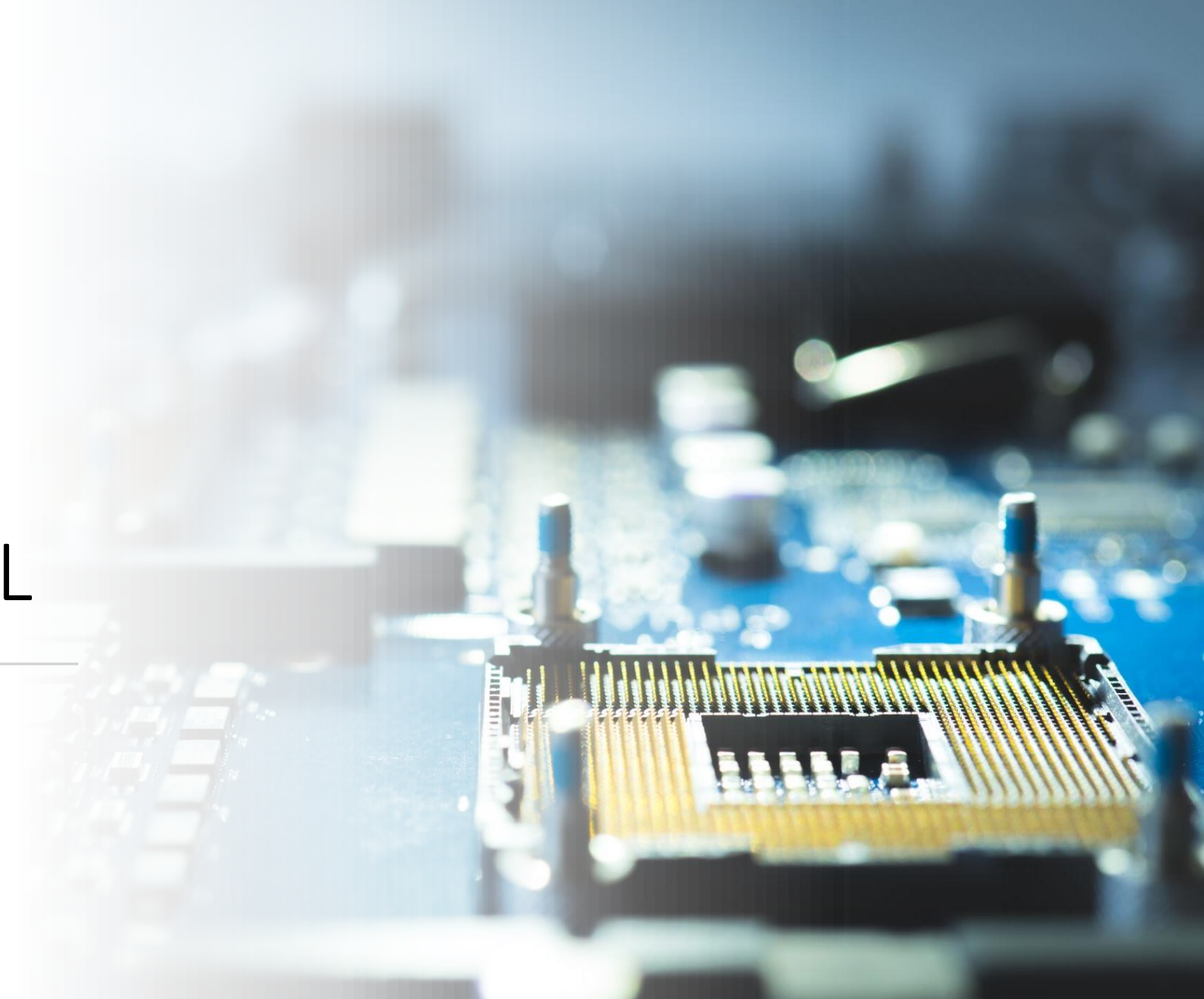
Additional SQL



The UPDATE Command

Introduction to the UPDATE command

The UPDATE command is used to define changes within your database tables. As you're probably aware, database information is not static…it's constantly changing depending on user feedback or input. As an example, assume that an administrator wanted to change specific data (maybe a username and password) for a particular employee within the employees table. To make these changes to an existing record in the table, an UPDATE statement would have to be created.

The UPDATE Command

Introduction to the UPDATE command

The UPDATE statement requires certain keywords, operators, and usually a WHERE clause to modify the specific record. To give you an example, consider the following statement:

```
UPDATE employees SET name = 'Zac' WHERE name = 'Zak'
```

Or

```
UPDATE employees SET name = 'Zac' WHERE employeeid = 1
```

In both scenarios, the employees table is modified. In the first example, all instances of Zak are changed to Zac. In the second example, the name field is replaced with Zac for the employeeid which is equal to 1.

The UPDATE Command

Introduction to the UPDATE command

The UPDATE statement generally uses the following elements:

- **UPDATE** - The UPDATE keyword is used to identify the statement or action you are attempting to perform on the database.
- **Table name** - The name of the table for which you want to update values for.
- **SET** – Specifies that you want to set a field to a particular value.
- **Fields** – The field that you want to set a new value for immediately follows the SET keyword. The equal sign (=) will immediately follow the name of the field.
- **Value** - The actual value that you want to set the field to.

The UPDATE Command

Introduction to the UPDATE command

Of course you could update multiple fields as well. If this were the case you simply add another field/value directly after the initial field/value, separating them by the comma symbol as follows:

```
UPDATE employees SET name = 'Zac', email = 'zruvalca@sdccd.edu'
WHERE employeeid = 1
```

The DELETE Command

Introduction to the DELETE command

The DELETE command can be used to remove unneeded records from the database. For instance, if you wanted to remove all employees from the employees table, we might write a statement as follows:

```
DELETE FROM employees
```

This statement removes all the employees from the employees table. Of course, this doesn't make much sense unless you fired the entire company! Instead, you might want to delete a specific employee. If this were the case, you could append a WHERE clause to your statement to remove only one record:

```
DELETE FROM employees WHERE employeeid = 1
```

The DELETE Command

Introduction to the DELETE command

The previous statement removes only the record where the employeeid is equal to 1. As was the case with the UPDATE example, you could also delete a user by name as follows:

```
DELETE FROM employees WHERE name = 'Agnes'
```

This statement removes all records from the employees table whose name field matches 'Agnes'.

SQL Expressions

Introduction to SQL expressions

In programming, expressions are anything that when calculated, result in a value. For instance, 1 + 1 = 2 is an example of an expression. Expressions in SQL work similarly. Consider the following data that could appear in the employees table:

employeeid	firstname	lastname
1	Wally	Smith
2	Wilbur	Walters
3	Tina	Stephenson
4	Agnes	Garcia

SQL Expressions

Introduction to SQL expressions

In the previous example you could use a simple SELECT statement to display the information exactly as it appears in the preceding table, or you could write an expression that appends the firstname and lastname fields together. The query would look like this:

```
SELECT employeeid, firstname & lastname AS name FROM employees
```

Notice the & operator (covered in the next lecture). The & operator is used to concatenate (join) two fields into one virtual field using the AS keyword. The results would display as follows:

employeeid	name
1	WallySmith
2	WilburWalters
3	TinaStephenson
4	AgnesGarcia

SQL Expressions

Introduction to SQL expressions

Notice that there is no space between the first and last names. To add a space, you need to add a literal string value as follows:

```
SELECT employeeid, firstname & ' ' & lastname AS name FROM employees
```

Adding the space results in a gap between the first and last names as follows:

employeeid	name
1	Wally Smith
2	Wilbur Walters
3	Tina Stephenson
4	Agnes Garcia

SQL Expressions

SQL aliases

You probably also noticed the AS keyword outlined after the lastname field. This allows you to create an **alias**. An alias is used when joining two fields together into one virtual field. The alias is then displayed in the result set.

```
SELECT employeeid, firstname & ' ' & lastname AS name FROM employees
```

Alias Column

employeeid	name
1	Wally Smith
2	Wilbur Walters
3	Tina Stephenson
4	Agnes Garcia

SQL Expressions

SQL expressions in MySQL and SQLite

Although the previous examples would work perfectly in some databases, MySQL and SQLite used different methods:

MySQL

```
SELECT employeeid, CONCAT(firstname,' ',lastname) AS name FROM employees
```

SQLite

```
SELECT employeeid, firstname || ' ' || lastname AS name FROM employees
```

Common SQL Functions

Introduction to SQL functions

Aside from using operators to manually construct expressions, SQL provides built-in functions (small blocks of code that can perform operations and return a value). Functions are available simply by making a call to them and passing the value and/or values on which you want the function to operate.

Common SQL Functions

Introduction to SQL functions

The functions outlined in this lecture represent a generic list of common SQL functions that most databases support. It's important to understand that not all databases support the same functions. Although in most cases all databases support similar functions, the way the function is written can differ syntactically from database to database. We'll explore the following functions and associated clauses:

- The COUNT function
- The AVG function
- The SUM function
- The MIN and MAX functions
- The GROUP BY clause
- The HAVING clause
- Other database specific functions

Common SQL Functions

The COUNT function

One of the most commonly used functions is the COUNT function. The COUNT function is used when you want to perform a simple count of records. Consider the following data from the employees table:

name	username	password
Wally	wwebmaster	password
Wilbur	wfounder	password
Tina	ttechie	abc123
Agnes	aaccountant	12345
Damon	ddeveloper	ispeakbinary

If you were building a password protected site complete with a login system that authenticates users, the COUNT function would be something to consider taking advantage of.

Common SQL Functions

The COUNT function

In the previous table you see that users, aside from storing their name and email, store their usernames and passwords within the employees table. Assuming usernames must be unique, we could write our query as follows:

```
SELECT COUNT(*) FROM employees WHERE username = 'wwebmaster' AND password = 'password'
```

In this scenario either a 1 or 0 will be returned in the query. Either someone matches that username and password combination or no one does. If the numeric value of 1 is returned back to your application, you would know that the user is valid and could let them pass. If 0 is returned back, you know that they are not a valid user.

Common SQL Functions

The AVG function

The AVG function can be used in SQL with numeric columns to return the average value of a numeric column. Let's assume that we had a table of records that looked like this:

product	price	instock
Widget	10.00	true
Gadget	16.00	true
Doodad	20.00	true

To return the average price of the three products, we could write a query that looked like this:

```
SELECT AVG(price) FROM products
```

The result of the query would be 15.33 since that's the average of the three products.

Common SQL Functions

The SUM function

The SUM function can be used in SQL with numeric columns to return the total (or sum) of a numeric column. Let's assume that we had a table of records that looked like this:

product	price	instock
Widget	10.00	true
Gadget	16.00	true
Doodad	20.00	true

We could write a query that looked like this:

```
SELECT SUM(price) FROM products
```

The result of the query would be 46.00 since that's the sum of 10.00 + 16.00 + 20.00.

Common SQL Functions

The MIN and MAX functions

Like the AVG and SUM functions, the MIN and MAX functions are also used to work with numeric values. The MIN function returns the smallest value of the selected column. The MAX function returns the largest value of the selected column. Consider the following data:

product	price	instock
Widget	10.00	true
Gadget	16.00	true
Doodad	20.00	true

To return the cheapest price of the three products, we could write the following query:

```
SELECT MIN(price) FROM products
```

The result of the query would be 10.00 since that's the cheapest of the three products.

Common SQL Functions

The GROUP BY statement

The GROUP BY statement is often used with aggregate functions (COUNT, MAX, MIN, SUM, AVG) to group the result-set by one or more columns. Consider the following data:

customerid	name	region
1	Mark Jones	United States
2	Sally Smith	United Kingdom
3	David Doe	United States

To list the number of customers in each country, we could write the following query:

```
SELECT COUNT(customerid), region FROM customers GROUP BY region
```

COUNT(customerid)	region
2	United States
1	United Kingdom

Common SQL Functions

The HAVING clause

The HAVING clause exists primarily because the WHERE keyword could not be used with aggregate functions. Consider the following data:

customerid	name	region
1	Mark Jones	United States
2	Sally Smith	United Kingdom
3	David Doe	United States

The following SQL statement lists the number of customers in each country but only includes countries with more than 1 customer:

```
SELECT COUNT(customerid), region FROM customers GROUP BY region HAVING COUNT(customerid) > 1
```

Common SQL Functions

Other database specific functions

So far we've covered functions that are generally supported by most database systems. Again, all databases usually have a subset of features that are proprietary to their systems. Proprietary functions allow you to work with dates and times, math, string manipulation, and much more.

As an example, suppose you wanted to find all tickets submitted within a particular date range. The SQL Server DATEADD function is one of several functions you could use:

```
SELECT * FROM tickets WHERE submitteddate > DATEADD(m, -1, Date())
```

Assuming the current date is 5/30/2018, all submitted help desk tickets beginning one month prior to today's date and up to today's date would be returned.

Common SQL Functions

Other database specific functions

Depending on the database that you choose, I encourage you to seek out that particular database's documentation to get an idea as to what additional functionality is supported outside of the generic functionality covered in this course.