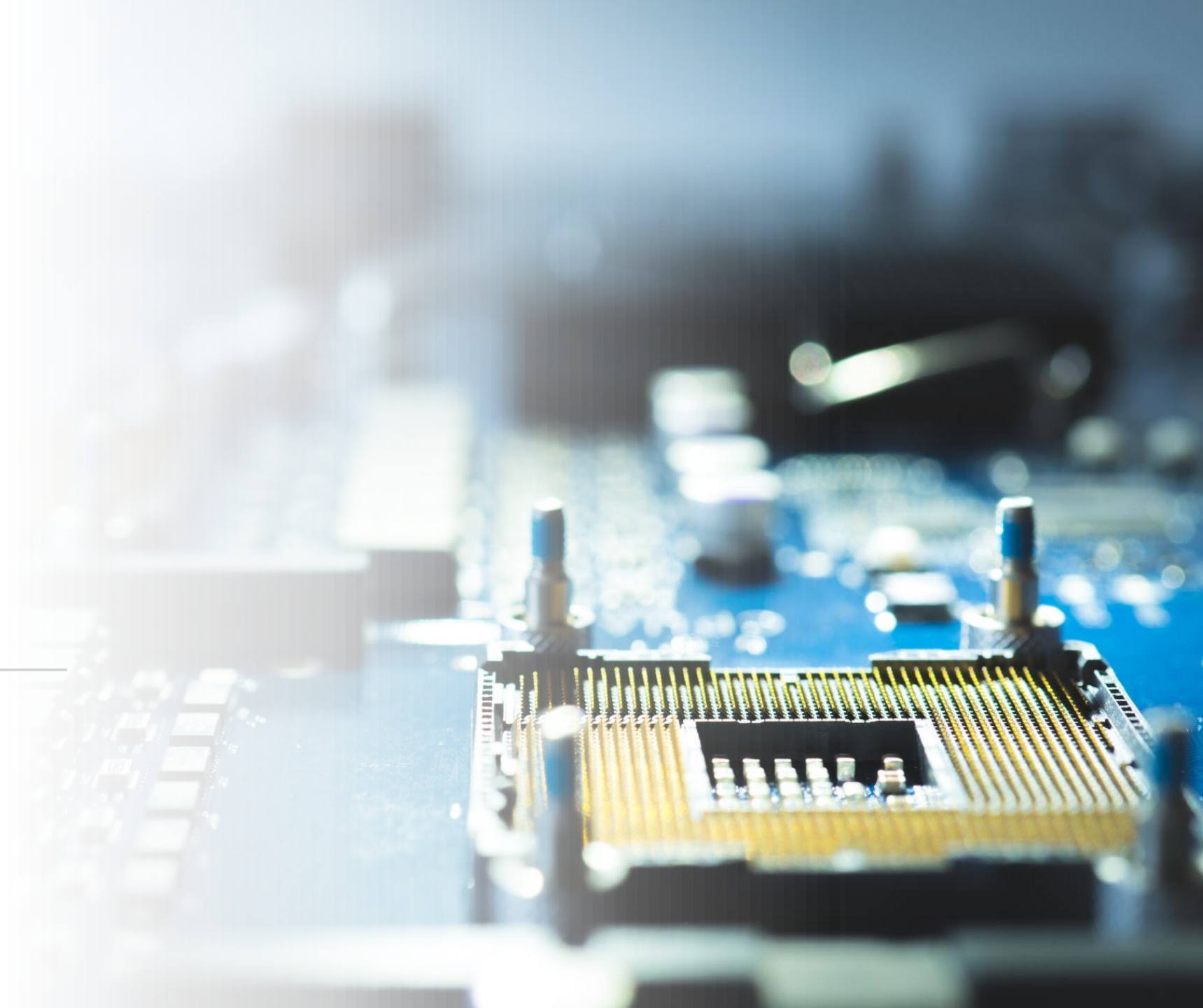




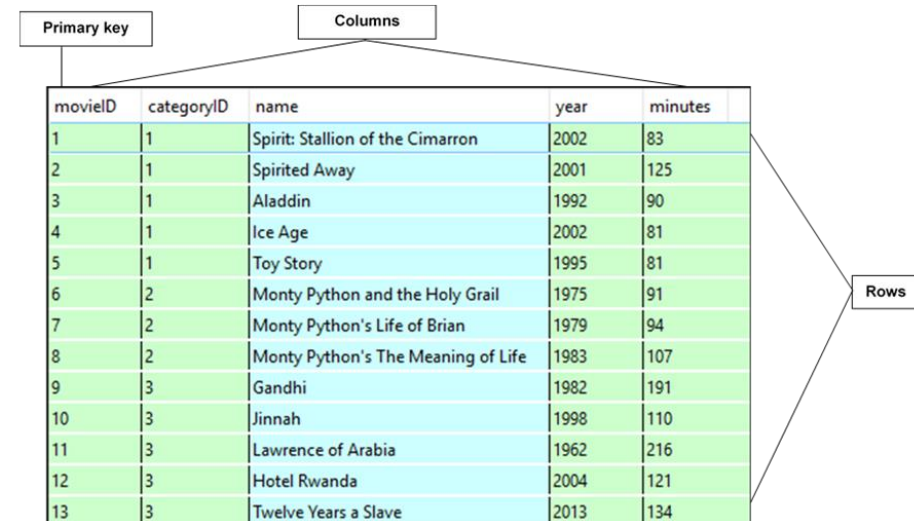
Database Components



Database Components

- A *relational database* consists of *tables*. Tables consist of *rows* and *columns*, which can also be referred to as *records* and *fields*.
- A table is typically modeled after a real-world entity, such as a product or customer, but it can also be modeled after an abstract concept, such as the data for a game.
- A column represents an attribute of the entity, such as a movie's name.
- A row contains a set of values for one instance of the entity, such as one movie.
- Most tables have a *primary key* that uniquely identifies each row in the table.
- The primary key is usually a single column, but it can also consist of two or more columns.

The Movie table

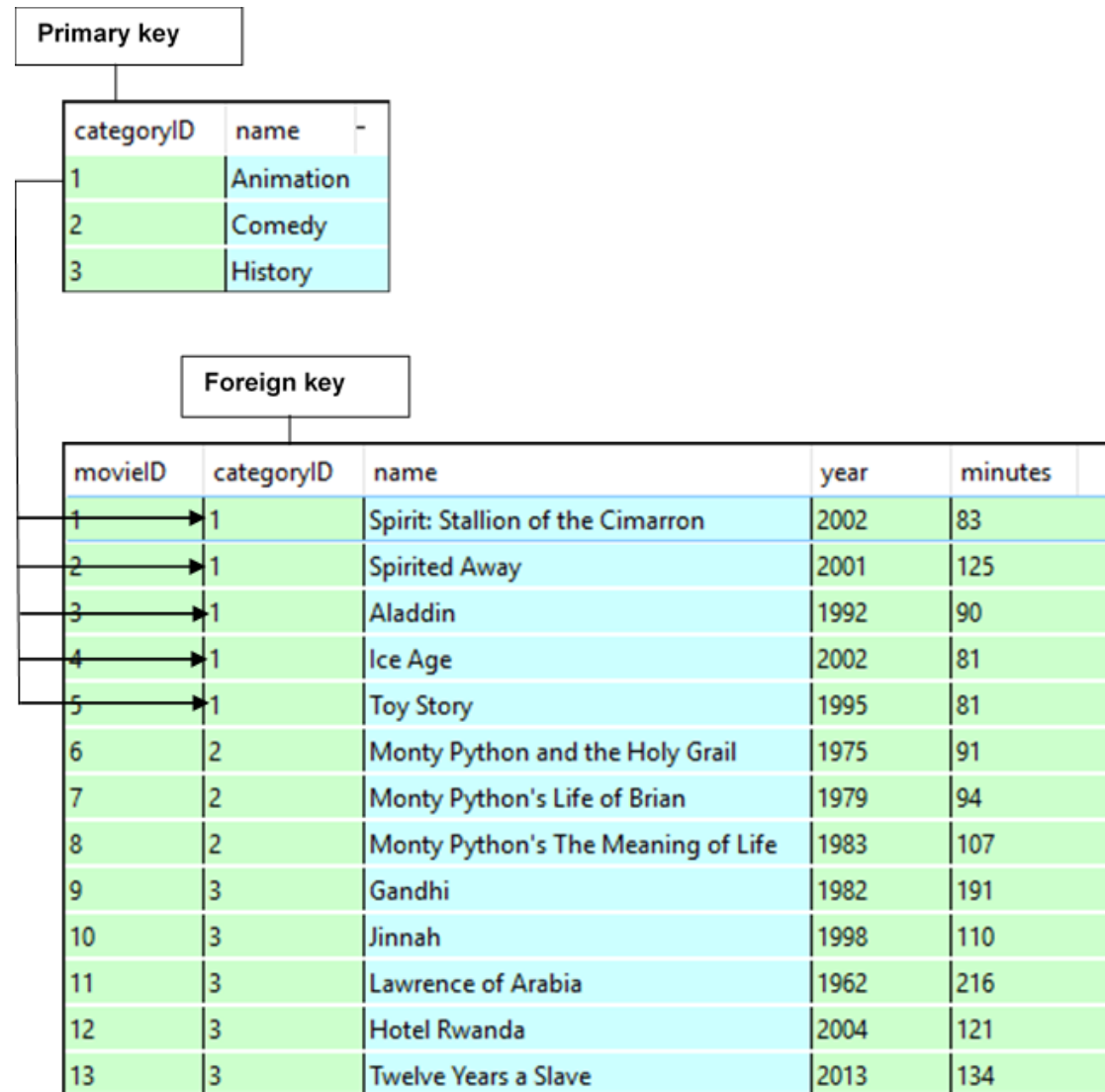


The diagram illustrates the structure of the 'Movie table'. It features a table with five columns: 'movieID', 'categoryID', 'name', 'year', and 'minutes'. The 'movieID' column is designated as the 'Primary key' by a label with a line pointing to it. A 'Columns' label is positioned above the table header. A 'Rows' label is placed to the right of the table, with a bracket indicating the vertical data entries. The table contains 13 rows of data, each representing a movie.

movieID	categoryID	name	year	minutes
1	1	Spirit: Stallion of the Cimarron	2002	83
2	1	Spirited Away	2001	125
3	1	Aladdin	1992	90
4	1	Ice Age	2002	81
5	1	Toy Story	1995	81
6	2	Monty Python and the Holy Grail	1975	91
7	2	Monty Python's Life of Brian	1979	94
8	2	Monty Python's The Meaning of Life	1983	107
9	3	Gandhi	1982	191
10	3	Jinnah	1998	110
11	3	Lawrence of Arabia	1962	216
12	3	Hotel Rwanda	2004	121
13	3	Twelve Years a Slave	2013	134

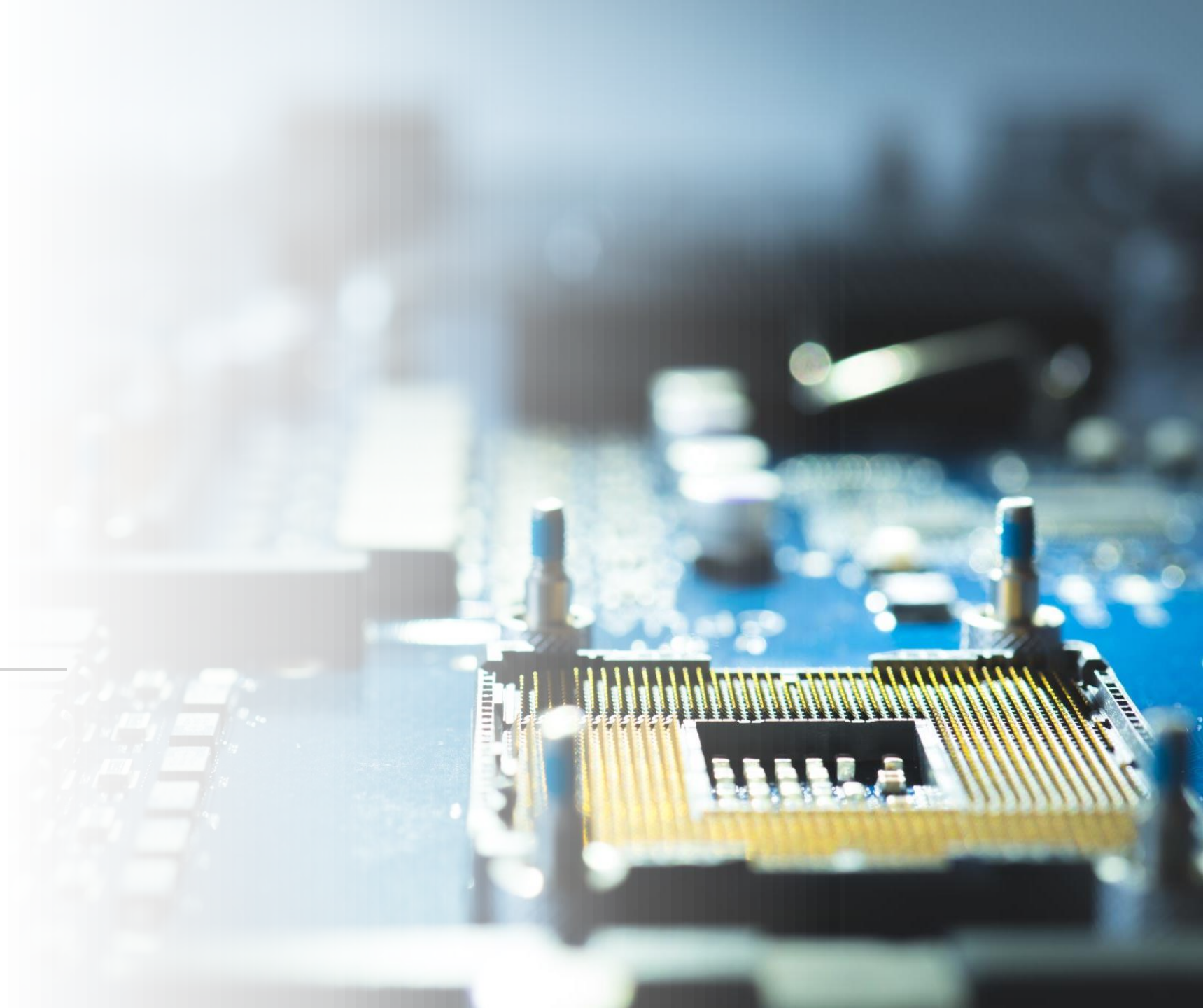
How the tables in a relational database are related

- The tables in a relational database are related to each other through their key columns. For example, the Category and Movie tables in this figure use the categoryID column to create the relationship between the two tables.
- The categoryID column in the Movie table is called a *foreign key* because it identifies a related row in the Category table. A table may contain one or more foreign keys.
- When you define a foreign key, you can't add rows to the table with the foreign key unless there's a matching primary key in the related table.
- The relationships between the tables in a database correspond to the relationships between the entities they represent. The most common type of relationship is a *one-to-many relationship* as illustrated by the Category and Movie tables.
- A table can also have a *one-to-one relationship* or a *many-to-many relationship* with another table.





Creating a Table



Creating a Table

The WHERE clause

The CREATE TABLE statement is used to create a new table in a database.

```
CREATE TABLE student (id INT, name VARCHAR(255))
```

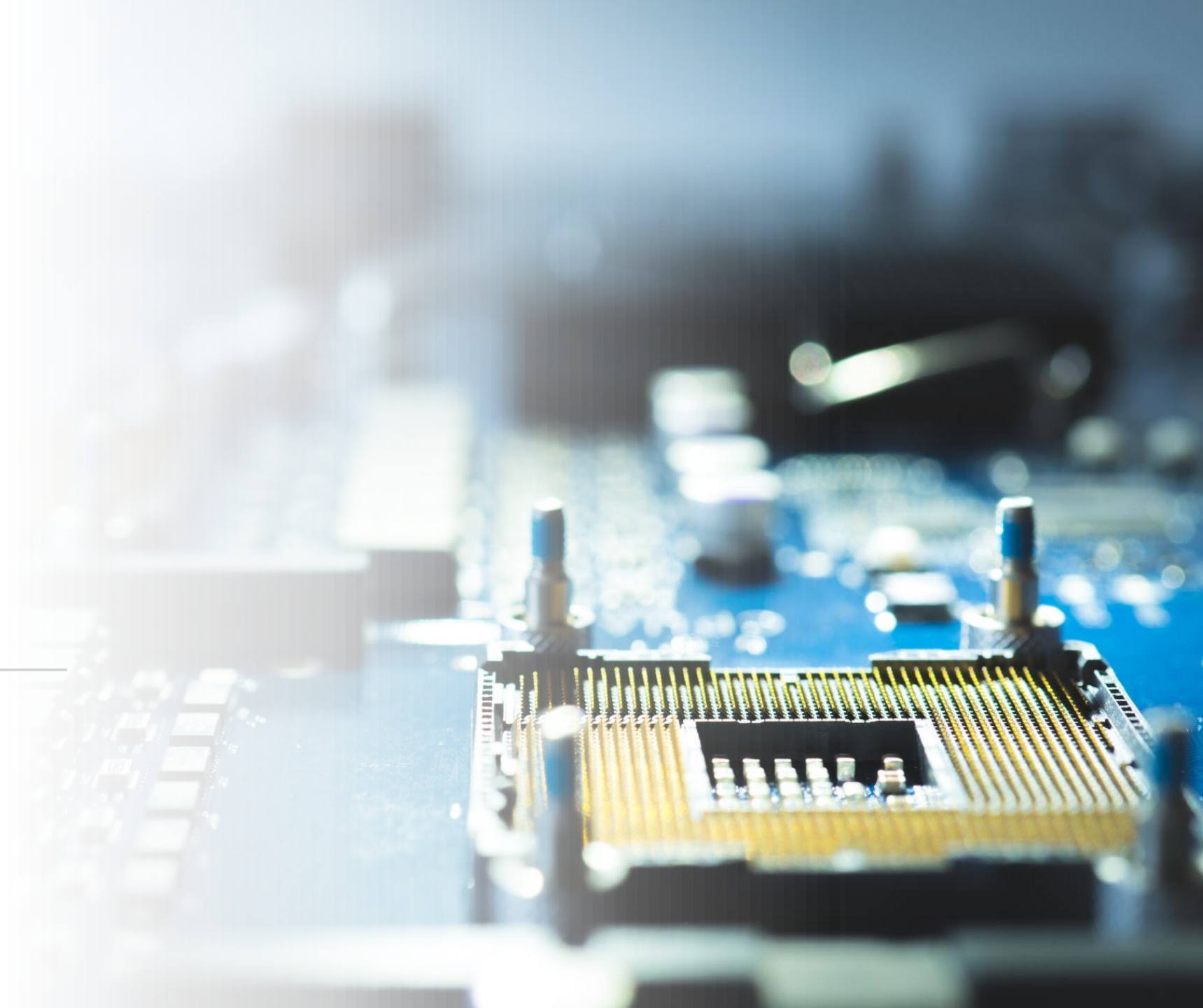
This statement uses the WHERE clause to limit the result returned to rows where the employee’s name matches 'Wally'. The following would be shown (assuming only one employee had that name):

employeeid	name	username	password	email	roleid
1	Wally	wwebmaster	password	wally@vectacorp.com	1

It’s important to note that the selection is made only when a certain criteria is true. If a record with the name of 'Wally' did not exist, it wouldn’t return anything.



Inserting Data



The INSERT INTO Command

Introduction to the INSERT INTO command

Before we get ahead of ourselves with the tickets table, let’s practice inserting employees into the employees table. If you can recall, the employees table resembles the following structure:

Field Name	Data Type
employeeid	int
name	varchar
username	varchar
password	varchar
email	varchar
roleid	int

When writing an INSERT statement we would want to include all of the fields in our statement and also be mindful of the data types that each field accepts. If you tried inserting a string of text into roleid for instance, an error would be produced.

The INSERT INTO Command

Introduction to the INSERT INTO command

Considering the field names and data types within the employees table, our INSERT statement might be written as follows:

```
INSERT INTO employees (name, username, password, email, roleid)
VALUES ('Zak', 'zruvalcaba', 'abc123', 'zruvalca@sdccd.edu', 1)
```

Notice that employeeid is missing from this statement. Again, this is the primary key and since it's an auto incrementing key, it is excluded from the statement.

The INSERT INTO Command

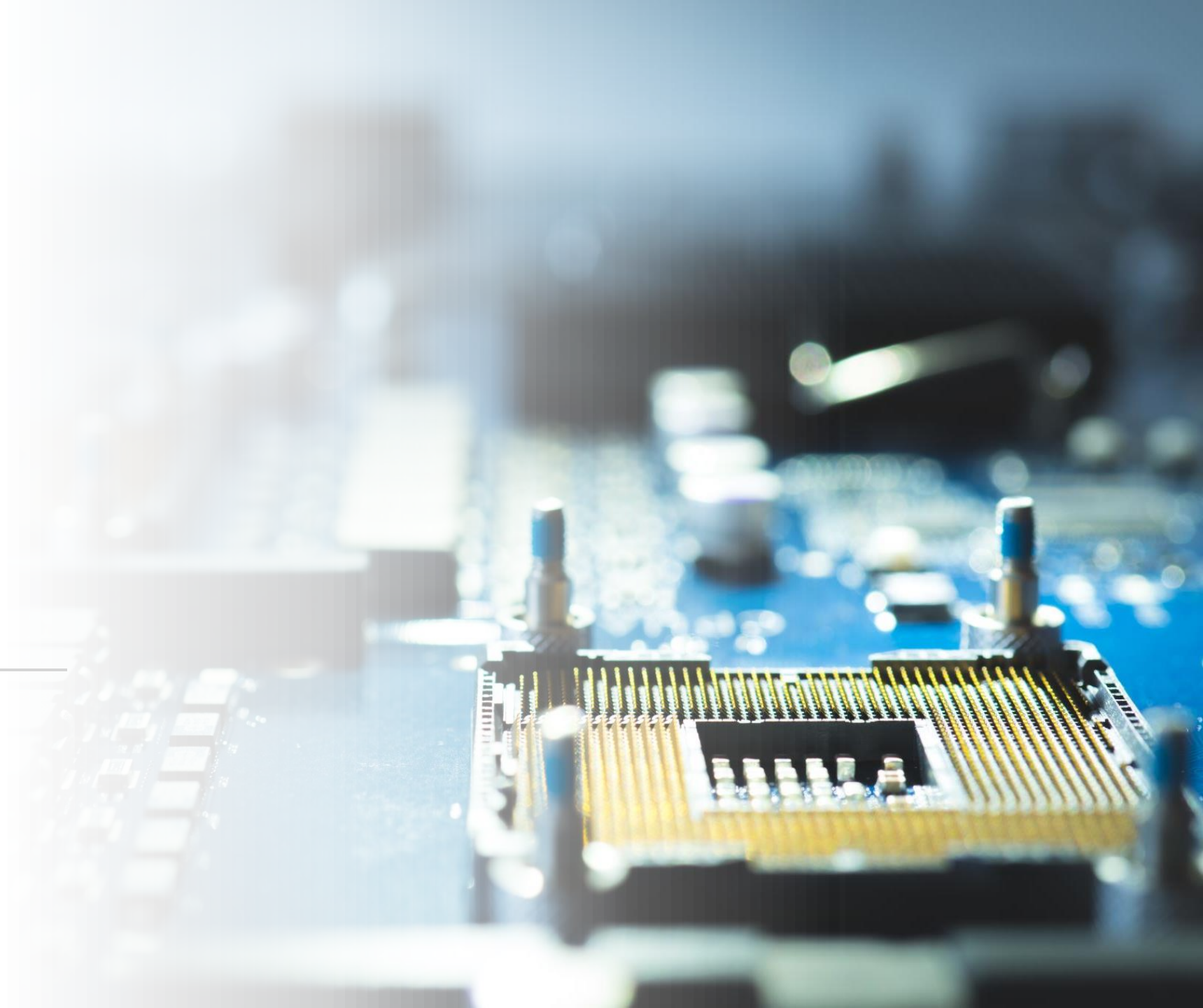
Introduction to the INSERT INTO command

The preceding statement inserts all the values you specified into the proper columns within the employees table. The INSERT statement generally uses the following elements:

- ❖ **INSERT** - The INSERT keyword is used to identify the statement or action you are attempting to perform on the database.
- ❖ **INTO** - The INTO keyword specifies that you are inserting something into a specific table.
- ❖ **Table name** - The name of the table into which you want to insert the values.
- ❖ **Fields** – Each field for which you are inserting a value into is listed. Note that each field name is comma delimited and is almost always listed in the order as it appears within the database table.
- ❖ **VALUES** - The actual values to be inserted. The values are also comma delimited and must be in the same order as the fields are listed.



Selecting Data



Selecting Data

- **The SELECT statement**
- The SELECT statement is used to select data from a database
- `SELECT column1, column2, ...
FROM table_name;`

Column1, column2 are the field names to select data from. To select all the fields use:

- `SELECT * FROM table_name;`

	AlbumId	Title	ArtistId
	Filter	Filter	Filter
1	1	For Those About To Rock We Salute ...	1
2	2	Balls to the Wall	2
3	3	Restless and Wild	2
4	4	Let There Be Rock	1
5	5	Big Ones	3
6	6	Jagged Little Pill	4
7	7	Facelift	5
8	8	Warner 25 Anos	6
9	9	Plays Metallica By Four Cellos	7
10	10	Audioslave	8

Filtering and Ordering Data

The WHERE clause

The most common SQL clause is the WHERE clause. The WHERE clause is used in conjunction with the SELECT statement to deliver a more refined search based on individual field criteria. This example could be used to extract a specific employee based on a first name:

```
SELECT * FROM employees WHERE name = 'Wally'
```

This statement uses the WHERE clause to limit the result returned to rows where the employee's name matches 'Wally'. The following would be shown (assuming only one employee had that name):

employeeid	name	username	password	email	roleid
1	Wally	wwebmaster	password	wally@vectacorp.com	1

It's important to note that the selection is made only when a certain criteria is true. If a record with the name of 'Wally' did not exist, it wouldn't return anything.

Filtering and Ordering Data

Looking for fields with null values

A field with a NULL value is a field with no value. If a field in a table is optional, it is possible to insert a new record or update a record without adding a value to this field. When this is the case, the field will be saved with a NULL value. Consider the following data:

employeeid	name	username	password	email	roleid
1	Wally	wwebmaster	password	wally@vectacorp.com	1
2	Wilbur	wfounder	password	wilbur@vectacorp.com	2
3	Tina	ttechie		tina@vectacorp.com	1

We could query all fields with null values as follows:

```
SELECT name FROM employees WHERE password IS NULL
```

In this case, Tina's record is returned.

Filtering and Ordering Data

The ORDER BY clause

The ORDER BY clause provides you with a quick way of sorting the results of your query in either ascending or descending order. Consider the following table of information:

employeeid	name	username	password	email	roleid
1	Wally	wwebmaster	password	wally@vectacorp.com	1
2	Wilbur	wfounder	password	wilbur@vectacorp.com	2
3	Tina	ttechie	abc123	tina@vectacorp.com	1
4	Agnes	aaccountant	12345	agnes@vectacorp.com	2
5	Damon	ddeveloper	ispeakbinary	damon@vectacorp.com	1

If you selected all the records by using a SELECT all statement (SELECT *), it would return all the results, ordering them based on the value in the employeeid field: 1 through 5.

Filtering and Ordering Data

The ORDER BY clause

Using the SELECT command with an ORDER BY clause allows you to sort based on a different field name:

```
SELECT * FROM employees ORDER BY name
```

The preceding statement would return results, alphabetized in ascending order by name. You can also order by multiple columns by adding a comma after the field name and entering a second field name:

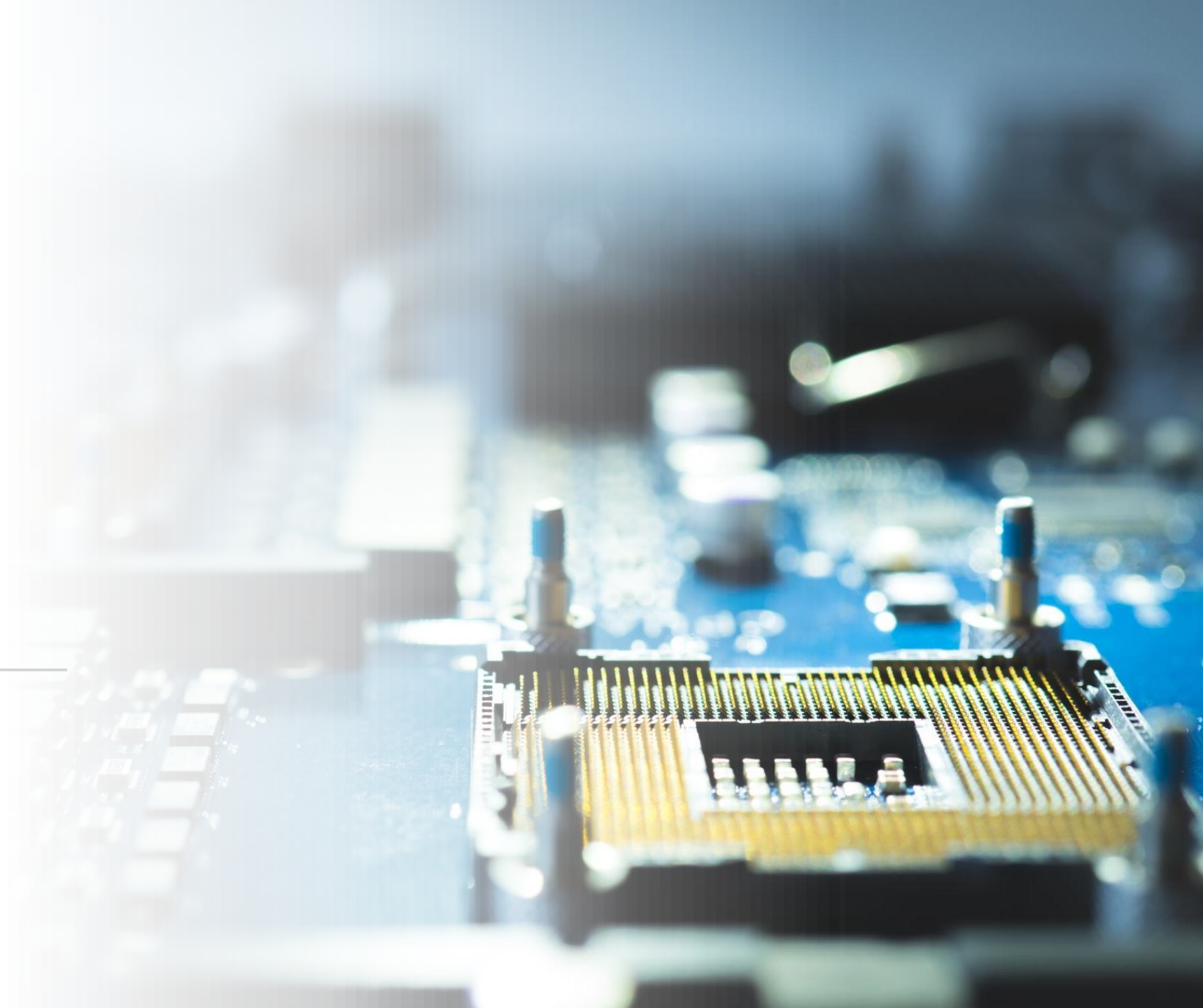
```
SELECT * FROM employees ORDER BY name, email
```

In this case, all records with identical name fields are sorted next by email. You can also adjust the ordering of the list. By default, results are returned in ascending (ASC) order. If you wanted the result in descending order, it would be written as follows:

```
SELECT * FROM employees ORDER BY name DESC
```



Joining Tables



Joining Related Tables

Introduction to SQL Joins

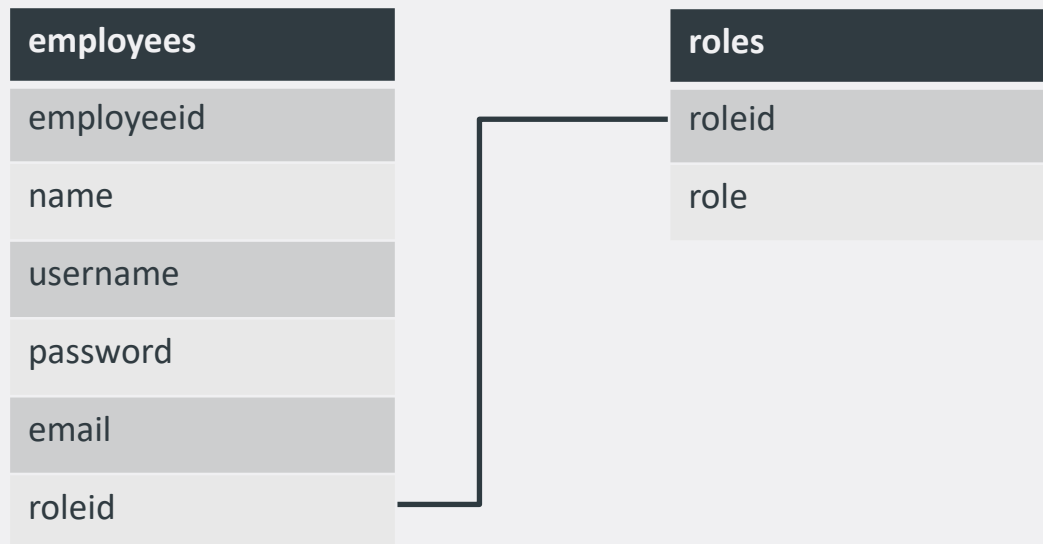
Up to this point, we've focused primarily on extracting data from a single table. Depending on how advanced your database becomes, at times you might want to extract data from multiple tables at once. If that's the case, you will need to use joins. Although there are several types of joins, two types will be covered here:

- ❖ INNER JOIN
- ❖ OUTER JOIN
- ❖ LEFT JOIN (not covered)
- ❖ RIGHT JOIN (not covered)
- ❖ FULL JOIN (not covered)
- ❖ SELF JOIN (not covered)
- ❖ UNION (not covered)

Joining Related Tables

Introduction to INNER JOIN

Of the different types of joins, the inner join is by far the most popular. An Inner join enables you to see all the records of two tables that have relationships established to one another within a single query. Remember that the employees and the roles tables have an established one-to-many relationship. The two tables in the database resemble the following:



Joining Related Tables

Introduction to INNER JOIN

Assume that you wanted to extract the information from the employees table for employeeid 2. Your SELECT statement would resemble the following:

```
SELECT * FROM employees WHERE employeeid = 2
```

This statement would produce the following result:

employeeid	name	username	password	email	roleid
2	Wilbur	wfounder	password	wilbur@vectacorp.com	2

Although the results are shown just fine, the issue resides within the roleid field. What exactly does 2 mean? The numeric value of 2 doesn't mean much to the end user. Instead we would want to extract the actual role of *Support* or *Developer* directly from the roles table. This is where the INNER JOIN command comes in.

Joining Related Tables

Introduction to INNER JOIN

By modifying the SQL statement slightly, we can easily extract data from both the employees and roles tables at the same time:

```
SELECT employees.*, roles.* FROM employees
INNER JOIN roles ON employees.roleid = roles.roleid
```

The preceding SQL statement would produce the following result:

employeeid	name	username	password	email	roleid	role
2	Wilbur	wfounder	password	wilbur@vectacorp.com	2	Support

Because of the INNER JOIN, we are able to effectively query both the employees and roles tables and return the data into a single, virtual result set.

Joining Related Tables

Introduction to INNER JOIN

Notice that the preceding table now becomes more efficient and manageable. Also notice that, rather than referencing the names of the tables, you used the `tableName.fieldname` notation. This is crucial when using joins; otherwise, you would end up with two `employeeids` without a direct reference to its corresponding table.

Also note the use of the `ON` operator in the preceding `INNER JOIN` statement. The `ON` operator instructs the statement to join two tables on a specific primary and foreign key pairing.

Joining Related Tables

Introduction to OUTER JOIN

Outer joins enable rows to be returned from a join in which one of the tables does not contain matching rows for the other table. As an example, suppose the roleid field in the employees table wasn't a required field. For instance, Tina is clearly a support technician and Damon and Wally are developers. The other employees however, may not have roles listed within the employees table. You could end up with data that resembled the following:

employeeid	name	username	password	email	roleid
1	Wally	wwebmaster	password	wally@vectacorp.com	2
2	Wilbur	wfounder	password	wilbur@vectacorp.com	
3	Tina	ttechie	abc123	tina@vectacorp.com	1
4	Agnes	aaccountant	12345	agnes@vectacorp.com	
5	Damon	ddeveloper	ispeakbinary	damon@vectacorp.com	2

Using an INNER JOIN in this scenario would yield inconsistent and unusable data.

Joining Related Tables

Introduction to OUTER JOIN

The OUTER JOIN would be a perfect choice when attempting to join two tables where the foreign key field may not be required. The statement would resemble the following:

```
SELECT employees.*, roles.* FROM employees  
OUTER JOIN roles ON employees.roleid = roles.roleid
```

In this case, all data is returned, even if no role is present for Wilbur and Agnes.