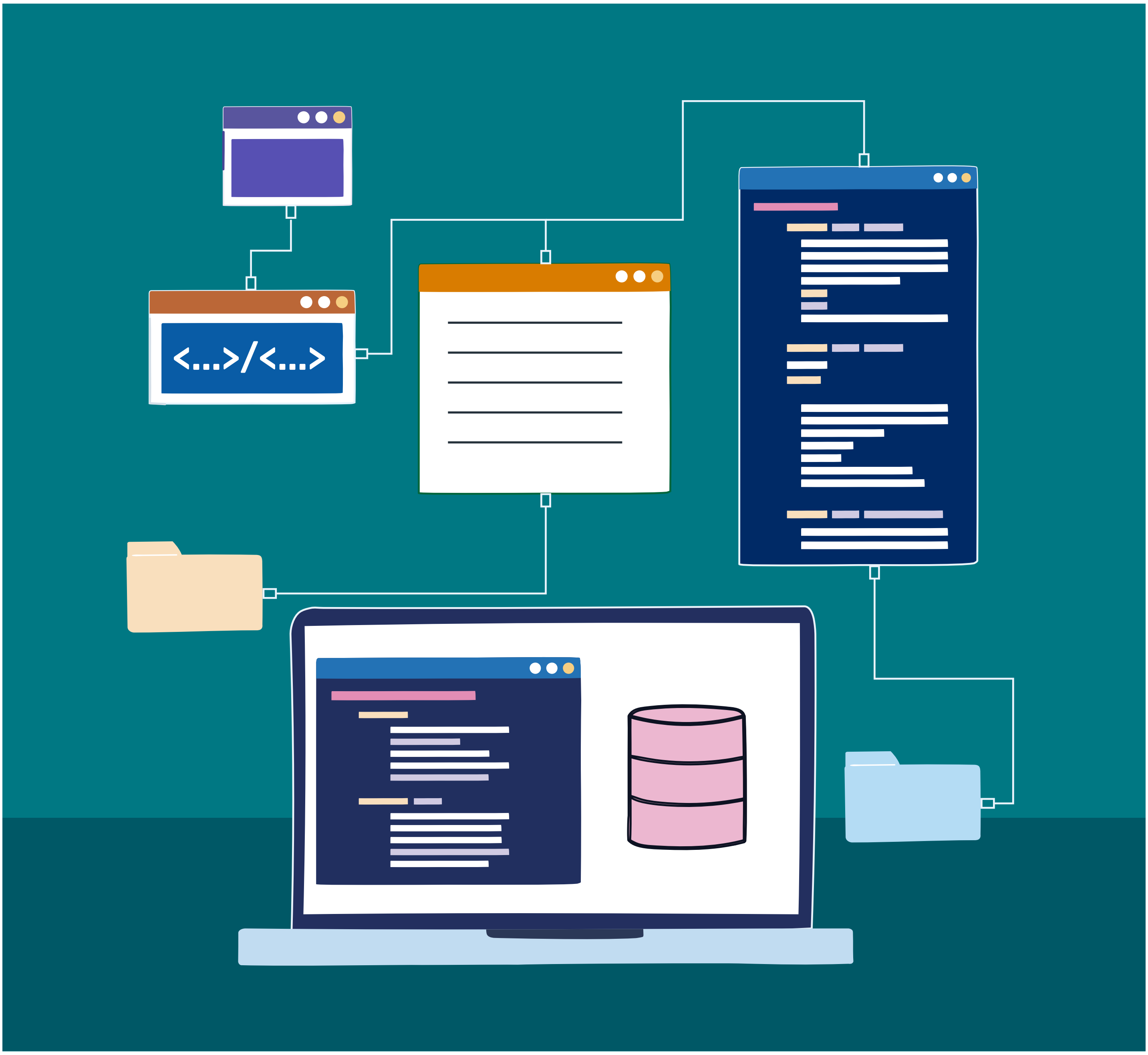
SCALER Topics

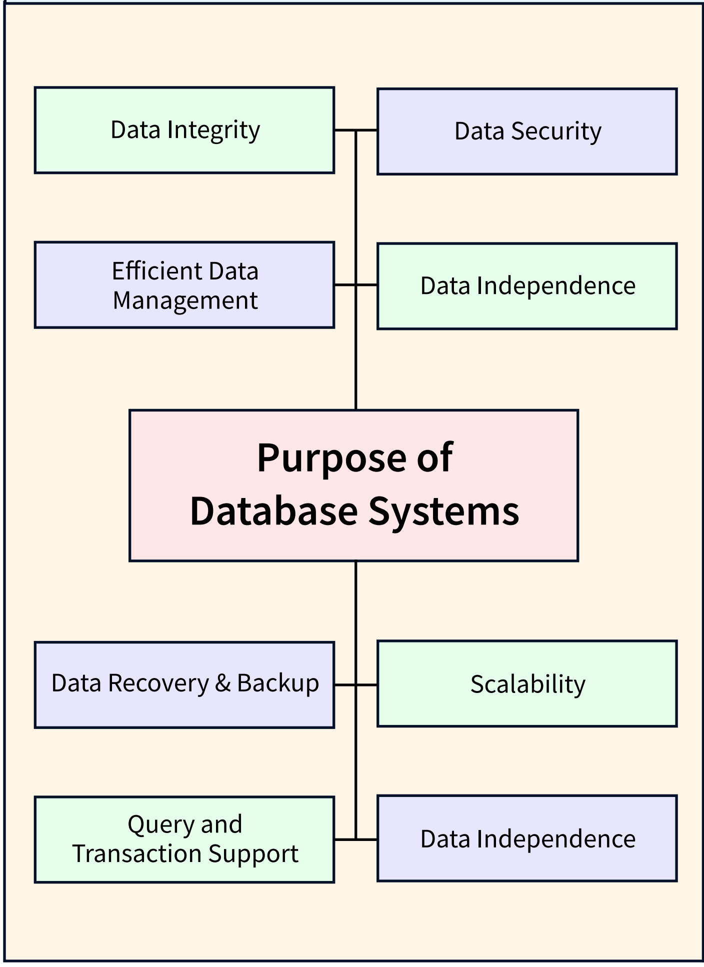
Cheatsheet

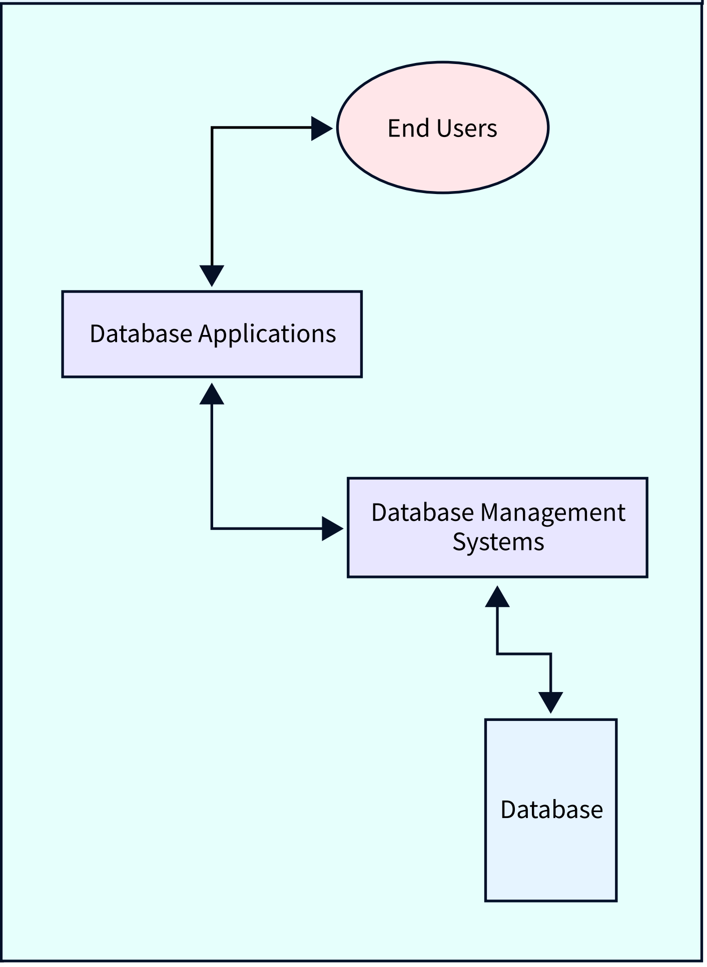
DATABASE MANAGEMENT SYSTEM

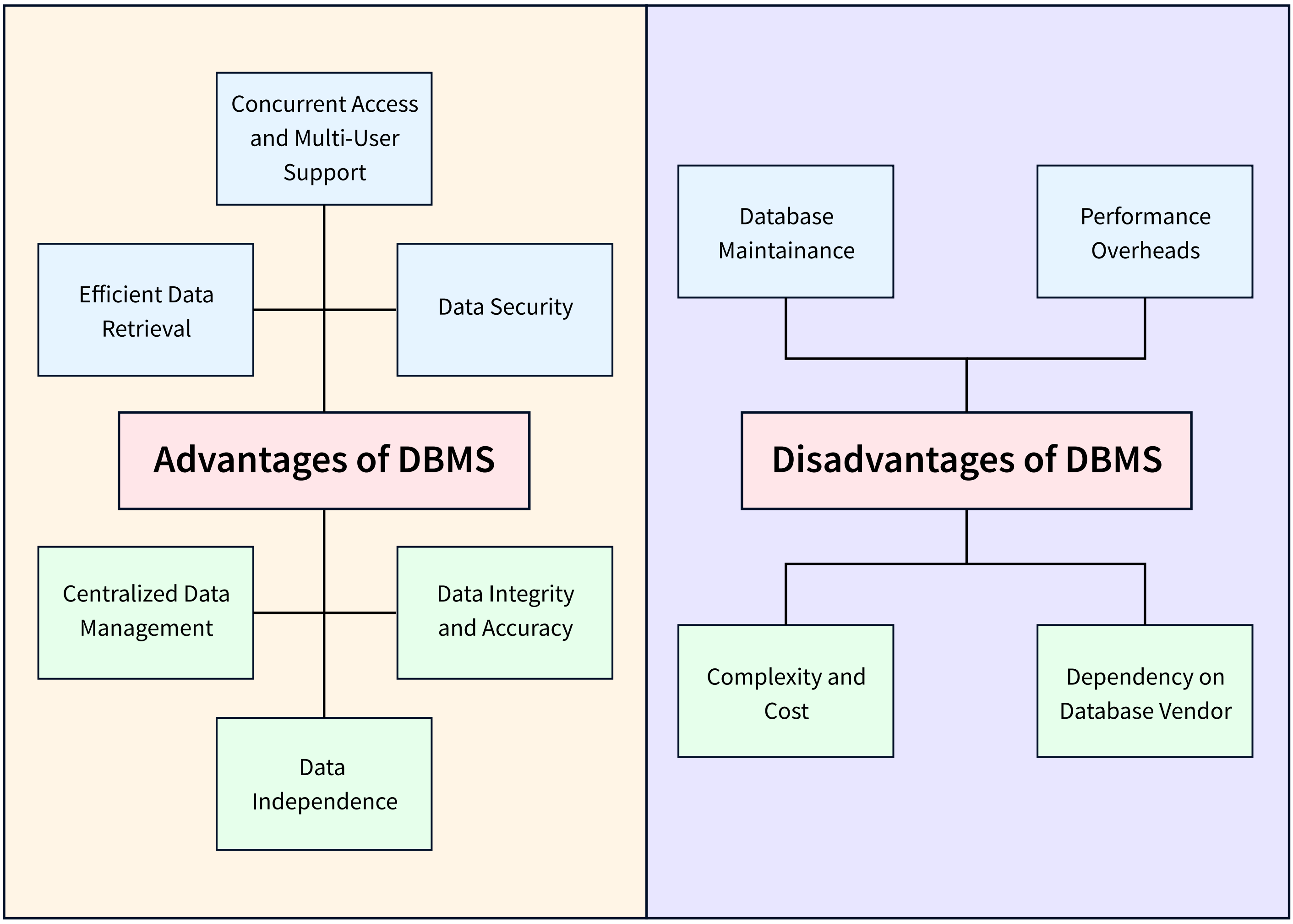


01 What is DBMS?

A Database System is an organized collection of data stored electronically, typically managed by a Database Management System (DBMS).







Advantages of DBMS

Concurrent Access and Multi-User Support

Efficient Data Retrieval

Data Security

Centralized Data Management

Data Integrity and Accuracy

Data Independence

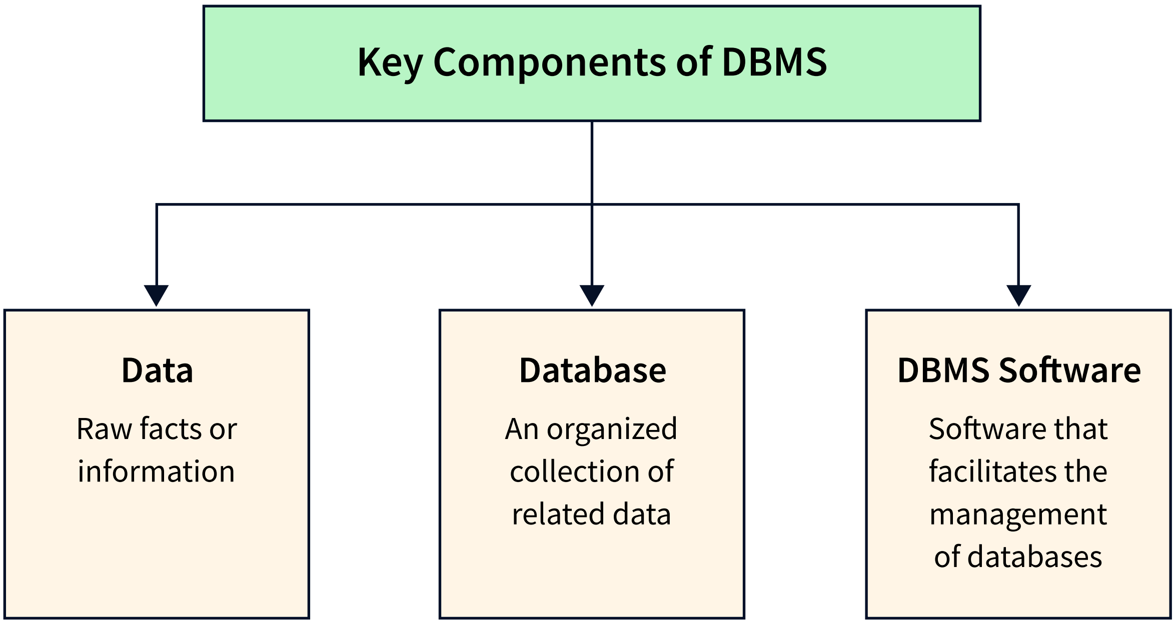
Disadvantages of DBMS

Database Maintainance

Performance Overheads

Complexity and Cost

Dependency on Database Vendor



Key Components of DBMS

Data

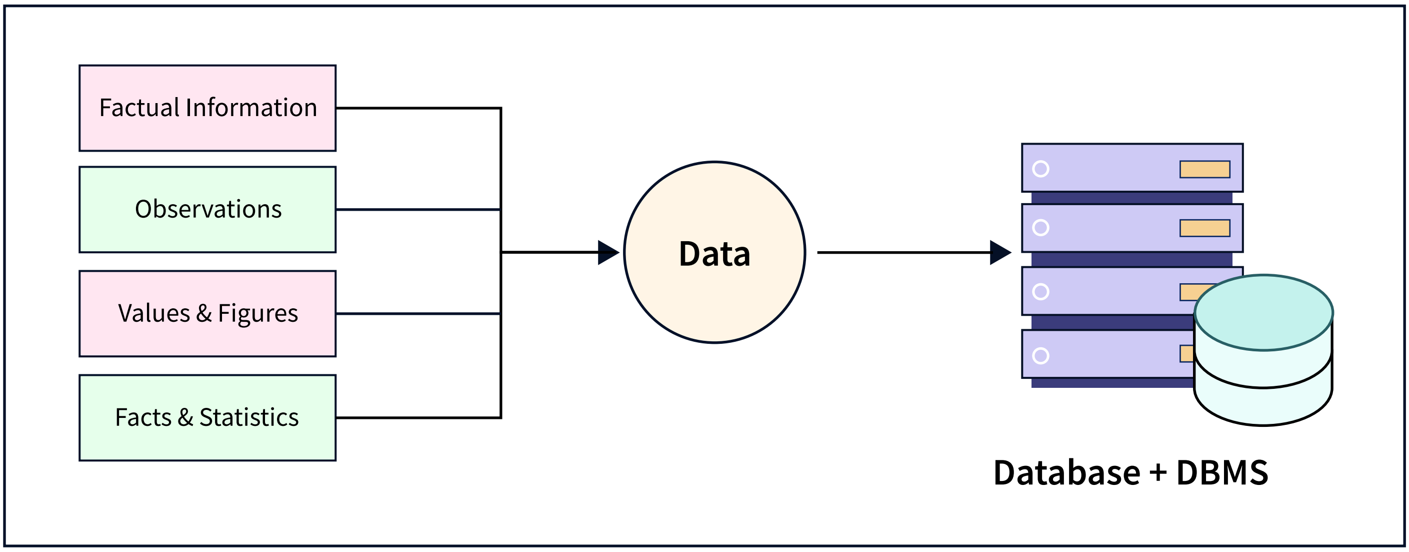
Raw facts or information

Database

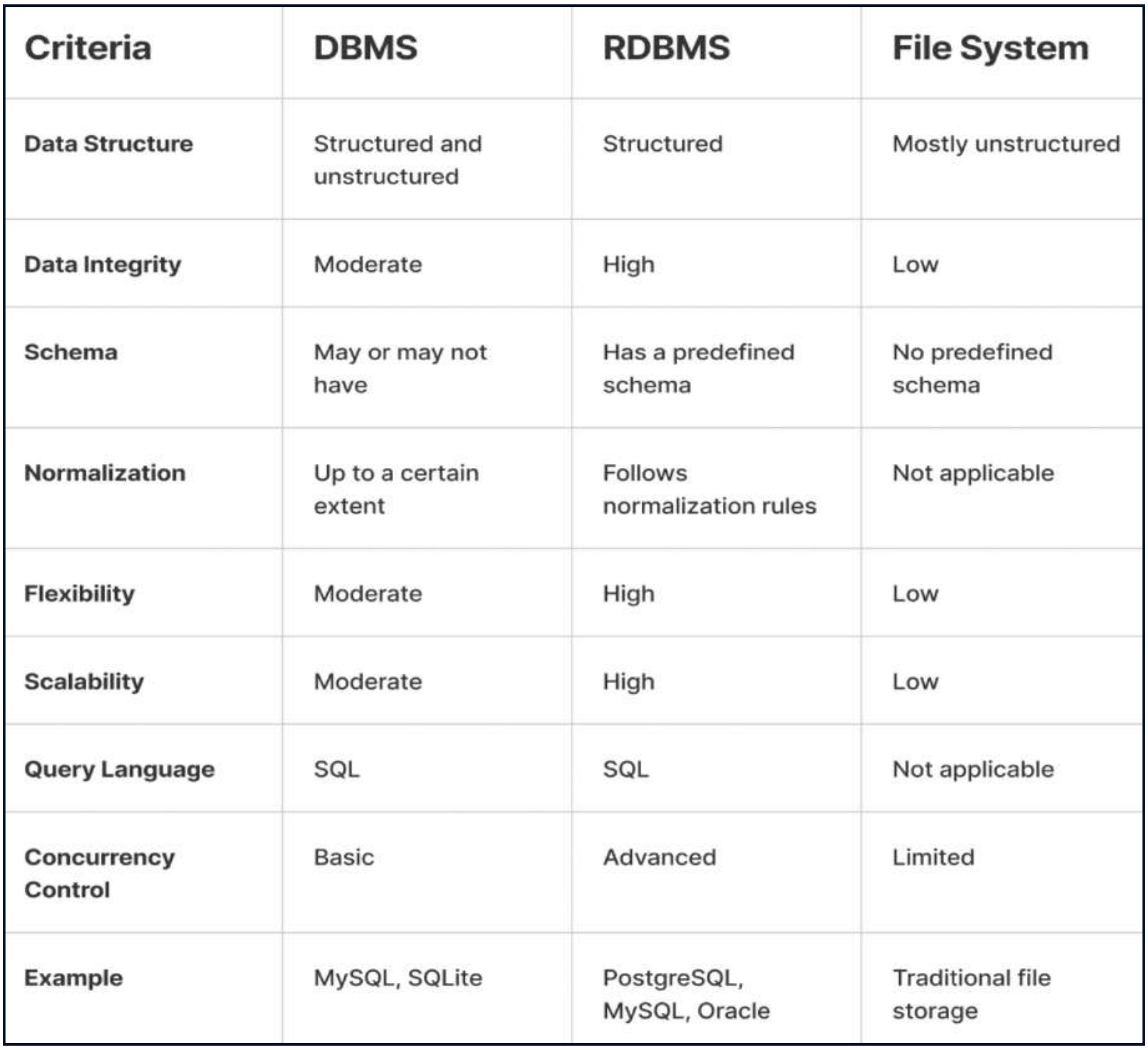
An organized collection of related data

DBMS Software

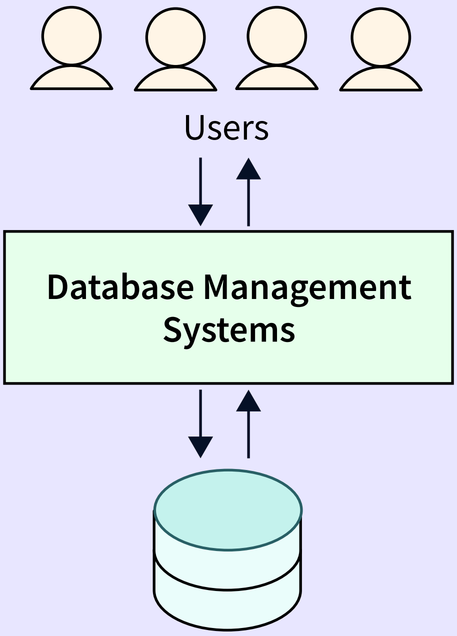
Software that facilitates the management of databases



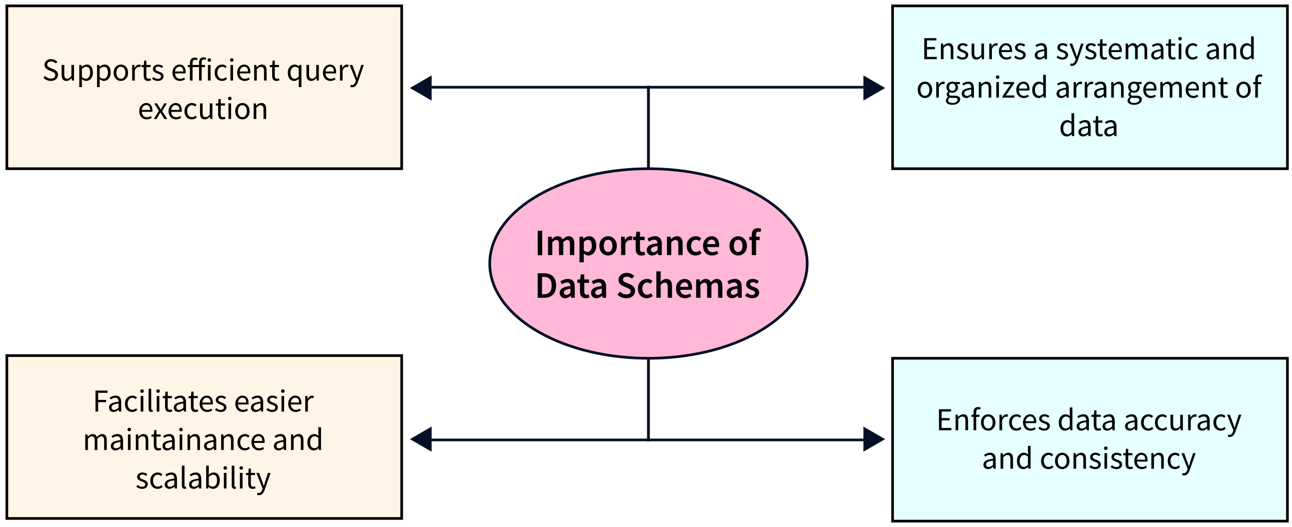
DBMS vs RDBMS vs File System



Criteria	DBMS	RDBMS	File System
Data Structure	Structured and unstructured	Structured	Mostly unstructured
Data Integrity	Moderate	High	Low
Schema	May or may not have	Has a predefined schema	No predefined schema
Normalization	Up to a certain extent	Follows normalization rules	Not applicable
Flexibility	Moderate	High	Low
Scalability	Moderate	High	Low
Query Language	SQL	SQL	Not applicable
Concurrency Control	Basic	Advanced	Limited
Example	MySQL, SQLite	PostgreSQL, MySQL, Oracle	Traditional file storage



- A Data Schema is a blueprint that defines the structure, organization, and relationships of data stored in a database. It provides a framework for representing and storing information in a systematic way.



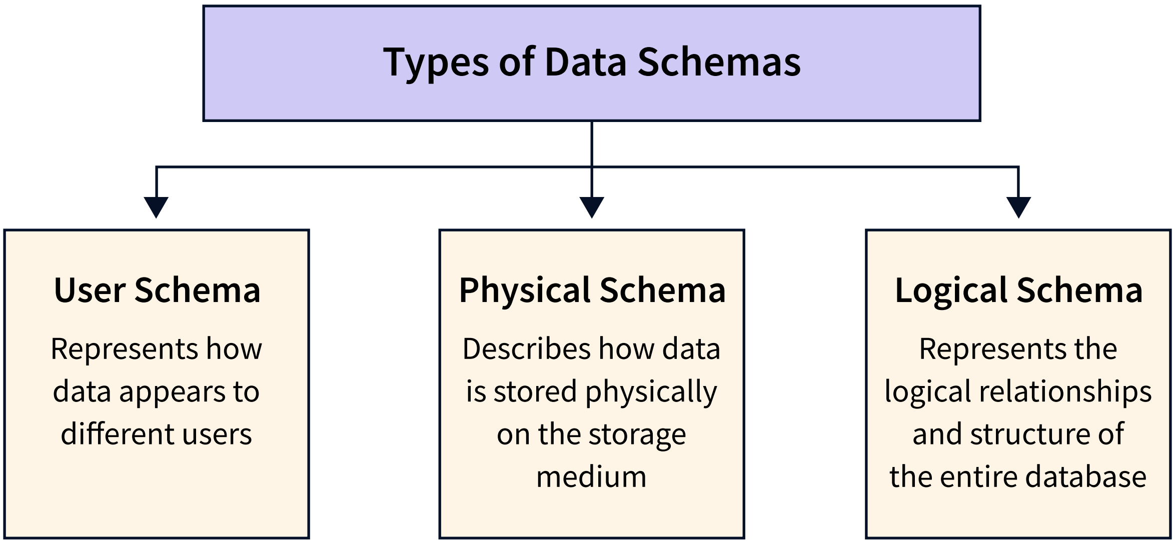
Importance of Data Schemas

Supports efficient query execution

Ensures a systematic and organized arrangement of data

Facilitates easier maintainance and scalability

Enforces data accuracy and consistency



Types of Data Schemas

User Schema

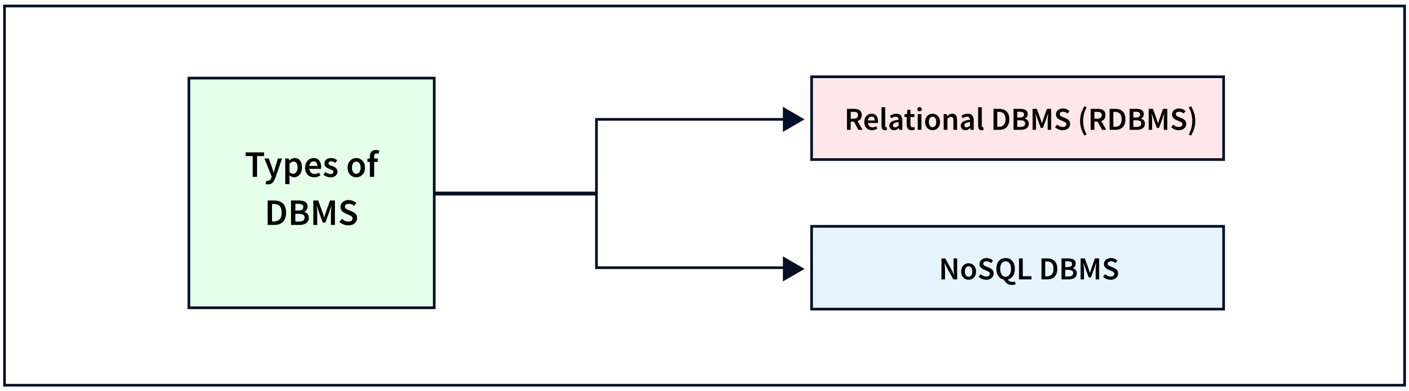
Represents how data appears to different users

Physical Schema

Describes how data is stored physically on the storage medium

Logical Schema

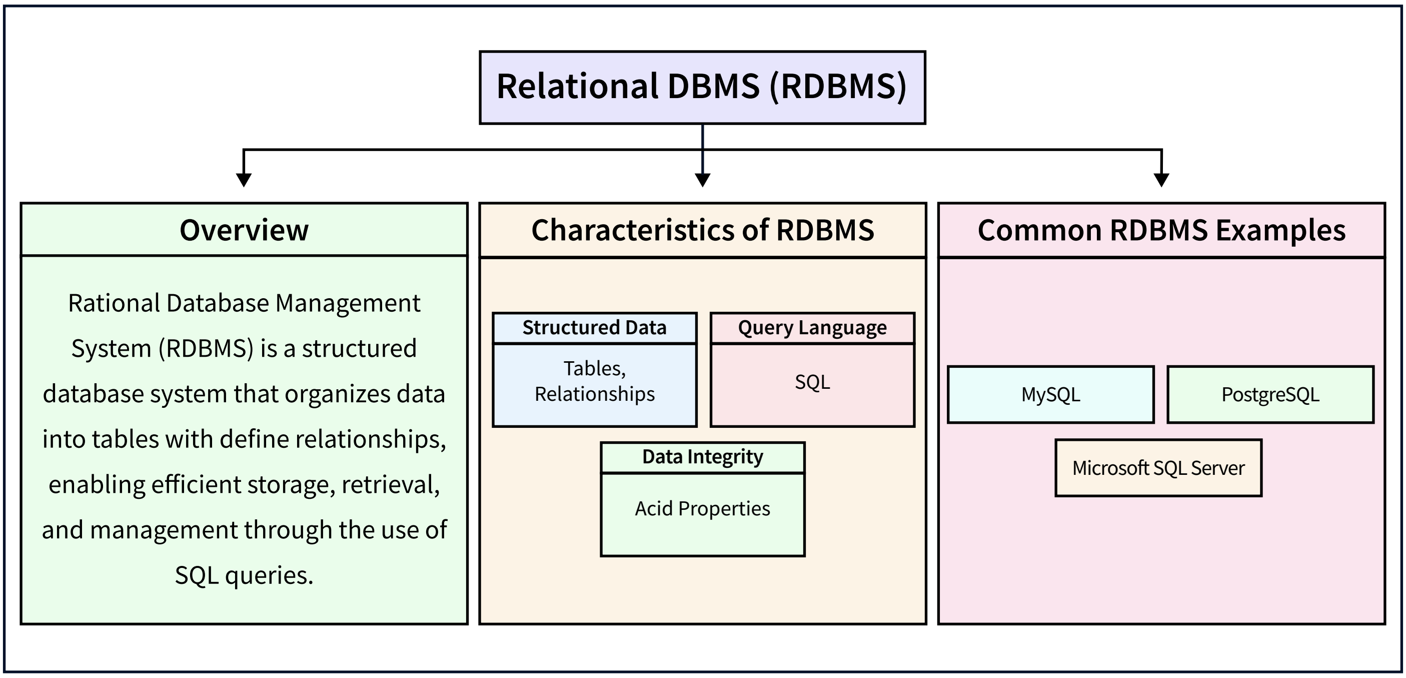
Represents the logical relationships and structure of the entire database



Types of DBMS

- Relational DBMS (RDBMS)
- NoSQL DBMS

Relational DBMS (RDBMS)



Relational DBMS (RDBMS)

Overview

Rational Database Management System (RDBMS) is a structured database system that organizes data into tables with define relationships, enabling efficient storage, retrieval, and management through the use of SQL queries.

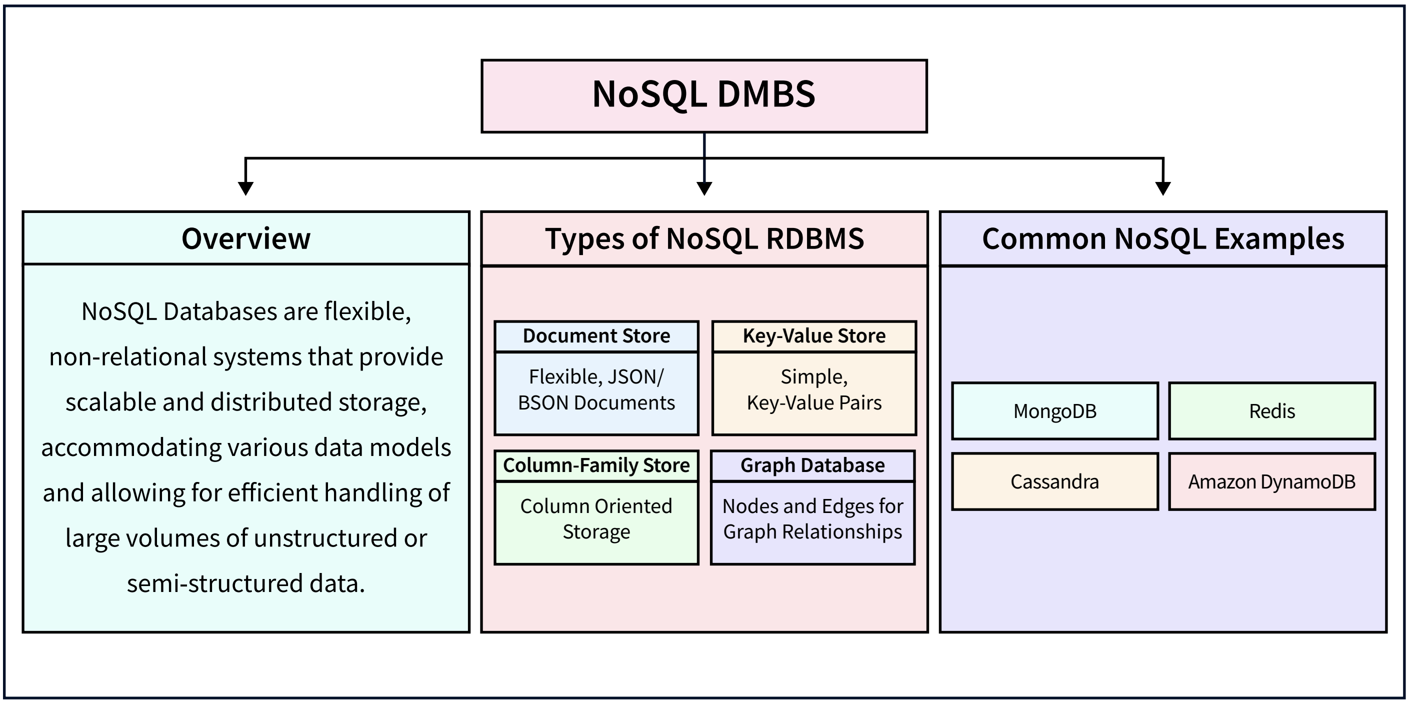
Characteristics of RDBMS

- Structured Data: Tables, Relationships
- Query Language: SQL
- Data Integrity: Acid Properties

Common RDBMS Examples

- MySQL
- PostgreSQL
- Microsoft SQL Server

NoSQL DBMS



NoSQL DMBS

Overview

NoSQL Databases are flexible, non-relational systems that provide scalable and distributed storage, accommodating various data models and allowing for efficient handling of large volumes of unstructured or semi-structured data.

Types of NoSQL RDBMS

Document Store
Flexible, JSON/ BSON Documents

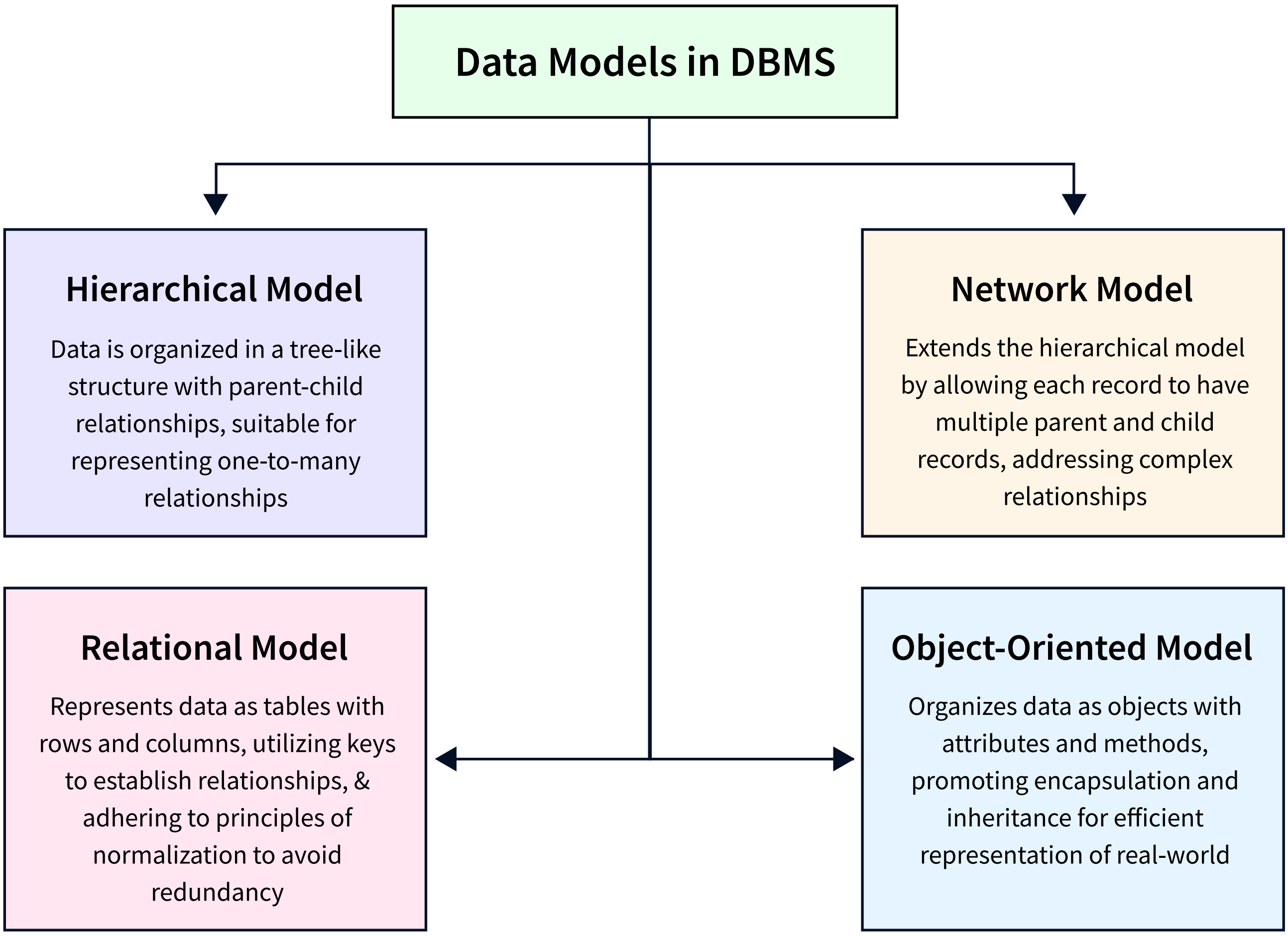
Key-Value Store
Simple, Key-Value Pairs

Column-Family Store
Column Oriented Storage

Graph Database
Nodes and Edges for Graph Relationships

Common NoSQL Examples

- MongoDB
- Redis
- Cassandra
- Amazon DynamoDB



Data Models in DBMS

Hierarchical Model

Data is organized in a tree-like structure with parent-child relationships, suitable for representing one-to-many relationships

Network Model

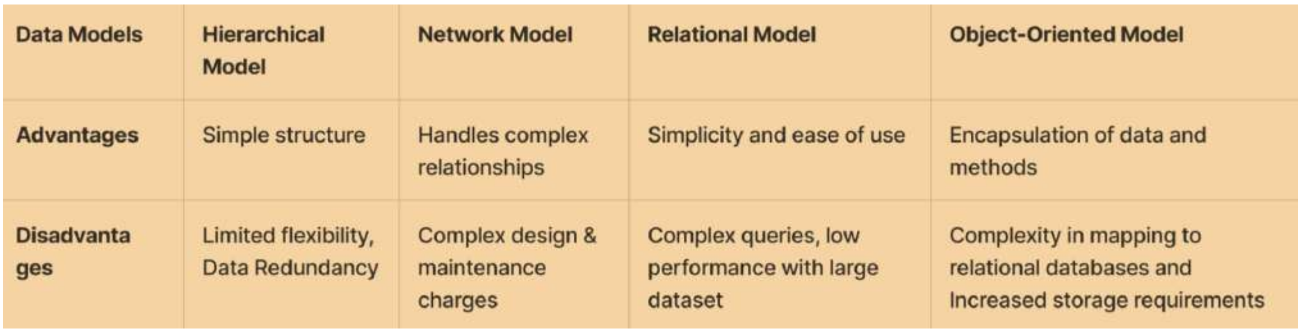
Extends the hierarchical model by allowing each record to have multiple parent and child records, addressing complex relationships

Relational Model

Represents data as tables with rows and columns, utilizing keys to establish relationships, & adhering to principles of normalization to avoid redundancy

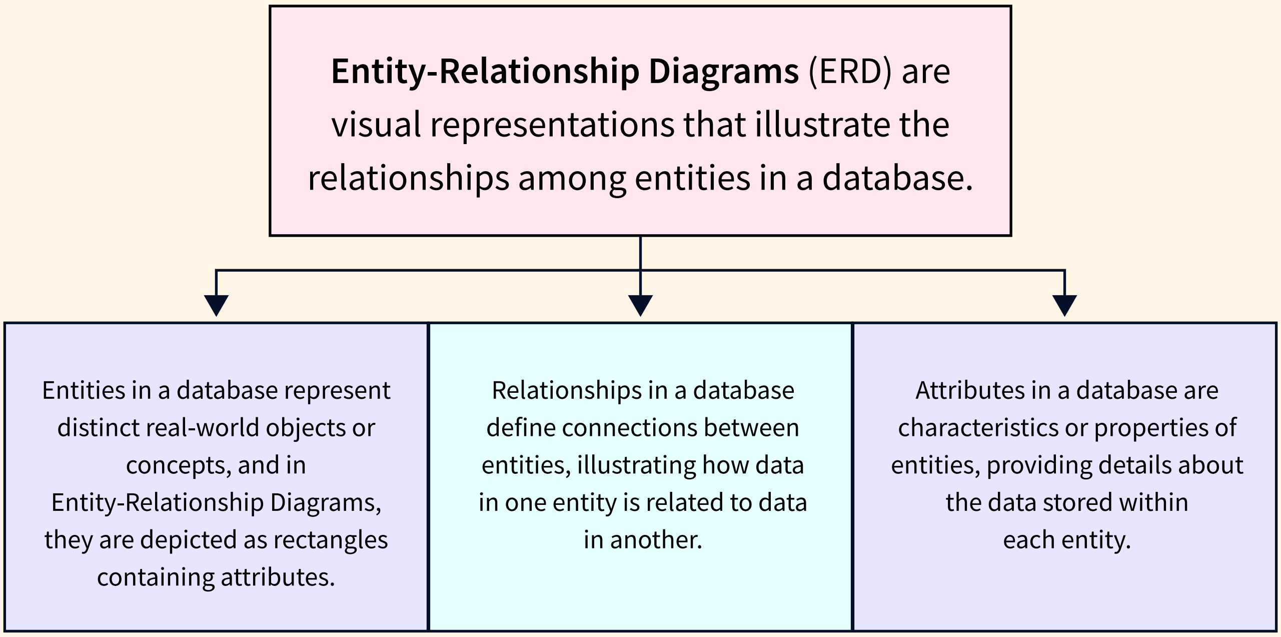
Object-Oriented Model

Organizes data as objects with attributes and methods, promoting encapsulation and inheritance for efficient representation of real-world



Data Models	Hierarchical Model	Network Model	Relational Model	Object-Oriented Model
Advantages	Simple structure	Handles complex relationships	Simplicity and ease of use	Encapsulation of data and methods
Disadvantages	Limited flexibility, Data Redundancy	Complex design & maintenance charges	Complex queries, low performance with large dataset	Complexity in mapping to relational databases and Increased storage requirements

02 Entity Relationships



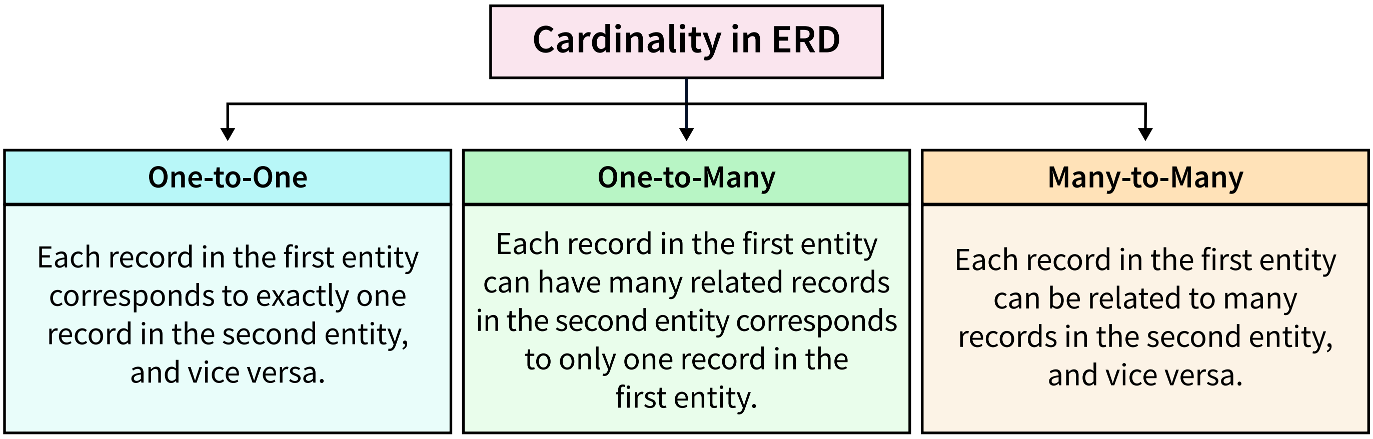
Entity-Relationship Diagrams (ERD) are visual representations that illustrate the relationships among entities in a database.

Entities in a database represent distinct real-world objects or concepts, and in Entity-Relationship Diagrams, they are depicted as rectangles containing attributes.

Relationships in a database define connections between entities, illustrating how data in one entity is related to data in another.

Attributes in a database are characteristics or properties of entities, providing details about the data stored within each entity.

Component	Symbol
Entity	▭
Relationships	◇
Attribute	⬭



Cardinality in ERD

One-to-One

Each record in the first entity corresponds to exactly one record in the second entity, and vice versa.

One-to-Many

Each record in the first entity can have many related records in the second entity corresponds to only one record in the first entity.

Many-to-Many

Each record in the first entity can be related to many records in the second entity, and vice versa.

Example : One-to-One

Employee
Employee_ID
Name
Department

Passport
Passport_number
Expiry_date

Each employee has exactly one passport, & each passport is associated with exactly one employee

Example : One-to-Many

Department
Department_ID
Department_Name

Employee
Employee_ID
Name

Each department can have many employees, but each employee belongs to only one department.

Example : Many-to-Many

Student
Student_ID
Name
Grade

Course
Course_ID
Course_Name

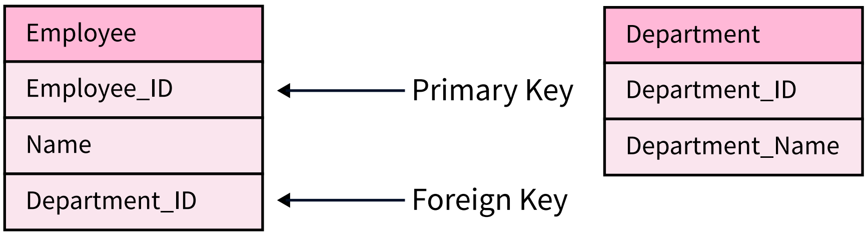
Each student can enroll in many courses, and each course can have many students.

03 Keys in DBMS

- Primary
- Foreign
- Composite
- Alternate
- Candidate
- Super

Primary Key	Foreign Key
A unique identifier for each record in an entity, often depicted in ERD as underlined attributes).	A field in one table that links to the primary key in another table, establishing relationships between entities.

Example:



Employee
Employee_ID
Name
Department_ID

Primary Key

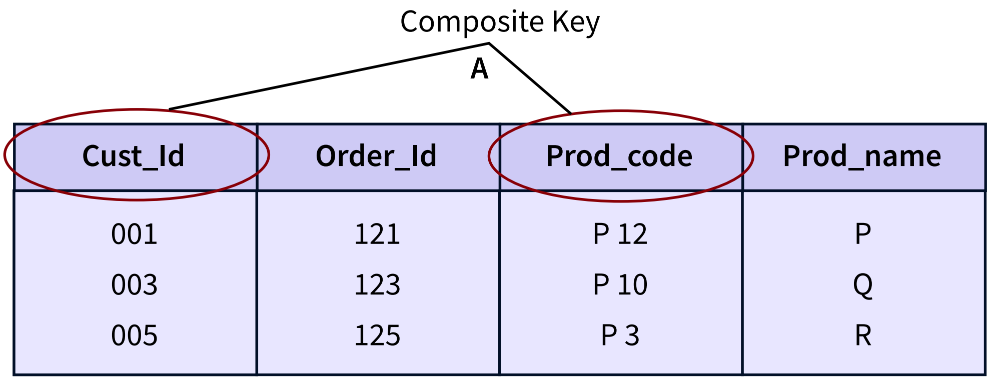
Foreign Key

Department
Department_ID
Department_Name

Relationship: Employee.Department_ID

Department.Department_ID

Composite Key
A composite key consists of two or more columns that, together, uniquely identify a record in a table. It's used when a single column is not sufficient to ensure uniqueness.

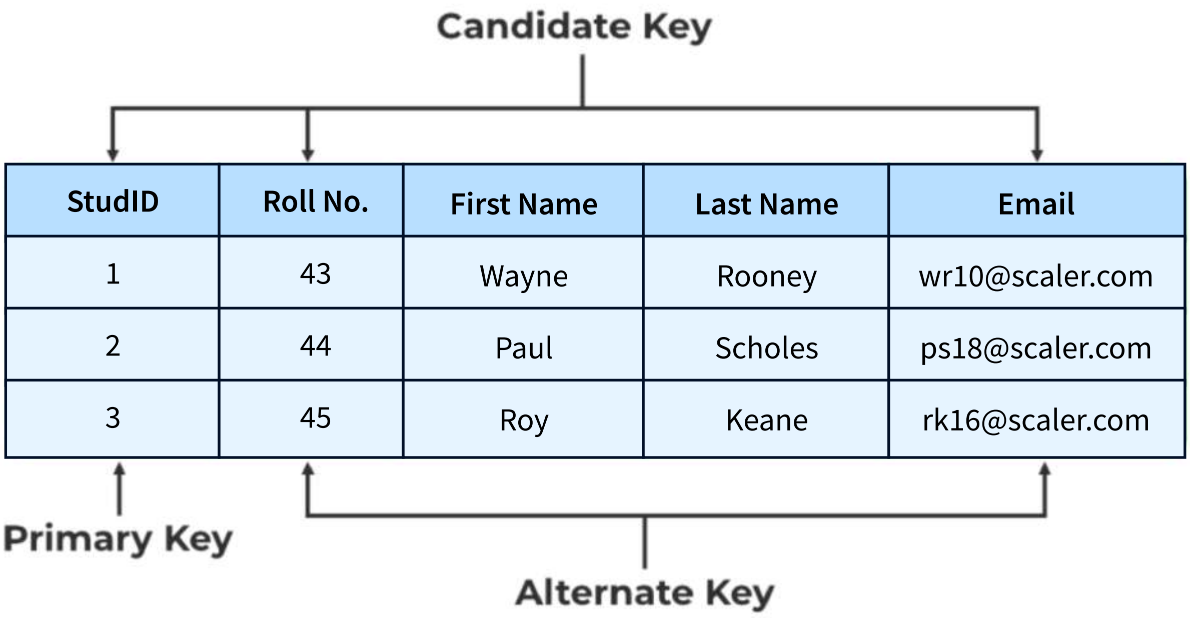


Cust_Id	Order_Id	Prod_code	Prod_name
001	121	P 12	P
003	123	P 10	Q
005	125	P 3	R

Combined value of these two columns is unique

Candidate Key	Alternate Key
A candidate key is a set of columns that can uniquely identify a record in a table. From these, one key is chosen as the primary key.	An alternate key is a candidate key that is not selected as the primary key. It can be used as a unique identifier if needed.

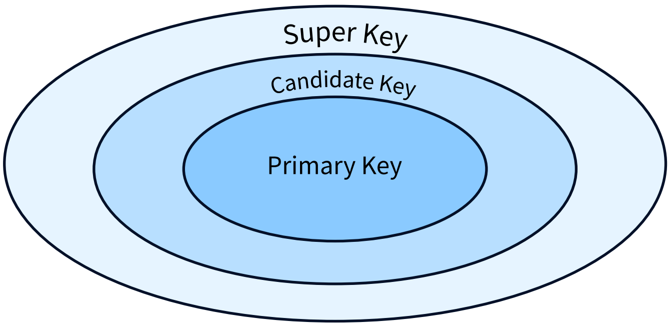
Example:



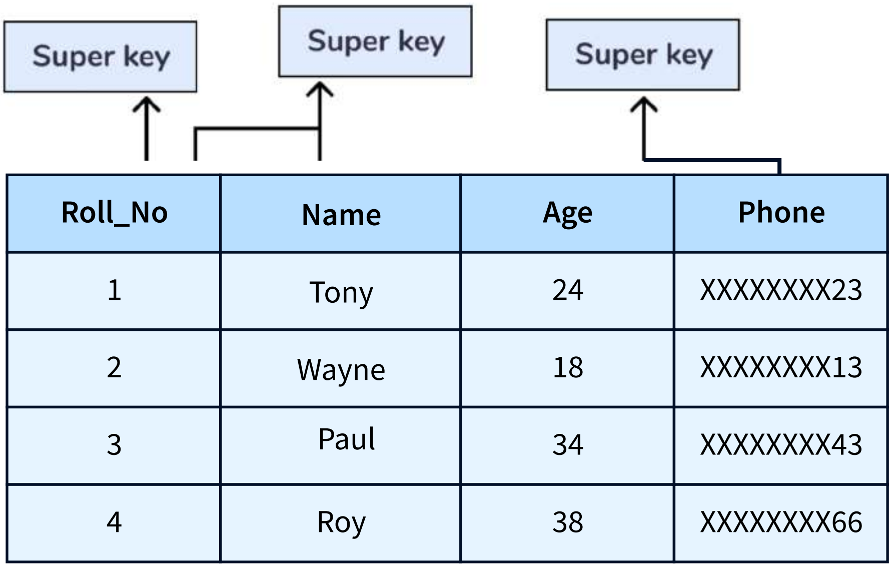
StudID	Roll No.	First Name	Last Name	Email
1	43	Wayne	Rooney	wr10@scaler.com
2	44	Paul	Scholes	ps18@scaler.com
3	45	Roy	Keane	rk16@scaler.com

Super Key

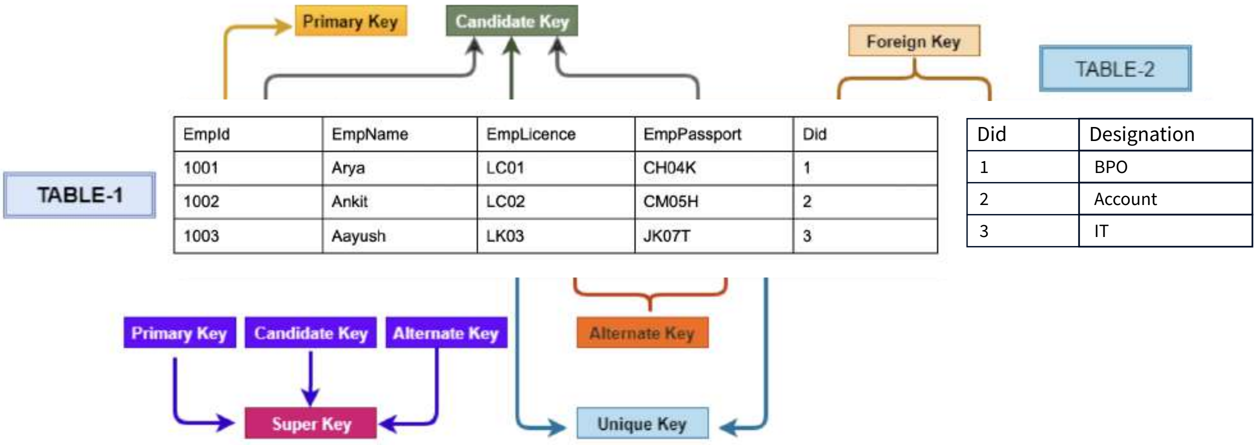
A super key is a set of one or more columns that can uniquely identify a record. It may contain more columns than necessary to uniquely identify a record.



Example



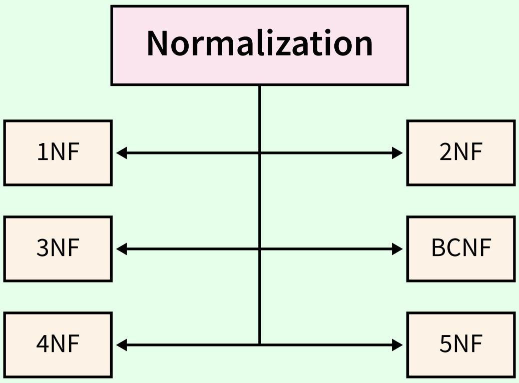
Roll_No	Name	Age	Phone
1	Tony	24	XXXXXXXX23
2	Wayne	18	XXXXXXXX13
3	Paul	34	XXXXXXXX43
4	Roy	38	XXXXXXXX66



EmpId	EmpName	EmpLicence	EmpPassport	Did
1001	Arya	LC01	CH04K	1
1002	Ankit	LC02	CM05H	2
1003	Aayush	LK03	JK07T	3

Did	Designation
1	BPO
2	Account
3	IT

04 Normalization



Normalization is a process of tidying up information in a database so that there's no unnecessary repetition, which can cause problems when adding, deleting, or updating data.

1NF (First Normal Form)

Ensures that each attribute in a table contains atomic values, and there is no repeating groups.

2NF (Second Normal Form)

Builds on 1NF and ensures that non-prime attributes are fully functionally dependent on the primary key.

3NF (Third Normal Form)

Further refines the normalization process by ensuring that no transitive dependencies exist.

BCNF (Boyce Codd Normal Form)

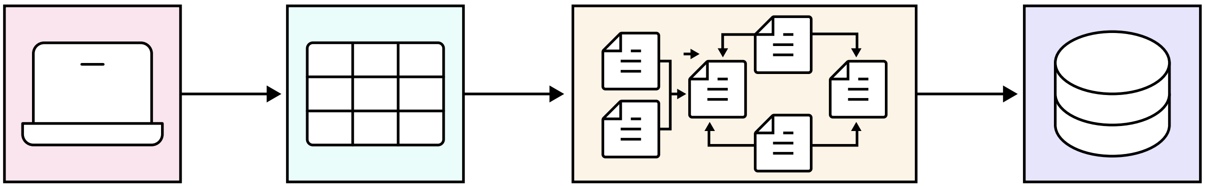
Ensures that there is no non-trivial functional dependencies of attributes on the primary key.

4NF (Forth Normal Form)

Extends normalization by addressing multi-valued dependencies.

5NF (Fifth Normal Form)

Handles cases where certain join dependencies exist in the database.



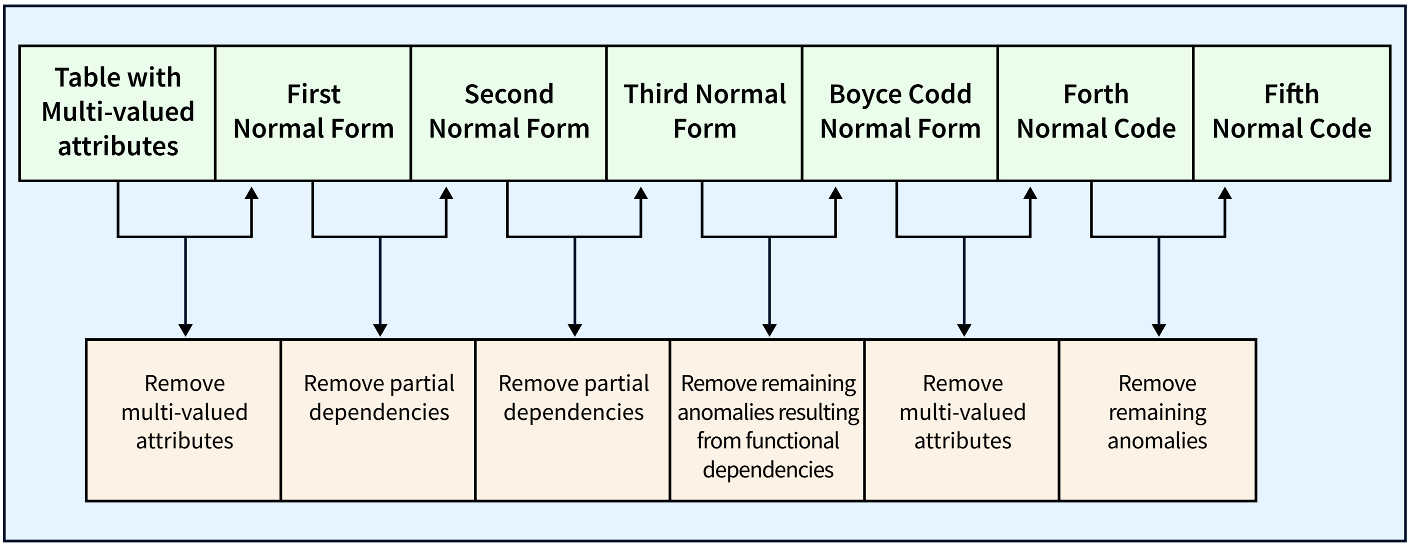


Table with Multi-valued attributes

First Normal Form

Second Normal Form

Third Normal Form

Boyce Codd Normal Form

Forth Normal Code

Fifth Normal Code

Remove multi-valued attributes

Remove partial dependencies

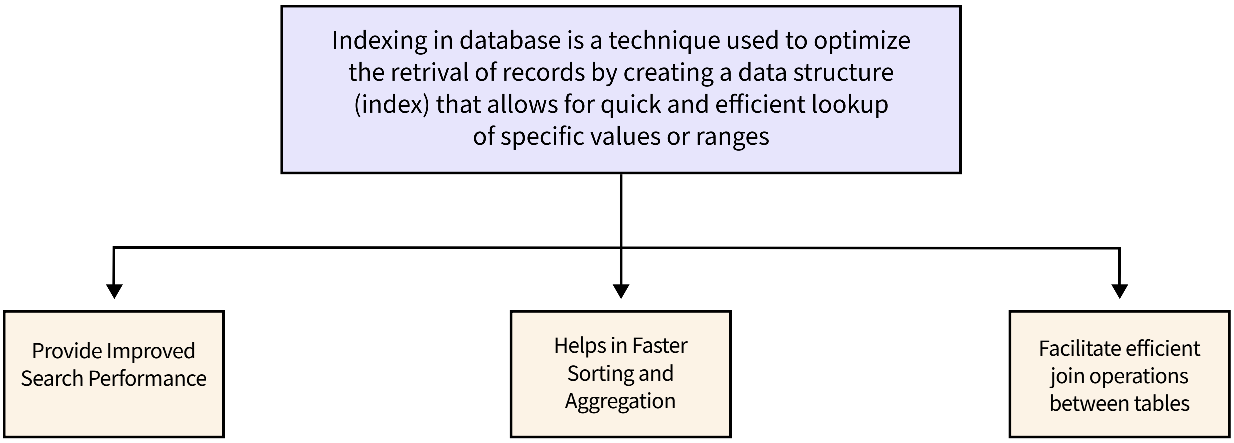
Remove partial dependencies

Remove remaining anomalies resulting from functional dependencies

Remove multi-valued attributes

Remove remaining anomalies

05 Indexing in DBMS

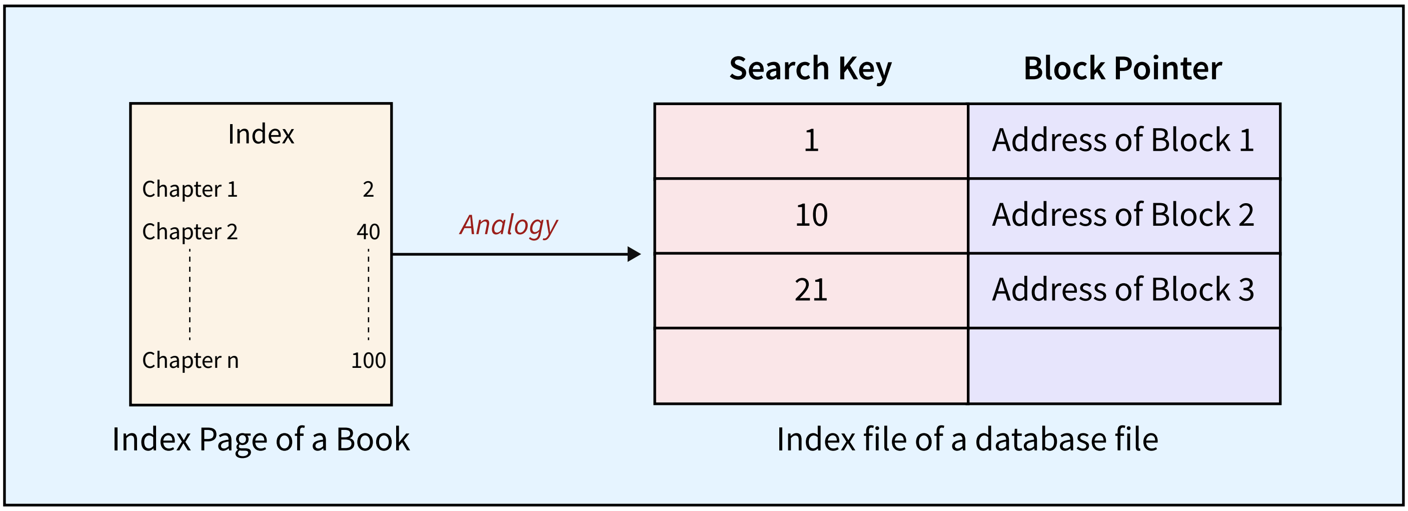


Indexing in database is a technique used to optimize the retrival of records by creating a data structure (index) that allows for quick and efficient lookup of specific values or ranges

Provide Improved Search Performance

Helps in Faster Sorting and Aggregation

Facilitate efficient join operations between tables



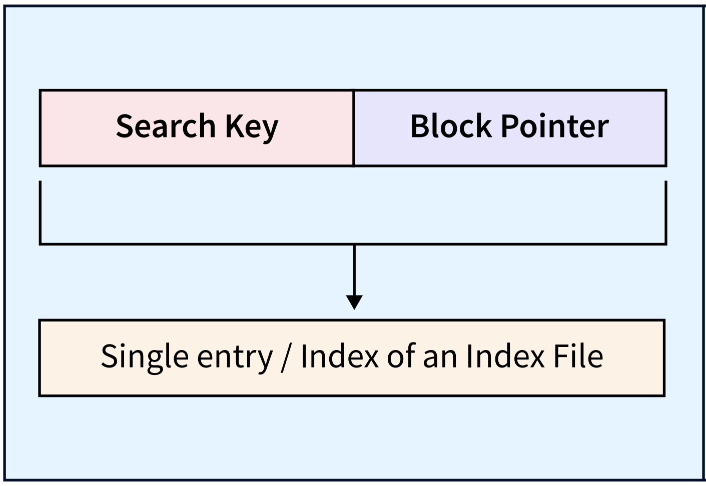
Index

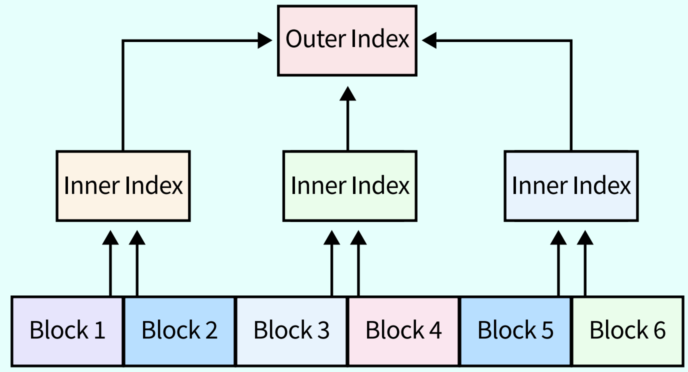
Chapter 1	2
Chapter 2	40
Chapter n	100

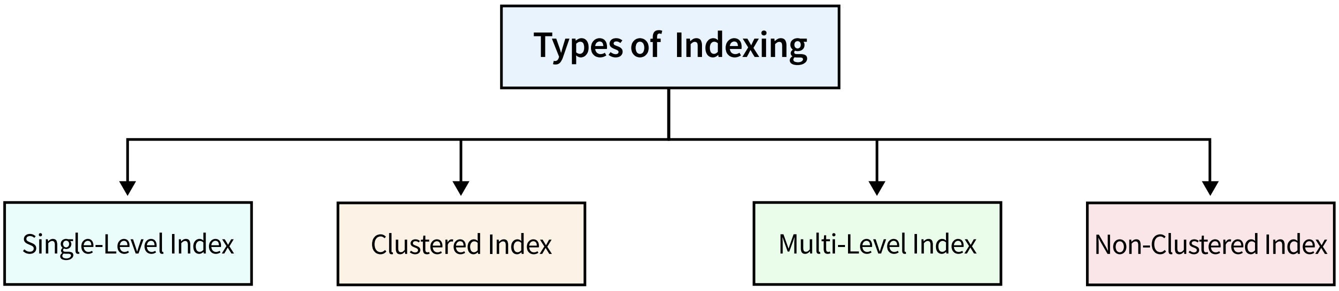
Index Page of a Book

Search Key	Block Pointer
1	Address of Block 1
10	Address of Block 2
21	Address of Block 3

Index file of a database file

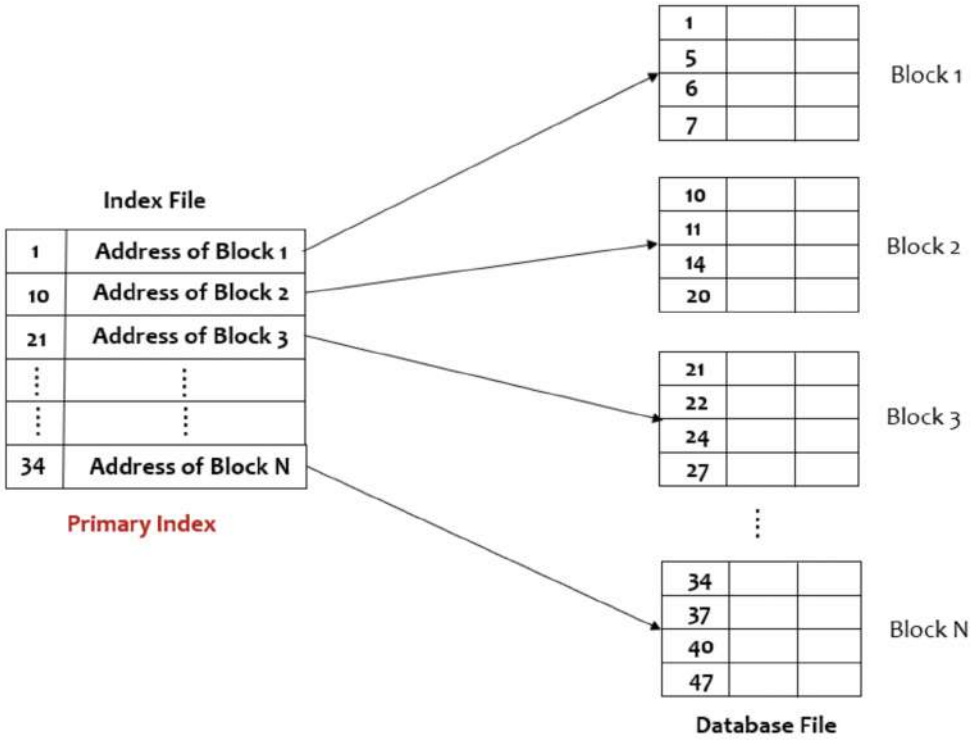






Primary Index/Single-Level Index

Uses a single index structure to locate the actual data

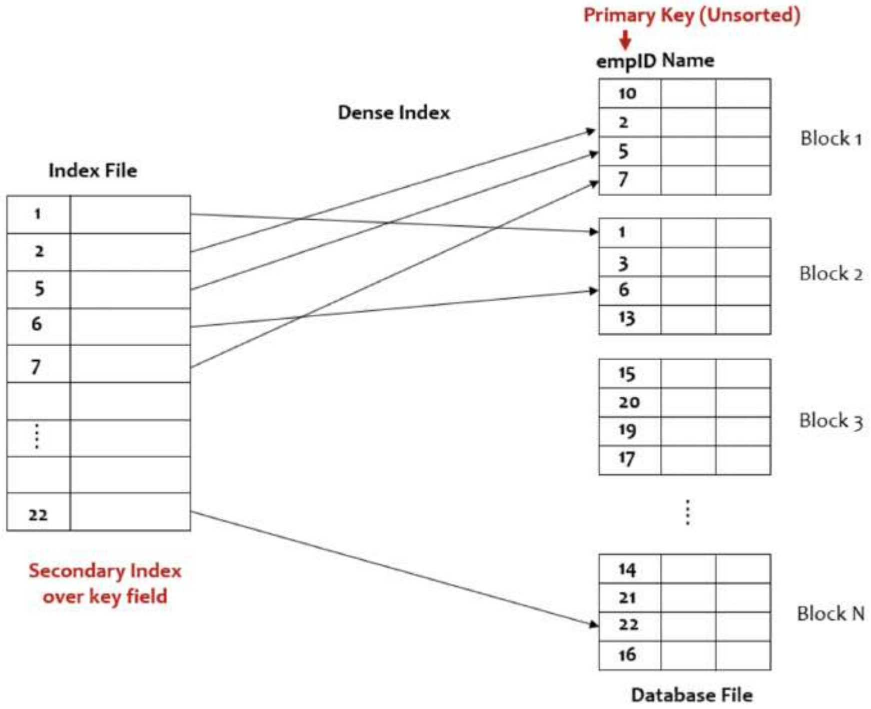


Index File

1	Address of Block 1
10	Address of Block 2
21	Address of Block 3
⋮	⋮
⋮	⋮
34	Address of Block N

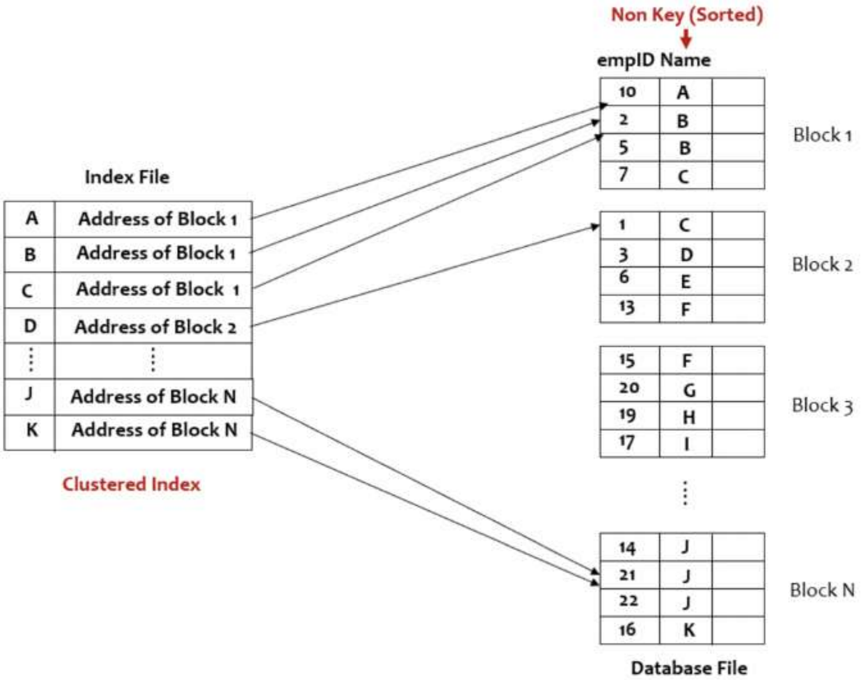
Secondary Indexing

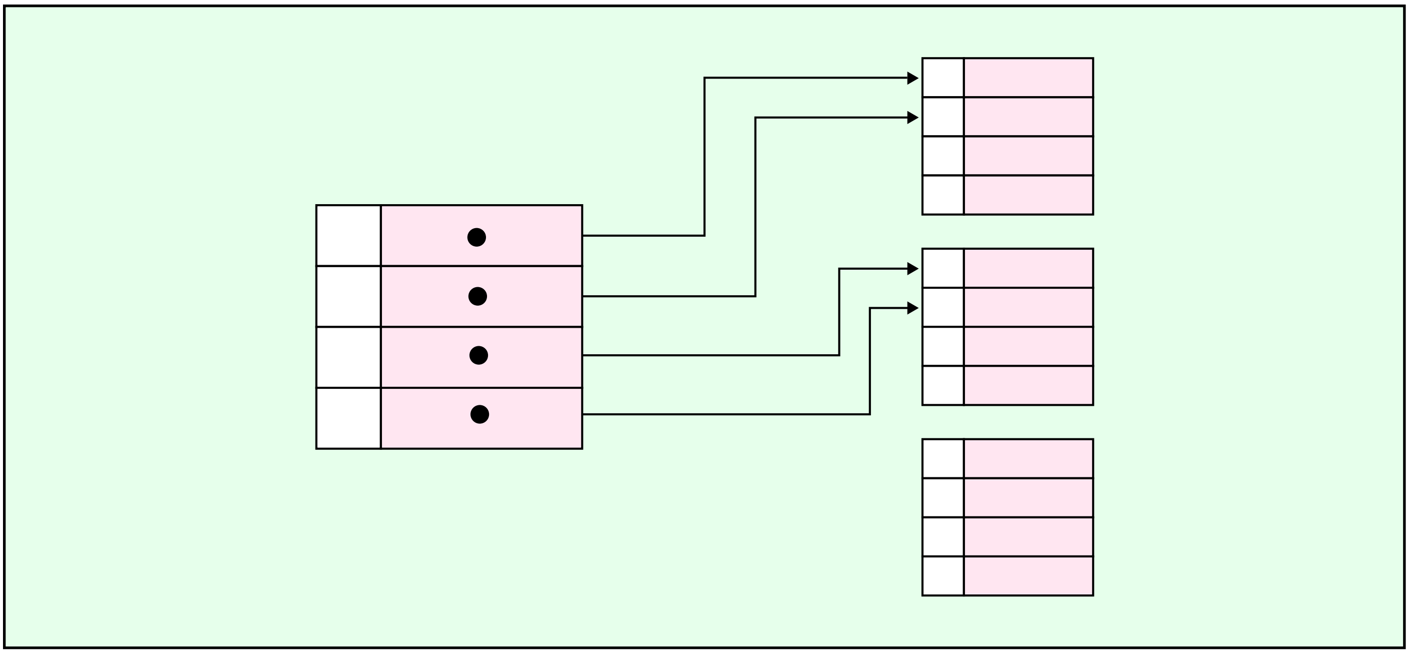
In secondary Indexing over the key field, the index is created on the unordered key field of the database file.
It is always a dense index.



Clustered Indexing

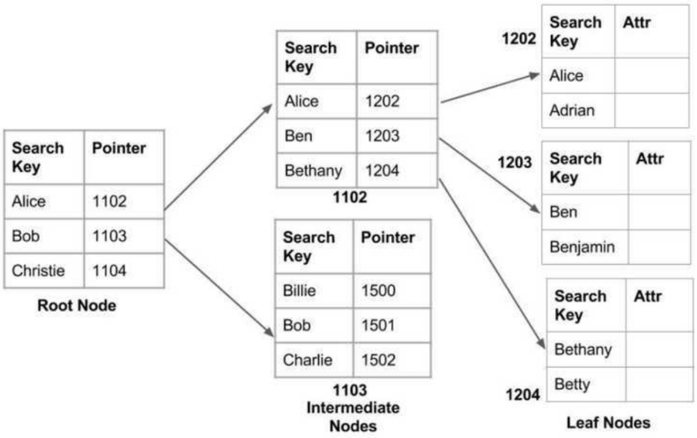
In clustered indexing, the index is created on the ordered nonkey field of the database file.

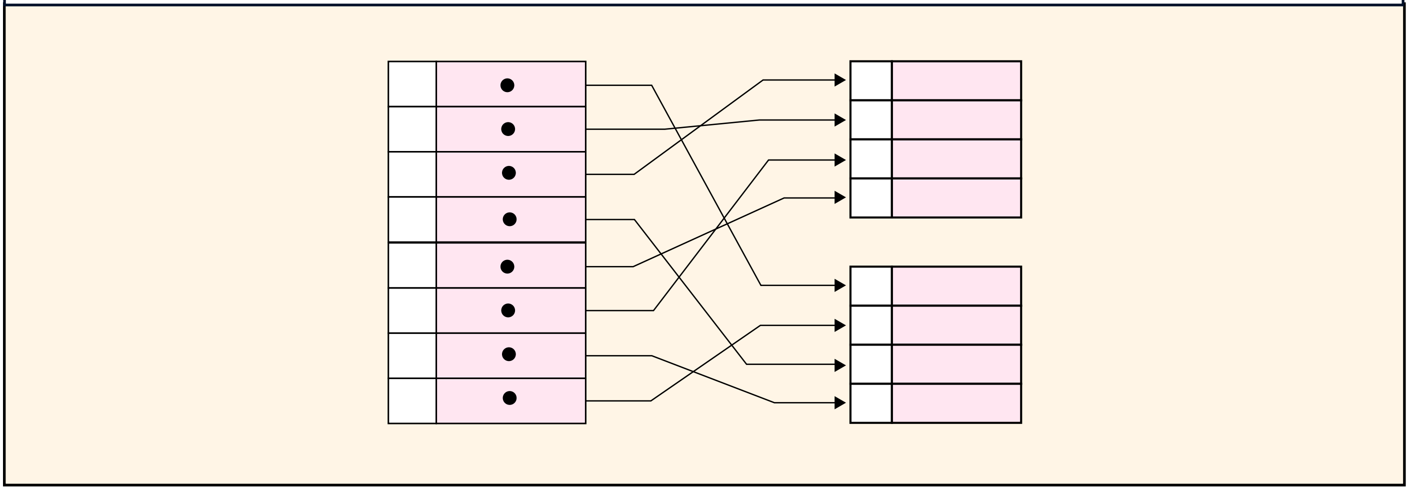




Non-clustered Indexing

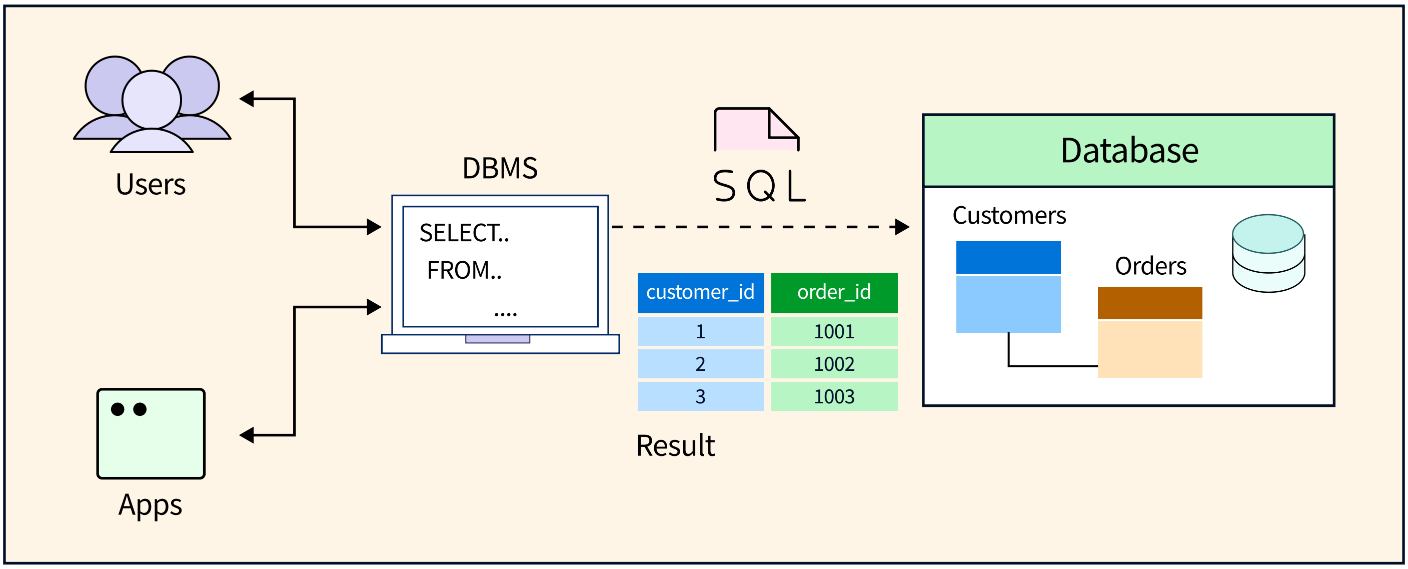
A non-clustered index just tells us where the data lies, i.e. it gives us a list of virtual pointers or references to the location where the data is actually stored. Data is not physically stored in the order of the index. Instead, data is present in leaf nodes





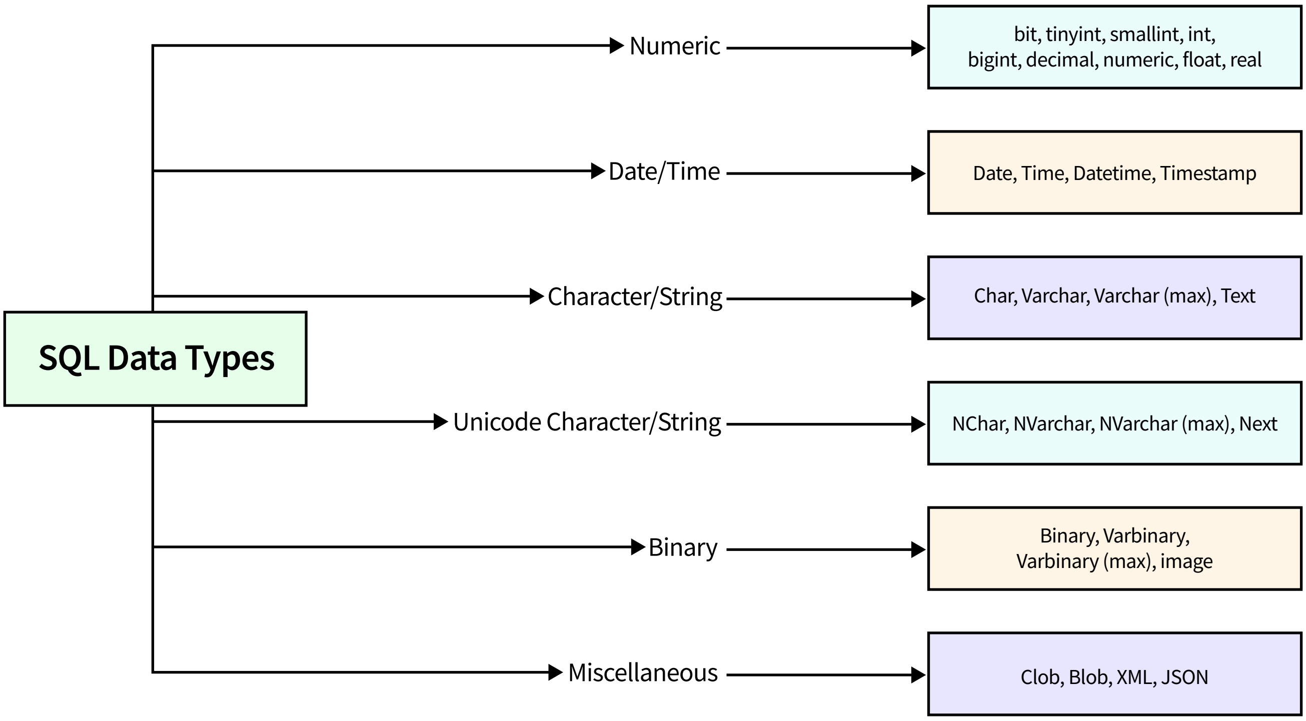
06 Structured Query Language

SQL is a standard programming language used for managing and manipulating relational databases. It provides a set of commands for interacting with databases, allowing users to define, query, update, and manage data efficiently.



customer_id	order_id
1	1001
2	1002
3	1003

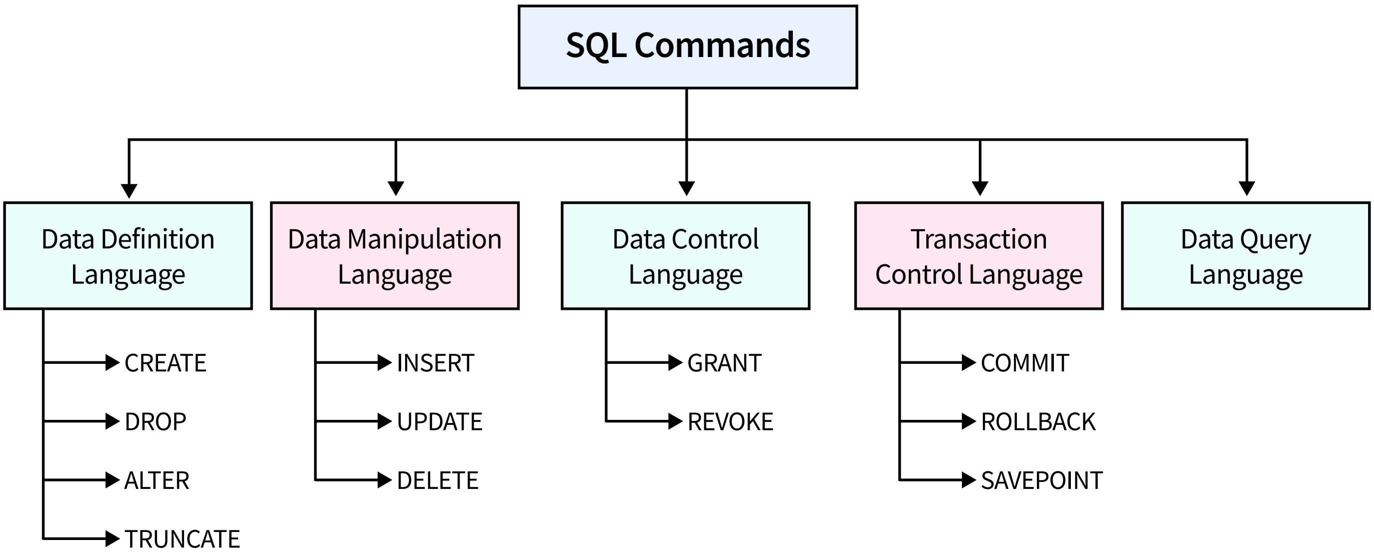
Result



SQL Data Types

- Numeric → bit, tinyint, smallint, int, bigint, decimal, numeric, float, real
- Date/Time → Date, Time, Datetime, Timestamp
- Character/String → Char, Varchar, Varchar (max), Text
- Unicode Character/String → NChar, NVarchar, NVarchar (max), Next
- Binary → Binary, Varbinary, Varbinary (max), image
- Miscellaneous → Clob, Blob, XML, JSON

SQL (Structured Query Language) commands are instructions that interact with a relational database management system (RDBMS). These commands allow users to perform various operations such as querying data, updating records, inserting new data, and managing the structure of a database



SQL Commands

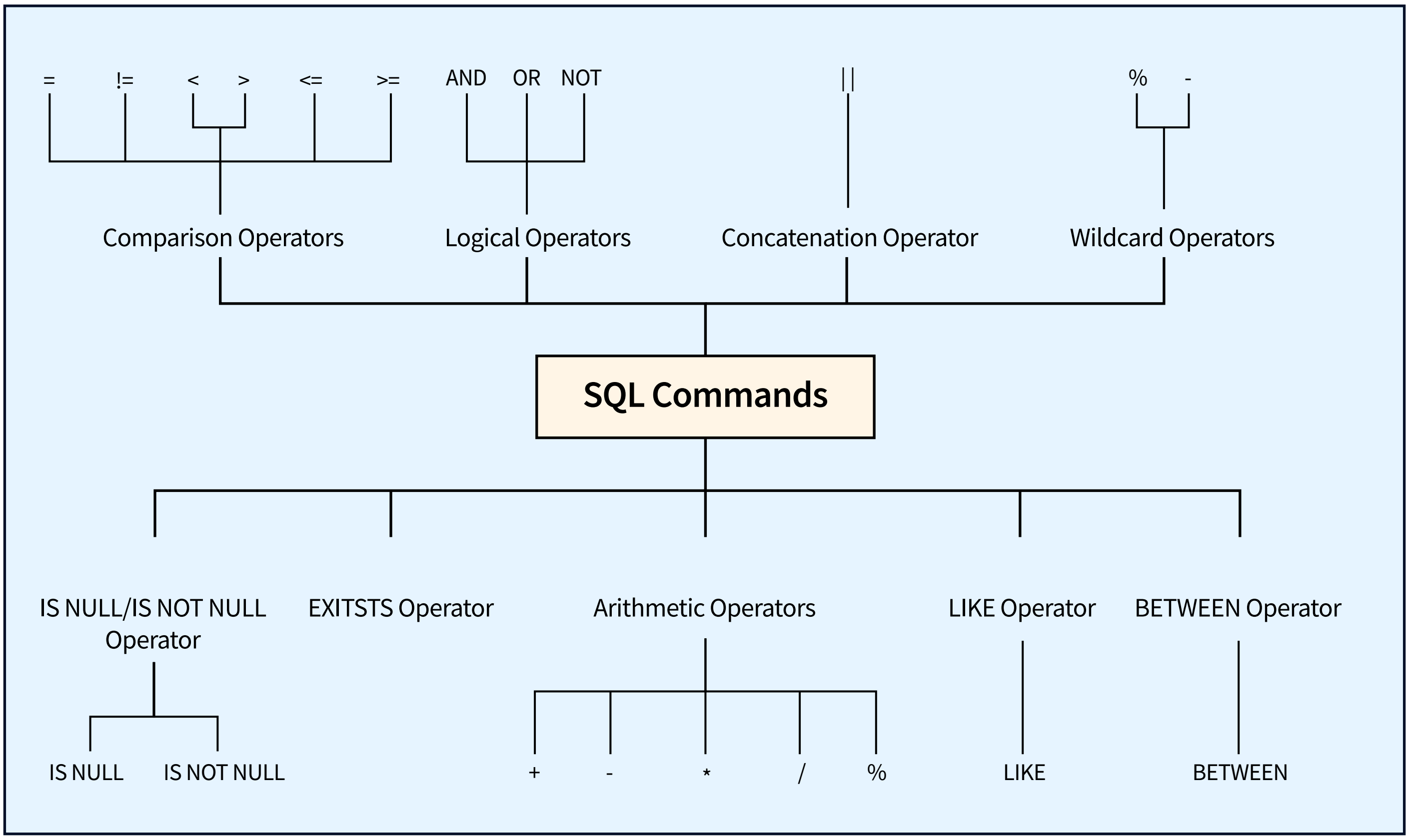
Data Definition Language
- CREATE
- DROP
- ALTER
- TRUNCATE

Data Manipulation Language
- INSERT
- UPDATE
- DELETE

Data Control Language
- GRANT
- REVOKE

Transaction Control Language
- COMMIT
- ROLLBACK
- SAVEPOINT

Data Query Language



Comparison Operators: = != < > <= >=

Logical Operators: AND OR NOT

Concatenation Operator: ||

Wildcard Operators: % -

SQL Commands

IS NULL/IS NOT NULL Operator: IS NULL, IS NOT NULL

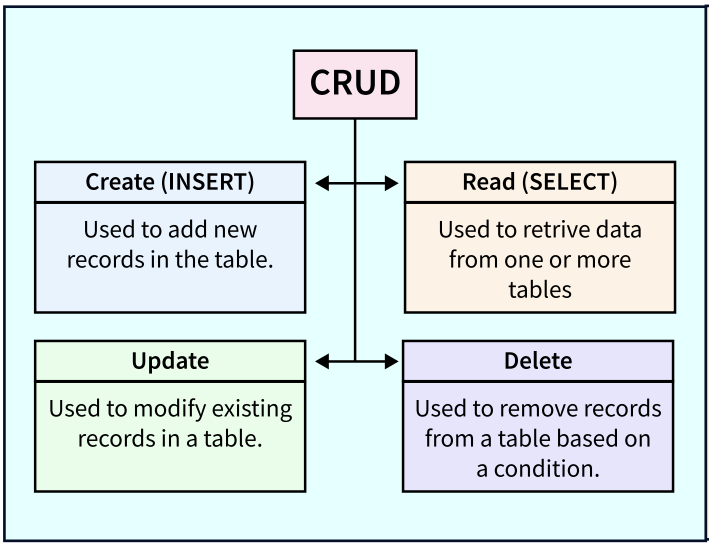
EXITSTS Operator

Arithmetic Operators: + - * / %

LIKE Operator: LIKE

BETWEEN Operator: BETWEEN

CRUD stands for create, read, update and delete which represent the fundamental operations performed on data in a database.



CRUD

Create (INSERT)

Used to add new records in the table.

Read (SELECT)

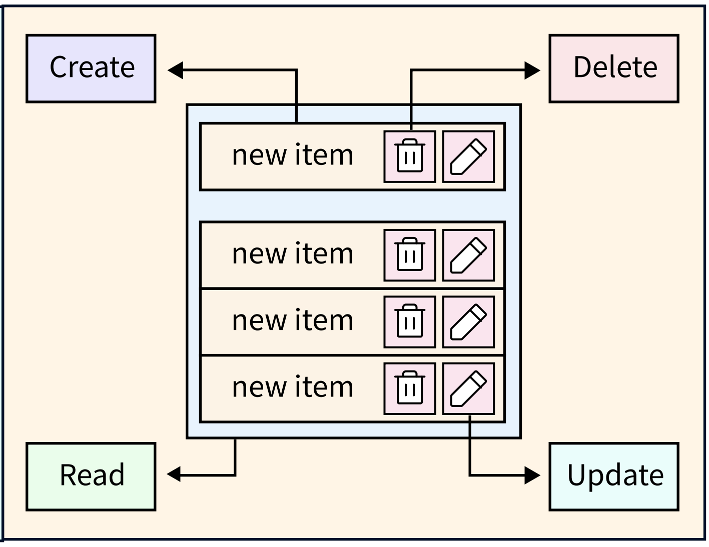
Used to retrive data from one or more tables

Update

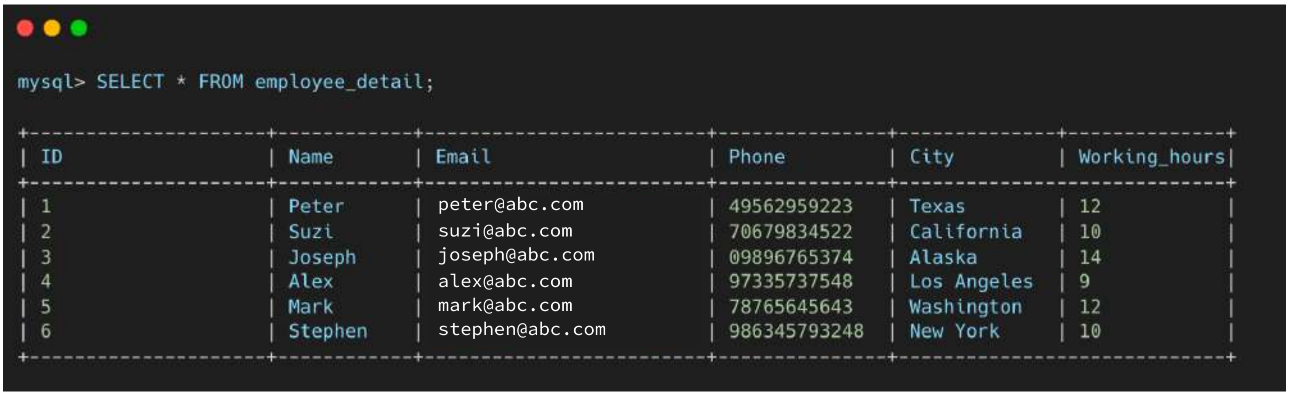
Used to modify existing records in a table.

Delete

Used to remove records from a table based on a condition.

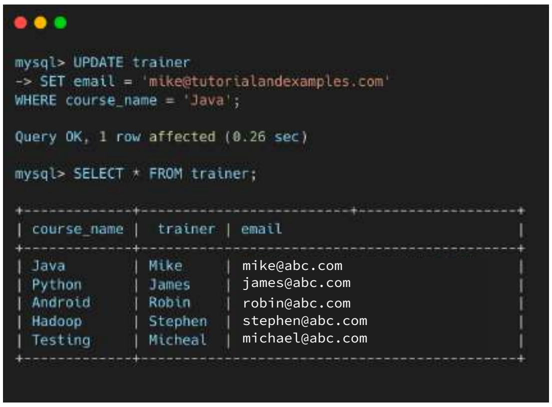


Read (SELECT) Example



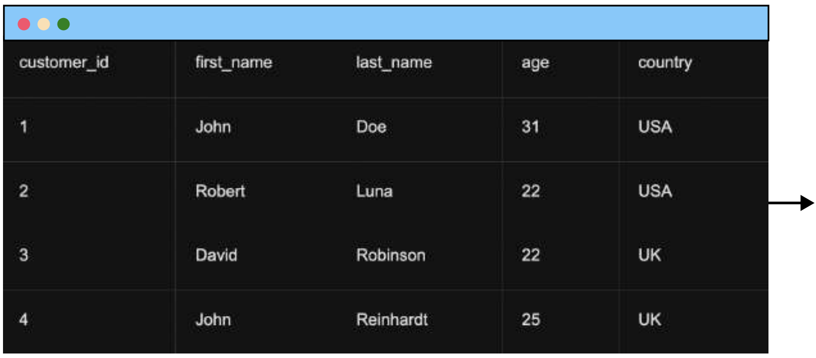
```
mysql> SELECT * FROM employee_detail;

+--------------------+-----------+-------------------------+-------------+-------------+--------------+
| ID                 | Name      | Email                   | Phone       | City        | Working_hours|
+--------------------+-----------+-------------------------+-------------+-------------+--------------+
| 1                  | Peter     | peter@abc.com           | 49562959223 | Texas       | 12           |
| 2                  | Suzi      | suzi@abc.com            | 70679834522 | California  | 10           |
| 3                  | Joseph    | joseph@abc.com          | 09896765374 | Alaska      | 14           |
| 4                  | Alex      | alex@abc.com            | 97335737548 | Los Angeles | 9            |
| 5                  | Mark      | mark@abc.com            | 78765645643 | Washington  | 12           |
| 6                  | Stephen   | stephen@abc.com         | 986345793248| New York    | 10           |
+--------------------+-----------+-------------------------+-------------+-------------+--------------+
```

UPDATE Example

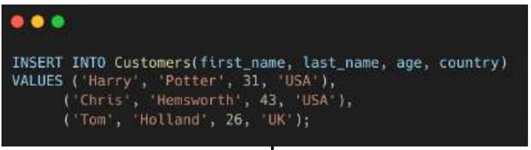


```
mysql> UPDATE trainer
-> SET email = 'mike@tutorialandexamples.com'
WHERE course_name = 'Java';

Query OK, 1 row affected (0.26 sec)

mysql> SELECT * FROM trainer;

+-------------+-----------------------+----------------------+
| course_name |  trainer | email                             |
+-------------+---------------------------------------------+
| Java        | Mike     | mike@abc.com                      |
| Python      | James    | james@abc.com                     |
| Android     | Robin    | robin@abc.com                     |
| Hadoop      | Stephen  | stephen@abc.com                   |
| Testing     | Micheal  | michael@abc.com                   |
+-------------+---------------------------------------------+
```

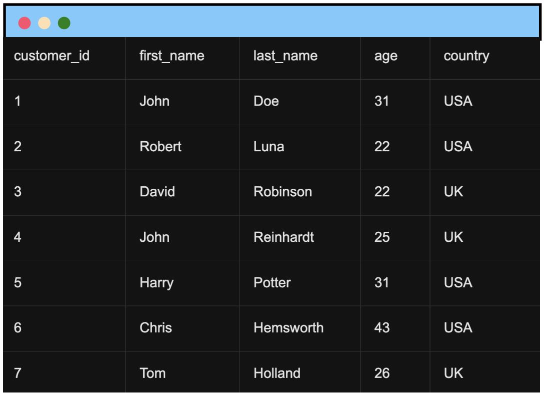
Create (INSERT) Example



customer_id	first_name	last_name	age	country
1	John	Doe	31	USA
2	Robert	Luna	22	USA
3	David	Robinson	22	UK
4	John	Reinhardt	25	UK

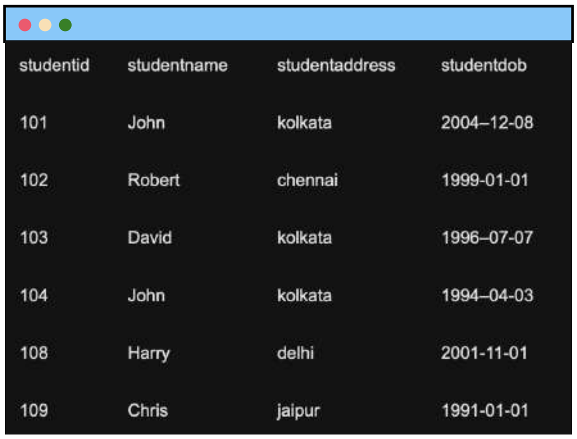


```
INSERT INTO Customers(first_name, last_name, age, country)
VALUES ('Harry', 'Potter', 31, 'USA'),
       ('Chris', 'Hemsworth', 43, 'USA'),
       ('Tom', 'Holland', 26, 'UK');
```

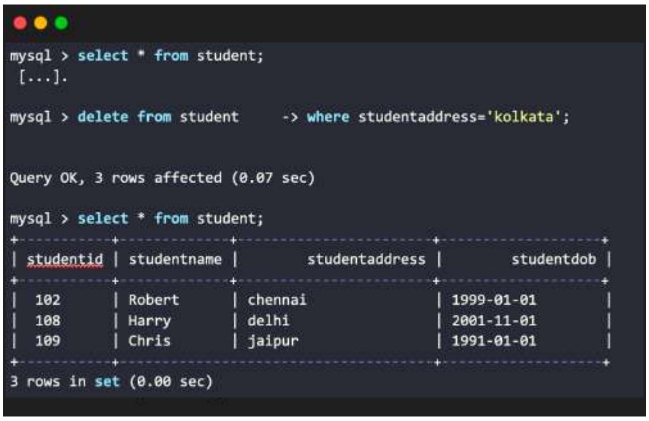


customer_id	first_name	last_name	age	country
1	John	Doe	31	USA
2	Robert	Luna	22	USA
3	David	Robinson	22	UK
4	John	Reinhardt	25	UK
5	Harry	Potter	31	USA
6	Chris	Hemsworth	43	USA
7	Tom	Holland	26	UK

DELETE Example



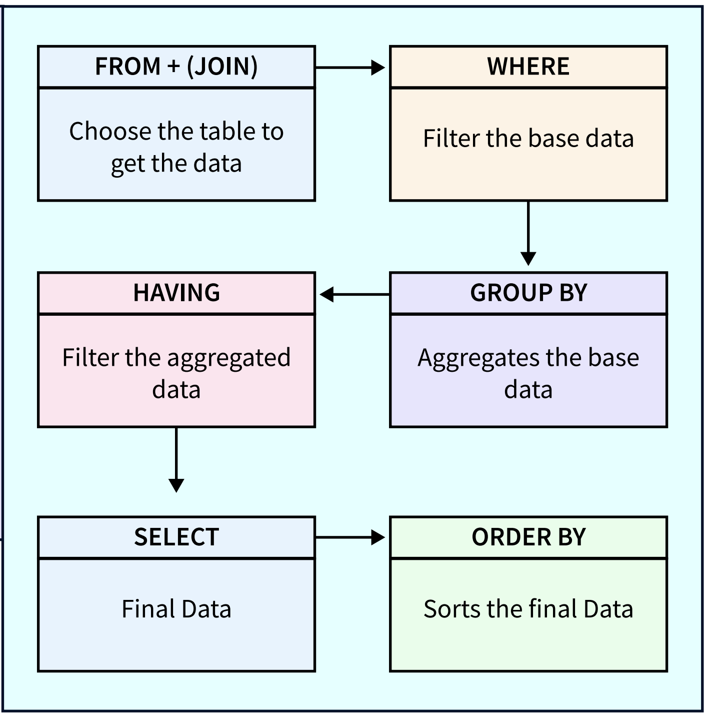
studentid	studentname	studentaddress	studentdob
101	John	kolkata	2004–12-08
102	Robert	chennai	1999-01-01
103	David	kolkata	1996–07-07
104	John	kolkata	1994–04-03
108	Harry	delhi	2001-11-01
109	Chris	jaipur	1991-01-01



```
mysql > select * from student;
 [...].

mysql > delete from student     -> where studentaddress='kolkata';


Query OK, 3 rows affected (0.07 sec)

mysql > select * from student;
+-----------+-------------+---------------------------+------------------+
| studentid | studentname |            studentaddress |        studentdob |
+-----------+-------------+---------------------------+------------------+
|  102      | Robert      | chennai                   | 1999-01-01       |
|  108      | Harry       | delhi                     | 2001-11-01       |
|  109      | Chris       | jaipur                    | 1991-01-01       |
+-----------+-------------+---------------------------+------------------+
3 rows in set (0.00 sec)
```

SQL Clauses are components of SQL statements that defines specific conditions or actions.

WHERE Clause	ORDER BY Clause
ARRAYTABLE	LIMIT Clause
GROUP BY Clause	INTO Clause
HAVING Clause	OPTION Clause



FROM + (JOIN): Choose the table to get the data

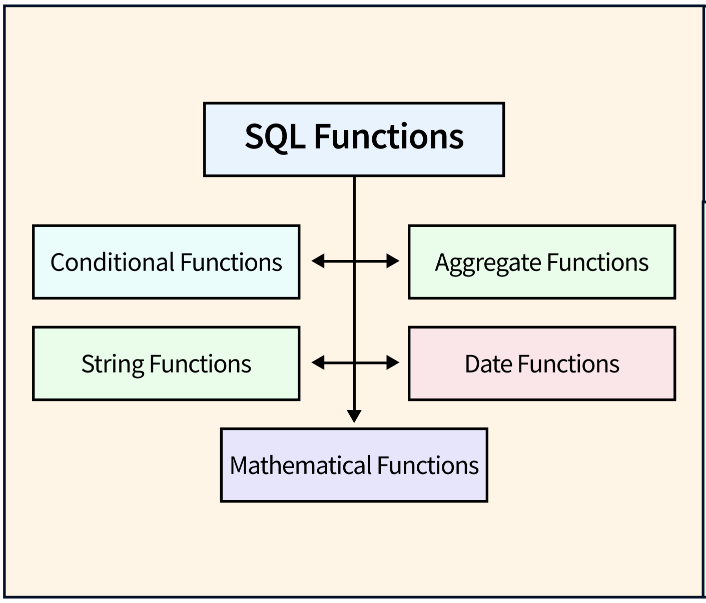
WHERE: Filter the base data

GROUP BY: Aggregates the base data

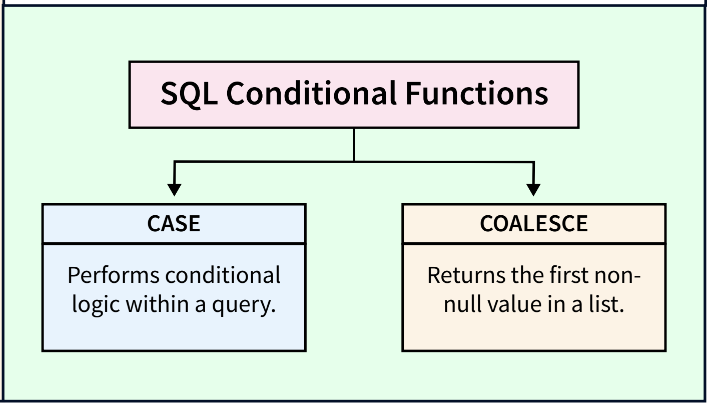
HAVING: Filter the aggregated data

SELECT: Final Data

ORDER BY: Sorts the final Data



Conditional Functions in SQL are used to introduce logic and make decisions based on a specified conditions within a query.



SQL Conditional Functions

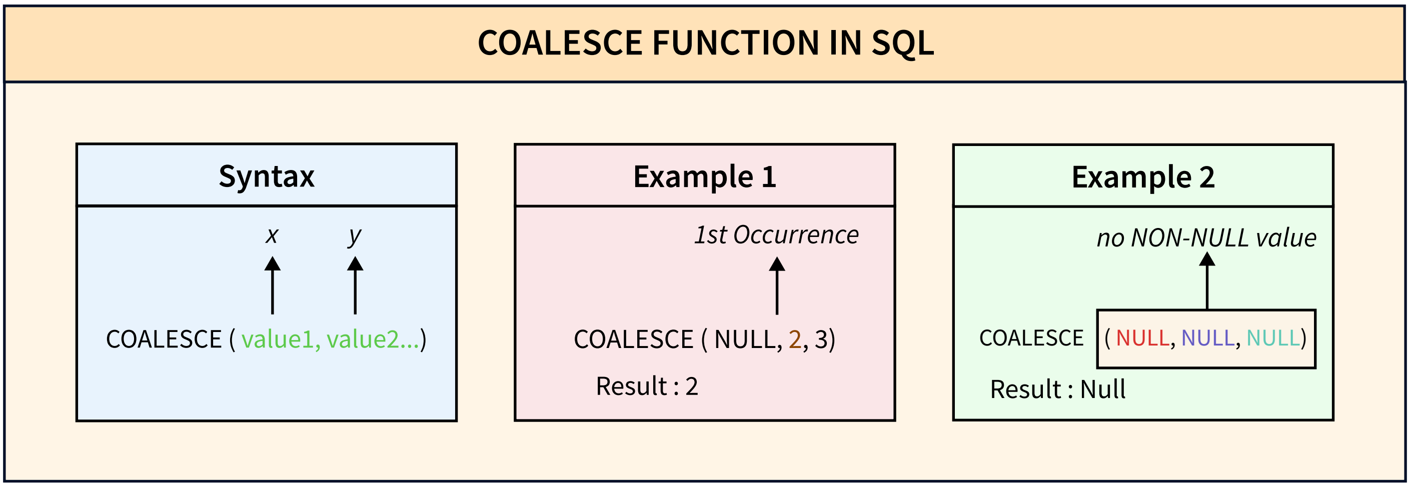
CASE: Performs conditional logic within a query.

COALESCE: Returns the first non-null value in a list.

CASE FUNCTION IN SQL

```
SELECT CASE WHEN -1<1 THEN TRUE ELSE
FALSE END AS RESULT;
```

RESULT	TRUE



COALESCE FUNCTION IN SQL

Syntax

x y

COALESCE (value1, value2...)

Example 1

1st Occurrence

COALESCE (NULL, 2, 3)

Result : 2

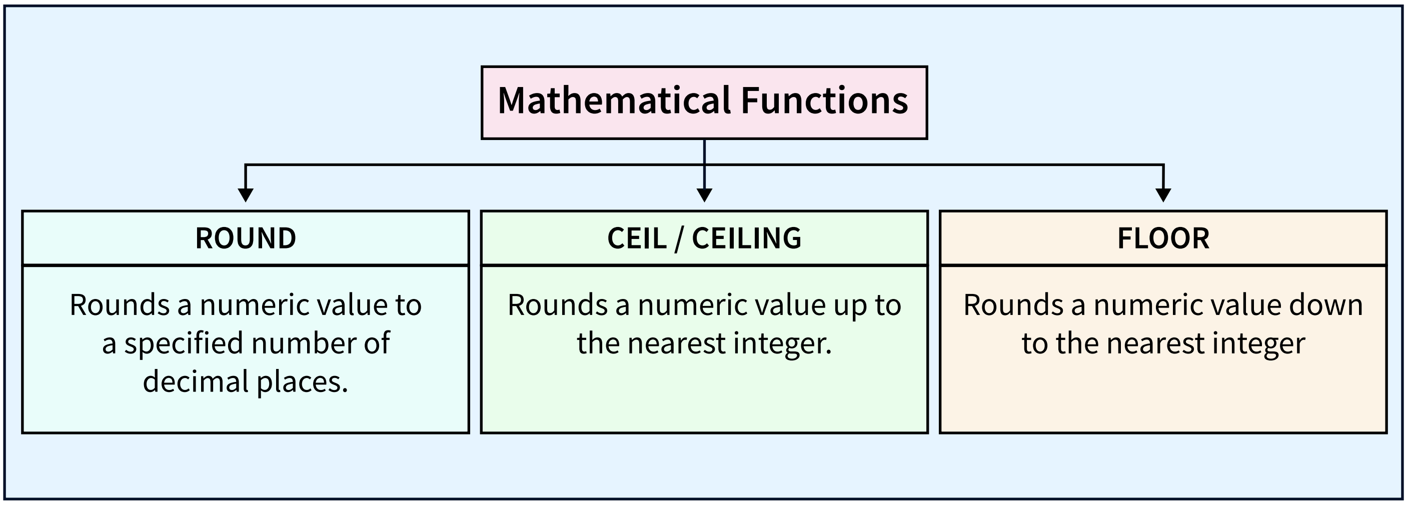
Example 2

no NON-NULL value

COALESCE (NULL, NULL, NULL)

Result : Null

Mathematical functions in SQL are used to perform various mathematcal operations on numeric data types.



Mathematical Functions

ROUND

Rounds a numeric value to a specified number of decimal places.

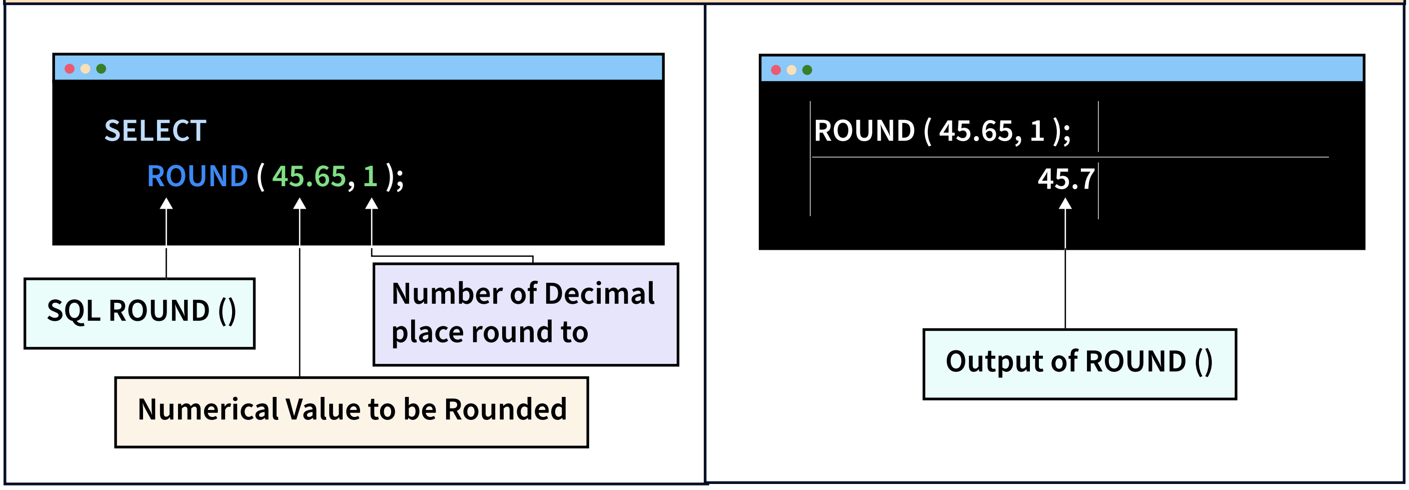
CEIL / CEILING

Rounds a numeric value up to the nearest integer.

FLOOR

Rounds a numeric value down to the nearest integer

ROUND Function in SQL

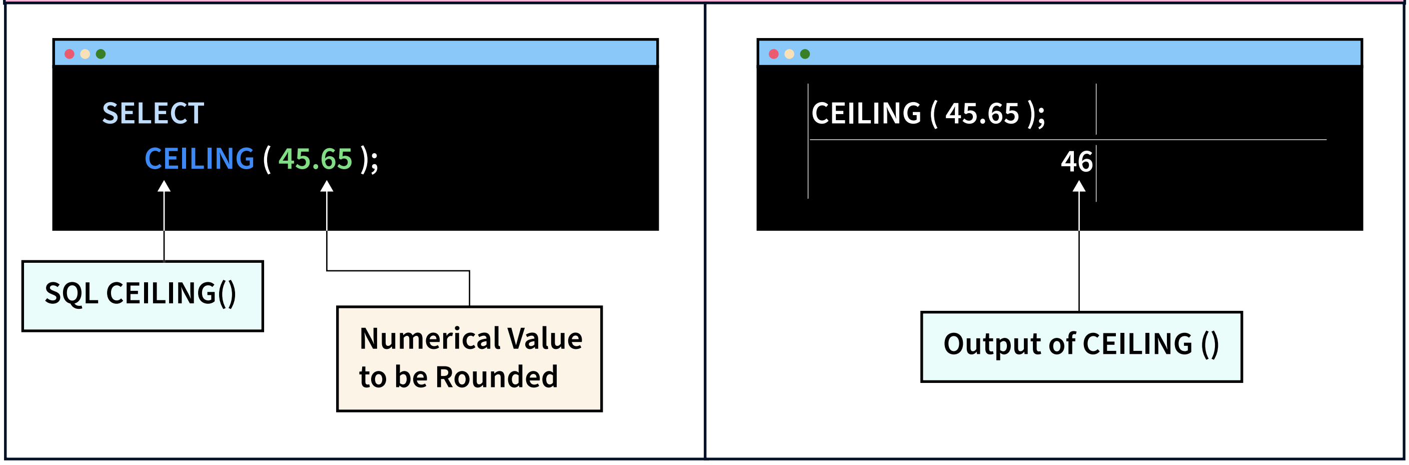


```
SELECT
    ROUND ( 45.65, 1 );
```

```
ROUND ( 45.65, 1 );
                45.7
```

SQL ROUND ()
Numerical Value to be Rounded
Number of Decimal place round to
Output of ROUND ()

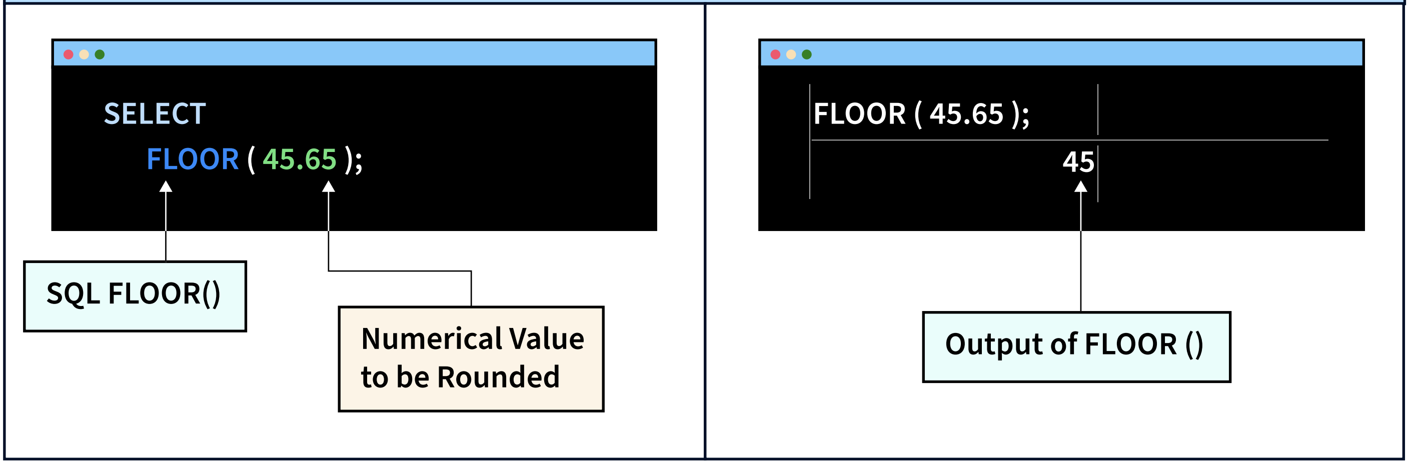
CEIL FUNCTION IN SQL



```
SELECT
    CEILING ( 45.65 );
```

```
CEILING ( 45.65 );
                  46
```

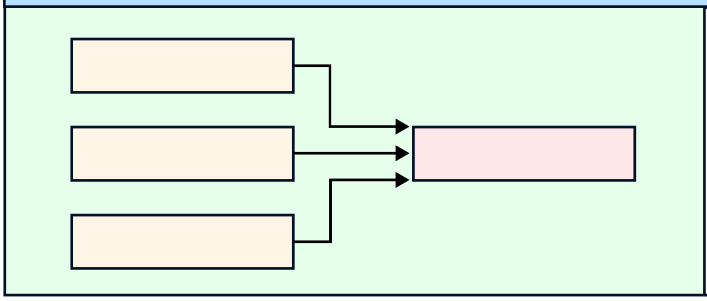
FLOOR FUNCTION IN SQL



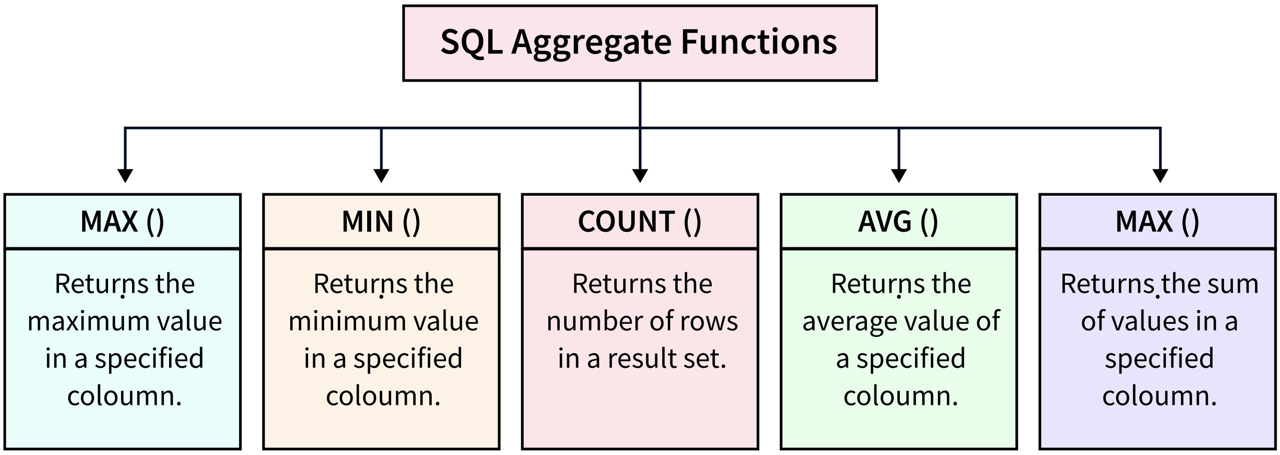
```
SELECT
    FLOOR ( 45.65 );
```

```
FLOOR ( 45.65 );
                45
```

Aggregate Functions



SQL Aggregate Functions perform calculations on a set of values and return a single result.



SQL Aggregate Functions

MAX ()

Returns the maximum value in a specified coloumn.

MIN ()

Returns the minimum value in a specified coloumn.

COUNT ()

Returns the number of rows in a result set.

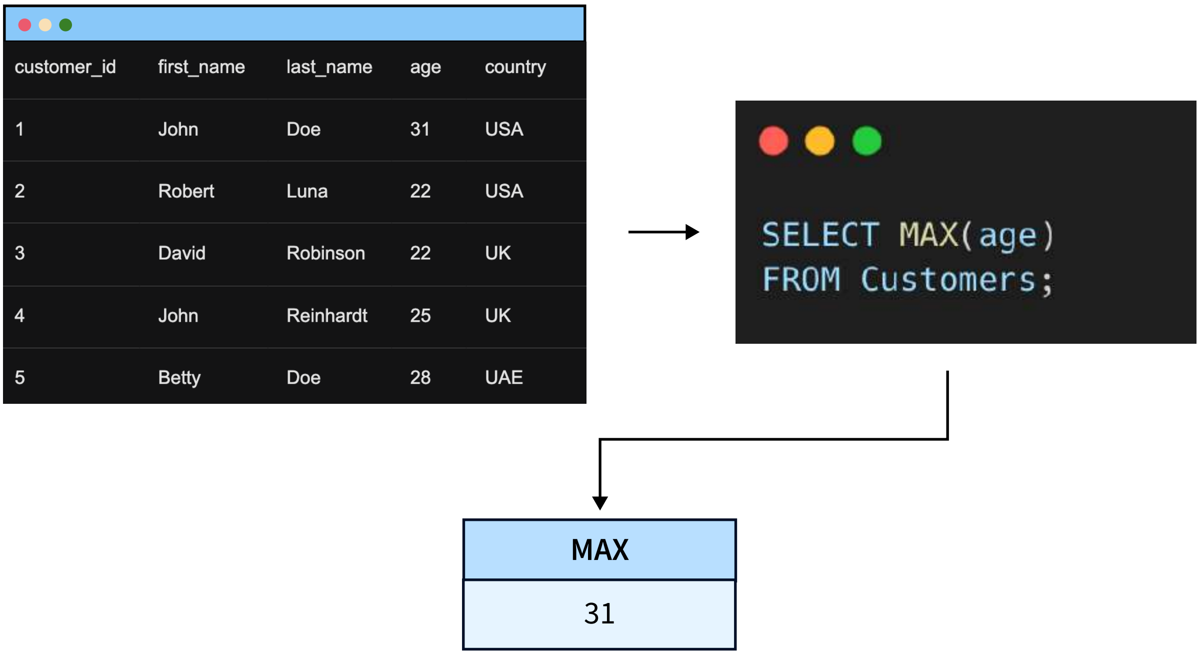
AVG ()

Returns the average value of a specified coloumn.

MAX ()

Returns the sum of values in a specified coloumn.

MAX Function in SQL

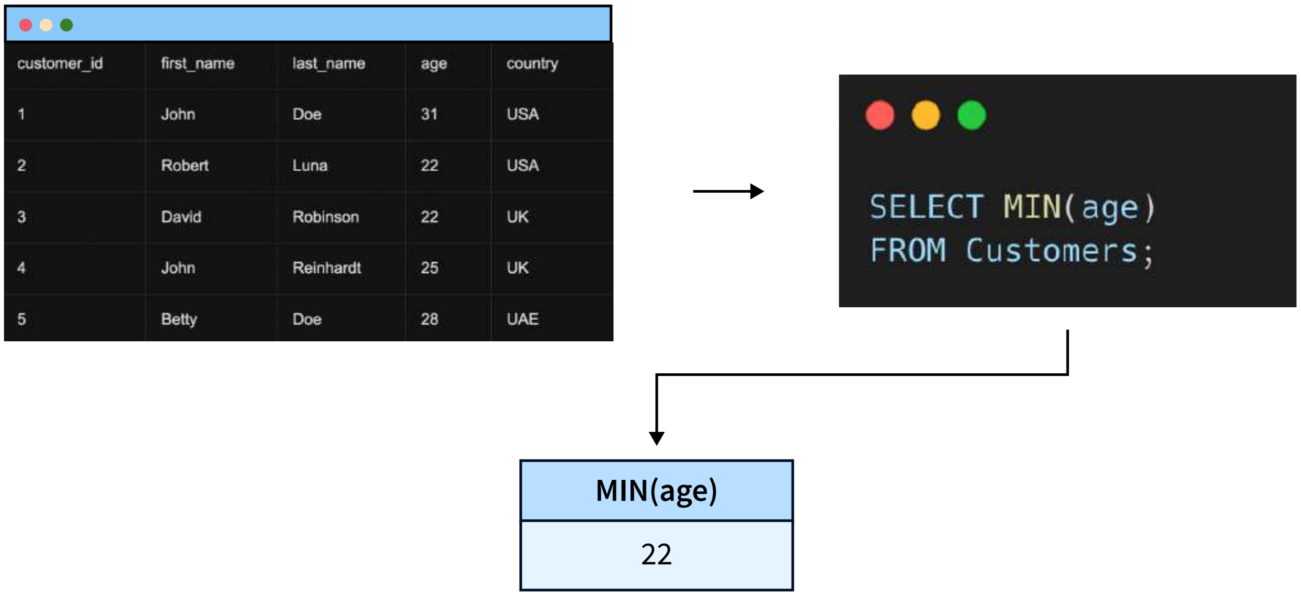


customer_id	first_name	last_name	age	country
1	John	Doe	31	USA
2	Robert	Luna	22	USA
3	David	Robinson	22	UK
4	John	Reinhardt	25	UK
5	Betty	Doe	28	UAE

```
SELECT MAX(age)
FROM Customers;
```

MAX
31

MIN Function in SQL

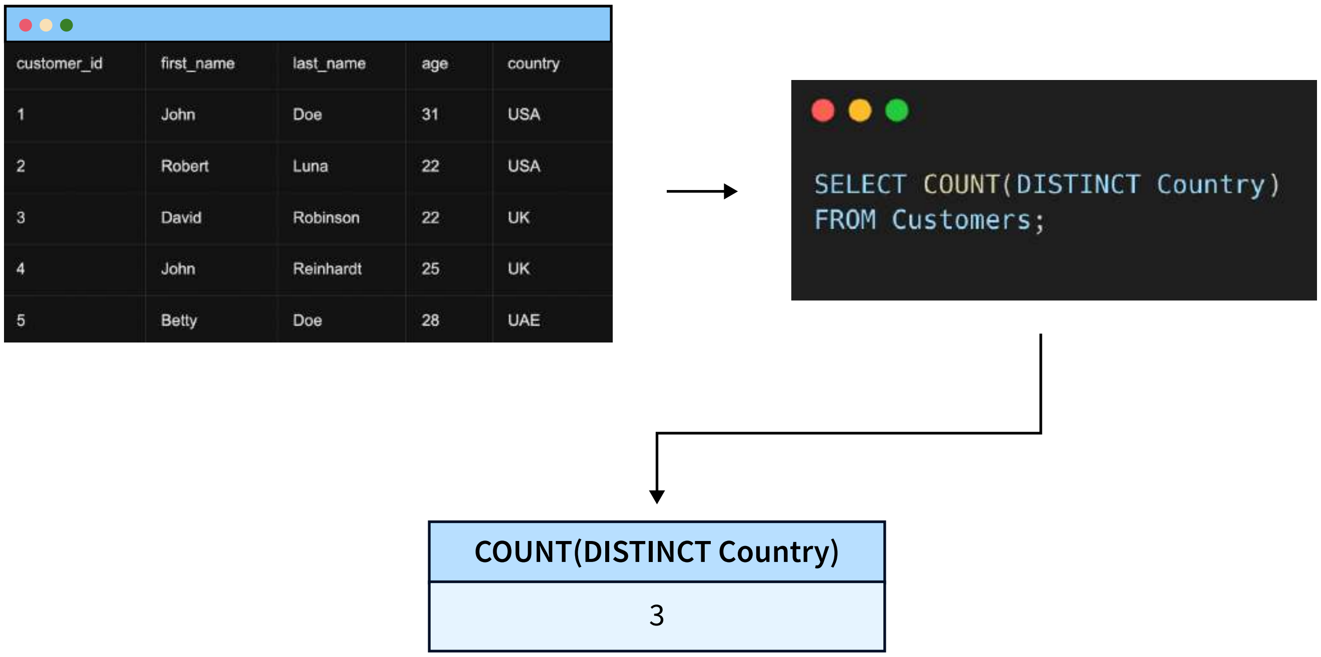


customer_id	first_name	last_name	age	country
1	John	Doe	31	USA
2	Robert	Luna	22	USA
3	David	Robinson	22	UK
4	John	Reinhardt	25	UK
5	Betty	Doe	28	UAE

```
SELECT MIN(age)
FROM Customers;
```

MIN(age)
22

COUNT Function in SQL

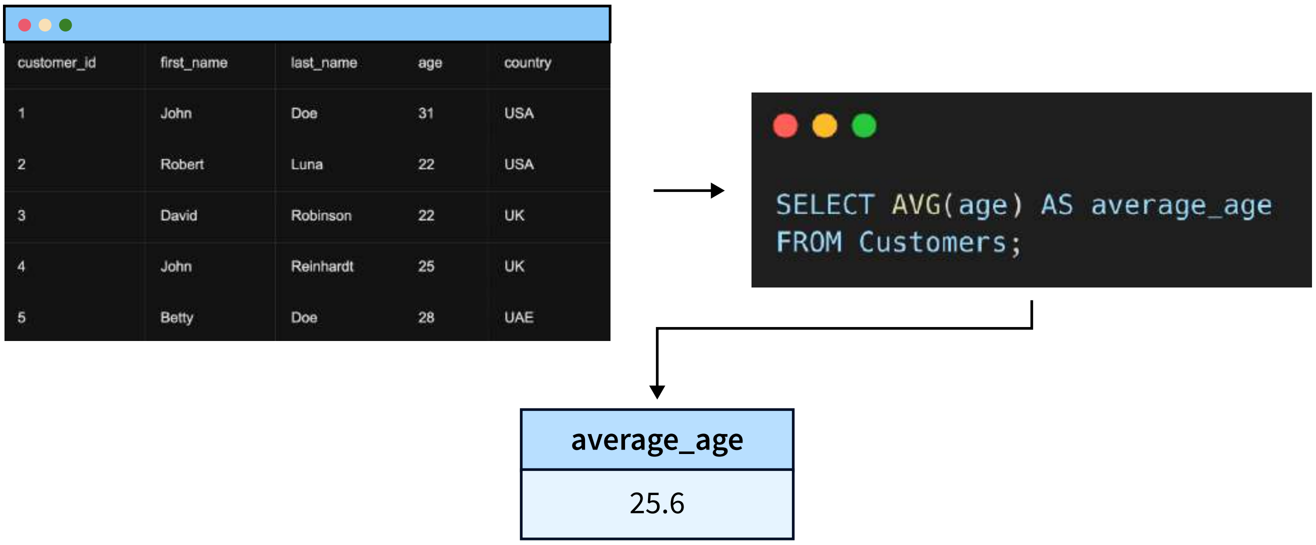


customer_id	first_name	last_name	age	country
1	John	Doe	31	USA
2	Robert	Luna	22	USA
3	David	Robinson	22	UK
4	John	Reinhardt	25	UK
5	Betty	Doe	28	UAE

```
SELECT COUNT(DISTINCT Country)
FROM Customers;
```

COUNT(DISTINCT Country)
3

AVG Function SQL

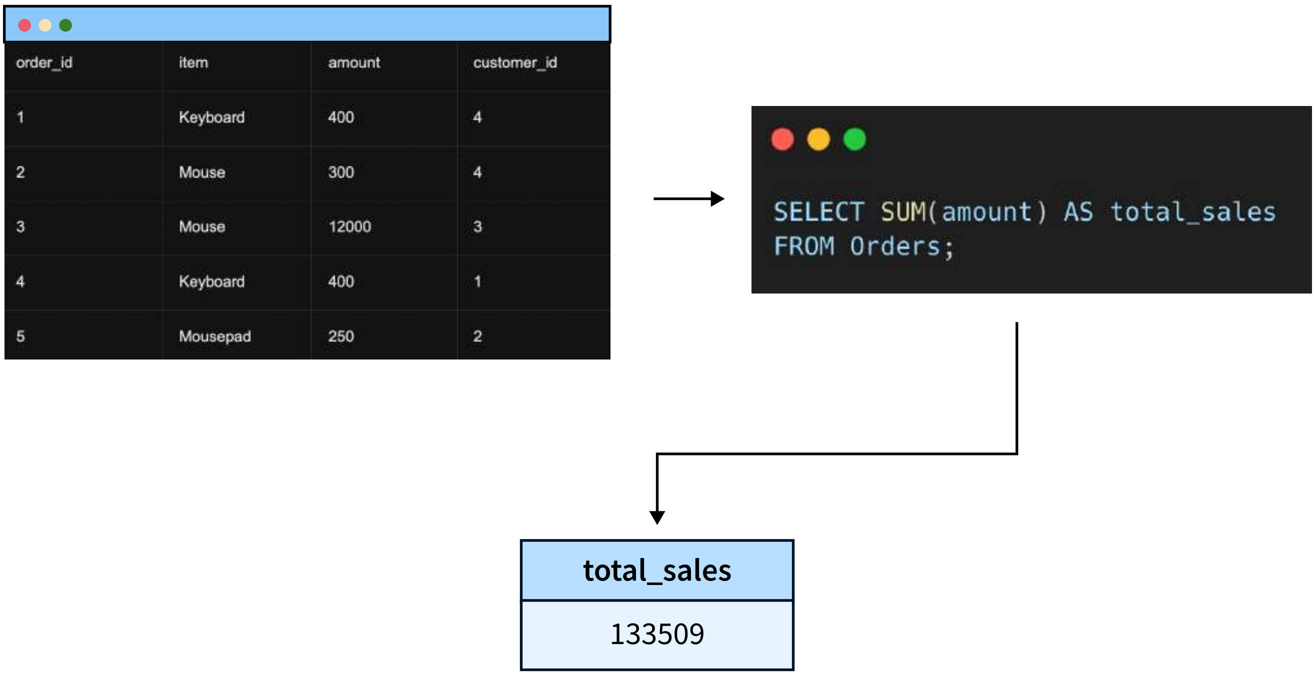


customer_id	first_name	last_name	age	country
1	John	Doe	31	USA
2	Robert	Luna	22	USA
3	David	Robinson	22	UK
4	John	Reinhardt	25	UK
5	Betty	Doe	28	UAE

```
SELECT AVG(age) AS average_age
FROM Customers;
```

average_age
25.6

SUM Function in SQL

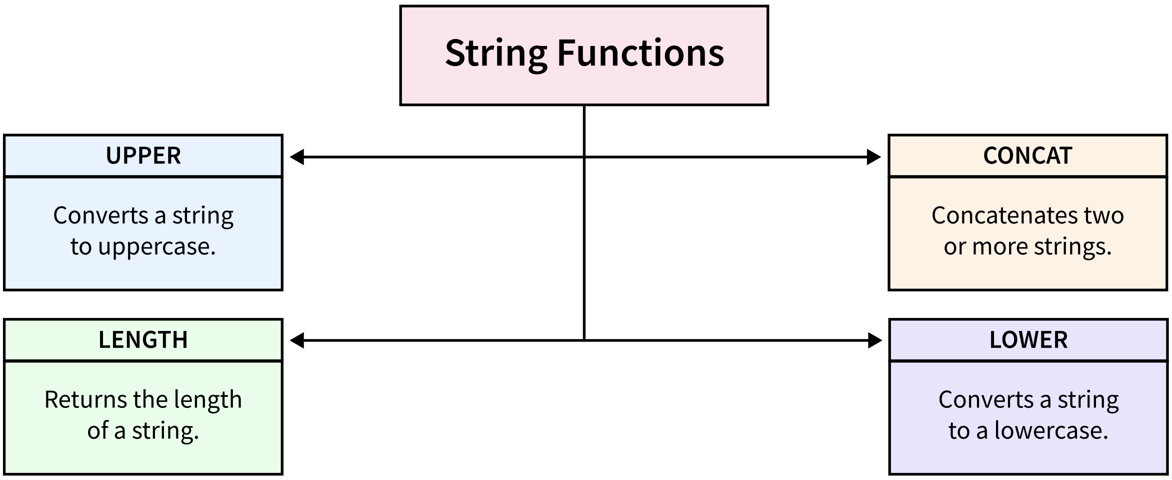


order_id	item	amount	customer_id
1	Keyboard	400	4
2	Mouse	300	4
3	Mouse	12000	3
4	Keyboard	400	1
5	Mousepad	250	2

```
SELECT SUM(amount) AS total_sales
FROM Orders;
```

total_sales
133509

String Function in SQL are used to manipulate and perform operations on character data, typically text values.



String Functions

UPPER

Converts a string to uppercase.

CONCAT

Concatenates two or more strings.

LENGTH

Returns the length of a string.

LOWER

Converts a string to a lowercase.

UPPER Function in SQL

Syntax :

UPPER (str)

Example :

UPPER ('test')

test → UPPER () → TEST

passing through function

LOWER Function in SQL

Syntax :

LOWER (str)

Example :

LOWER ('TEST')

TEST → LOWER () → test

passing through function

CONCAT Function in SQL

Syntax :

CONCAT (string1, string2, string 3, ...)

Example :

concat (" May be A-Z ", " or ", " may be 0-9 ")

Output

May be A-Z or may be 0-9

LENGTH Function in SQL

Syntax :

LENGTH (str)

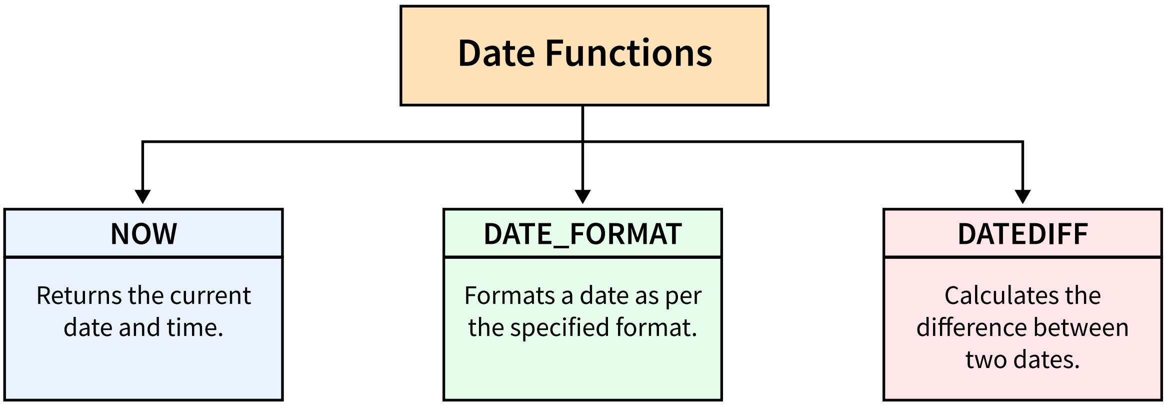
Example :

LENGTH ('TEST')

TEST → LENGTH () → 4

passing through function

Date functions in SQL are used to perform operations on date and time data types. These functions help in manipulating, extracting, and formatting date values.



Date Functions

NOW

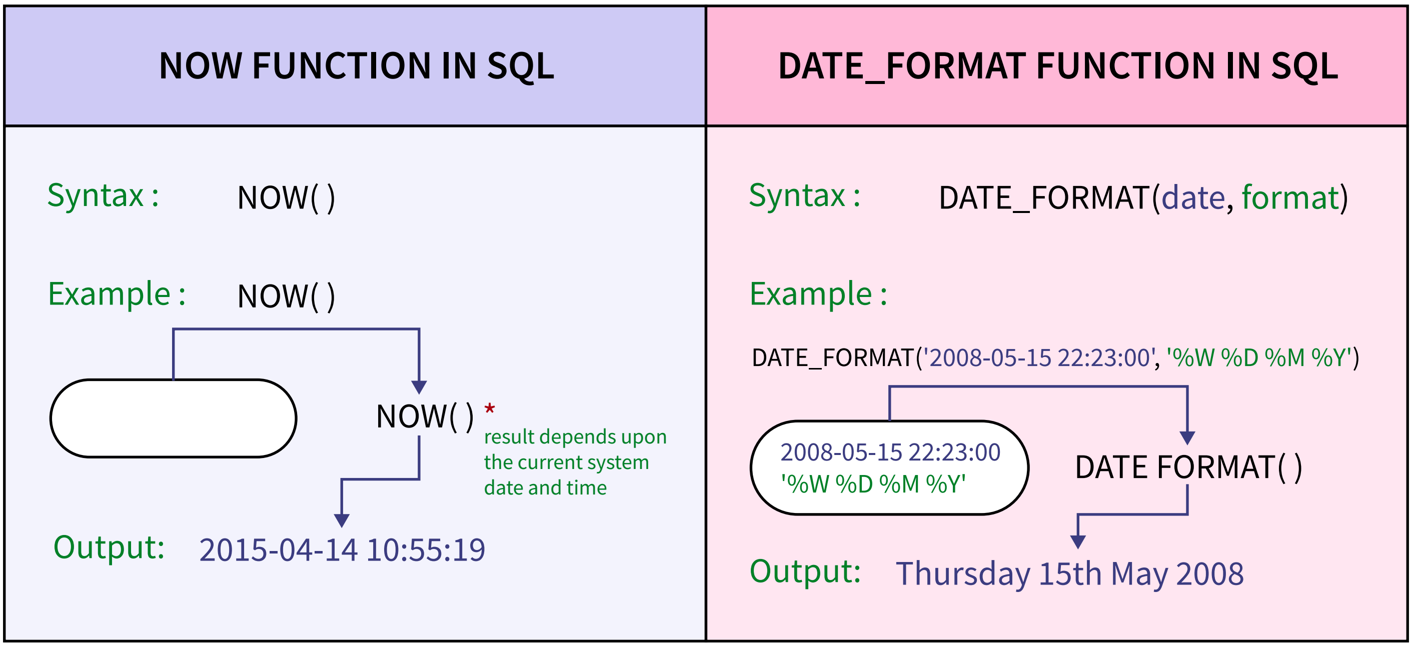
Returns the current date and time.

DATE_FORMAT

Formats a date as per the specified format.

DATEDIFF

Calculates the difference between two dates.



NOW FUNCTION IN SQL

Syntax : NOW( )

Example : NOW( )

NOW( ) * result depends upon the current system date and time

Output: 2015-04-14 10:55:19

DATE_FORMAT FUNCTION IN SQL

Syntax : DATE_FORMAT(date, format)

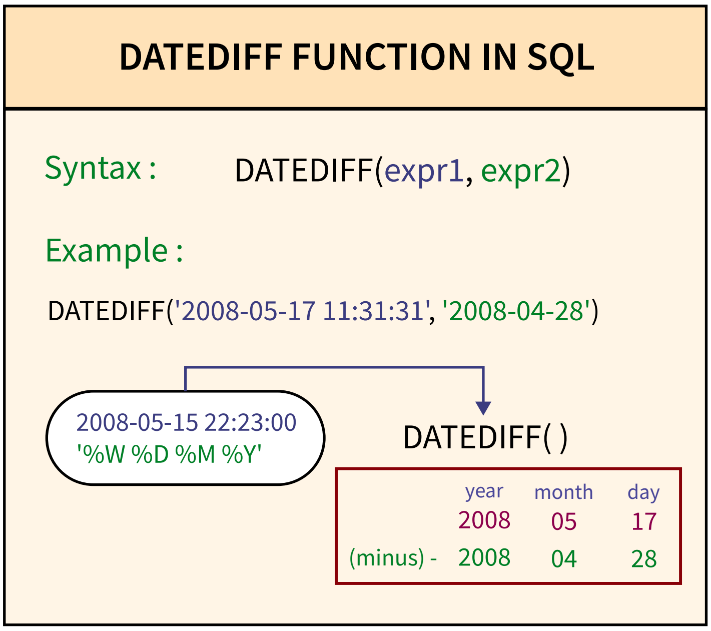
Example :

DATE_FORMAT('2008-05-15 22:23:00', '%W %D %M %Y')

2008-05-15 22:23:00
'%W %D %M %Y'

DATE FORMAT( )

Output: Thursday 15th May 2008



DATEDIFF FUNCTION IN SQL

Syntax : DATEDIFF(expr1, expr2)

Example :

DATEDIFF('2008-05-17 11:31:31', '2008-04-28')

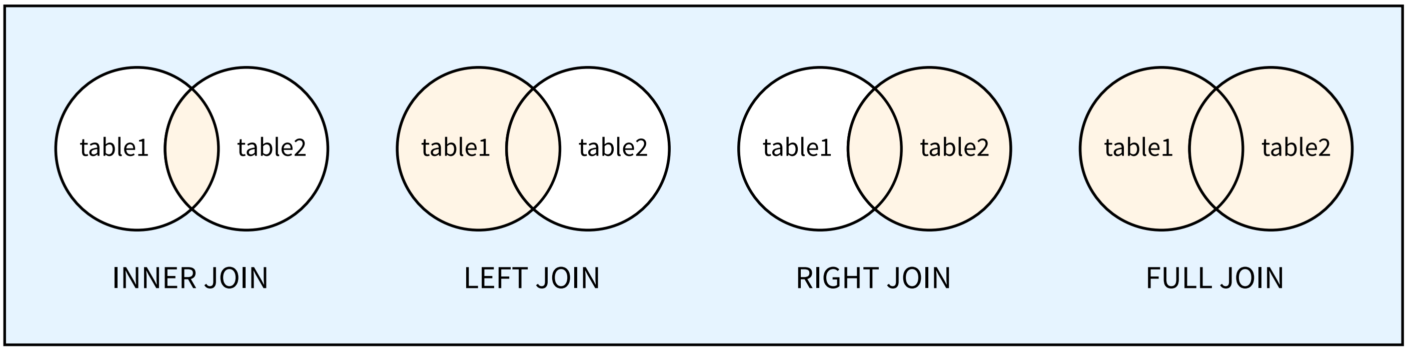
2008-05-15 22:23:00
'%W %D %M %Y'

DATEDIFF( )

	year	month	day
	2008	05	17
(minus) -	2008	04	28

Joins in SQL

SOL JOIN is used to combine rows from two or more tables based on a related column between them.



INNER JOIN

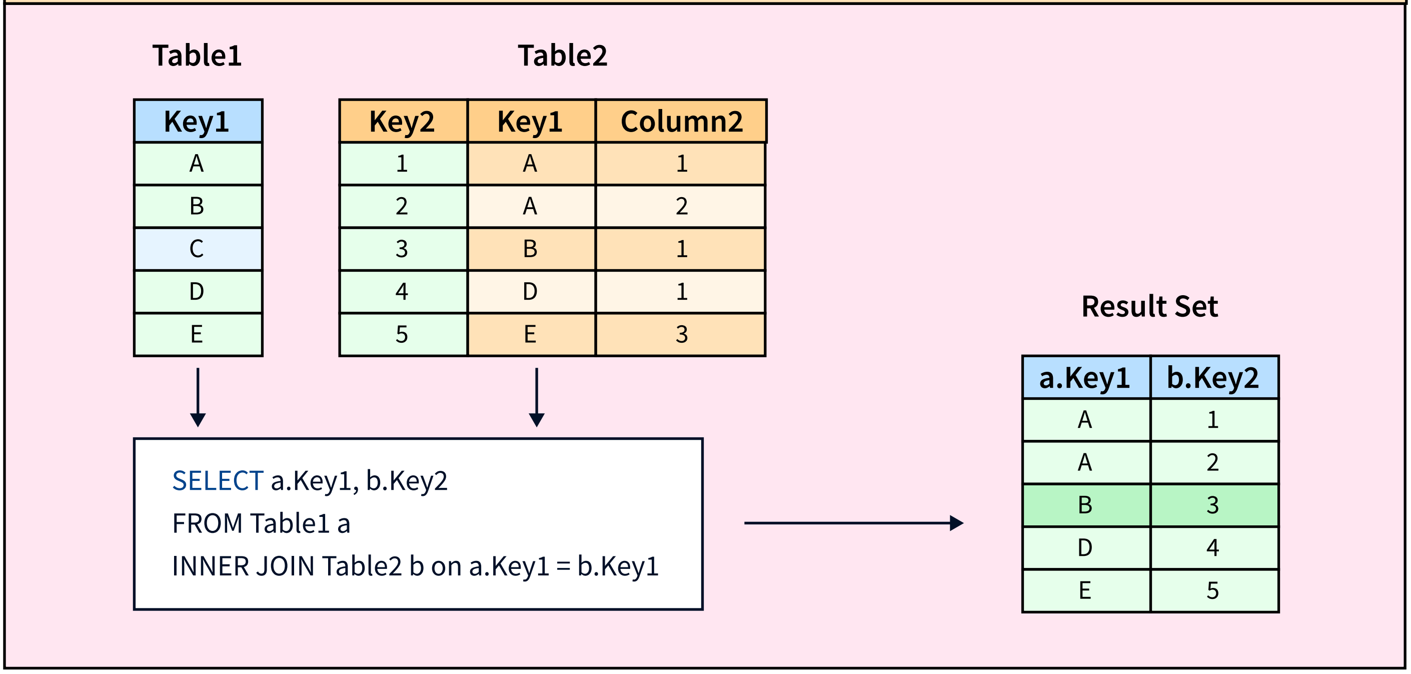


Table1

Key1
A
B
C
D
E

Table2

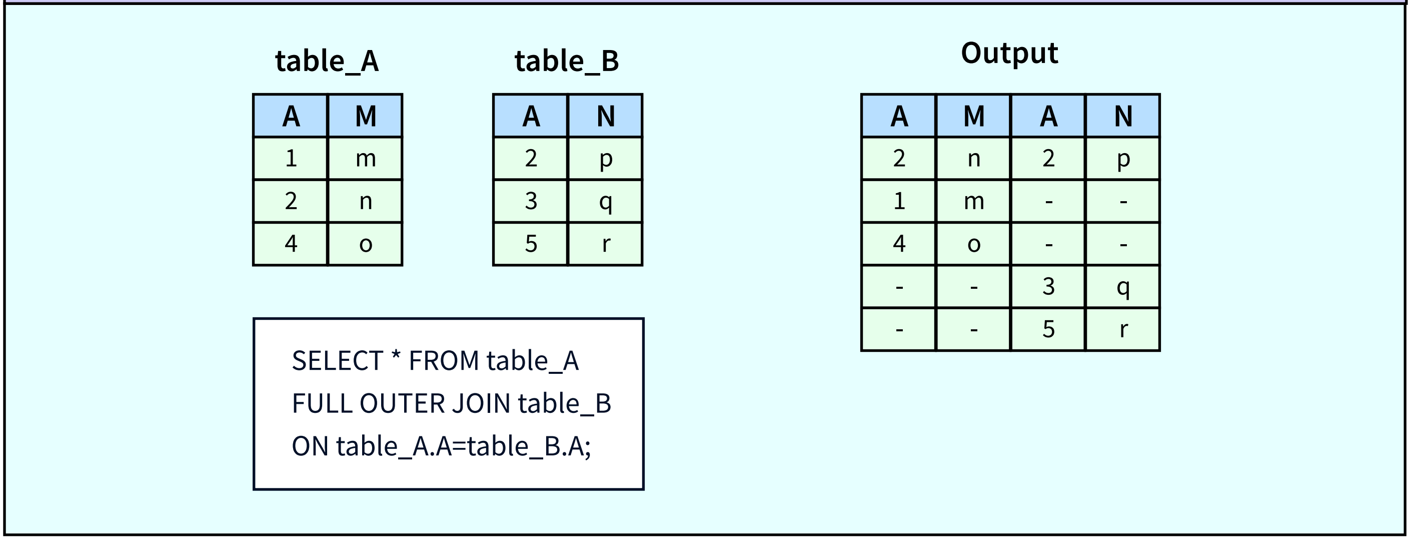
Key2	Key1	Column2
1	A	1
2	A	2
3	B	1
4	D	1
5	E	3

```
SELECT a.Key1, b.Key2
FROM Table1 a
INNER JOIN Table2 b on a.Key1 = b.Key1
```

Result Set

a.Key1	b.Key2
A	1
A	2
B	3
D	4
E	5

OUTER JOIN



table_A

A	M
1	m
2	n
4	o

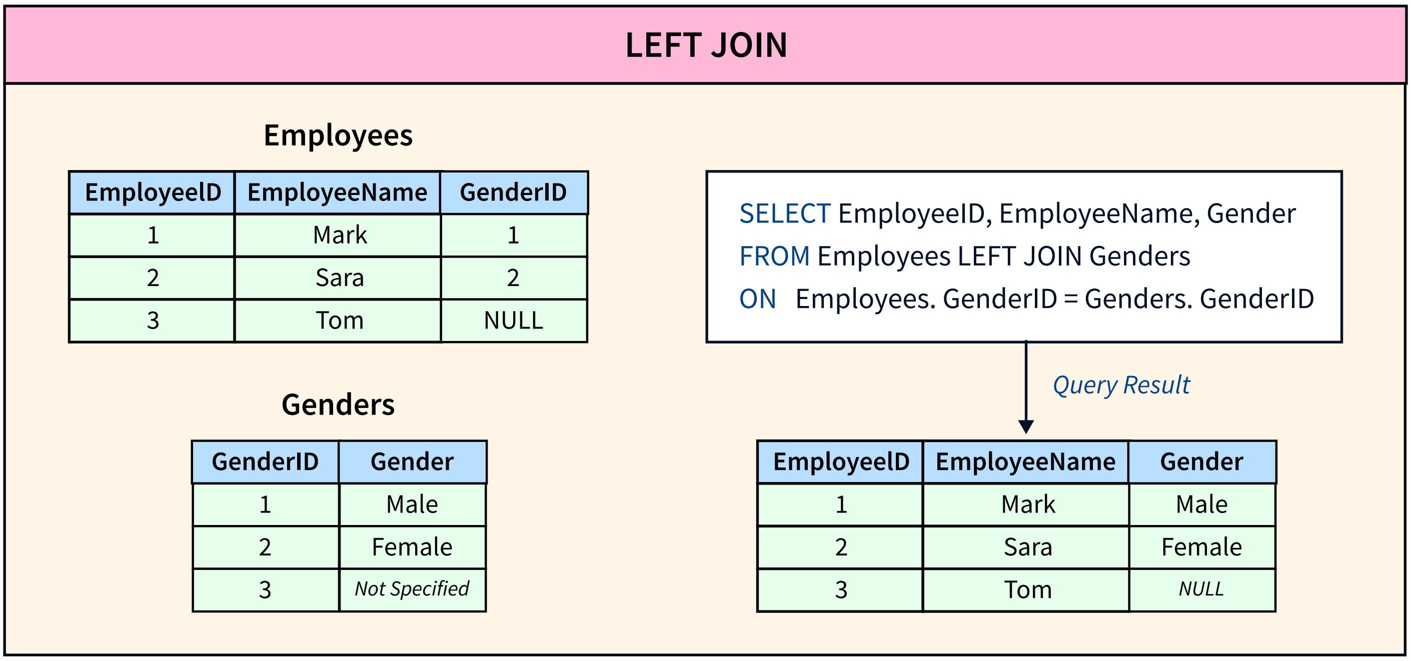
table_B

A	N
2	p
3	q
5	r

```
SELECT * FROM table_A
FULL OUTER JOIN table_B
ON table_A.A=table_B.A;
```

Output

A	M	A	N
2	n	2	p
1	m	-	-
4	o	-	-
-	-	3	q
-	-	5	r



LEFT JOIN

Employees

EmployeeID	EmployeeName	GenderID
1	Mark	1
2	Sara	2
3	Tom	NULL

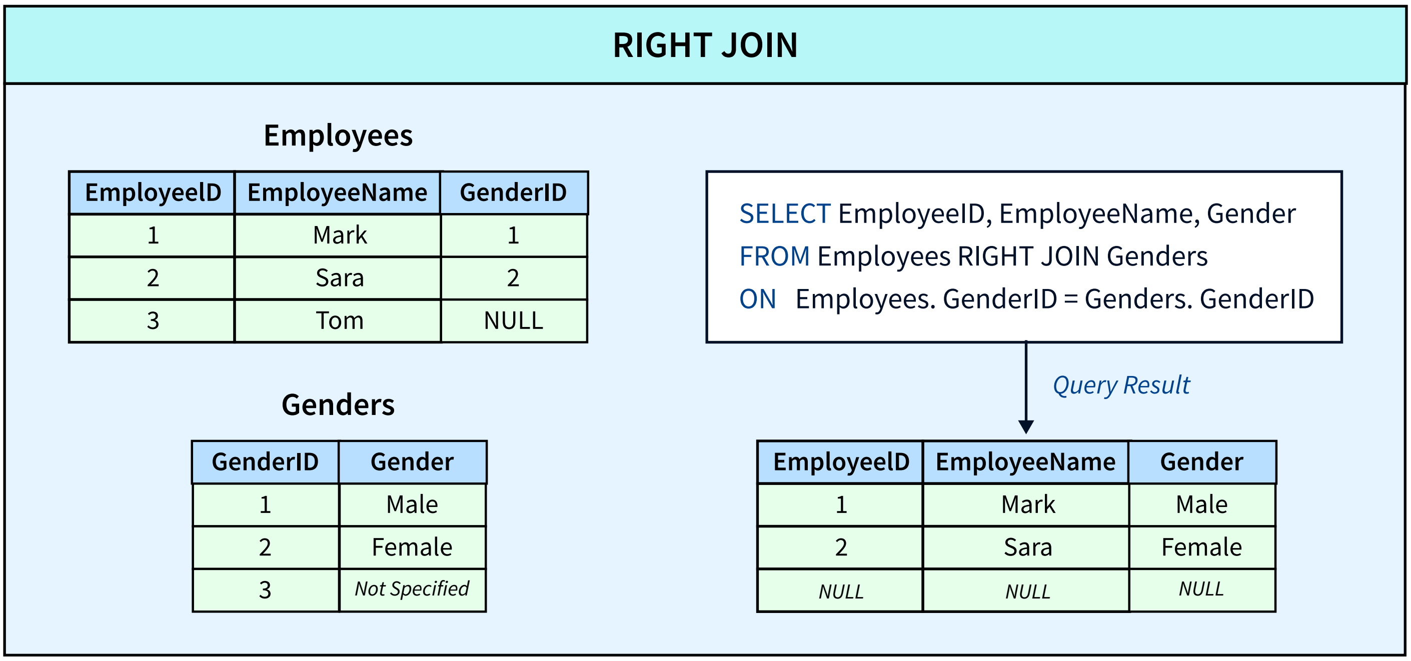
Genders

GenderID	Gender
1	Male
2	Female
3	*Not Specified*

```sql
SELECT EmployeeID, EmployeeName, Gender
FROM Employees LEFT JOIN Genders
ON  Employees. GenderID = Genders. GenderID
```

Query Result

EmployeeID	EmployeeName	Gender
1	Mark	Male
2	Sara	Female
3	Tom	*NULL*



RIGHT JOIN

Employees

EmployeeID	EmployeeName	GenderID
1	Mark	1
2	Sara	2
3	Tom	NULL

Genders

GenderID	Gender
1	Male
2	Female
3	*Not Specified*

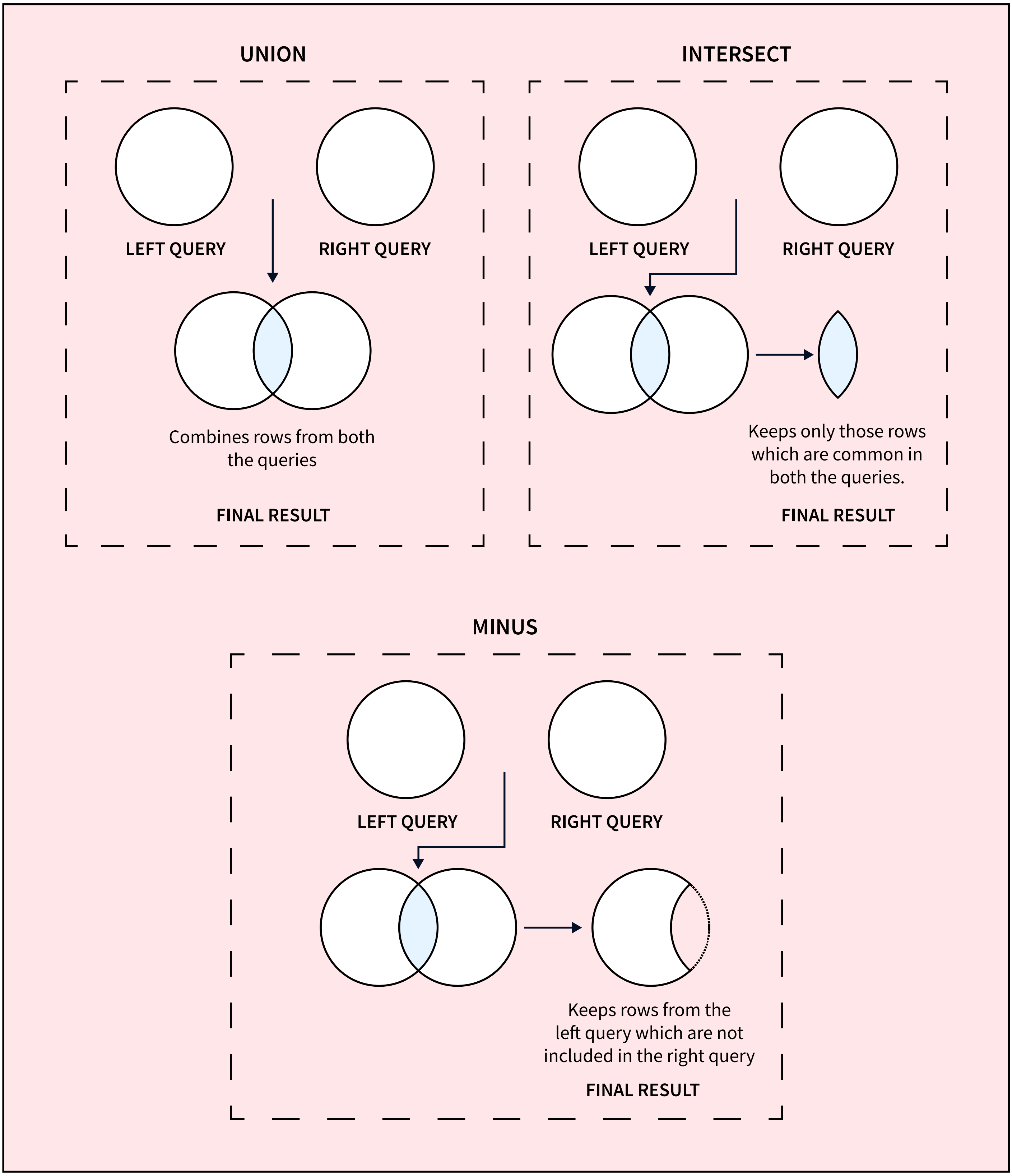
```sql
SELECT EmployeeID, EmployeeName, Gender
FROM Employees RIGHT JOIN Genders
ON  Employees. GenderID = Genders. GenderID
```

Query Result

EmployeeID	EmployeeName	Gender
1	Mark	Male
2	Sara	Female
NULL	*NULL*	*NULL*

SQL SET OPERATIONS

SQL Set Operations are used to combine the results of two or more SELECT queries into a single result set. The common set operations include UNION, INTERSECT, and EXCEPT/MINUS (depending on the SQL variant).

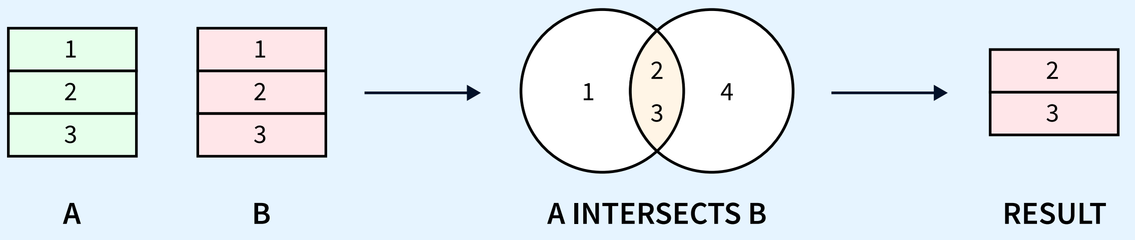


Combines rows from both the queries

Keeps only those rows which are common in both the queries.

Keeps rows from the left query which are not included in the right query

INTERSECT EXAMPLE

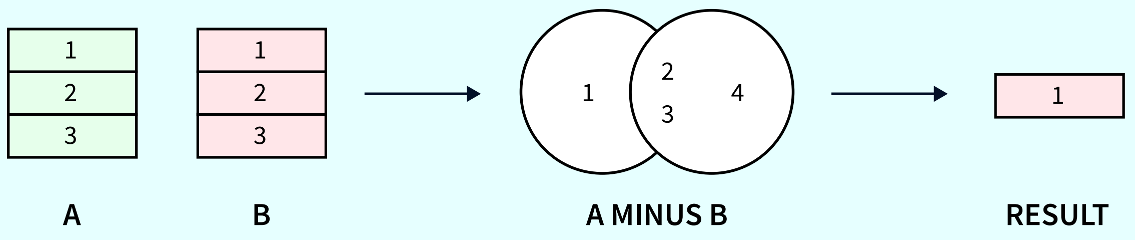


A
1
2
3

B
1
2
3

RESULT
2
3

MINUS EXAMPLE

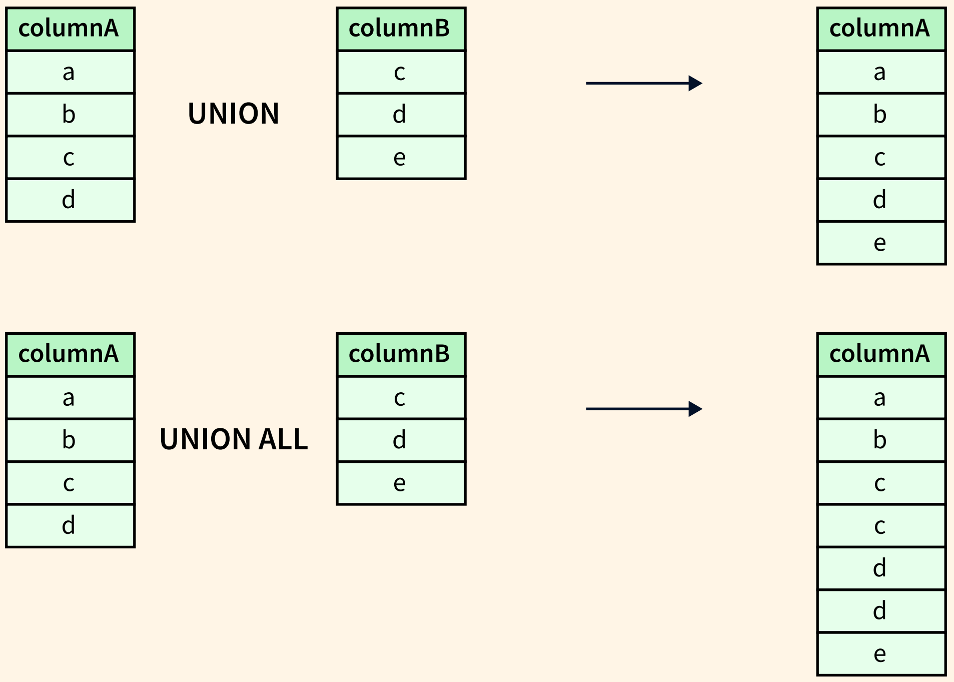


A
1
2
3

B
1
2
3

RESULT
1

UNION / UNION ALL EXAMPLE



columnA
a
b
c
d

UNION

columnB
c
d
e

columnA
a
b
c
d
e

columnA
a
b
c
d

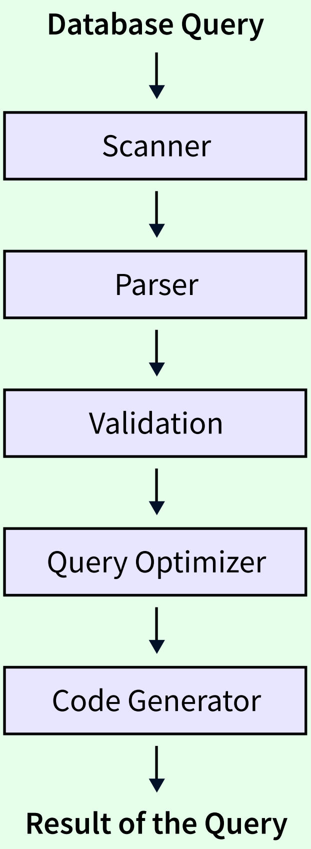
UNION ALL

columnB
c
d
e

columnA
a
b
c
c
d
d
e

SQL QUERY PROCESSING

SQL Query Processing involves parsing, optimizing, and executing SQL queries to retrieve data from databases.



Database Query

Scanner

Parser

Validation

Query Optimizer

Code Generator

Result of the Query

Scanner:

In the SOL Query Processing, the scanner step involves tokenizing the SQL query, breaking it into individual units (tokens) such as keywords, identifiers, literals, and symbols, to facilitate further parsing and analysis.

Parser:

The SQL query is analyzed to ensure its syntax is correct. The query is broken down into components such as keywords, table names, columns, and conditions.

Validation:

During the validation step in SQL Query Processing, the database management system ensures the correctness and integrity of the SQL query by verifying the existence of referenced tables and columns, checking data types, and validating syntax and semantics.

Query Optimization:

The DBMS generates alternative execution plans and selects the most efficient one based on factors like indexing, join strategies & available resources

Code Generator:

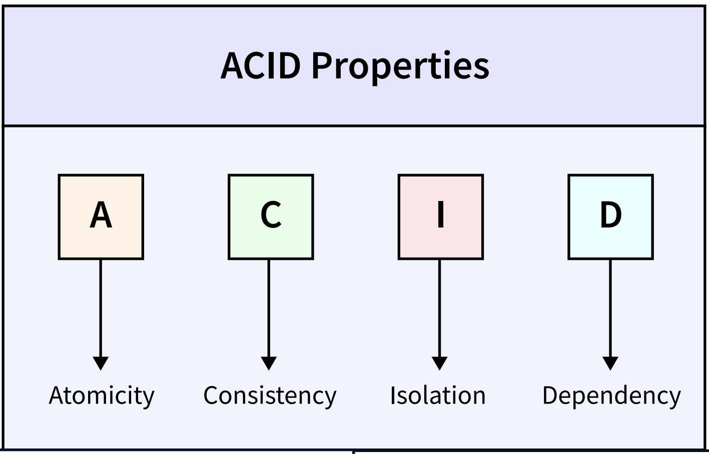
Translates the optimized query plan into executable code, determining how to access and manipulate data in the underlying database system for efficient query execution.

Database Processor:

Executes the generated code, interacting with the underlying database to retrieve, manipulate, or store data according to the user's query.

07 Transaction Management

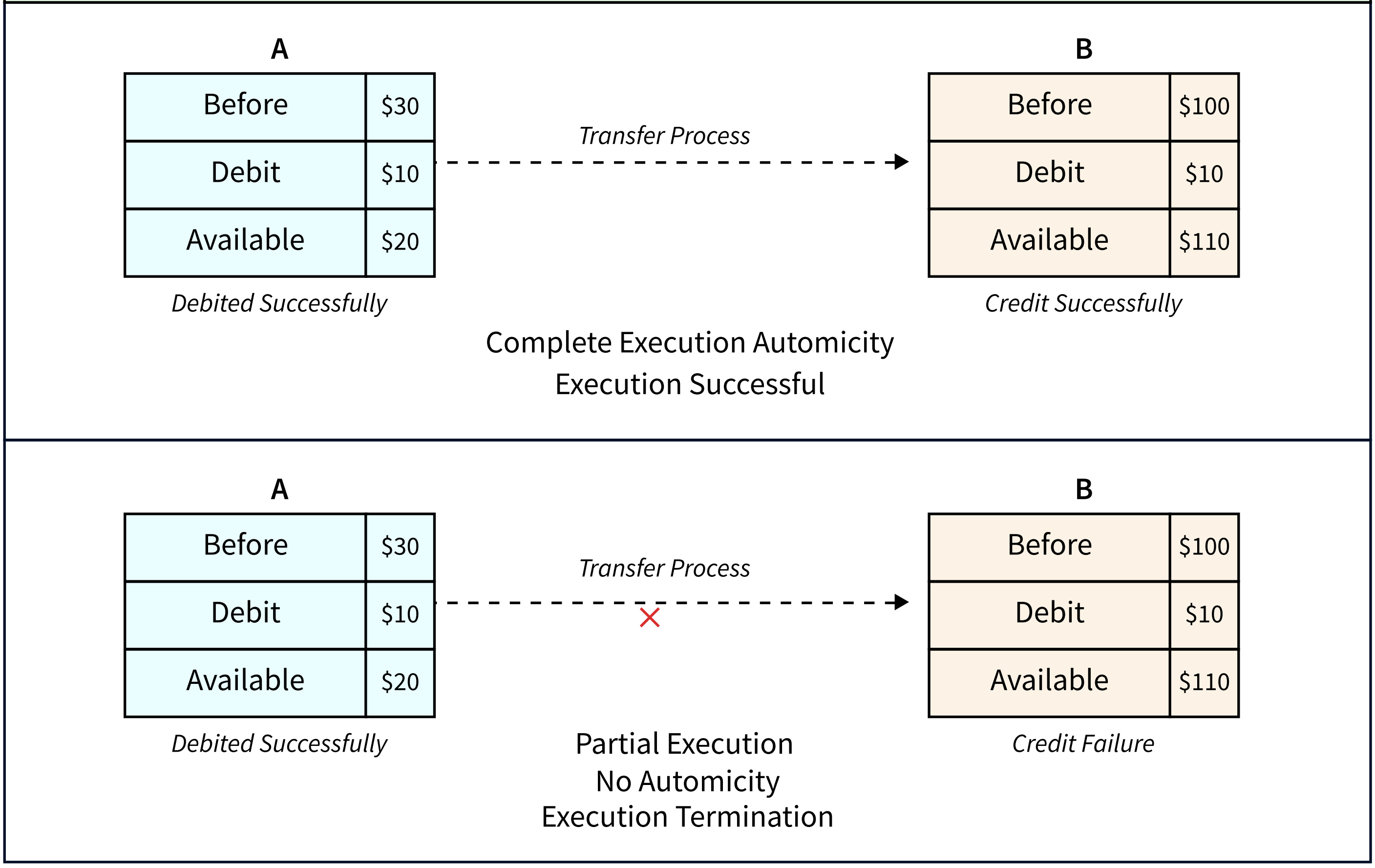
A transaction is a sequence of one or more operations (SQL statements) that are executed as a single, indivisible unit of work. Transaction management in DBMS refers to the process of ensuring the reliable and consistent execution of database transactions.



Atomicity	Consistecy
Each transaction is either properly carried out or the database reverts back to the state before the transaction started.	The database must be in a consistent state before and after the transaction.
Isolation	**Durability**
Multipe transaction occur independently without interference.	Successful transaction are persisted even in the case of system failure.

1 Atomicity	2 Consistecy
If any statement in the transaction fails, the entire transaction fails, and the database is left unchanged.	Transaction must meet all protocols defined by the ssystem -- no partially completed transactions.
3 Isolation	**4 Durability**
No transaction has acccess to any other transaction that is unfinished. Each transaction is independent.	Once a transaction has been committed, it will remain committed through the use of transaction logs and backups.

Atomicity



A

Before	$30
Debit	$10
Available	$20

Debited Successfully

Transfer Process

B

Before	$100
Debit	$10
Available	$110

Credit Successfully

Complete Execution Automicity
Execution Successful

A

Before	$30
Debit	$10
Available	$20

Debited Successfully

Transfer Process

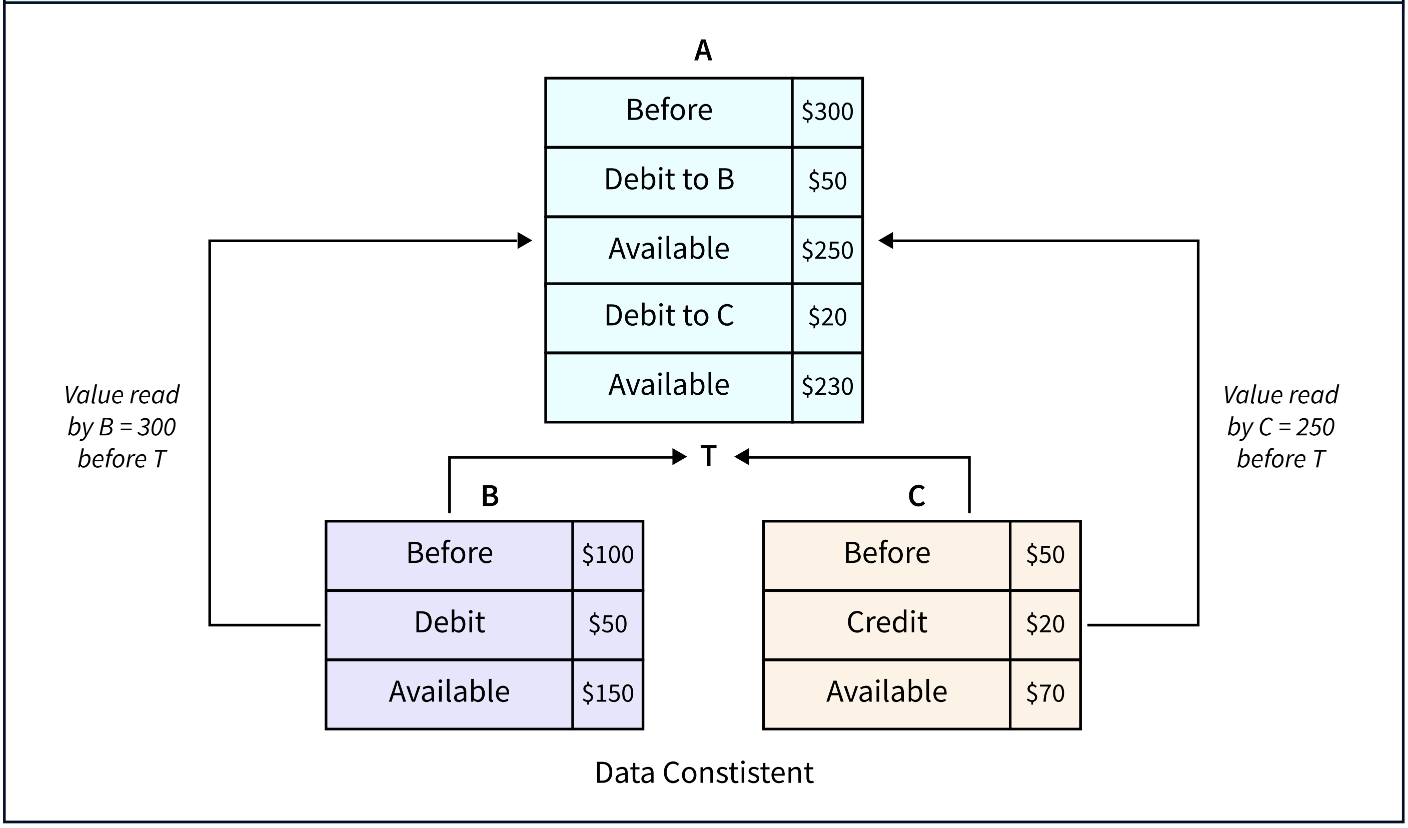
B

Before	$100
Debit	$10
Available	$110

Credit Failure

Partial Execution
No Automicity
Execution Termination

Consistency



A

Before	$300
Debit to B	$50
Available	$250
Debit to C	$20
Available	$230

Value read by B = 300 before T

Value read by C = 250 before T

T

B

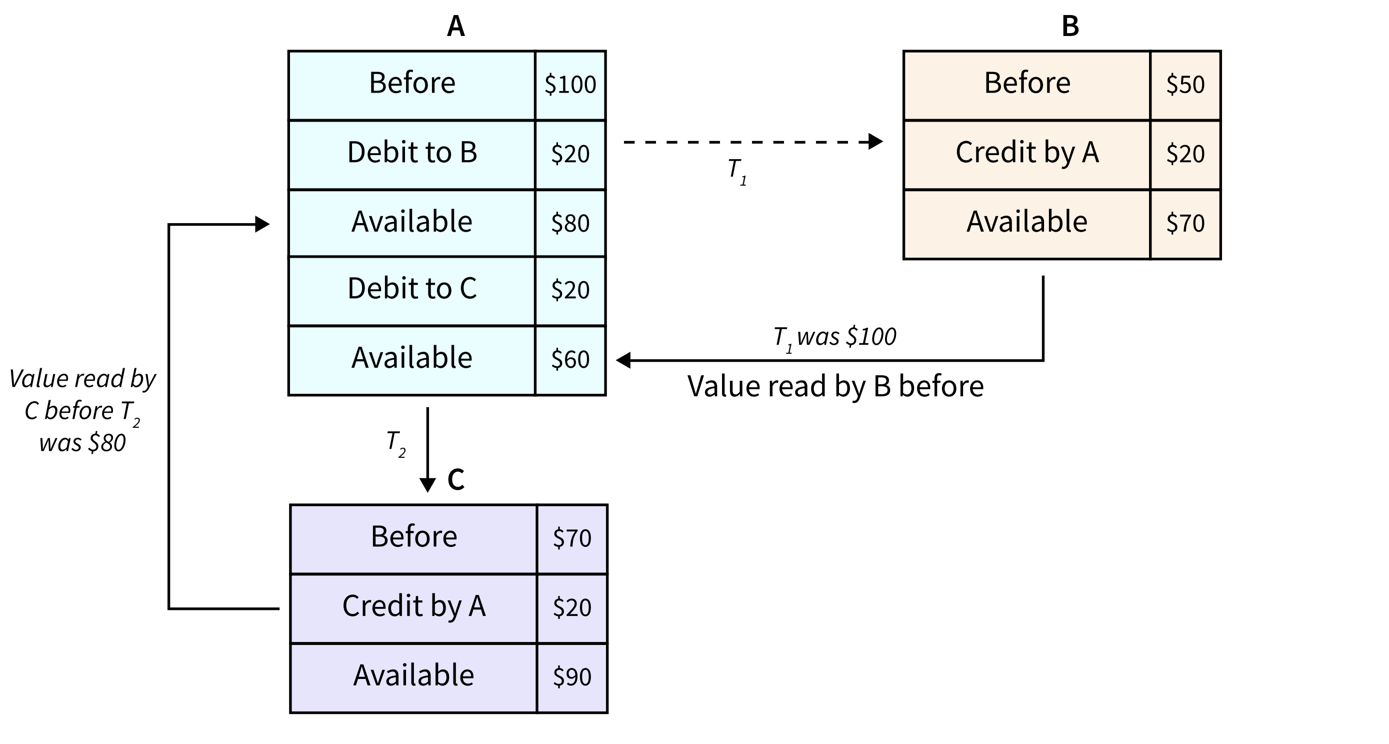
Before	$100
Debit	$50
Available	$150

C

Before	$50
Credit	$20
Available	$70

Data Constistent

Isolation



A

Before	$100
Debit to B	$20
Available	$80
Debit to C	$20
Available	$60

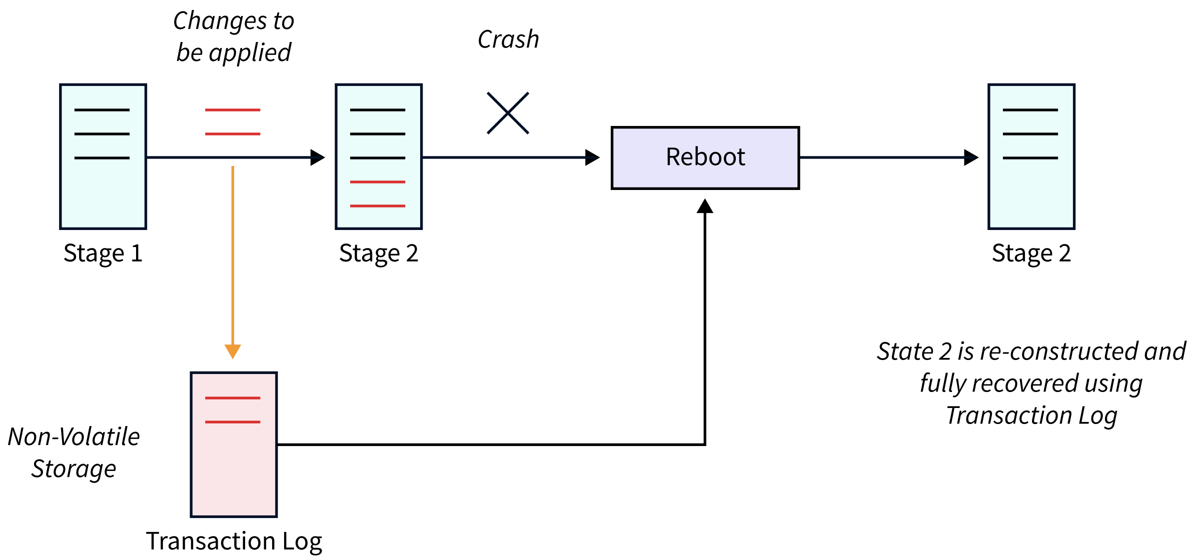
B

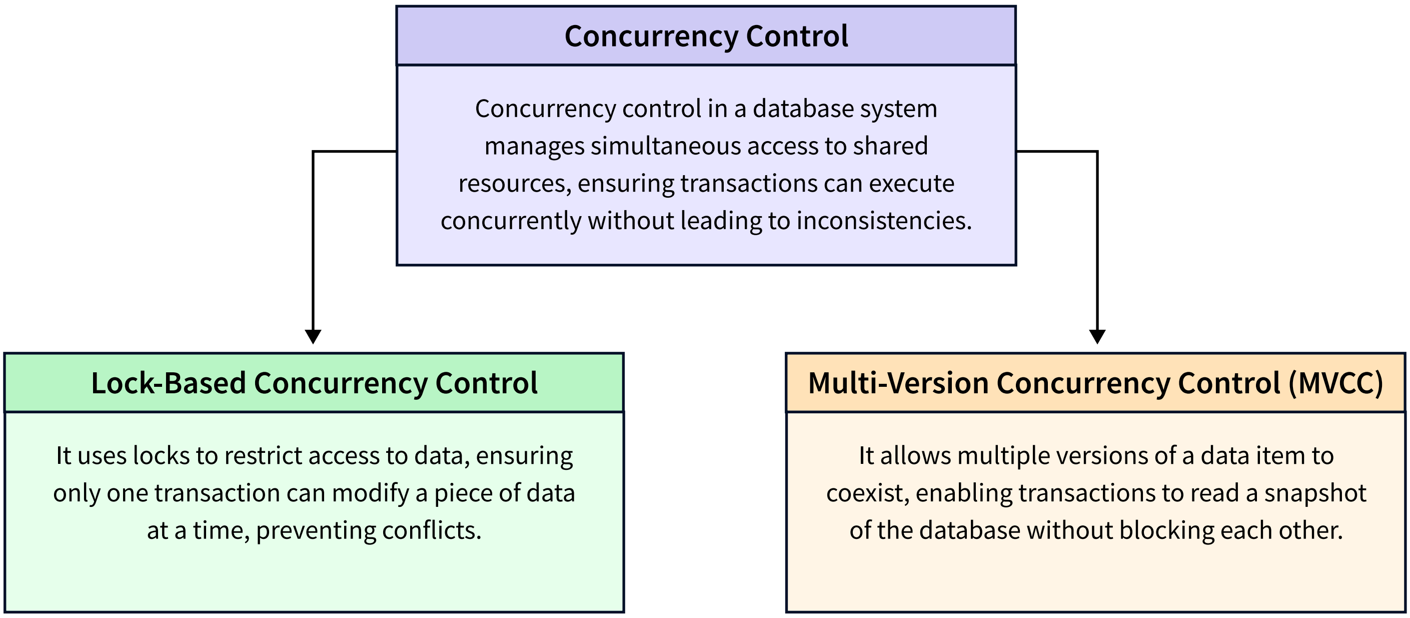
Before	$50
Credit by A	$20
Available	$70

C

Before	$70
Credit by A	$20
Available	$90

Durability





Concurrency Control

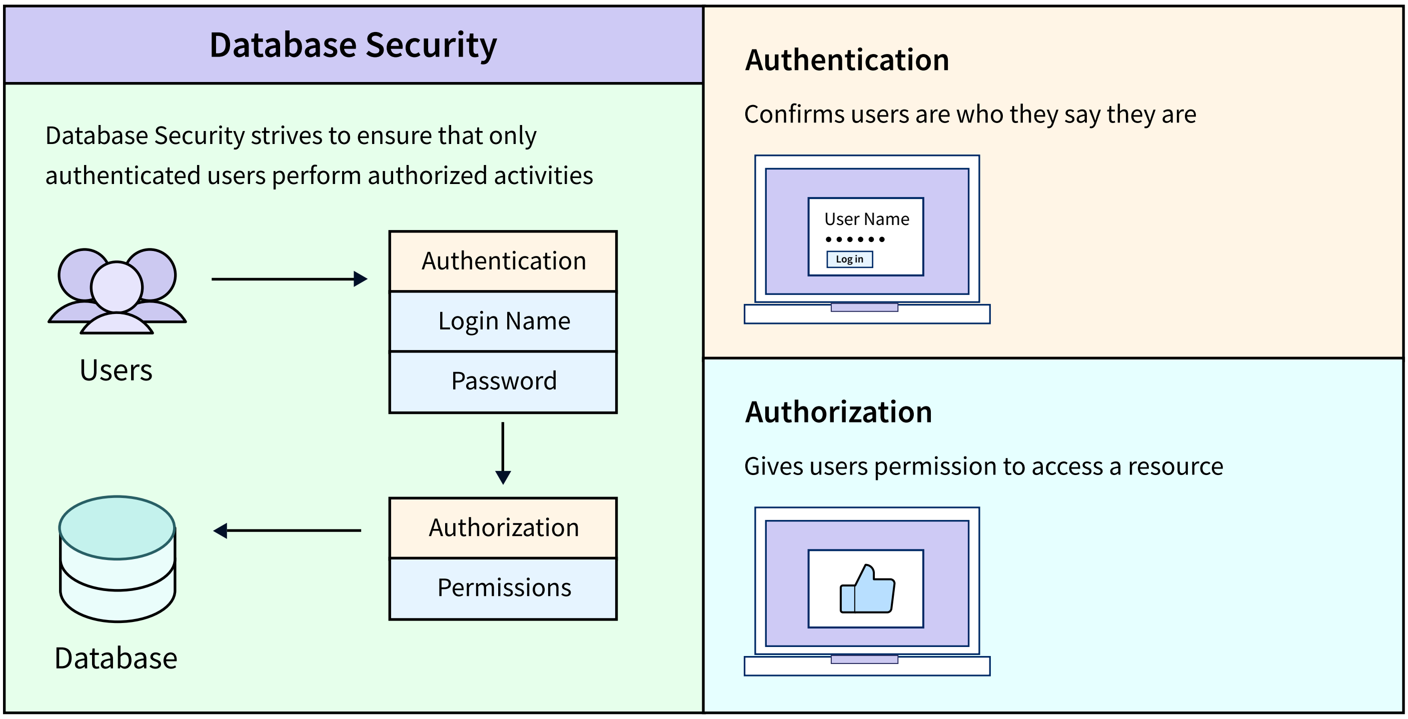
Concurrency control in a database system manages simultaneous access to shared resources, ensuring transactions can execute concurrently without leading to inconsistencies.

Lock-Based Concurrency Control

It uses locks to restrict access to data, ensuring only one transaction can modify a piece of data at a time, preventing conflicts.

Multi-Version Concurrency Control (MVCC)

It allows multiple versions of a data item to coexist, enabling transactions to read a snapshot of the database without blocking each other.



Database Security

Database Security strives to ensure that only authenticated users perform authorized activities

Authentication

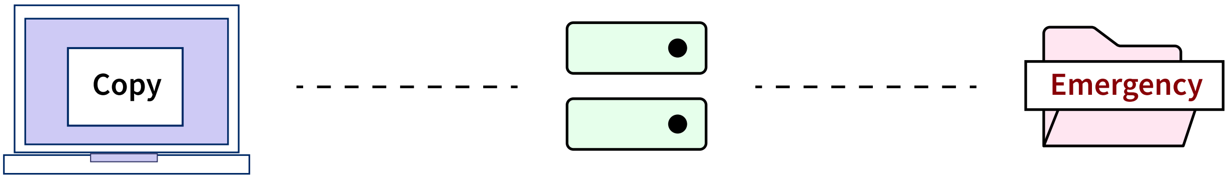
Confirms users are who they say they are

Authorization

Gives users permission to access a resource

DATA BACKUP

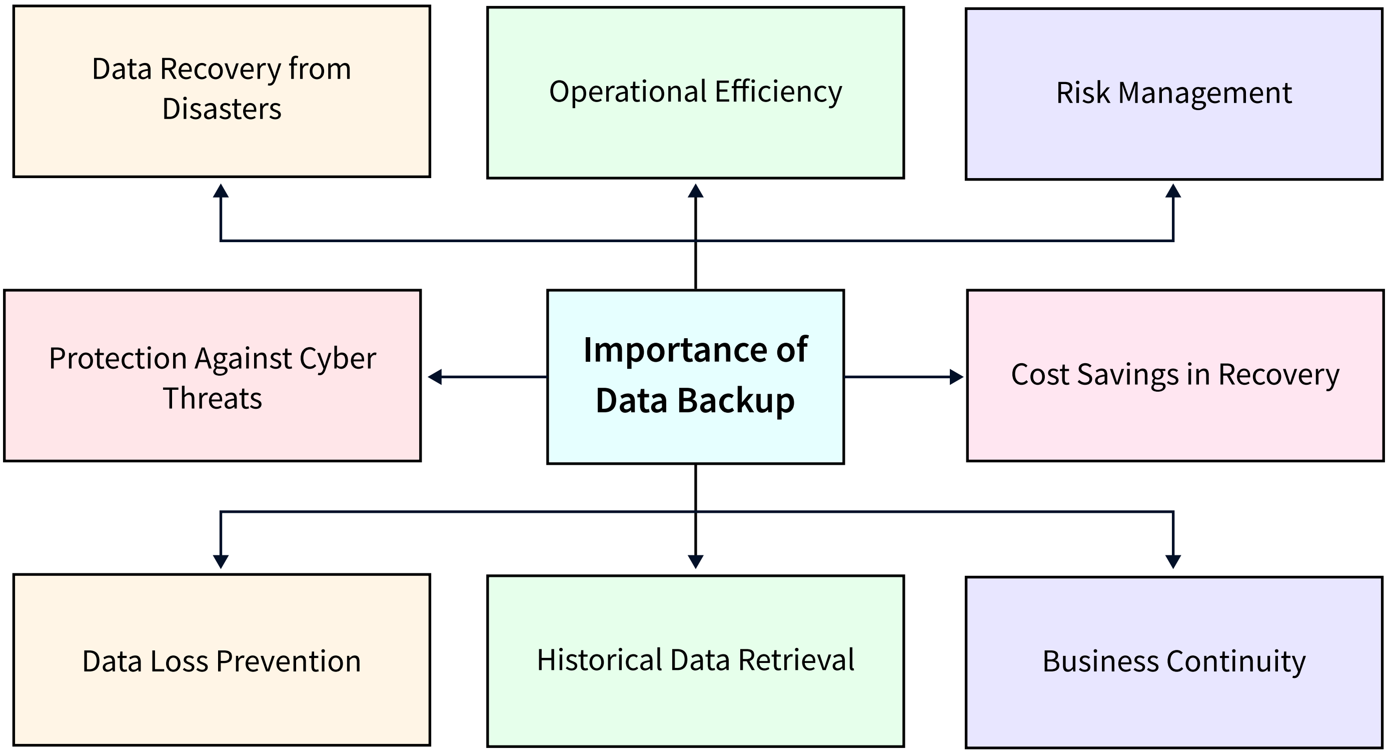
A data backup is a copy or archive of important information from your device.



Create a copy of your important information.

Store it in a secure, separate location.

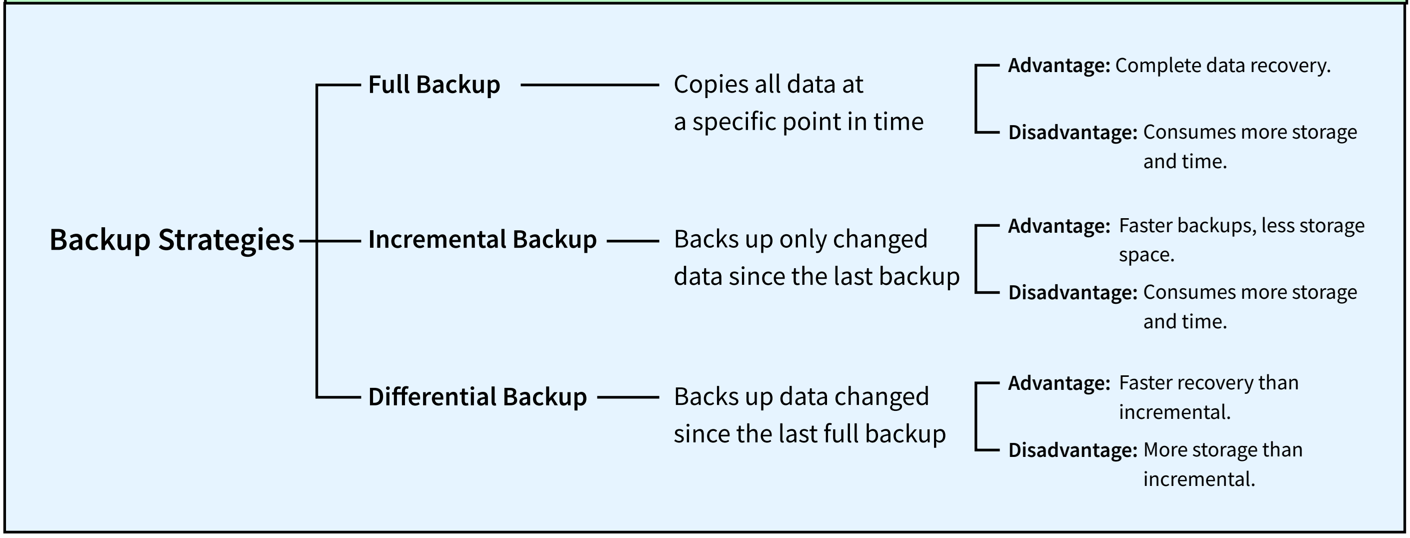
Recognize the backup as a restoration method for your device.



Importance of Data Backup

- Data Recovery from Disasters
- Operational Efficiency
- Risk Management
- Protection Against Cyber Threats
- Cost Savings in Recovery
- Data Loss Prevention
- Historical Data Retrieval
- Business Continuity

Backup Strategies

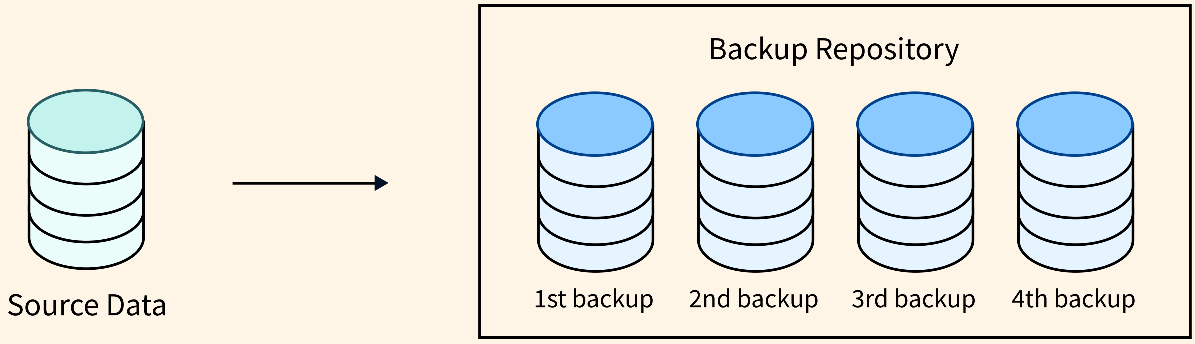


Backup Strategies

- **Full Backup** — Copies all data at a specific point in time
  - **Advantage:** Complete data recovery.
  - **Disadvantage:** Consumes more storage and time.
- **Incremental Backup** — Backs up only changed data since the last backup
  - **Advantage:** Faster backups, less storage space.
  - **Disadvantage:** Consumes more storage and time.
- **Differential Backup** — Backs up data changed since the last full backup
  - **Advantage:** Faster recovery than incremental.
  - **Disadvantage:** More storage than incremental.

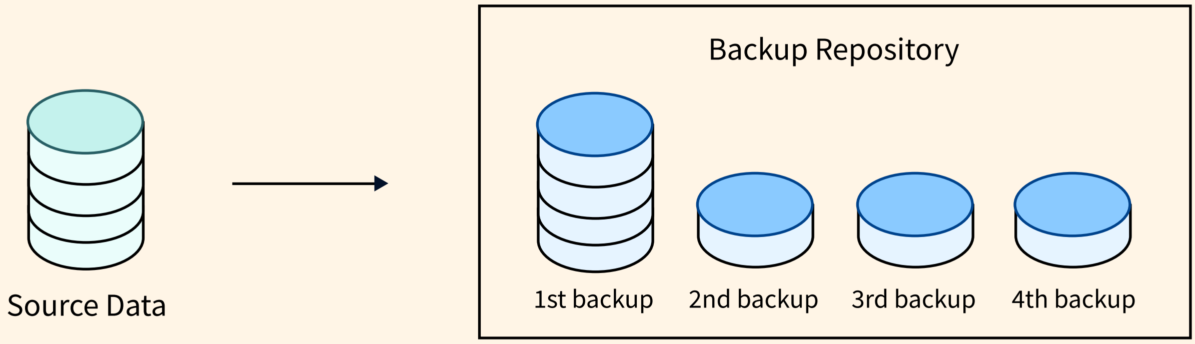
Full backup

Data is copied in its entirety every time.



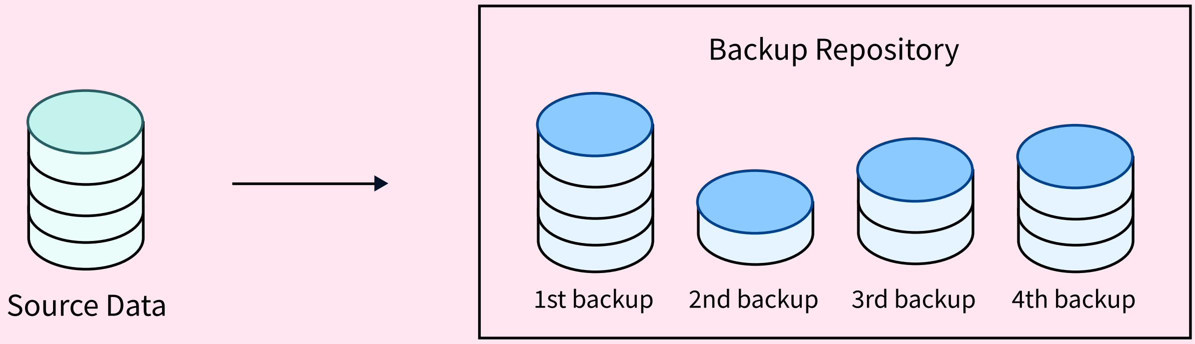
Incremental Backup

Data is copied in its entirety to begin with, and then only new or updated data is backed up each time a backup is initiated after that.

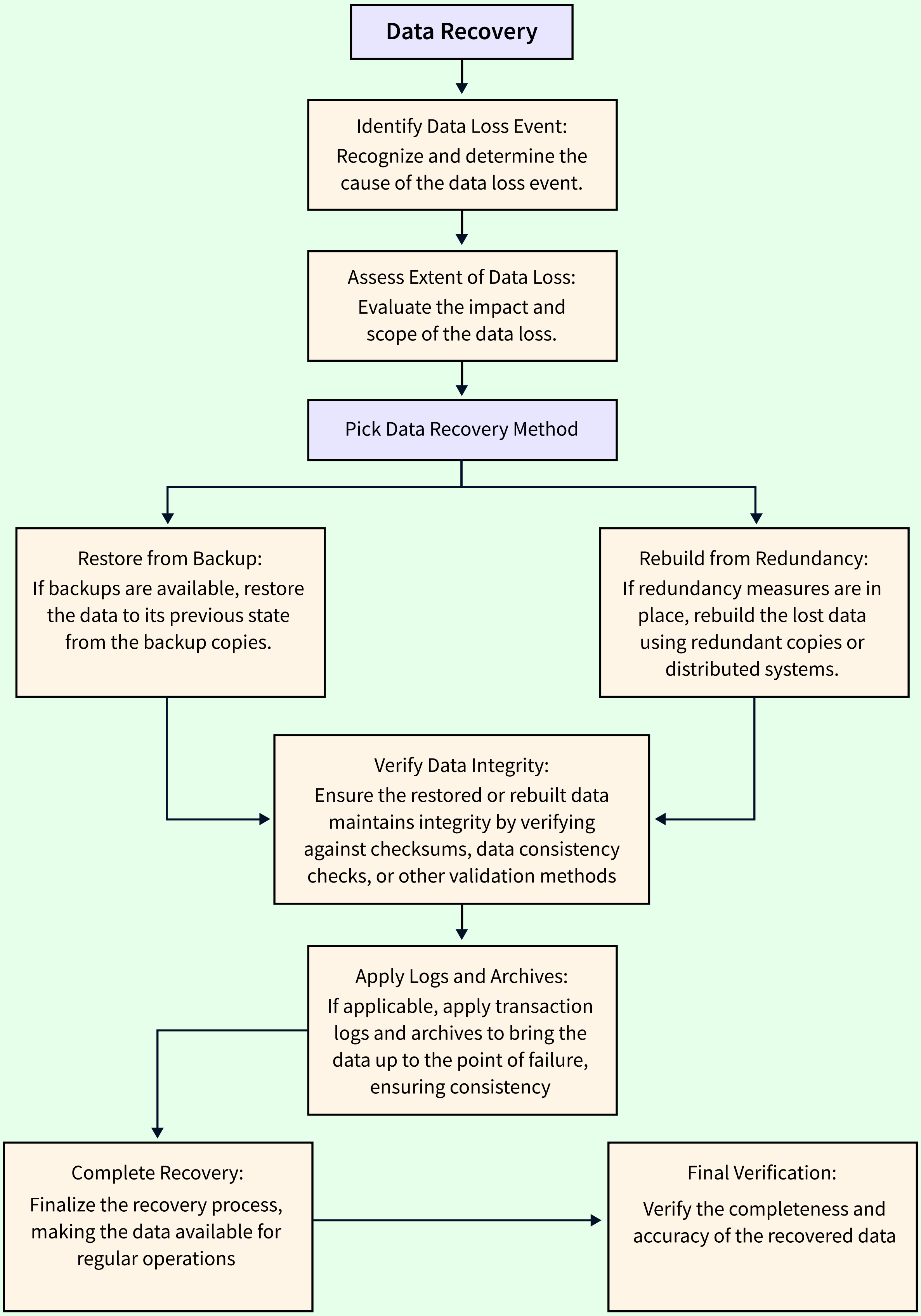


Differential Backup

Data is copied in its entirety to begin with, and then only sets of backup with a change are backed up each time a backup is initiated after that.



Data Recovery



Data Recovery

Identify Data Loss Event: Recognize and determine the cause of the data loss event.

Assess Extent of Data Loss: Evaluate the impact and scope of the data loss.

Pick Data Recovery Method

Restore from Backup: If backups are available, restore the data to its previous state from the backup copies.

Rebuild from Redundancy: If redundancy measures are in place, rebuild the lost data using redundant copies or distributed systems.

Verify Data Integrity: Ensure the restored or rebuilt data maintains integrity by verifying against checksums, data consistency checks, or other validation methods

Apply Logs and Archives: If applicable, apply transaction logs and archives to bring the data up to the point of failure, ensuring consistency

Complete Recovery: Finalize the recovery process, making the data available for regular operations

Final Verification: Verify the completeness and accuracy of the recovered data

SCALER TOPICS

Unlock your potential in software development with **FREE COURSES** from **SCALER TOPICS!**

Register now and take the first step towards your future Success!

C++ for Beginners

5.9k enrolled ₹ Free

Java for Beginners

6.8k enrolled ₹ Free

That's not it. Explore 20+ Courses by clicking below

Explore Other Courses

Practice **CHALLENGES**
and become 1% better everyday

CIFAR-10 Image Classification Using PyTorch

Article

No. Of Questions : 3

Go to Challenge >

How to Build a Snake Game in JavaScript?

Article

No. Of Questions : 3

Go to Challenge >

Explore Other Challenges