SQL For Testers

Introduction

Importance of Databases and SQL

Database Components

Tables, Schemas, Columns, and Keys

- Constraints and Views
- Stored Procedures and Triggers

Structuring the Database

Introducing Database Normalization

- Advancing Database Normalization
- SQL Data Types

Data Retrieval

Query Execution Process

- Guidelines for Writing Query
- Basic Query Syntax
- Learn Data Retrieval Through Practice

Data Definition

Data Definition Language (DDL)

- Learn DDL Through Practice
- Running the DDL Script

Data Manipulation

Data Manipulation Language (DML)

- Learn DML Through Practice
- Running the DML Script

Transforming Data and Functions

Aggregates and Joins

- Learn Aggregates and Joins Through Practice
- Data Presentation
- Learn Data Presentation Through Practice
- Data Transformation
- Learn Data Transformation Through Practice

Database Administration

Backup, Recovery, and Database Restore

- Learn Backup, Recovery, and Restore Through Practice
- Login, Users, and Roles
- Learn Server Logins, Users, and Roles Through Practice

Understand Normalization Rules

Testing database normalization involves verifying that your database schema adheres to the rules of normalization.

While there isn’t a direct automated tool to check normalization, you can perform manual checks and write scripts to validate your schema. Let’s explore some practical steps:

First Normal Form (1NF):

- Ensure each column contains atomic (indivisible) values.
- Write a script to check if any columns have multiple values (e.g., comma-separated lists).

Second Normal Form (2NF):

- Verify that non-key attributes depend fully on the entire primary key.
- Write a script to identify partial dependencies.

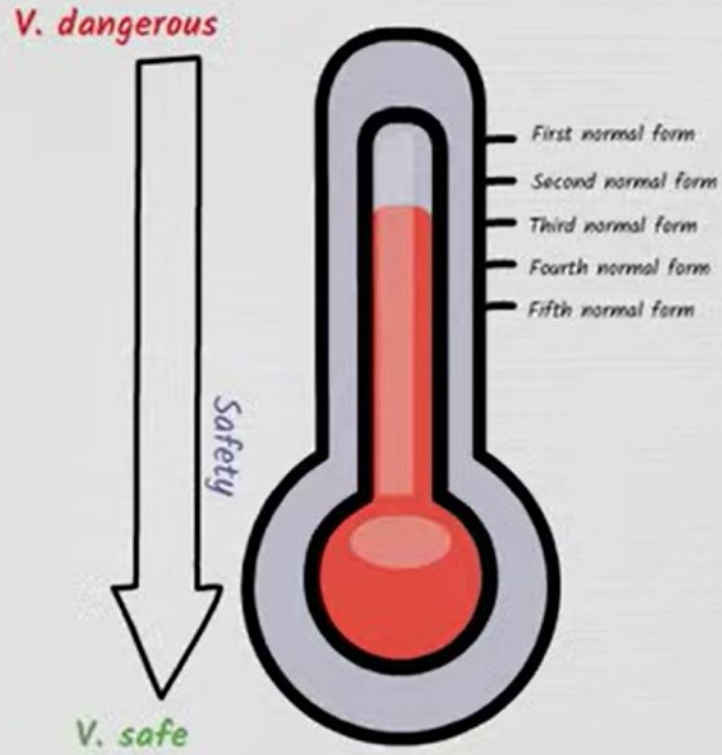
Third Normal Form (3NF):

- Check for transitive dependencies.
- Write a script to find columns that depend on other non-key attributes.
- Run Validation Queries:
- Execute your scripts against your database.

Identify any violations of normalization rules.

Automate Checks (Optional):

The normal forms

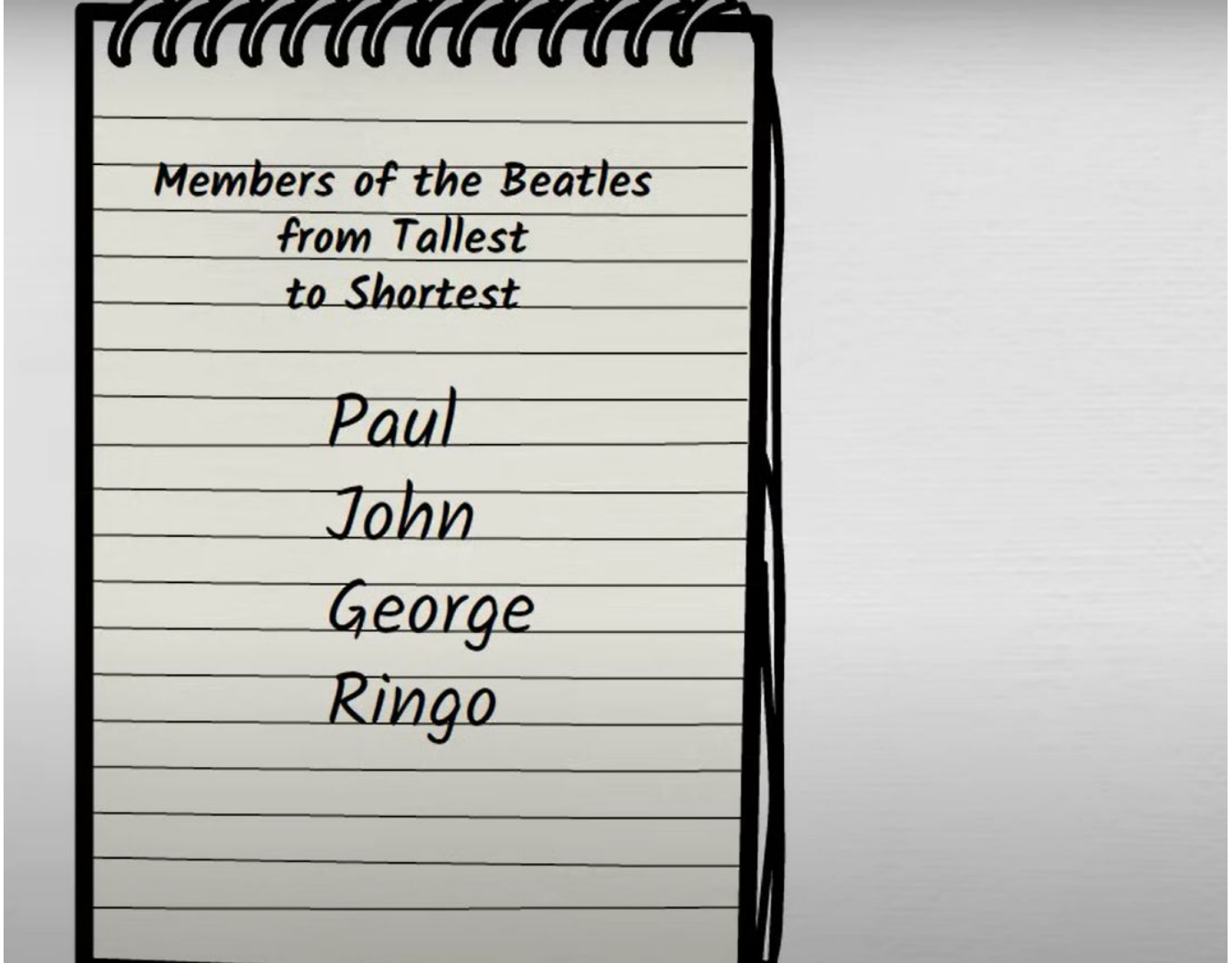


Who were the members of the Beatles?

John, Paul, George, and Ringo

Paul, John, Ringo, and George

(equivalent)


Members of the Beatles
from Tallest
to Shortest
Paul
John
George
Ringo

Using row order to convey information violates INF.

Beatle_Height

Beatle	Height_In_Cm
George	178
John	179
Ringo	170
Paul	180

Beatle_Height

Beatle	Height_In_Cm
George	178
John	179
Ringo	Somewhere between 168 and 171
Paul	180

Beatle_Height

Beatle	Height_In_Cm (integer)
George	178
John	179
Ringo	Somewhere between 168 and 171
Paul	180

Mixing data types within the same column violates INF

(and the DB platform won't let you do it anyway)

We should make this the primary key...

Beatle_Height

Beatle	Height_In_Cm
George	178
John	179
Ringo	170
Paul	180

```
ALTER TABLE Beatle_Height
ADD PRIMARY KEY (Beatle);
```

Making "Beatle" the primary key prevents this:

Beatle	Height_In_Cm
George	178
John	

Player_ID	Inventory
jdog21	2 amulets, 4 rings
gila19	18 copper coins
trev73	3 shields, 5 arrows, 30 copper coins, 7 rings

- Repeating group

Player_Inventory

Player_ID	Quantity_1	Item_Type_1	Quantity_2	Item_Type_2	Quantity_3	Item_Type_3	Quantity_4	Item_Type_4
jdog21	2	amulets	4	rings				
gila19	18	copper coins						
trev73	3	shields	5	arrows	30	copper coins	7	rings

Player_Inventory

Player_ID	Item_Type	Item_Quantity
jdog21	amulets	2
jdog21	rings	4
gila19	copper coins	18
trev73	shields	3
trev73	arrows	5
trev73	copper coins	30
trev73	rings	7

First Normal Form Rules:

1) Using row order to convey information is not permitted

2) Mixing data types within the same column is not permitted

3) Having a table without a primary key is not permitted

4) Repeating groups are not permitted

2nd Normal Form Rules

3rd Normal Form Rules

4th Normal Form Rules

5th Normal Form Rules

Querying data

- SELECT FROM – show you how to use a simple `SELECT FROM` statement to query the data from a single table.
- SELECT – learn how to use the SELECT statement without referencing a table.

Sorting data

- ORDER BY

Filtering data

- WHERE – learn how to use the WHERE clause to filter rows based on specified conditions.
- SELECT DISTINCT – show you how to use the DISTINCT operator in the SELECT statement to eliminate duplicate rows in a result set.
- AND – introduce you to the AND operator to combine Boolean expressions to form a complex condition for filtering data.
- OR– introduce you to the OR operator and show you how to combine the OR operator with the AND operator to filter data.
- IN – show you how to use the IN operator in the WHERE clause to determine if a value matches any value in a set.
- NOT IN – negate the IN operator using the NOT operator to check if a value doesn't match any value in a set.
- BETWEEN – show you how to query data based on a range using the BETWEEN operator.
- LIKE – query database on pattern matching using wildcards such as % and _.
- LIMIT – use LIMIT to limit the number of rows returned by the SELECT statement
- IS NULL – test whether a value is NULL or not by using the IS NULL operator.

Joining tables

- Table & Column Aliases – introduce you to table and column aliases.
- Joins – give you an overview of joins supported in MySQL including inner join, left join, and right join.
- INNER JOIN – query rows from a table that has matching rows in another table.
- LEFT JOIN – return all rows from the left table and matching rows from the right table or null if no matching rows are found in the right table.
- RIGHT JOIN – return all rows from the right table and matching rows from the left table or null if no matching rows are found in the left table.
- Self-join – join a table to itself using a table alias and connect rows within the same table using inner join and left join.
- CROSS JOIN – make a Cartesian product of rows from multiple tables.

WHY JOINS?

- Typically in a relational database, data is organized into various tables made of attributes (columns) and records (rows).
- In each table there exist a column that is the **primary key** which is a column where each entry uniquely represents a single row in that table. This is usually the ID (short for identifier) column.
- A column in a table that establishes an association with another *table's primary key via shared values is called a* **foreign key**. Foreign keys are also typically titled IDs but prepended with the name of the referenced table.
- This concept is applied when combining two or more tables together using a JOIN

lt_id	lt-data1
1	LEFT-DATA1
2	LEFT-DATA2
3	LEFT-DATA3

rt_id	rt_fk	rt_data1
1	1	RIGHT-DATA1
2	1	RIGHT-DATA2
3	4	RIGHT-DATA3
4	2	RIGHT-DATA4

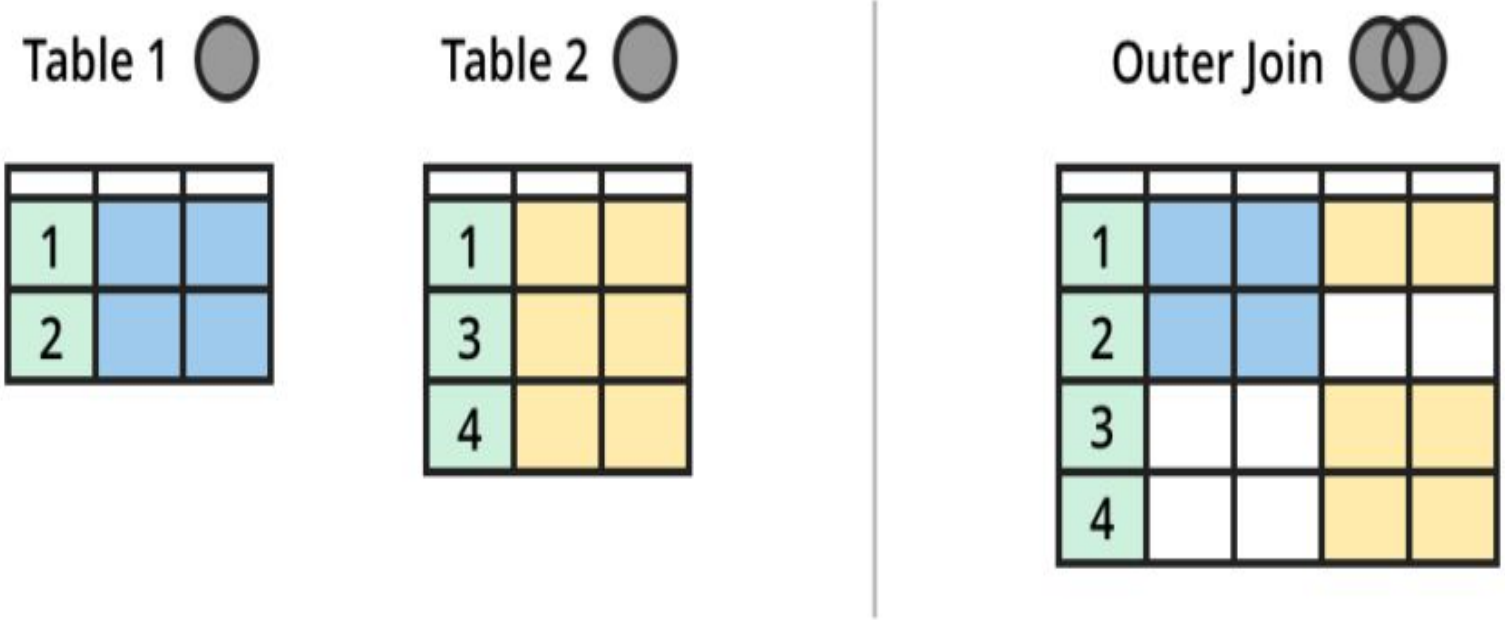
User Table – Table 1

ID (Primary Key)	Name	Address
1	Sally Select	123 Join Dr
2	Frank From	25 Where St

Event Table – Table 2

User_ID (Foreign Key)	ID (Primary Key)	Action
1	A	LOGIN
3	B	VIEW PAGE
4	C	LOGIN

Outer Join



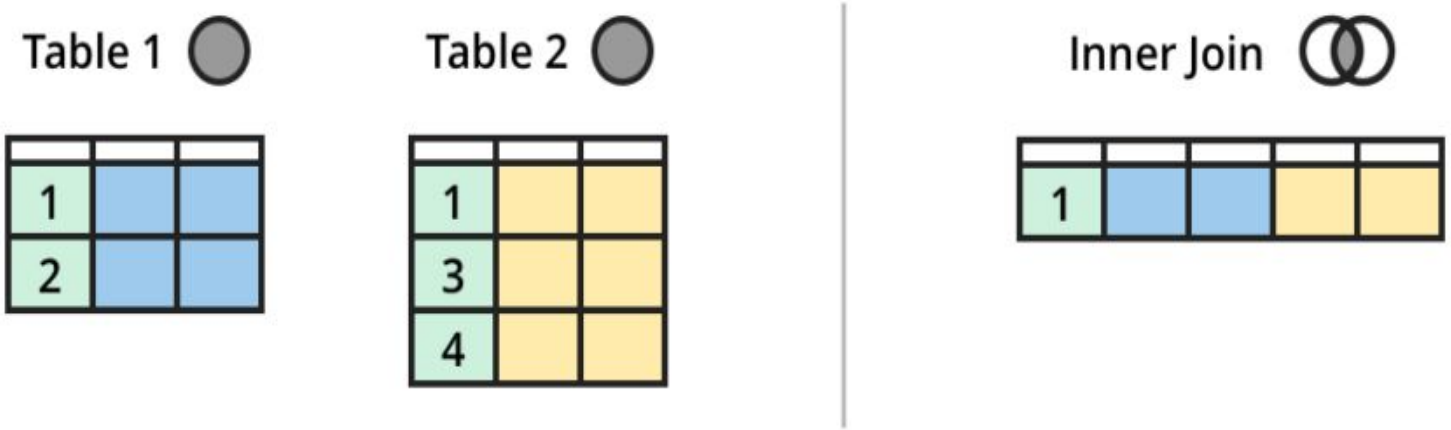
Inner Join

SELECT Orders.OrderID, Customers.CustomerName, Orders.OrderDate

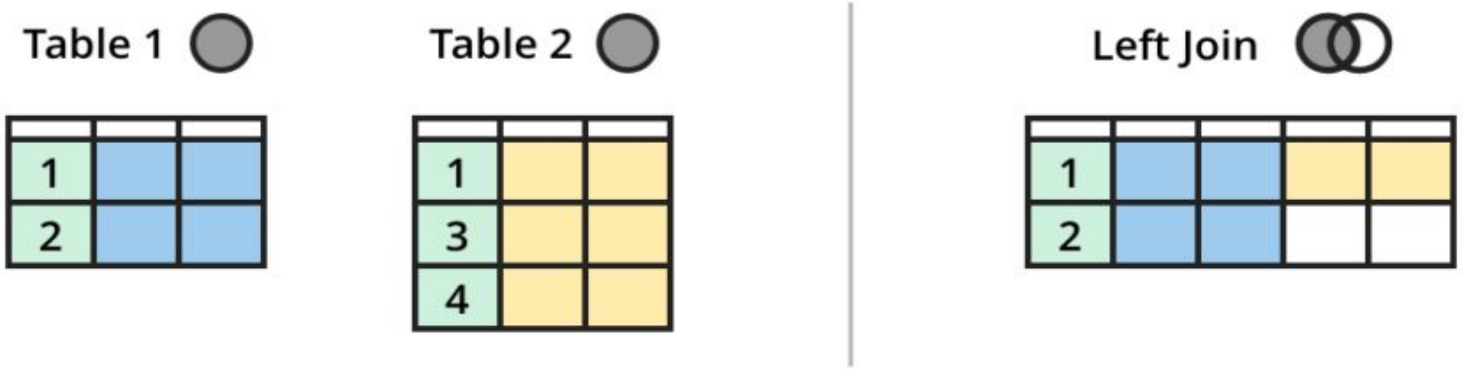
FROM

Orders INNER JOIN Customers ON Orders.CustomerID=Customers.CustomerID;

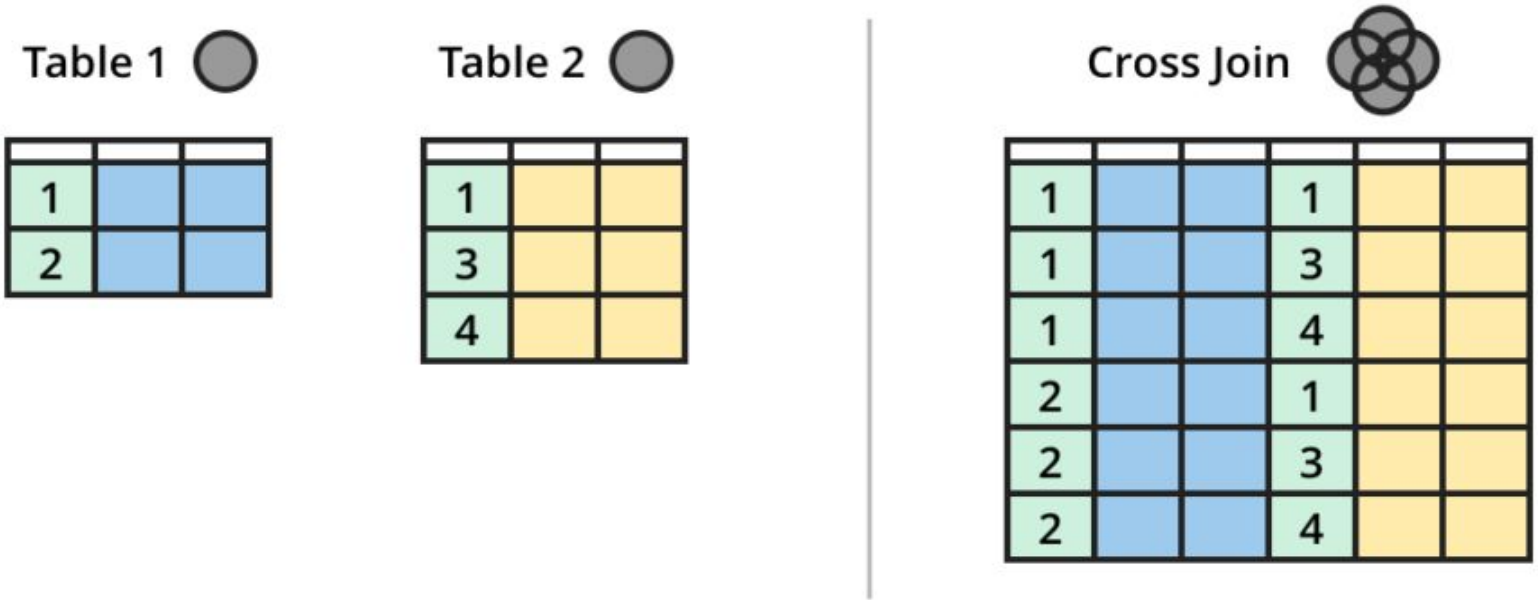
INNER JOIN: This returns records that have matching values in both tables



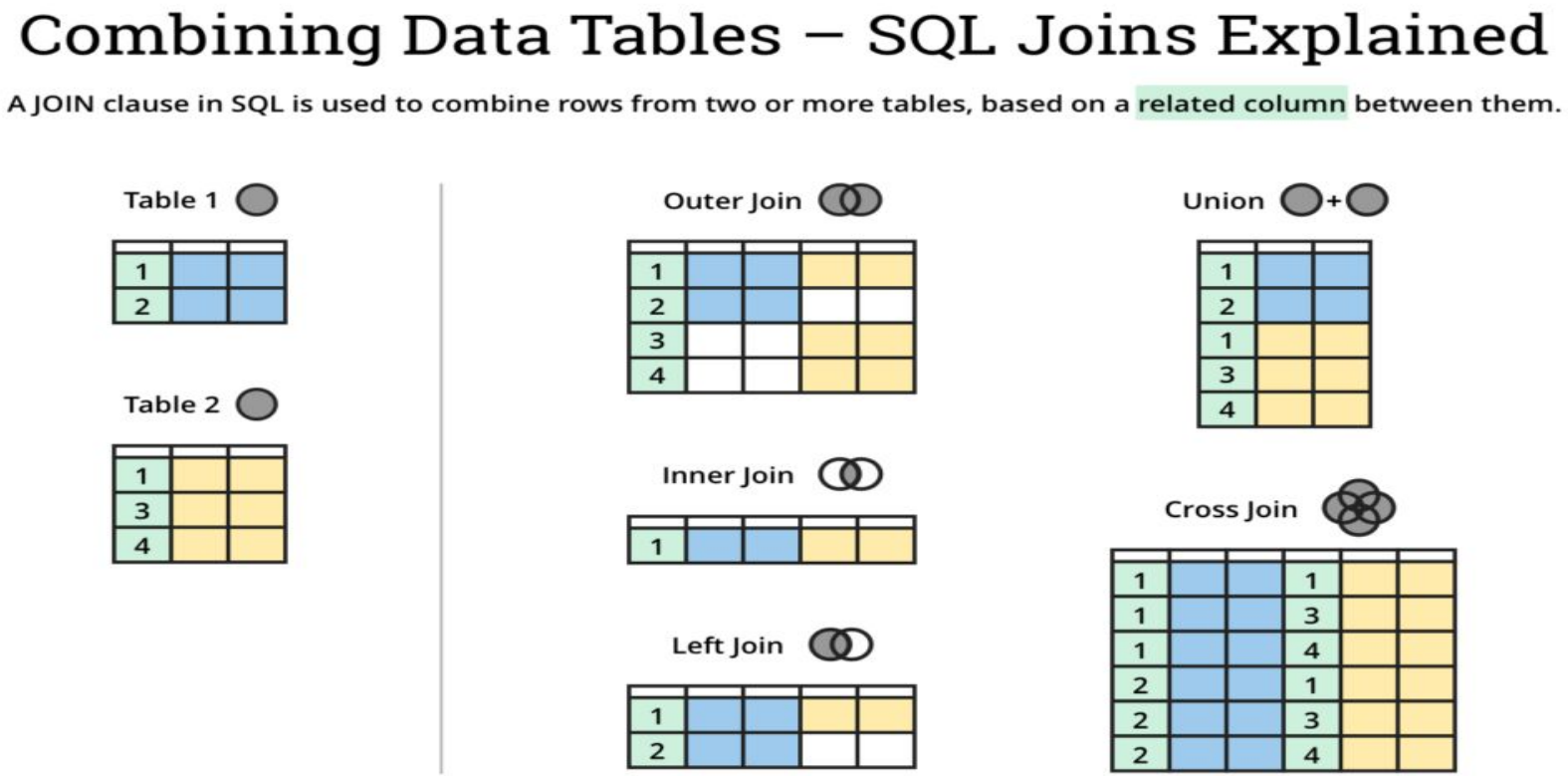
Left Join



Cross Join



Joins Summary



Indexing makes columns faster to query by creating pointers to where data is stored within a database. Imagine you want to find a piece of information that is within a large database. To get this information out of the database the computer will look through every row until it finds it. If the data you are looking for is towards the very end, this query would take a long time to run.

This took 3 comparisons to find the right answer instead of 8 in the unindexed data.

Indexes allow us to create sorted lists without having to create all new sorted tables, which would take up a lot of storage space.

SELECT * FROM friends WHERE name = 'Zack';

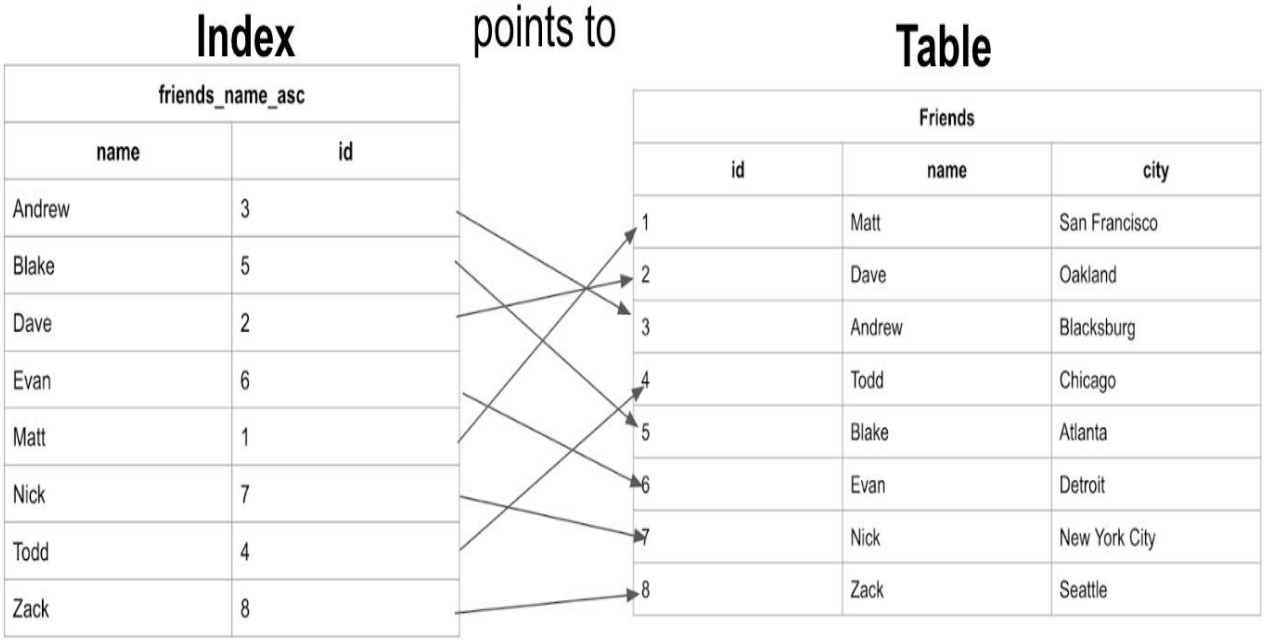
friends		
id	**name**	**city**
1	Matt	San Francisco
2	Dave	Oakland
3	Andrew	Blacksburg
4	Todd	Chicago
5	Blake	Atlanta
6	Evan	Detroit
7	Nick	New York City
8	Zack	Seattle

SELECT * FROM friends WHERE name = 'Zack';

friends_name_asc	
Name	**Index**
Andrew	3
Blake	5
Dave	2
Evan	6
Matt	1
Nick	7
Todd	4
Zack	8

Whats is an Index

An index is a structure that holds the field the index is sorting and a pointer from each record to their corresponding record in the original table where the data is actually stored.



Types of Indexes

There are two types of databases indexes:

Clustered

- Clustered indexes are the unique index per table that uses the primary key to organize the data that is within the table.
- Clustered indexes do not have to be explicitly declared.
- Created when the table is created.
- Use the primary key sorted in ascending order.

Non-clustered

- Non-clustered indexes are sorted references for a specific field, from the main table, that hold pointers back to the original entries of the table.

```
CREATE [UNIQUE | FULLTEXT | SPATIAL] INDEX index_name
    [index_type]
    ON tbl_name (key_part,...)
    [index_option]
    [algorithm_option | lock_option] ...

key_part: {col_name [(length)] | (expr)} [ASC | DESC]

index_option: {
    KEY_BLOCK_SIZE [=] value
  | index_type
  | WITH PARSER parser_name
  | COMMENT 'string'
  | {VISIBLE | INVISIBLE}
  | ENGINE_ATTRIBUTE [=] 'string'
  | SECONDARY_ENGINE_ATTRIBUTE [=] 'string'
}

index_type:
    USING {BTREE | HASH}

algorithm_option:
    ALGORITHM [=] {DEFAULT | INPLACE | COPY}

lock_option:
    LOCK [=] {DEFAULT | NONE | SHARED | EXCLUSIVE}
```

Summary

- Indexing can vastly reduce the time of queries
- Every table with a primary key has one clustered index
- Every table can have many non-clustered indexes to aid in querying
- Non-clustered indexes hold pointers back to the main table
- Not every database will benefit from indexing
- Not every index will increase the query speed for the database