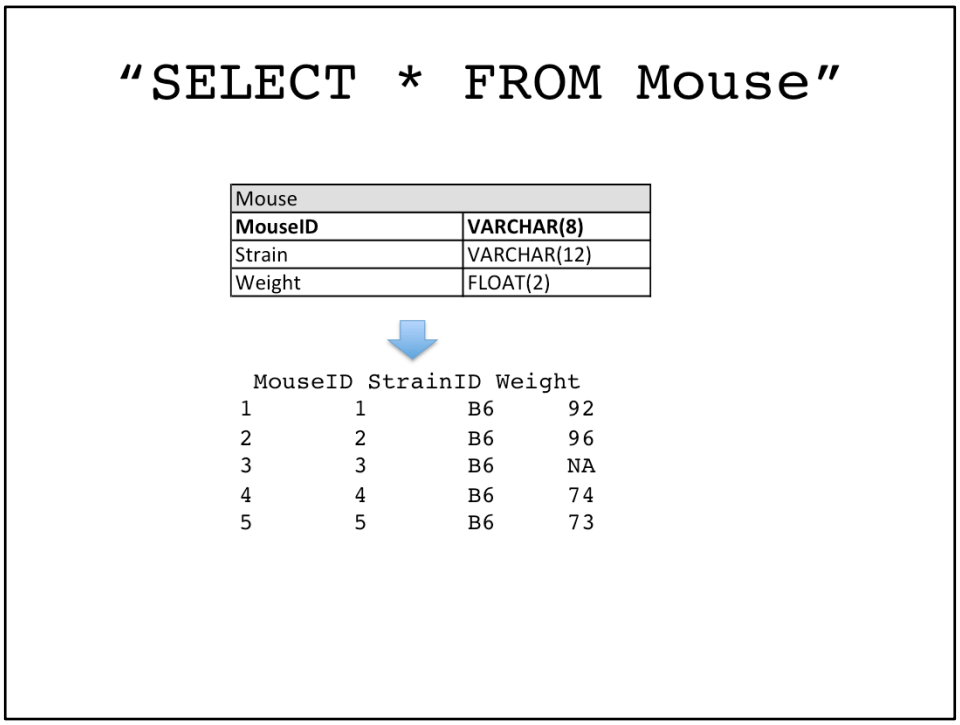


I mentioned that there is additional software on top of these tables. In fact, it's a specialized language (called Structured Query Language, or SQL) based on set-theoretical operations that enable us to filter and merge tables based on criteria.

We can leverage SQL to return and merge portions of these tables based on our criteria. For example, we might want to merge the Mouse and Assignment tables, but only for a particular date.

SELECT statement

- Selects the tables you want to use
- Can also create aliases (shortcuts) to make your query more understandable
- Examples:
  - `"SELECT * FROM Mouse"`
    - Selects everything from the Mouse table
  - `"SELECT Strain, MouseID FROM Mouse"`
    - Selects only the Strain and MouseID fields

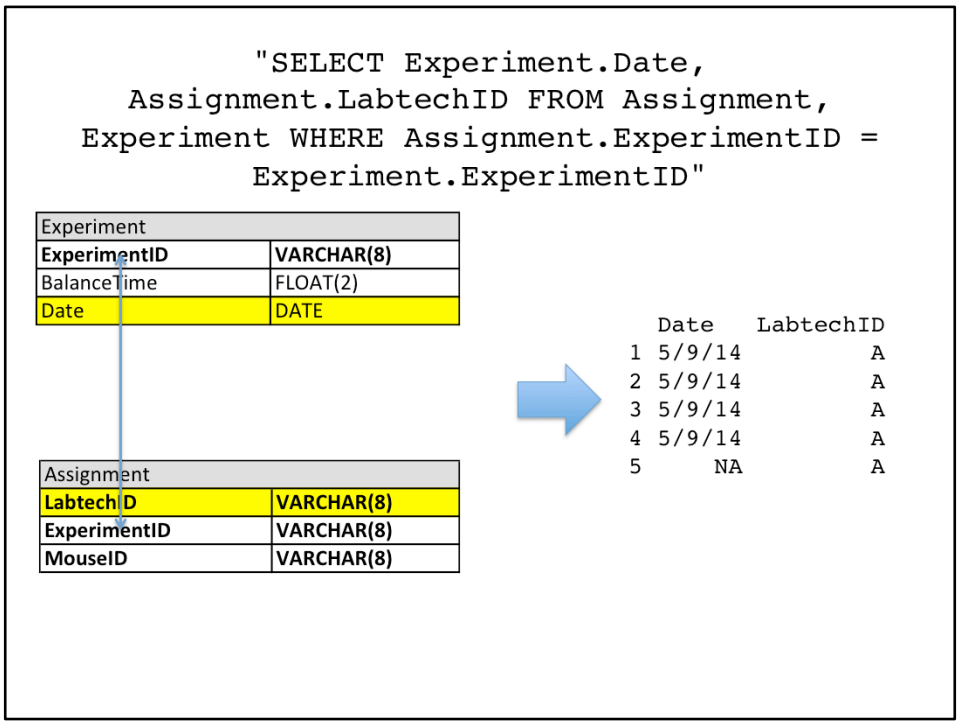


Here we use the SELECT statement to grab all fields from the Mouse table and return it into memory.

WHERE

- Select filtering criteria and joining criteria goes here
- Examples:
  - “SELECT * FROM Mouse WHERE Weight < 60”
  - "SELECT Experiment.Date, Assignment.LabtechID FROM Assignment, Experiment WHERE Assignment.ExperimentID = Experiment.ExperimentID"

The WHERE part of the SQL statement is where you can add filtering and joining steps.



Let's take the second example apart. The first thing is that we want the Date field (from the Experiment Table) and the LabtechID field (from the Assignment Table). But to get this information, we will need to merge on fields that exist in both tables, the ExperimentID field. Our WHERE clause does the merging step.

Note that in our example, we don't return the merged field. Returning the merged field is optional, but is a good check to ensure that our join went correctly.