SQL Level 1

Part 2

Pattern Matching

Using SQL Wildcards

Pattern Matching

SQL wildcards can substitute for one or more characters when searching data. There are 2 wildcards:

underscore _ represents exactly one character/number.
Can use multiple times for specific number of characters ___

percent sign % represents one or more characters/numbers (can be none).

SQL Server has some additional wildcards. Use the links below to learn more:

- [ ] Wildcard - Character(s) to Match
- [^] Wildcard - Character(s) Not to Match

Examples

- How many customers have an @gmail.com address?
- How many user's name starts with a B?

Wildcard Example

```
WHERE name LIKE '___ty'
```

marty 

Doherty 

Betty 

Case Sensitivity in PostgreSQL

In PostgreSQL:

- **LIKE** is case sensitive (case matters)
- **ILIKE** is case insensitive (case does not matter, it's ignored)

PostgreSQL Wildcard Example

```
WHERE name LIKE 'B%'
```

Brian 

bob 

PostgreSQL Wildcard Example

```
WHERE name ILIKE 'B%'
```

Brian 

bob 

Case Sensitivity in SQL Server

SQL Server is case insensitive by default.
LIKE does not indicate case sensitivity, it's the database column itself.

To see a column's case sensitivity:

1. In the **Object Explorer**, Right–click a column name & choose **Properties**.
2. In **Extended Properties**, look at **Collation**:

- The CI in **SQL_Latin1_General_CP1_CI_AS** means it's case insensitive
- The CS in **SQL_Latin1_General_CP1_CS_AS** means it's case sensitive

SQL Server Wildcard Example

```
WHERE name LIKE 'B%'
```

Brian 

bob 

SQL Server Wildcard Example

```
WHERE name LIKE 'B%'
COLLATE SQL_Latin1_General_CP1_CS_AS
```

Brian 

bob 

Combining Filters

Logical Operators

Logical operators are keywords used to connect multiple search conditions in a WHERE clause.

Operator	Description
AND	Requires both specified conditions are met (true) for a record to be included in the result.
OR	Requires at least one of the specified conditions are met (true) for the record to be included in the result
NOT	Selects rows for the result which do **not** meet the specified criteria.

Logical Operators Examples

```
SELECT * FROM products
WHERE name LIKE 'B%'
AND price < 10;
```

Logical Operators Examples

```
SELECT * FROM products
WHERE name LIKE 'B%'
OR price < 10;
```

Logical Operators Examples

```
SELECT * FROM products
WHERE NOT name LIKE 'B%';
```

Exercise

Open the file “2.0 LIKE and Wildcards.sql” (in SQL Level 1 folder)

Database Terminology

We won’t get into database design, but there are some key concepts you should understand.

Understanding Relationships

Even if you never create a database and tables (you'll likely only work on ones that are set up for you), understanding relationships within a database are essential for building SQL queries.

Primary & Foreign Keys

Identifying Rows & Connecting Tables

Primary Key

- The **primary key** uniquely identifies each row in a table.
- A primary key must be a unique value (the same value is never used on multiple rows).
- A primary key cannot have NULL values.
- A table can have only one primary key (which may consist of single or multiple fields). When multiple fields are used as a primary key, they are called a composite key.

primary key

user_id	name	email
1	John Smith	johnsmith@gmail.com
2	Daisy Jones	daisyjones@yahoo.com
3	Sue Parker	sueparker@outlook.com
4	John Smith	john-smith@yahoo.com

Foreign Key

A **foreign key** links 2 tables together (like a cross-reference).

A **foreign key** in one table, refers to a **primary key** in another table.

ORDERS TABLE

primary key — foreign key

order_id	user_id
1	U3
2	U1
3	U2
4	U2

USERS TABLE

primary key

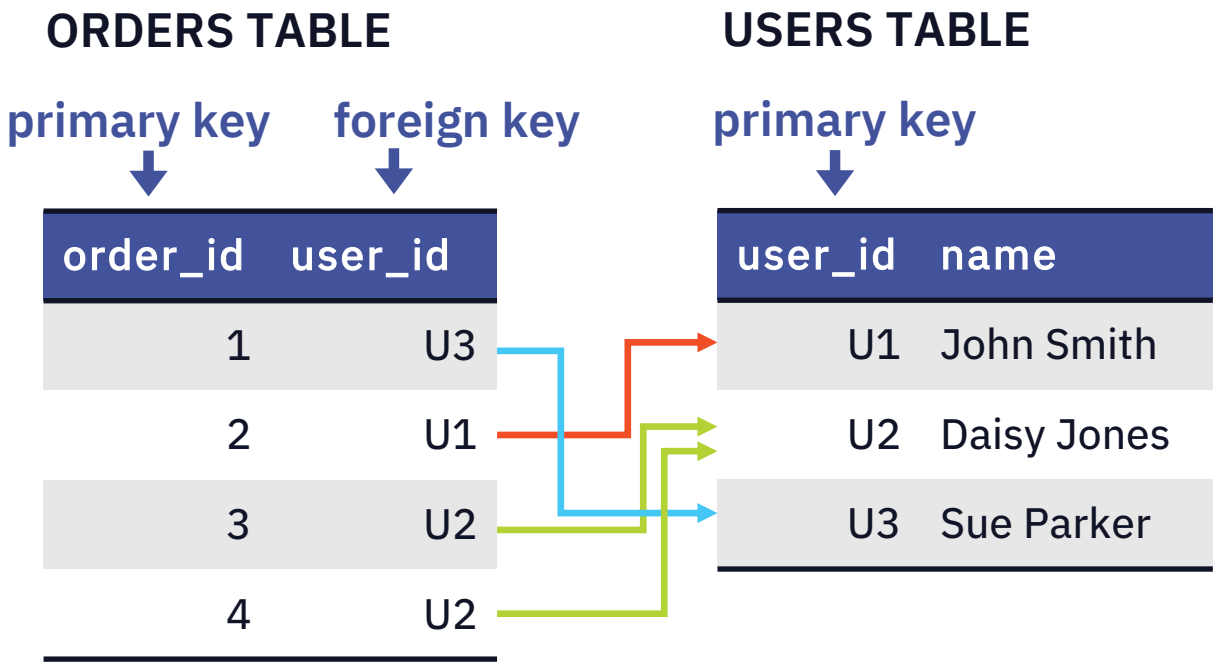
user_id	name
U1	John Smith
U2	Daisy Jones
U3	Sue Parker

Each order is made be a user.
That user's info is stored in a different table.

We connect the order to the user's info using foreign and primary keys.

Foreign Key

A **foreign key** value can be used multiple times in that column.

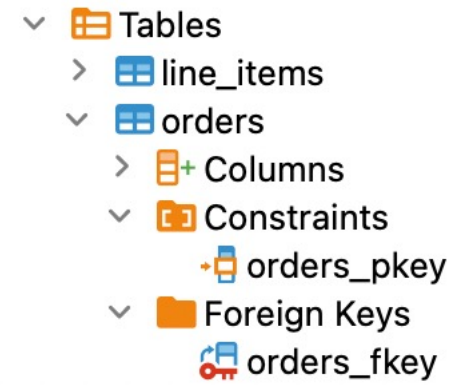


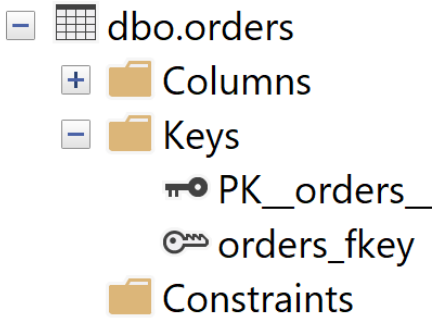
Each order has a unique order_id (primary key), that is associated with a user_id (foreign key).

While each order is unique, the same user can place multiple orders. So the user_id may be repeated across different order_ids (because it refers to the same user many times).

Rules for Foreign Keys

- There can be multiple foreign keys in a table.
- NULL values are allowed in a foreign key.
  Example: When a user_id is deleted, rows for orders referring to that user_id would be set to NULL.
- The relationship established between two tables is known as a referential integrity constraint. Referential integrity requires that values in a foreign key column must be present in the matching primary key, or they must be NULL.

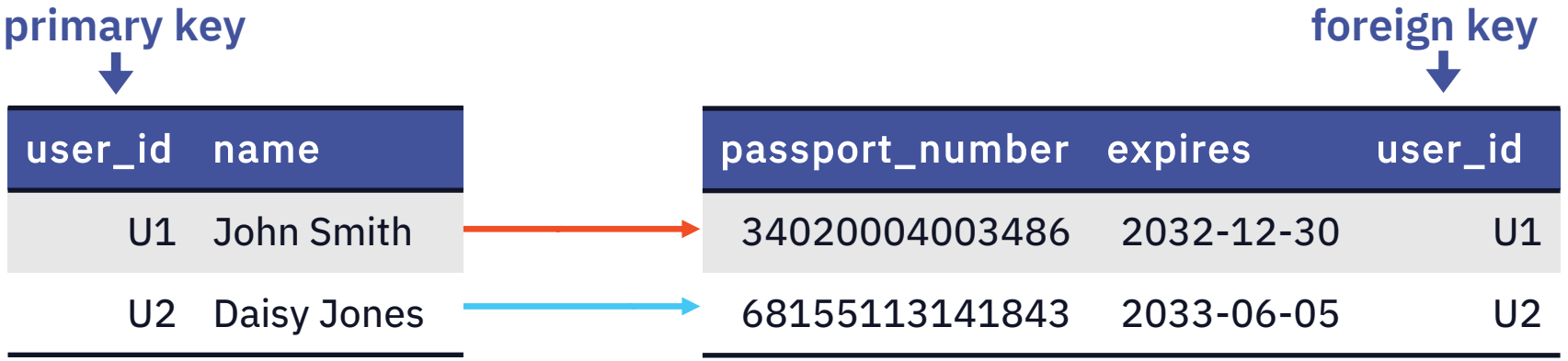




One-to-One Relationship

In a **one-to-one** relationship, one record in a table is associated with only one record in another table.

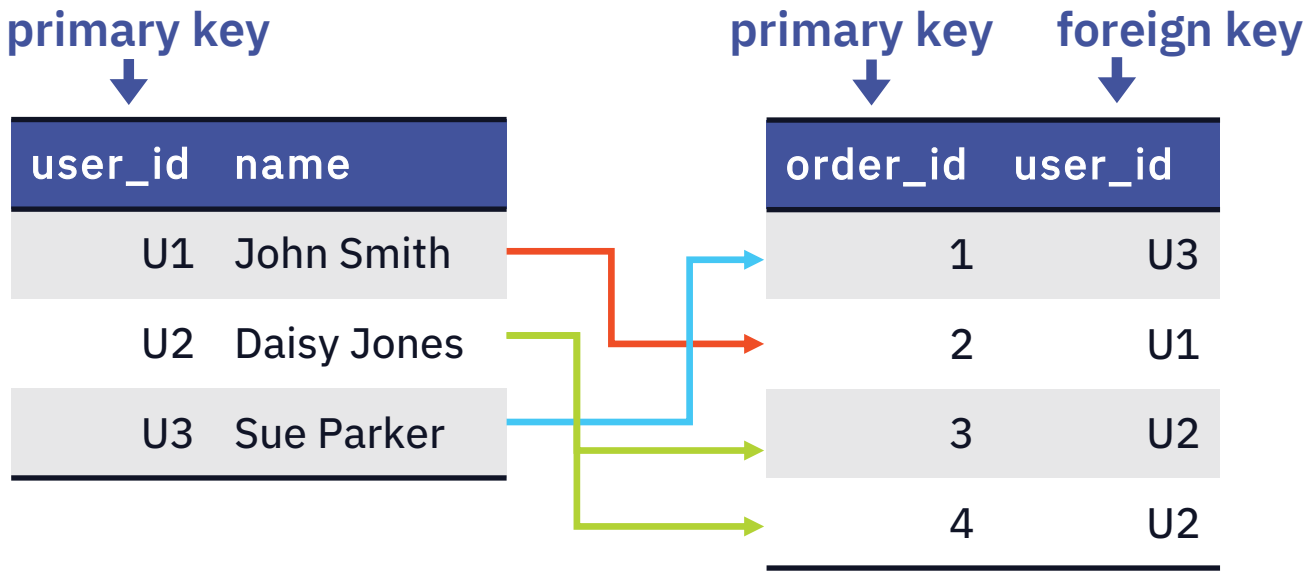
Example: A person has only one passport number. A passport number is given to only one person.



One-to-Many Relationship

In a **one-to-many** relationship, one record in a table can be associated with many records in another table.

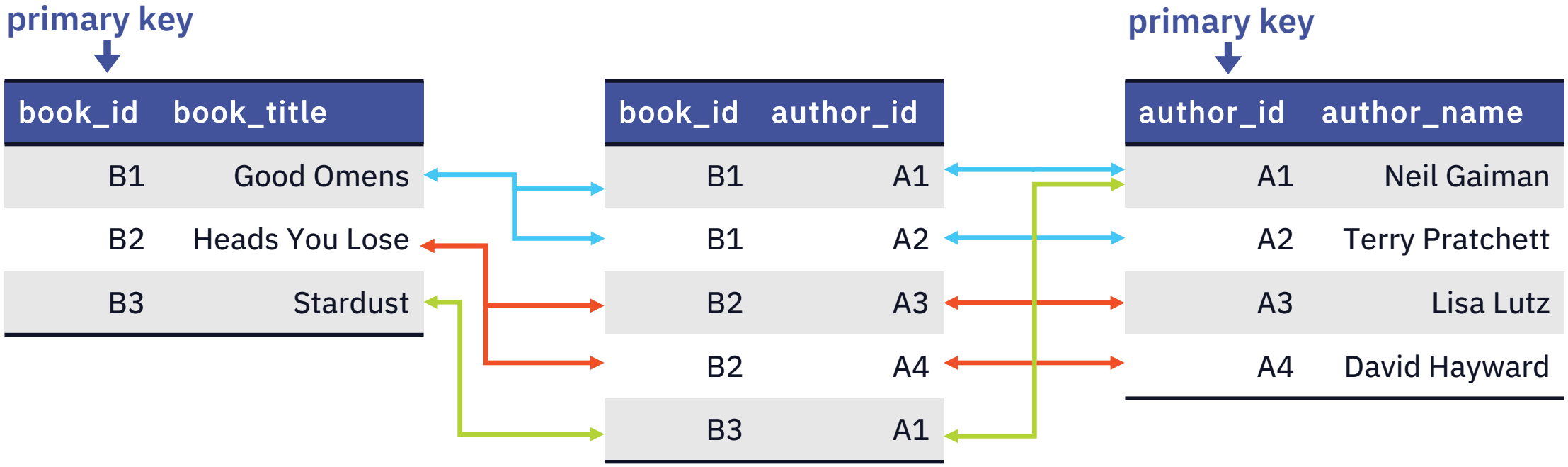
Example: A user can make multiple orders. An order can be made by only one user.



Many-to-Many Relationship

In a **many-to-many** relationship, multiple records in a table are associated with multiple records in another table.

Example: Books can have multiple authors. Authors can write multiple books.



SQL Server: ER Diagrams

An ER (Entity Relation) Diagram is a visual representation of the tables, their columns, and relationships. To create an ER Diagram:

1. In the **Object Explorer,** inside a database Right–click on **Database Diagrams** and choose **Create Database Diagram**.
2. If you get a message “This database does not have one or more support objects... Do you wish to create them?” click **Yes**.
3. Select the tables you want (Shift–click to select a range, Ctrl–click to select specific tables).
4. Click **Add**, then click **Close**.

To change the size of the ER diagram, Right–click in an empty area and use **Zoom**.

You can save the diagram with **File > Save Diagram**. You re-open it by double-clicking the saved diagram in the **Object Explorer’s** folder for **Database Diagrams**.

SQL Server Only Exercise

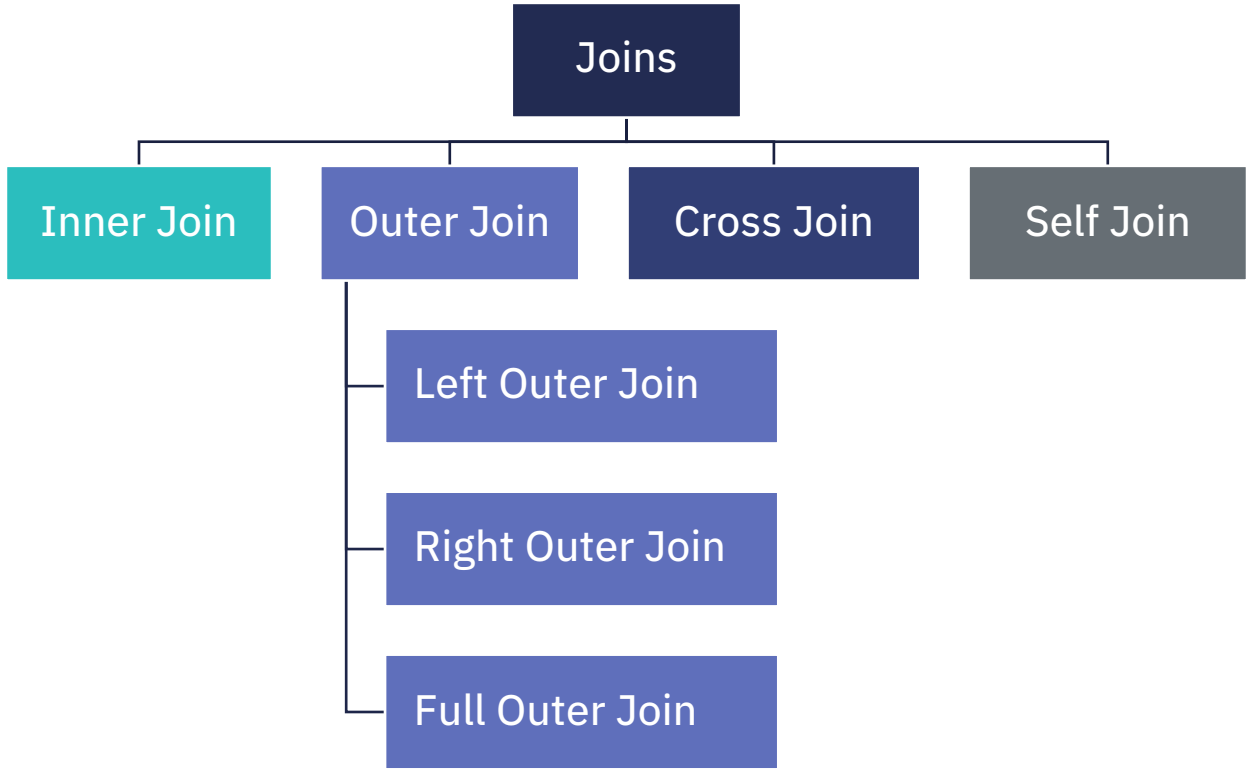
Create an ER Diagram using the instructions on the previous slide

Joins

Combining Data From Multiple Tables

Joins

- A **JOIN** combines data from multiple tables.
- A **JOIN** enables you to use a single **SELECT** statement to query two or more tables simultaneously.
- Usually joins are made on primary and foreign keys.

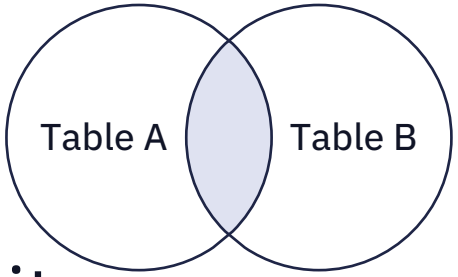


Primary Key Connects to Foreign Key

Rows are matched via the primary key and foreign key (which is a connection between 2 tables).

The primary key of the first table usually shares the same name as the foreign key of the secondary table.

Inner Joins



- An **INNER JOIN** is the most basic type of join, where it matches column values that are common between tables.
- You're matching every instance of a value in a field in the first table, to every instance of that value in the second table.
- If any primary/foreign key is missing, the row is ignored.
- Results may be smaller than either table being joined.

Inner Join Example

```
SELECT * FROM employees
JOIN departments
ON employees.dept_id = departments.dept_id;
```

primary key

foreign key

emp_id	emp_name	dept_id	manager_id
E1	Amy Schumm	D1	E5
E2	Riley Heaney	D3	E5
E3	Tyler Monahan	D2	E6
E4	Chad Schmidt		E1
E5	Elma Stanton	D1	E6
E6	Bruce Rice		

primary key

dept_id	dept_name
D1	Sales
D2	HR
D3	IT
D4	Support
D5	Administration

Inner Join Example

primary key

foreign key

emp_id	emp_name	dept_id	manager_id
E1	Amy Schumm	D1	E5
E2	Riley Heaney	D3	E5
E3	Tyler Monahan	D2	E6
E4	Chad Schmidt	NULL	E1
E5	Elma Stanton	D1	E6
E6	Bruce Rice	NULL	NULL

primary key

dept_id	dept_name
D1	Sales
D2	HR
D3	IT
D4	Support
D5	Administration

Combine the tables, leaving out rows with **NULL** values

```
SELECT * FROM employees
JOIN departments
ON employees.dept_id = departments.dept_id;
```

emp_id	emp_name	dept_id	manager_id	dept_id	dept_name
E1	Amy Schumm	D1	E5	D1	Sales
E2	Riley Heaney	D3	E5	D3	IT
E3	Tyler Monahan	D2	E6	D2	HR
E5	Elma Stanton	D1	E6	D1	Sales

Aliases in a Join

Aliases are a nickname for a table or column in a query.
Usually, one or two letters to make a query shorter.

```
SELECT * FROM employees AS e
JOIN departments AS d
ON e.dept_id = d.dept_id;
```

Here we use an alias as a short name to keep the code cleaner.
NOTE: **AS** is optional and is often left out.

Using Table Aliases on Column Names

```
SELECT e.emp_name, d.dept_name
FROM employees e
JOIN departments d
ON e.dept_id = d.dept_id;
```

emp_id	emp_name	dept_id	manager_id
E1	Amy Schumm	D1	E5
E2	Riley Heaney	D3	E5
E3	Tyler Monahan	D2	E6
E4	Chad Schmidt		E1
E5	Elma Stanton	D1	E6
E6	Bruce Rice		

dept_id	dept_name
D1	Sales
D2	HR
D3	IT
D4	Support
D5	Administration

Column Aliases

Aliases can also be used to rename a column in the result set.

```
SELECT emp_name AS "Employee Name"
FROM employees;
```

In SQL Server square brackets [ ] can also be used instead of double quotes.

Syntax for Aliases

COLUMN ALIASES:

- `emp_name AS Name`
- `emp_name Name`
- `emp_name AS "Employee Name"`
- `emp_name "Employee Name"`

TABLE ALIASES:

- `employees AS e`
- `employees e`

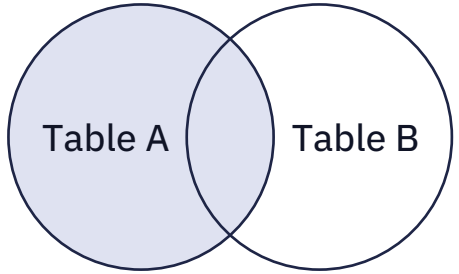
Exercise

Open the file “2.1 Inner Joins and Aliases.sql” (in SQL Level 1 folder)

Inner Joins vs. Outer Joins

- **INNER JOIN**: returns matching rows (data matches in both tables).
- **OUTER JOIN**: returns matching and non-matching rows (missing values appear as NULL).

Outer Left Join



primary key

foreign key

primary key

emp_id	emp_name	dept_id	manager_id
E1	Amy Schumm	D1	E5
E2	Riley Heaney	D3	E5
E3	Tyler Monahan	D2	E6
E4	Chad Schmidt	NULL	E1
E5	Elma Stanton	D1	E6
E6	Bruce Rice	NULL	NULL

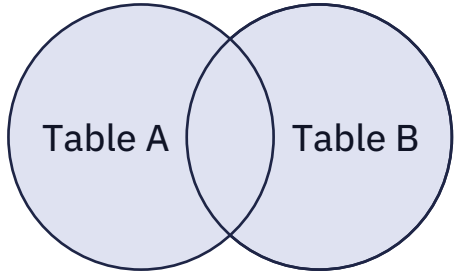
dept_id	dept_name
D1	Sales
D2	HR
D3	IT
D4	Support
D5	Administration

emp_id	emp_name	dept_id	manager_id	dept_id	dept_name
E1	Amy Schumm	D1	E5	D1	Sales
E2	Riley Heaney	D3	E5	D3	IT
E3	Tyler Monahan	D2	E6	D2	HR
E4	Chad Schmidt	NULL	E1	NULL	NULL
E5	Elma Stanton	D1	E6	D1	Sales
E6	Bruce Rice	NULL	NULL	NULL	NULL

```
SELECT * FROM employees e
LEFT JOIN departments d
ON e.dept_id = d.dept_id;
```

Combine the tables, keeping all rows from the left (first) table.

Full Outer Join



emp_id	emp_name	dept_id	manager_id
E1	Amy Schumm	D1	E5
E2	Riley Heaney	D3	E5
E3	Tyler Monahan	D2	E6
E4	Chad Schmidt	NULL	E1
E5	Elma Stanton	D1	E6
E6	Bruce Rice	NULL	NULL

dept_id	dept_name
D1	Sales
D2	HR
D3	IT
D4	Support
D5	Administration

emp_id	emp_name	dept_id	manager_id	dept_id	dept_name
E1	Amy Schumm	D1	E5	D1	Sales
E2	Riley Heaney	D3	E5	D3	IT
E3	Tyler Monahan	D2	E6	D2	HR
E4	Chad Schmidt	NULL	E1	NULL	NULL
E5	Elma Stanton	D1	E6	D1	Sales
E6	Bruce Rice	NULL	NULL	NULL	NULL
NULL	NULL	NULL	NULL	D5	Administration
NULL	NULL	NULL	NULL	D4	Support

```
SELECT * FROM employees e
FULL JOIN departments d
ON e.dept_id = d.dept_id;
```

Combine the tables, keeping all rows from both tables.

Most Common Types of Joins

INNER JOIN:

- Returns rows with values that match in both tables.

LEFT, RIGHT, and FULL OUTER JOINs:

- Returns matching and non-matching rows (non-matching rows have NULL on one side).
- They simply change the side where we keep unmatched rows (the left, right, or both sides).

Other Types of Joins

CROSS JOIN: Returns a paired combination of each row of the first table with each row of the second table.

- Can produce very large results without proper filtering.
- Less commonly used, so we're not going to cover.

In SQL Level 2 we discuss:

SELF JOIN: Matches one part of a table with another part of the same table.

UNION: Combines data across tables to remove duplicates.

Exercise

Open the file “2.2 Outer Joins and NULLS.sql” (in SQL Level 1 folder)