Introduction to Databases

Concepts in Focus

- Data
- Database
- Database Management System(DBMS)
  - Advantages
- Types of Databases
  - Relational Database
  - Non-Relational Database

Data


Data

Any sort of information that is stored is called data.

Examples:

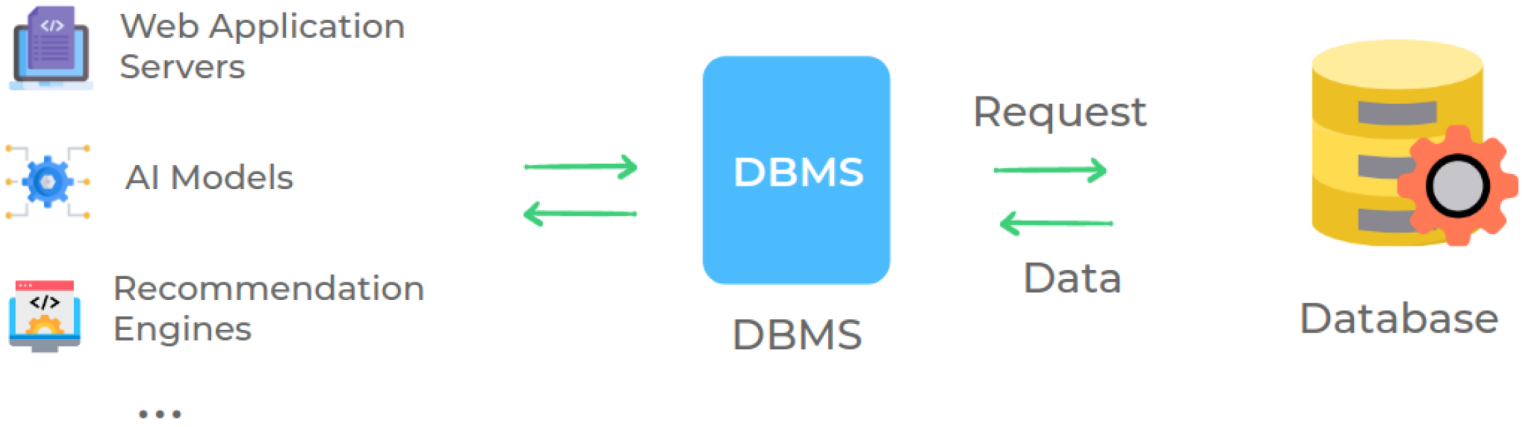
- Messages & multimedia on WhatsApp
- Products and orders on Amazon
- Contact details in telephone directory, etc.

Database

An organised collection of data is called a database.

Database Management System (DBMS)

A software that is used to easily store and access data from the database in a secure way.



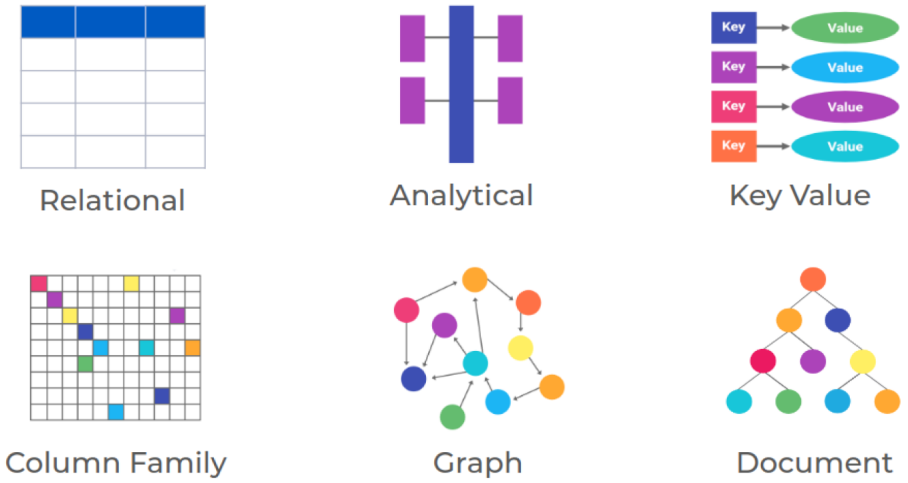
Advantages

- **Security:** Data is stored & maintained securely.
- **Ease of Use:** Provides simpler ways to create & update data at the rate it is generated and updated respectively.
- **Durability and Availability:** Durable and provides access to all the clients at any point in time.

- **Performance:** Quickly accessible to all the clients(applications and stakeholders).

Types of Databases

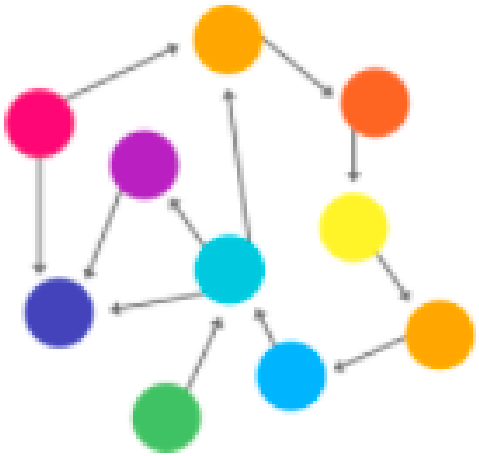
There are different types of databases based on how we organize the data.



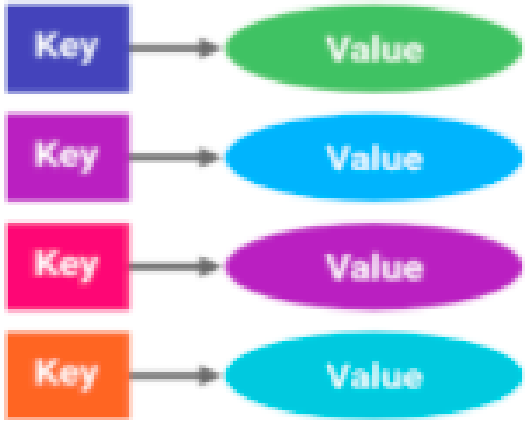
Relational Database

In relational databases, the data is organised in the form of tables.

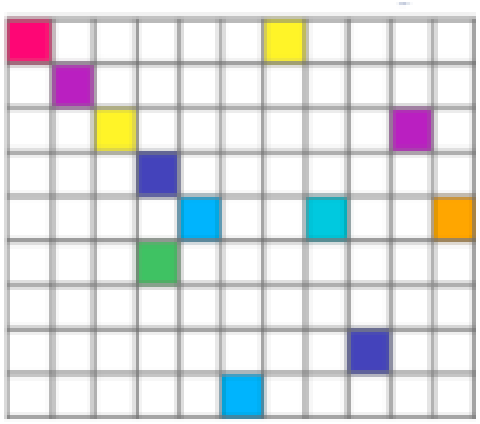
Non-Relational Database



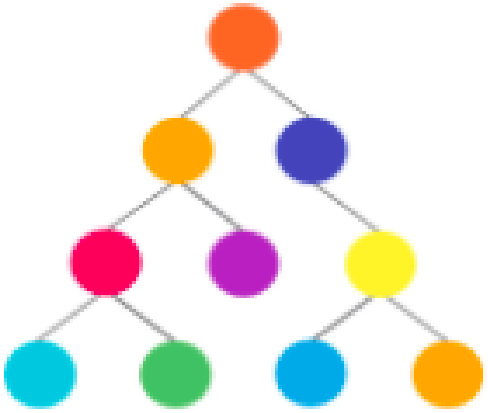
Graph



Key Value



Column Family



Document

These four types are commonly referred as non-relational databases.

Note

- Choice of database depends on our requirements.
- Relational database is the most commonly used database.

Relational DBMS

A

`Relational DBMS` is a DBMS designed specifically for relational databases. Relational databases organise the data in the form of tables.

Examples: Oracle, PostgreSQL, MySQL, SQLite, SQL Server, IBM DB2, etc.

Non-Relational DBMS

A

`Non-relational DBMS` is a DBMS designed specifically for non-relational databases. Non-relational databases store the data in a non-tabular form.

Examples: Elasticsearch, CouchDB, DynamoDB, MongoDB, Cassandra, Redis, etc.

Submit Feedback

Introduction to SQL

We have already learnt that databases and DBMS are key to organising and analysing data for business uses.

From here on, let's get busy working around with databases using SQL!

- SQL stands for `Structured Query Language`
- SQL is used to perform operations on Relational DBMS.
- SQL is declarative. Hence, easy to learn.

SQL provides multiple clauses (commands) to perform various operations like create, retrieve, update and delete the data.

The first step towards working with the database would be creating a table.

Create Table

Creates a new table in the database.

Syntax

SQL

```sql
CREATE TABLE table_name (
  column1 type1,
  column2 type2,
  ...
);
```

Here,

`type1` and `type2` in the syntax are the datatypes of `column1` and `column2` respectively. Datatypes that are supported in SQL are mentioned below.

Example

Create a

`player` table to store the following details of players.

column_name	data_type
name	VARCHAR(200)
age	INT/INTEGER
score	INT/INTEGER

SQL

```sql
CREATE TABLE player (
  name VARCHAR(200),
  age INTEGER,
  score INTEGER
);
```

We can check the details of the created table at any point in time using the

`PRAGMA` command (mentioned below).

Try it Yourself!

Assume that we have to build a database that stores all the information about the students in a school, subjects, exam schedules, etc. Lets build a few tables to store the data!

1. Create a `student` table to store the following details of students.

details	data_type
name	VARCHAR(200)

details	data_type
date_of_birth	DATE
address	TEXT

2. Create an `exam_schedule` table to store the information about exams.

details	data_type
name	VARCHAR(200)
course	VARCHAR(200)
exam_date_time	DATETIME
duration_in_sec	INT
pass_percentage	FLOAT

Data Types

Following data types are frequently used in SQL.

Data Type	Syntax
Integer	INTEGER / INT
Float	FLOAT

Data Type	Syntax
String	VARCHAR
Text	TEXT
Date	DATE
Time	TIME
Datetime	DATETIME
Boolean	BOOLEAN

Note

1. Boolean values are stored as integers 0 (FALSE) and 1 (TRUE).
2. Date object is represented as: ‘YYYY-MM-DD’
3. Datetime object is represented as: ‘YYYY-MM-DD HH:MM:SS’

PRAGMA

`PRAGMA TABLE_INFO` command returns the information about a specific table in a database.

Syntax

SQL

```sql
PRAGMA TABLE_INFO(table_name);
```

Example

Let's find out the information of the

`employee` table that's present in the database.

SQL

```
1  PRAGMA TABLE_INFO(employee);
```

Output

cid	name	type	notnull	dflt_value	pk
0	employee_id	INTEGER	0		0
1	name	VARCHAR(200)	0		0
2	salary	INTEGER	0		0

Note

If the given table name does not exist, `PRAGMA TABLE_INFO` doesn't give any result.

Try it Yourself!

Try checking out the information of the tables that you have created above.

Inserting Rows

`INSERT` clause is used to insert new rows in a table.

Syntax

SQL

```sql
INSERT INTO
    table_name (column1, column2,..., columnN)
VALUES
    (value11, value12,..., value1N),
    (value21, value22,..., value2N),
    ...;
```

Any number of rows from 1 to n can be inserted into a specified table using the above syntax.

Database

Let's learn more about the INSERT clause by going hands-on on the

`player` and `match_details` tables that store the details of players and matches in a tournament respectively.

- `player` table stores the name, age and score of players.
- `match_details` table stores name of team, opponent team name, place, date and match result

Examples

1. Insert `name` , `age` and `score` of 2 players in the `player` table.

SQL

```sql
INSERT INTO
  player (name, age, score)
```

```
  player (name, age, score)
VALUES
  ("Rakesh", 39, 35),
  ("Sai", 47, 30);
```

Upon executing the above code, both the entries would be added to the

`player` table.

Let's view the added data!

We can retrieve the inserted data by using the following command.

SQL

```sql
SELECT *
FROM player;
```

We shall know more about retrieving data in further cheat sheets.

2. Similarly, let's insert the details of 2 matches in the match_details table.

SQL

```sql
INSERT INTO
  match_details (team_name, played_with, venue, date, is_won)
VALUES
  ("CSK", "MI", "Chennai", "2020-04-21", true),
  ("SRH", "RR", "Hyderabad", "2020-04-23", true);
```

Note

1. Boolean values can be either given as (TRUE or FALSE) or (1 or 0). But in the database, the values are stored as 1 or 0.
2. Date object is represented as: 'YYYY-MM-DD'
3. Datetime object is represented as: 'YYYY-MM-DD HH:MM:SS'

Possible Mistakes

Mistake 1

The number of values that we're inserting must match with the number of column names that are specified in the query.

SQL

```sql
INSERT INTO
  player(name, age, score)
VALUES
  ("Virat", 31);
```

SQL

```
Error: 2 values for 3 columns
```

Mistake 2

We have to specify only the existing tables in the database.

SQL

```sql
INSERT INTO
  players_information(name, age, score)
VALUES
  ("Virat", 31, 30);
```

```
Error: no such table: players_information
```

SQL

Mistake 3

Do not add additional parenthesis

`()` post `VALUES` keyword in the code.

```sql
INSERT INTO
  player (name, age, score)
VALUES
  (("Rakesh", 39, 35), ("Sai", 47, 30));
```

SQL

```
Error: 2 values for 3 columns
```

SQL

Mistake 4

While inserting data, be careful with the datatypes of the input values. Input value datatype should be same as the column datatype.

```sql
INSERT INTO
  player(name, age, score)
VALUES
  ("Virat", 31, "Hundred");
```

SQL

Warning

If the datatype of the input value doesn't match with the datatype of column, SQLite doesn't raise an error.

Try it Yourself!

- Three new players have joined the tournament. Try inserting the players' data in the `player` table.

name	age	score
Ram	28	70
Sita	25	30
Ravi	30	53

Retrieving Data

`SELECT` clause is used to retrieve rows from a table.

Database

The database consists of a

`player` table that stores the details of players who are a part of a tournament. `player` table stores the name, age and score of players.

Let's explore more about the SELECT clause using the database!

Selecting Specific Columns

To retrieve the data of only specific columns from a table, add the respective column names in the SELECT clause.

Syntax

SQL

```sql
SELECT
  column1,
  column2,
  ...,
  columnN
FROM
  table_name;
```

Example

Let's fetch the

`name` and `age` of the players from the `player` table.

```sql
SELECT
  name,
  age
FROM
  player;
```

Output

name	age
Virat	32
Rakesh	39
Sai	47

Selecting All Columns

Sometimes, we may want to select all the columns from a table. Typing out every column name, for every time we have to retrive the data, would be a pain. We have a shortcut for this!

Syntax

```sql
SELECT *
FROM table_name;
```

Example

Get all the data of players from the

`player` table.

SQL

```sql
SELECT *
FROM player;
```

Output

name	age	score
Virat	32	50
Rakesh	39	35
Sai	47	30
---	---	---

Selecting Specific Rows

We use WHERE clause to retrieve only specific rows.

Syntax

SQL

```sql
SELECT *
FROM table_name
WHERE condition;
```

`WHERE` clause specifies a condition that has to be satisfied for retrieving the data from a database.

Example

Get

`name` and `age` of the player whose `name` is "Ram" from the `player` table.

SQL

```sql
SELECT *
FROM player
WHERE name="Sai";
```

Output

name	age	score
Sai	47	30

Try it Yourself!

The database consists of an

`employee` table that stores the `employee_id` , `name` and `salary` of employees. Let's fetch data for the following queries.

1. Get all the data from the `employee` table.
2. Get `name` and `salary` of all the employees from the `employee` table.
3. Get `employee_id` and `salary` whose `name` is "Raju" from the `employee` table.

Concepts in Focus

- Update Rows
- SQL – Case Insensitive

Database

The database consists of a

`player` table that stores the details of players who are a part of a tournament. `player` table stores the name, age and score of players.

Update Rows

`UPDATE` clause is used to update the data of an existing table in database. We can update all the rows or only specific rows as per the requirement.

Update All Rows

Syntax

SQL

```sql
UPDATE
  table_name
SET
  column1 = value1;
```

Example:

Update the

`score` of all players to 100 in the `player` table.

SQL

```sql
UPDATE
  player
SET
  score = 100;
```

Update Specific Rows

Syntax

SQL

```sql
UPDATE
  table_name
SET
  column1 = value1
WHERE
  column2 = value2;
```

Example

Update the

`score` of "Ram" to 150 in the `player` table.

SQL

```sql
UPDATE
  player
SET
  score = 150
WHERE
  name = "Ram";
```

Try it Yourself!

The database contains a

`student` table that stores the information of name, percentage and scholarship amount of students.

1. Update the `scholarship_amount` of all students to 15000 in the `student` table.
2. Update the `scholarship_amount` of "Raju" to 25000 in the `student` table.

SQLite is Case Insensitive!

- Query 1

```sql
SELECT
  *
FROM
  player;
```

- Query 2

```sql
select
  *
from
  player;
```

Note

Best Practice: Both Query 1 and Query 2 gives the same output. But, it is recommended to write keywords in upper case to make the query more readable. Prefer Query 1 format over Query 2.

Concepts in Focus

- Delete Rows
- Drop Table

Database

The database consists of a

`player` table that stores the details of players who are a part of a tournament. `player` table stores the name, age and score of players.

Delete Rows

`DELETE` clause is used to delete existing records from a table.

Delete All Rows

Syntax

SQL

```sql
DELETE FROM
  table_name;
```

Example

Delete all the rows from the

`player` table.

SQL

```sql
DELETE FROM
```

```
  player;
```

Delete Specific Rows

Syntax

SQL

```sql
DELETE FROM
  table_name
WHERE
  column1 = value1;
```

Example

Delete "Shyam" from the

`player` table. Note: We can uniquely identify a player by name.

SQL

```sql
DELETE FROM
  player
WHERE
  name = "Shyam";
```

Warning

We can not retrieve the data once we delete the data from the table.

Drop Table

`DROP` clause is used to delete a table from the database.

Syntax

```sql
DROP TABLE table_name;
```

Example

Delete

`player` table from the database.

```sql
DROP TABLE player;
```

MARKED AS COMPLETE

Alter Table

`ALTER` clause is used to add, delete, or modify columns in an existing table. Let's learn more about ALTER clause using the following database.

Database

The database consists of a

`player` table that stores the details of players who are a part of a tournament. `player` table stores the name, age and score of players.

Add Column

Syntax

```SQL
ALTER TABLE
  table_name
ADD
  column_name datatype;
```

- You can use `PRAGMA TABLE_INFO(table_name)` command to check the updated schema of the table.

Example

Add a new column

`jersey_num` of type `integer` to the `player` table.

```SQL
ALTER TABLE
  player
ADD
```

```
4    jersey_num INT;
```

Note

Default values for newly added columns in the existing rows will be NULL.

Rename Column

Syntax

SQL

```
1  ALTER TABLE
2    table_name RENAME COLUMN c1 TO c2;
```

Example

Rename the column `jersey_num` in the `player` table to `jersey_number` .

SQL

```
1  ALTER TABLE
2    player RENAME COLUMN jersey_num TO jersey_number;
```

Drop Column

Syntax

```sql
ALTER TABLE
  table_name DROP COLUMN column_name;
```

Example

Remove the column

`jersey_number` from the `player` table.

```sql
ALTER TABLE
  player DROP COLUMN jersey_number;
```

Note

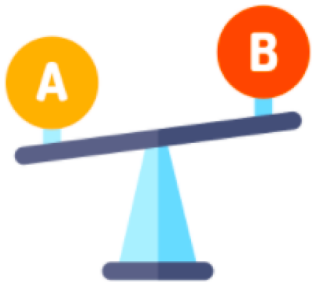
DROP COLUMN is not supported in some DBMS, including SQLite.

Try it Yourself!

1. Add a new column `id` of type `int` to the player table.
2. Update the name of column `id` to `player_id` in the player table.

Comparison Operators

In a typical e-commerce scenario, users would generally filter the products with good ratings, or want to purchase the products of a certain brand or of a certain price.
Let's see how comparison operators are used to filter such kind of data using the following database.



Database

The database contains a

`product` table that stores the data of products like name, category, price, brand and rating. You can check the schema and data of `product` table in the code playground.

Comparison Operators

Operator	Description
=	Equal to

Operator	Description
<>	Not equal to
<	Less than
<=	Less than or equal to
>	Greater than
>=	Greater than or equal to

Examples

1. Get all the details of the products whose `category` is "Food" from the `product` table.

SQL

```sql
SELECT
  *
FROM
  product
WHERE
  category = "Food";
```

Output

name	category	price	brand	rating
Chocolate Cake	Food	25	Britannia	3.7
Strawberry Cake	Food	60	Cadbury	4.1

name	category	price	brand	rating
Chocolate Cake	Food	60	Cadbury	2.5
...	...	...	...	...

2. Get all the details of the products that does not belong to `Food` category from the `product` table.

SQL

```sql
SELECT
  *
FROM
  product
WHERE
  category <> "Food";
```

Output

name	category	price	brand	rating
Blue Shirt	Clothing	750	Denim	3.8
Blue Jeans	Clothing	800	Puma	3.6
Black Jeans	Clothing	750	Denim	4.5
...	...	...	...	...

Similarly, we can use other comparison operators like **greater than (>)**, **greater than or equal to (>=), less than (<), less than or equal to (<=)** to filter the data as per the requirement.

Try it Yourself!

Put your learning into practice and try fetching the products based on the conditions mentioned.

Write a query for each of the below conditions.

1. `rating` greater than 4.5
2. `price` is less than or equal to 1000
3. `brand` is "Puma"
4. that does not belong to "Gadgets" `category`

String Operations

Consider the case of e-commerce platforms. We generally search for the products on the basis of product name. But while searching, we need not enter the full name. For example, typing “mobiles” in a search bar will fetch thousands of results. How to get the data on the basis of only a part of the string? Let’s learn about it!

Database

The database contains a

`product` table that stores the data of products like name, category, price, brand and rating. You can check the schema and data of `product` table in the code playground.

LIKE Operator

`LIKE` operator is used to perform queries on strings. This operator is especially used in `WHERE` clause to retrieve all the rows that match the given pattern.

We write

`patterns` using the following `wildcard characters` :

Symbol	Description	Example
Percent sign (`%`)	Represents zero or more characters	`ch%` finds ch, chips, chocolate..
Underscore (`_`)	Represents a single character	`_at` finds mat, hat and bat

Common Patterns

Pattern	Example	Description
Exact Match	WHERE name LIKE "mobiles"	Retrieves products whose name is exactly equals to "mobiles"
Starts With	WHERE name LIKE "mobiles%"	Retrieves products whose name starts with "mobiles"
Ends With	WHERE name LIKE "%mobiles"	Retrieves products whose name ends with "mobiles"
Contains	WHERE name LIKE "%mobiles%"	Retrieves products whose name contains with "mobiles"
Pattern Matching	WHERE name LIKE "a_%"	Retrieves products whose name starts with "a" and have at least 2 characters in length

Syntax

SQL

```sql
SELECT
    *
FROM
    table_name
WHERE
    c1 LIKE matching_pattern;
```

Examples

1. Get all the products in the "Gadgets" category from the `product` table.

SQL

```sql
SELECT
  *
FROM
  product
WHERE
```

```
    category LIKE "Gadgets";
```

Output

name	category	price	brand	rating
Smart Watch	Gadgets	17000	Apple	4.9
Smart Cam	Gadgets	2600	Realme	4.7
Smart TV	Gadgets	40000	Sony	4.0
Realme Smart Band	Gadgets	3000	Realme	4.6

2. Get all the products whose `name` starts with "Bourbon" from the `product` table.

SQL

```sql
SELECT
  *
FROM
  product
WHERE
  name LIKE "Bourbon%";
```

Here

`%` represents that, following the string "Bourbon", there can be 0 or more characters.

Output

name	category	price	brand	rating
Bourbon Small	Food	10	Britannia	3.9

name	category	price	brand	rating
Bourbon Special	Food	15	Britannia	4.6
Bourbon With Extra Cookies	Food	30	Britannia	4.4

3. Get all smart electronic products i.e., `name` contains "Smart" from the `product` table.

SQL

```sql
SELECT
  *
FROM
  product
WHERE
  name LIKE "%Smart%";
```

Here,

`%` before and after the string "Smart" represents that there can be 0 or more characters succeeding or preceding the string.

Output

name	category	price	brand	rating
Smart Watch	Gadgets	17000	Apple	4.9
Smart Cam	Gadgets	2600	Realme	4.7
Smart TV	Gadgets	40000	Sony	4
Realme Smart Band	Gadgets	3000	Realme	4.6

4. Get all the products which have exactly 5 characters in `brand` from the `product` table.

```sql
SELECT
  *
FROM
  product
WHERE
  brand LIKE "_____";
```

Output

name	category	price	brand	rating
Blue Shirt	Clothing	750	Denim	3.8
Black Jeans	Clothing	750	Denim	4.5
Smart Watch	Gadgets	17000	Apple	4.9
...	...	...	...	...

Note

The *percent sign(%)* is used when we are not sure of the number of characters present in the string.
If we know the exact length of the string, then the wildcard character *underscore(_)* comes in handy.

Try it Yourself!

Put your learning into practice and try fetching the products based on the different patterns:

Write a query for each of the below patterns.

- `category` is exactly equal "Food".
- `name` containing "Cake".
- `name` ends with "T-Shirt".
- `name` contains "Chips".
- `category` contains exactly 4 characters.

Logical Operators

So far, we've used comparison operators to filter the data. But in real-world scenarios, we often have to retrieve the data using several conditions at once. For example, in the case of e-commerce platforms, users often search for something like:
Get shoes from the Puma brand, which have ratings greater than 4.0 and price less than 5000.

With logical operators, we can perform queries based on multiple conditions. Let's learn how with the following database.

Database

The database contains a

`product` table that stores the data of products like name, category, price, brand and rating. You can check the schema and data of `product` table in the code playground.

AND, OR, NOT

Operator	Description
AND	Used to fetch rows that satisfy two or more conditions.
OR	Used to fetch rows that satisfy at least one of the given conditions.
NOT	Used to negate a condition in the `WHERE` clause.

Syntax

SQL

```sql
SELECT
  *
FROM
```

```
FROM
  table_name
WHERE
  condition1
  operator condition2
  operator condition3
  ...;
```

Examples

1. Get all the details of the products whose
   - `category` is "Clothing" *and*
   - `price` less than or equal to 1000 from the `product` table.

SQL

```sql
SELECT
  *
FROM
  product
WHERE
  category = "Clothing"
  AND price <= 1000;
```

Output

name	category	price	brand	rating
Blue Shirt	Clothing	750	Denim	3.8
Blue Jeans	Clothing	800	Puma	3.6
Black Jeans	Clothing	750	Denim	4.5

name	category	price	brand	rating
...	...	...	...	...

2. *Ignore* all the products with `name` containing "Cake" from the list of products.

SQL

```sql
SELECT
  *
FROM
  product
WHERE
  NOT name LIKE "%Cake%";
```

Output

name	category	price	brand	rating
Blue Shirt	Clothing	750	Denim	3.8
Blue Jeans	Clothing	800	Puma	3.6
Black Jeans	Clothing	750	Denim	4.5
---	---	---	---	---

Try it Yourself!

- Fetch all the products with `price` less than 20000 and `brand` is "Apple".
- Fetch all the products with `rating` greater than 4.0 or `brand` is "Britannia".
- Ignore all the products with `category` containing "Food" in `product` table.

Multiple Logical Operators

We can also use the combinations of logical operators to combine two or more conditions. These compound conditions enable us to fine-tune the data retrieval requirements.

Precedence

- When a query has multiple operators, operator precedence determines the sequence of operations.


NOT
AND
OR
High
Low

Order of precedence:

- NOT
- AND
- OR

Example

Fetch the products that belong to

- *Redmi* `brand` and `rating` greater than *4* or
- the products from *OnePlus*

  `brand`

```SQL
SELECT
  *
FROM
  product
WHERE
  brand = "Redmi"
  AND rating > 4
  OR brand = "OnePlus";
```

- In the above query,

  `AND` has the precedence over `OR` . So, the above query is equivalent to:

```SQL
SELECT
    *
FROM
    product
WHERE
```

```
WHERE
    (brand = "Redmi"
    AND rating > 4)
    OR brand = "OnePlus";
```

Quick Tip : It is suggested to always use parenthesis to ensure correctness while grouping the conditions.

Try it Yourself!

- Fetch all the products from "Clothing" category whose `name` does not contain "Jeans".
- Fetch all the products from "Puma" and "Denim" brands excluding the products with `name` containing "Shirts".
- Fetch all the products with `price` less than 100 or the products from "Food" category excluding the ones with `name` containing "Chocolate".

IN and BETWEEN Operators

Consider the case of a typical e-commerce scenario. Users generally search for the products that belong to a list of brands, or the products that lie within a particular price range.

In such scenarios, we use the IN operator to check if a value is present in the list of values. And, BETWEEN operator is used to check if a particular value exists in the given range.

Let’s learn about these operators in detail using the following database.

Database

The database contains a

`product` table that stores the data of products like name, category, price, brand and rating. You can check the schema and data of `product` table in the code playground.

IN Operator

Retrieves the corresponding rows from the table if the value of column(c1) is present in the given values(v1,v2,..).

Syntax

SQL

```sql
SELECT
  *
FROM
  table_name
WHERE
  c1 IN (v1, v2,..);
```

Example

Get the details of all the products from

`product` table, where the `brand` is either "Puma", "Mufti", "Levi's", "Lee" or "Denim".

SQL

```sql
SELECT
  *
FROM
  product
```

```
5  WHERE
6    brand IN ( "Puma", "Levi's", "Mufti", "Lee", "Denim");
```

Output

name	category	price	brand	rating
Blue Shirt	Clothing	750	Denim	3.8
Blue Jeans	Clothing	800	Puma	3.6
Black Jeans	Clothing	750	Denim	4.5
...	...	...	...	...

Try it Yourself!

- Get all the products from `product` table, that belong to "Britannia", "Lay's", "Cadbury" brands from the "Food" category.

BETWEEN Operator

Retrieves all the rows from table that have cloumn(c1) value present between the given range(v1 and v2).

Syntax

SQL

```
1  SELECT
2    *
```

```
FROM
  table_name
WHERE
  c1 BETWEEN v1
  AND v2;
```

Note

`BETWEEN` operator is inclusive, i.e., both the lower and upper limit values of the range are included.

Example

Find the products with

`price` ranging from 1000 to 5000.

SQL

```sql
SELECT
  name,
  price,
  brand
FROM
  product
WHERE
  price BETWEEN 1000
  AND 5000;
```

Output

name	price	brand
Blue Shirt	1000	Puma

name	price	brand
Smart Cam	2600	Realme
Realme Smart Band	3000	Realme

Possible Mistakes

1. When using the `BETWEEN` operator, the first value should be less than second value. If not, we'll get an incorrect result depending on the DBMS.

SQL

```sql
SELECT
  name,
  price,
  brand
FROM
  product
WHERE
  price BETWEEN 500
  AND 300;
```

Output

name	price	brand

2. We have to give both lower limit and upper limit while specifying range.

SQL

```sql
SELECT
  name,
  price,
  brand
```

```sql
FROM
  product
WHERE
  price BETWEEN
  AND 300;
```

SQL

```
Error: near "AND": syntax error
```

3. The data type of the column for which we're using the `BETWEEN` operator must match with the data types of the lower and upper limits.

SQL

```sql
SELECT
  name,
  price,
  brand
FROM
  product
WHERE
  name BETWEEN 300
  AND 500;
```

Output

name	price	brand

Try it Yourself!

- Get all the products from `product` table with `rating` greater than 4.3 and less than 4.8

ORDER BY and DISTINCT

In any e-commerce application, users have the option of sorting the products based on price, rating, etc. Also, for any product, users could know all the distinct brands available for the product.
Let's learn how to retrieve such ordered results and unique data!

Database

The database contains a

`product` table that stores the data of products like name, category, price, brand and rating. You can check the schema and data of `product` table in the code playground.

ORDER BY

We use

`ORDER BY` clause to order rows. By default, `ORDER BY` sorts the data in the `ascending order` .

Syntax

SQL

```sql
SELECT
  column1,
  column2,
..columnN
FROM
  table_name [WHERE condition]
ORDER BY
  column1 ASC / DESC,
  cloumn2 ASC / DESC;
```

Example

Get all products in the order of lowest

`price` and highest `rating` in "Puma" brand.

SQL

```sql
SELECT
  name,
  price,
  rating
FROM
  product
WHERE
  brand = "Puma"
ORDER BY
  price ASC,
```

Expand

Output

name	price	rating
Black Shirt	600	4.8

name	price	rating
Blue Jeans	800	3.6
Blue Shirt	1000	4.3

Try it Yourself!

- Get all the shirts from

  `product` table in the descending order of their `rating` and in the ascending order of `price` .

  Note: Assusme the products as shirts, if the

  `name` contains "Shirt".

DISTINCT

`DISTINCT` clause is used to return the distinct i.e unique values.

Syntax

SQL

```sql
SELECT
  DISTINCT column1,
  column2,..
  columnN
FROM
  table_name
WHERE
  [condition];
```

Example

- Get all the brands present in the `product` table.

```sql
SELECT
  DISTINCT brand
FROM
  product
ORDER BY
  brand;
```

SQL

Output

Brand
Absa
Apple
...

Try it Yourself!

- Get a list of distinct categories available in the `product` table

Pagination

E-commerce applications like Amazon or Flipkart hold millions of products. But, the user does not require all the available products every time s/he accesses the application. Infact, fetching all the products takes too long and consumes huge amount of data.

Using pagination, only a chunk of the data can be sent to the user based on their request. And, the next chunk of data can be fetched only when the user asks for it.

- We use `LIMIT` & `OFFSET` clauses to select a chunk of the results

Let's understand more about pagination concept using the following databse.

Database

The database contains a

`product` table that stores the data of products like name, category, price, brand and rating. You can check the schema and data of `product` table in the code playground.

LIMIT

`LIMIT` clause is used to specify the `number of rows(n)` we would like to have in result.

Syntax

```sql
SELECT
  column1,
  column2,..
  columnN
FROM
  table_name
LIMIT n;
```

Example

Get the details of 2 top-rated products from the brand "Puma".

```sql
SELECT
  name,
  price,
  rating
FROM
  product
WHERE
  brand = "Puma"
ORDER BY
  rating DESC
```

Expand

Output

name	price	rating
Black Shirt	600	4.8
Blue Shirt	1000	4.3

Try it Yourself!

- Get the 3 lowest priced products from the brand "Puma".

Note

If the limit value is greater than the total number of rows, then all rows will be retrieved.

OFFSET

`OFFSET` clause is used to specify the position (from nth row) from where the chunk of the results are to be selected.

Syntax

SQL

```sql
SELECT
  column1,
  column2,..
  columnN
FROM
  table_name
OFFSET n;
```

Example

Get the details of 5 top-rated products, starting from 5th row.

SQL

```sql
SELECT
  name,
  price,
  rating
FROM
```

```
  product
ORDER BY
  rating DESC
LIMIT 5
OFFSET 5;
```

Output

name	price	rating
Strawberry Cake	10	4.6
Bourbon Special	15	4.6
Realme Smart Band	3000	4.6
Harry Potter and the Goblet of Fire	431	4.6
Black Jeans	750	4.5

Possible Mistakes

- Using `OFFSET` before the `LIMIT` clause.

SQL

```
SELECT
  *
FROM
  product OFFSET 2
LIMIT 4;
```

SQL

```
Error: near "2": syntax error
```

- Using only `OFFSET` clause.

SQL

```sql
SELECT
  *
FROM
  product
OFFSET 2;
```

SQL

```
Error: near "2": syntax error
```

Note

`OFFSET` clause should be placed after the `LIMIT` clause. Default `OFFSET` value is 0.

Try it Yourself!

- Get the details of 5 top-rated products, starting from 10th row.

Aggregations

Consider the case of sports tournaments like cricket. Players' performances are analysed based on their batting average, maximum number of sixes hit, the least score in a tournament, etc.

We perform aggregations in such scenarios to combine multiple values into a single value, i.e., individual scores to an average score.

Let's learn more about aggregations to perform insightful analysis using the following database.

Database

The database consists of a player_match_details table that stores the information of players' details in a match like name, match, score, year, number of fours and sixes scored.

Schema

SQL

```sql
player_match_details (
  name VARCHAR(250),
  match VARCHAR(10),
  score INTEGER,
  fours INTEGER,
  sixes INTEGER,
  year INTEGER
);
```

Aggregation Functions

Combining multiple values into a single value is called aggregation. Following are the functions provided by SQL to perform aggregations on the given data:

Aggregate Functions	Description
COUNT	Counts the number of values
SUM	Adds all the values
MIN	Returns the minimum value
MAX	Returns the maximum value
AVG	Calculates the average of the values

Syntax

SQL

```sql
SELECT
  aggregate_function(c1),
  aggregate_function(c2)
FROM
  TABLE;
```

Note : We can calculate multiple aggregate functions in a single query.

Examples

1. Get the total runs scored by "Ram" from the `player_match_details` table.

name	match	score	fours	sixes	year
Ram	CSK vs DD	55	4	2	2011
Joseph	MI vs RR	58	4	2	2012
Lokesh	RCB vs KKR	55	3	4	2013
David	MI vs RR	63	4	2	2014
Viraj	KKR vs SRH	72	6	4	2010
Shyam	SRH vs RCB	62	3	2	2011
Stark	RR vs SRH	54	5	1	2012
Ram	CSK vs SRH	59	4	4	2013
Joseph	SRH vs RR	52	3	2	2014

SQL

```sql
SELECT
  SUM(score)
FROM
  player_match_details
WHERE
  name = "Ram";
```

Output

SUM(score)
221

SUM(score)

2. Get the highest and least scores among all the matches that happened in the year 2011.

```sql
SELECT
  MAX(score),
  MIN(score)
FROM
  player_match_details
WHERE
  year = 2011;
```

Output

MAX(score)	MIN(score)
75	62

COUNT Variants

- Calculate the total number of matches played in the tournament.

Variant 1

```sql
SELECT COUNT(*)
FROM player_match_details;
```

Variant 2

```sql
SELECT COUNT(1)
FROM player_match_details;
```

Variant 3

```sql
SELECT COUNT()
FROM player_match_details;
```

Output of Variant 1, Variant 2 and Variant 3

All the variants, i.e.,

`Variant 1` , `Variant 2` and `Variant 3` give the same result: `18`

Special Cases

- When `SUM` function is applied on non-numeric data types like strings, date, time, datetime etc., `SQLite` DBMS returns `0.0` and `PostgreSQL` DBMS returns `None` .
- Aggregate functions on strings and their outputs

Aggregate Functions	Output
MIN, MAX	Based on lexicographic ordering
SUM, AVG	0 (depends on DBMS)
COUNT	Default behavior

- `NULL` values are ignored while computing the aggregation values
- When aggregate functions are applied on only

  `NULL` values:

Aggregate Functions	Output
MIN	NULL
MAX	NULL
SUM	NULL
COUNT	0
AVG	NULL

Alias

Using the keyword

`AS` , we can provide alternate temporary names to the columns in the output.

Syntax

SQL

```sql
SELECT
  c1 AS a1,
  c2 AS a2,
  ...
FROM
  table_name;
```

Examples

- Get all the names of players with column name as "player_name".

SQL

```sql
SELECT
  name AS player_name
FROM
  player_match_details;
```

Output

player_name
Ram
Joseph

- Get the average score of players as "avg_score".

SQL

```sql
SELECT
  AVG(score) AS avg_score
FROM
  player_match_details;
```

Output

avg_score
60

Try it Yourself!

- Get the average score of "Ram" in the year 2011.
- Get the least score among all the matches.
- Get the highest score among the scores of all players in 2014.
- Get the total number of sixes hit as `sixes_hit`

Group By with Having

Previously, we have learnt to perform aggregations on all the rows of a table. Now, we shall look at how to split a table into multiple groups and apply aggregation on each group.

The GROUP BY keyword in SQL is used to group rows which have the same values for the mentioned attributes. You can perform aggregations on these groups to get finer analytics.

HAVING keyword is used to further refine the data by filtering the aggregated values. Let's explore more about GROUP BY and HAVING clauses with the following database.

Database

The database consists of

`player_match_details` table which stores name, match, score, year, number of fours and sixes scored.

- Schema

SQL

```sql
CREATE TABLE player_match_details (
   name VARCHAR(250),
   match VARCHAR(250),
   score INTEGER,
   fours INTEGER,
   sixes INTEGER,
   year INTEGER
);
```

GROUP BY

The

`GROUP BY` clause in SQL is used to group rows which have same values for the mentioned attributes.

Syntax

SQL

```sql
SELECT
  c1,
  aggregate_function(c2)
FROM
  table_name
GROUP BY c1;
```

Example

Get the total score of each player in the database.

SQL

```sql
SELECT
  name, SUM(score) as total_score
FROM
  player_match_details
GROUP BY name;
```

Output

name	total_score
David	105

name	total_score
Joseph	116
Lokesh	186
...	...

Try it Yourself!

- Get the maximum score of each player.
- Get the total number of sixes hit by each player.

GROUP BY with WHERE

We use WHERE clause to filter the data before performing aggregation.

Syntax

SQL

```sql
SELECT
  c1,
  aggregate_function(c2)
FROM
  table_name
WHERE
  c3 = v1
GROUP BY c1;
```

Example

Get the number of half-centuries scored by each player

SQL

```sql
SELECT
 name, COUNT(*) AS half_centuries
FROM
 player_match_details
WHERE score >= 50
GROUP BY name;
```

Output

name	half_centuries
David	1

name	half_centuries
Joseph	2
Lokesh	3
...	...

Try it Yourself!

- Get year wise number of half-centuries scored by each player.

HAVING

`HAVING` clause is used to filter the resultant rows after the application of `GROUP BY` clause.

Syntax

SQL

```sql
SELECT
  c1,
  c2,
  aggregate_function(c1)
FROM
  table_name
GROUP BY
  c1, c2
HAVING
  condition;
```

Example

Get the

`name` and number of `half_centuries` of players who scored more than one half century.

SQL

```sql
SELECT
  name,
  count(*) AS half_centuries
FROM
  player_match_details
WHERE
  score >= 50
GROUP BY
  name
HAVING
```

Expand

Output

name	half_centuries
Lokesh	2
Ram	3

Try it Yourself!

- Get the name and number of half-centuries scored by each player who scored at least a half-century in two matches.

Note

WHERE vs HAVING: WHERE is used to filter rows and this operation is performed before grouping. HAVING is used to filter groups and this operation is performed after grouping.

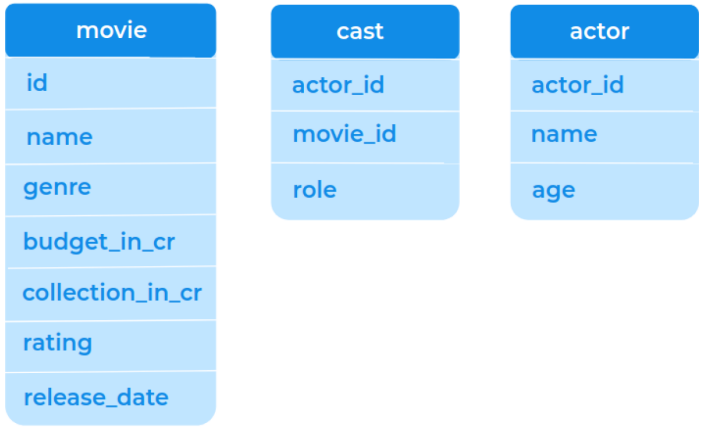
Expressions in Querying

We can write **expressions** in various SQL clauses. Expressions can comprise of various data types like integers, floats, strings, datetime, etc.

Let's learn more about expressions using the following database.

Database

The IMDb stores various *movies*, *actors* and *cast* information.



Using Expressions in SELECT Clause

1. Get profits of all movies.

Note: Consider profit as the difference between collection and budget.

```sql
SELECT
    id, name, (collection_in_cr-budget_in_cr) as profit
FROM
    movie;
```

Output

id	name	profit
1	The matrix	40.31
2	Inception	67.68
3	The Dark Knight	82.5
...	...	...

Note

We use "||" operator to concatenate strings in sqlite3

2. Get the movie `name` and `genre` in the following format: `movie_name - genre` .

```sql
SELECT
  name || " - " || genre AS movie_genre
FROM
  movie;
```

Output

movie_genre
The Matrix - Sci-fi
Inception - Action
The Dark Knight - Drama
Toy Story 3 - Animation
...

Using Expressions in WHERE Clause

1. Get all the movies with a profit of at least 50 crores.

SQL

```sql
SELECT
   *
FROM
   movie
WHERE
   (collection_in_cr - budget_in_cr) >= 50;
```

Output

id	name	genre	budget_in_cr	collection_in_cr	rating	release_date
2	Inception	Action	16.0	83.68	8.8	2010-07-14

id	name	genre	budget_in_cr	collection_in_cr	rating	release_date
3	The Dark Knight	Action	18.0	100.5	9.0	2008-07-16
4	Toy Story 3	Animation	20.0	106.7	8.5	2010-06-25
...	...	...	...	...	...	...

Using Expressions in UPDATE Clause

1. Scale down ratings from 10 to 5 in movie table.

SQL

```sql
UPDATE movie
SET rating = rating/2;
```

You can check the updation of movie ratings by retriving the data from the table.

Expressions in HAVING Clause

Get all the genres that have average profit greater than 100 crores.

SQL

```sql
SELECT
  genre
FROM
```

```
4    movie
5  GROUP BY
6    genre
7  HAVING
8    AVG(collection_in_cr - budget_in_cr) >= 100;
```

Output

genre
Action
Animation
Mystery
...

Try it Yourself!

Question 1

Get the profit of every movie with

`rating` greater than 8.0

Expected Output

id	name	genre	budget_in_cr	collection_in_cr	rating	release_date	
1	The Matrix	Sci-fi	6.3	46.43	8.7	1999-04-31	
2	Inception	Action	16	83.68	8.8	2010-07-16	
3	The Dark Knight	Drama	18	100.5	9	2008-07-18	
...	...	...	...	...	...	...	

Question 2

Get all the movies having a

`profit` of at least 30 crores, and belong to "Action", "Animation" or "Drama" genres.

Expected Result

id	name	genre	budget_in_cr	collection_in_cr	rating	release_date
2	Inception	Action	16	83.68	8.8	2010-07-16
3	The Dark Knight	Drama	18	100.5	9	2008-07-18
4	Toy Story 3	Animation	20	106.7	8.5	2010-06-25
...	...	...	...	...	...	...

Question 3

Scale up the ratings from 5 to 100 in the movie table.

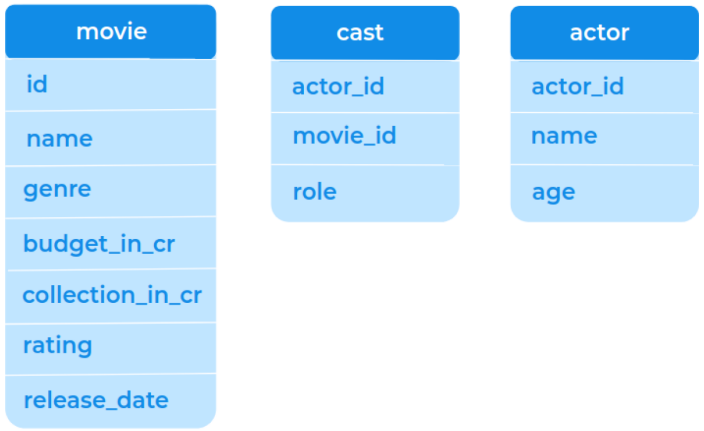
SQL Functions

SQL provides many built-in functions to perform various operations over data that is stored in tables.
Let's look at a few most commonly used functions in the industry using the following database.

Database

The IMDb dataset stores the information of movies, actors and cast.

Schema



Date Functions

strftime()

strftime() function is used to **extract month, year**, etc. from a date/datetime field based on a **specified format**.

Syntax

```sql
strftime(format, field_name)
```

Example

Get the movie title and release year for every movie in the database

```sql
SELECT name, strftime('%Y', release_date)
FROM
    movie;
```

In the above query, we extract year from

`release_date` by writing `strftime('%Y', release_date)` in `SELECT` clause.

Let's understand various formats in date functions with an example.
Consider the datetime

`2021-02-28 08:30:05`

format	description	output format	Example
%d	Day of the month	01 - 31	28
%H	Hour	00 - 24	08
%m	Month	01 - 12	02
%j	Day of the year	001 - 365	59
%M	Minute	00-59	30

format	description	output format	Example
...	...	...	...

Example

Get the number of movies released in each month of the year 2010

SQL

```sql
SELECT
    strftime('%m', release_date) as month,
    COUNT(*) as total_movies
FROM
    movie
WHERE
    strftime('%Y', release_date) = '2010'
GROUP BY
    strftime('%m', release_date);
```

Output

month	total_movies
03	2
05	1
06	3
..	..

As the above example, using strftime(), we can perform weekly, monthly or annual analysis deriving finer insights from the data.

Try it Yourself!

Question 1

Get the number of "Action" movies released in the year *2010*.

Expected Output

total_movies
4

Question 2

Get all the movies that are released in summer, i.e., between *April* and *June*.

Expected Output

name
The Matrix
Toy Story 3
Shutter Island
...

Question 3

Get the month in which the highest number of movies are released.

Expected Output

month	total_movies
06	6

CAST Function

CAST function is used to **typecast** a value to a desired data type.

Syntax

SQL

```sql
CAST(value AS data_type);
```

Example

Get the number of movies released in each month of the year 2010

SQL

```sql
SELECT strftime('%m', release_date) as month,
       COUNT(*) as total_movies
FROM
    movie
WHERE
    CAST(strftime('%Y', release_date) AS INTEGER) = 2010
GROUP BY
    strftime('%m', release_date);
```

Output

month	total_movies
03	2
05	1
06	3
..	..

Here,

`CAST(strftime('%Y', release_date) AS INTEGER)` converts the year in string format to integer format.

Try it Yourself!

Question 1

Get all the leap years in the database. An year can be marked as a leap year if

1 .It is divisible by 4 and not divisible by 100

2. Or if it is divisible by 400

Expected Output

year
1972
2008
2016

year

Other Common Functions

Arithmetic Functions

SQL Function	Behavior
FLOOR	Rounds a number to the nearest integer below its current value
CEIL	Rounds a number to the nearest integer above its current value
ROUND	Rounds a number to a specified number of decimal places

You can refer the following table to further understand how floor, ceil and round work in general.

	2.3	3.9	4.0	5.5
FLOOR	2	3	4	5
CEIL	3	4	4	6
ROUND	2	4	4	6

Let's understand how these functions can be used.

Examples

1. Fetch the ceil, floor and round (to 1 decimal) values of the collections of all movies.

SQL

```sql
SELECT
    name,
  ROUND(collection_in_cr, 1) AS RoundedValue,
  CEIL(collection_in_cr) AS CeilValue,
  FLOOR(collection_in_cr) AS FloorValue
FROM
  movie;
```

Output

name	RoundedValue	CeilValue	FloorValue
The Matrix	46.4	47	46
Inception	83.7	84	83
The Dark Knight	100.5	101	100
...	...	...	...

String Functions

SQL Function	Behavior
UPPER	Converts a string to upper case
LOWER	Converts a string to lower case

When you are not sure about the case (upper/lower) of the movie name, you can write a query as below to search for all the avengers movies irrespective of the case.

SQL

```sql
SELECT
   name
FROM
  movie
  WHERE UPPER(name) LIKE UPPER("%avengers%");
```

Output

name
Avengers: Age of Ultron
Avengers: Endgame
Avengers: Infinity War

Note

Usually, UPPER() AND LOWER() functions can help you to perform case-insensitive searches.

Try it Yourself!

Question 1

For each movie, get the ceil, floor and round(to 1 decimal) values of budget.

Expected Output

name	round_value	ceil_value	floor_value
The Matrix	6.3	7	6
Inception	16	16	16
The Dark Knight	18	18	18
...	...	...	...

Question 2

Get all the movie names that are released in 2010 and belong to "Action" genre.

Note:

Try using the sql functions learnt so far

Expected Output

movie_name
Inception
Iron Man 2
Thor

movie_name
Spider-Man: Homecoming
...

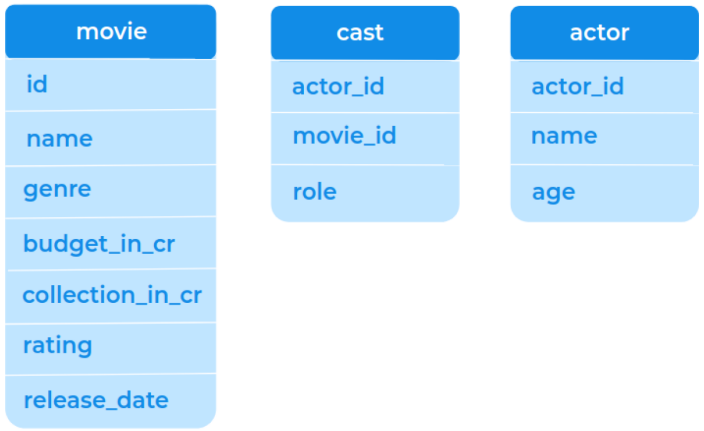
CASE Clause

SQL provides CASE clause to perform **conditional operations**. This is similar to the **switch case / if-else** conditions in other programming languages.

Let's learn more about the usage of CASE clause using the given database

Database

The IMDb dataset which consists of **movies, actors and cast**. You can refer to the database in the code playground for a better understanding.



CASE Clause

Each condition in the

`CASE` clause is evaluated and results in corresponding value when the first condition is met.

Syntax

```sql
SELECT c1, c2
CASE
    WHEN condition1 THEN value1
    WHEN condition2 THEN value2
    ...
    ELSE value
    END AS cn
FROM table;
```

Note

1. In CASE clause, if no condition is satisfied, it returns the value in the ELSE part. If we do not specify the ELSE part,

   `CASE clause` results in NULL

2. We can use CASE in various clauses like SELECT, WHERE, HAVING, ORDER BY and GROUP BY.

Example

Calculate the tax amount for all movies based on the profit. Check the following table for tax percentages.

profit	tax_percentage
<=100 crores	10% of profit
100< <=500 crores	15% of profit
> 500 crores	18% of profit

```sql
SELECT id, name,
  CASE
    WHEN collection_in_cr - budget_in_cr <= 100 THEN collection_in_cr - budget_in_cr * 0.1
    WHEN (collection_in_cr - budget_in_cr > 100
    AND collection_in_cr - budget_in_cr < 500) THEN collection_in_cr - budget_in_cr * 0.15
    ELSE collection_in_cr - budget_in_cr * 0.18
  END AS tax_amount
FROM
  movie;
```

SQL

Output

id	name	tax_amount
1	The Matrix	45.8
2	Inception	82.08
3	The Dark Knight	98.7
...	...	...

Try it Yourself

Question 1

Categorise movies as following.

rating	category
< 5	Poor

rating	category
5 <= _ <= 7	Average
7 <	Good

CASE with Aggregates

CASE statements can also be used together with aggregate functions

Example

- Get the number of movies with rating greater than or equal to 8, and the movies with rating less than 8, and are released between 2015 and 2020.

SQL

```sql
SELECT
  count(
    CASE
      WHEN rating >= 8 THEN 1
    END
  ) AS above_eight,
  count(
    CASE
      WHEN rating < 8 THEN 1
    END
```

Expand

Output

above_eight	below_eight
4	2

Try it Yourself!

- Get the number of movies with collection greater than or equal to 100 crores, and the movies with collection less than 100 crores.

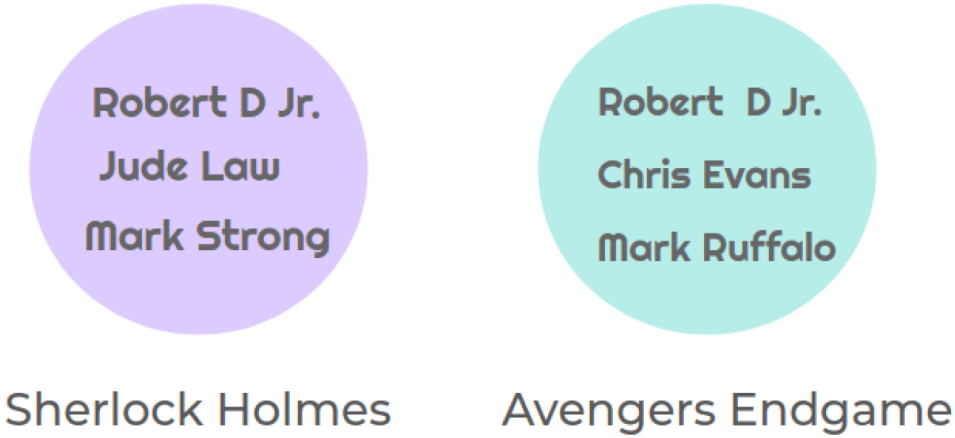
Output

above_100_cr	below_100_cr
13	7

The SQL Set operation is used to combine the two or more SQL queries.

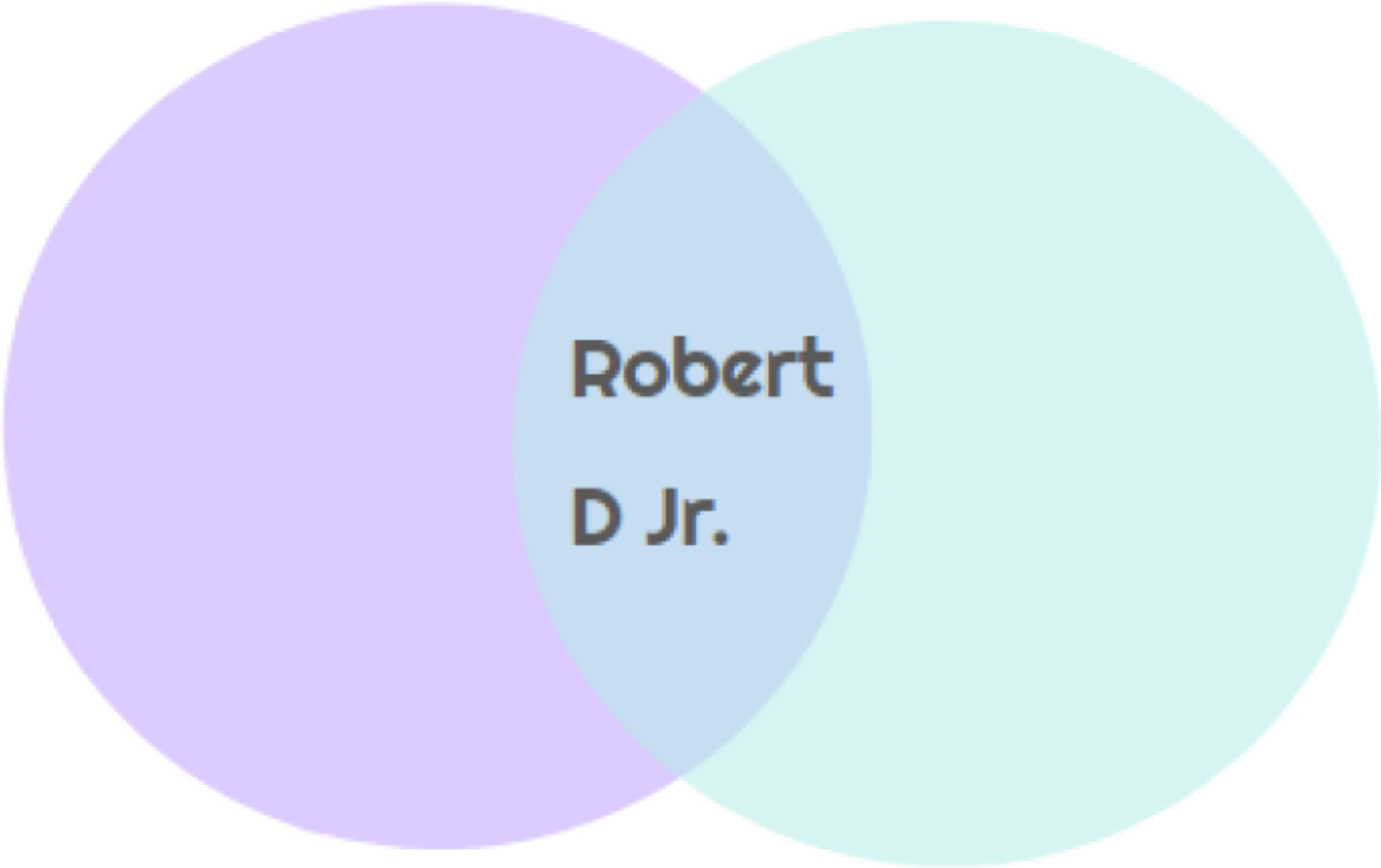
Let us understand common set operators by performing operations on two sets

- cast in "Sherlock Holmes" movie
- cast in "Avengers Endgame" movie



Common Set Operators

INTERSECT


Robert
D Jr.

Actors who acted in both Sherlock Holmes and Avengers Endgame

Result: Robert D Jr.

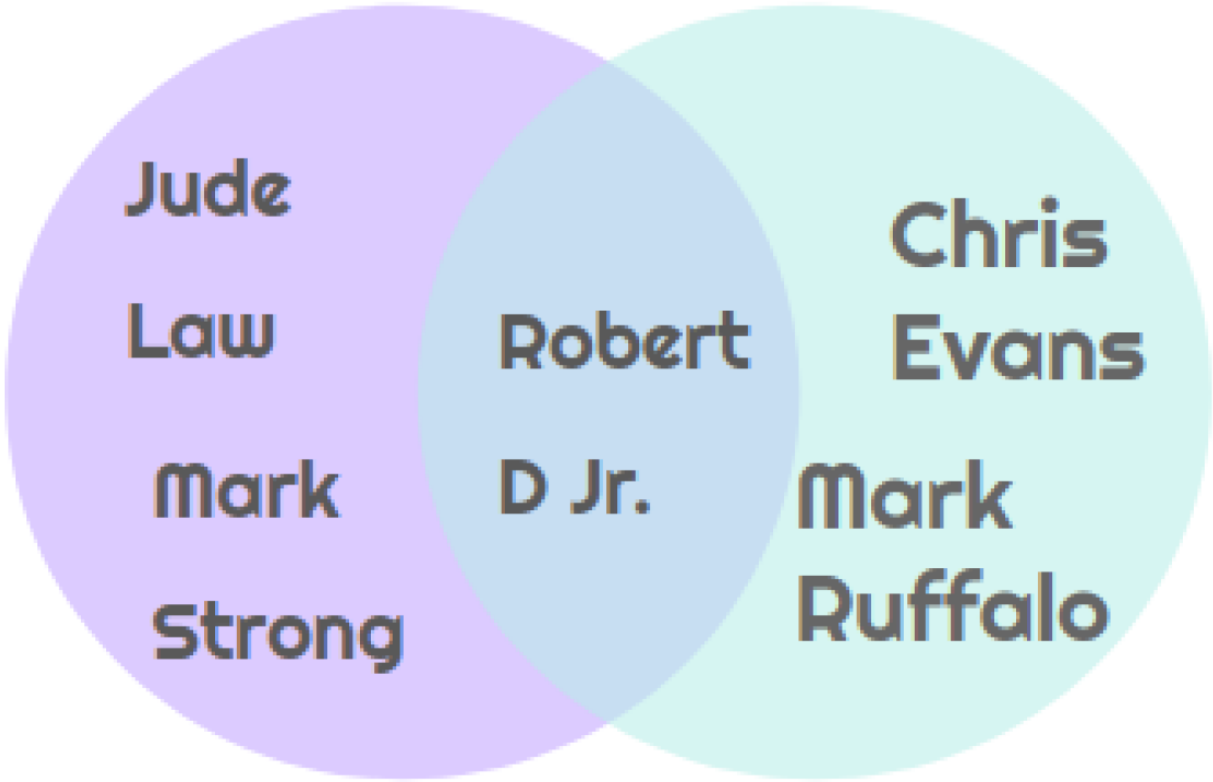
MINUS


Jude Law
Mark
Strong

Actors who acted in Sherlock Holmes and not in Avengers Endgame

Result: Jude Law, Mark Strong

UNION



Unique actors who acted in Sherlock Holmes or in Avengers Endgame

Result: Jude Law, Mark Strong, Robert D Jr, Chris Evans, Mark Ruffalo

UNION ALL

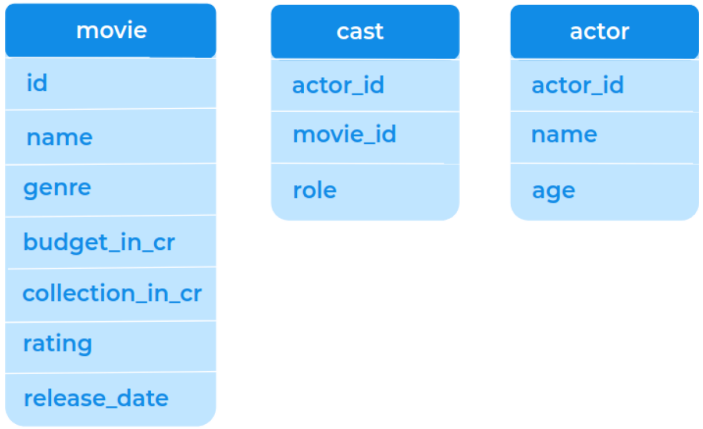
Doesn't eliminate duplicate results

Result: Jude Law, Mark Strong, Robert D Jr, Robert D Jr, Chris Evans, Mark Ruffalo

Let's learn more about CASE clause using the given database

Database

The IMDb dataset which consists of **movies, actors and cast**. You can refer to the database in the code playground for a better understanding.



Applying Set Operations

We can apply these set operations on the one or more sql queries to combine their results

Syntax

```SQL
SELECT
    c1, c2
FROM
    table_name
SET_OPERATOR
SELECT
    c1, c2
FROM
    table_name;
```

Basic rules when combining two sql queries using set operations

- Each SELECT statement must have the same number of columns
- The columns must have similar data types

- The columns in each SELECT statement must be in the same order

Examples

- Get ids of actors who acted in both Sherlock Holmes(id=6) and Avengers Endgame(id=15)?

SQL

```sql
SELECT actor_id
FROM cast
WHERE movie_id=6
INTERSECT
SELECT actor_id
FROM cast
WHERE movie_id=15;
```

Output

actor_id
6

- Get ids of actors who acted in Sherlock Holmes(id=6) and not in Avengers Endgame(id=15)?

SQL

```sql
SELECT actor_id
FROM cast
WHERE movie_id=6
EXCEPT
SELECT actor_id
FROM cast
WHERE movie_id=15;
```

Output

actor_id
16
21

- Get distinct ids of actors who acted in Sherlock Holmes(id=6) or Avengers Endgame(id=15).

SQL

```sql
SELECT actor_id
FROM cast
WHERE movie_id=6
UNION
SELECT actor_id
FROM cast
WHERE movie_id=15;
```

Output

actor_id
6
8
16
21
22

- Get ids of actors who acted in Sherlock Holmes(id=6) or Avengers Endgame(id=15).

SQL

```sql
SELECT actor_id
FROM cast
WHERE movie_id=6
UNION ALL
SELECT actor_id
FROM cast
WHERE movie_id=15;
```

Output

actor_id
6
16
21
6
8
22

Try it Yourself!

1. Get all the movie ids in which actors Robert Downey Jr. (id=6) & Chris Evans(id=22) have been casted
2. Get all the movie ids in which actor Robert Downey Jr. (id=6) is casted and not Chris Evans(id=22)
3. Get all the unique movie ids in which either actor Robert Downey Jr. (id=6) or Chris Evans(id=22) is casted

ORDER BY Clause in Set Operations

ORDER BY clause can appear only once at the end of the query containing multiple SELECT statements.

While using Set Operators, individual SELECT statements cannot have ORDER BY clause. Additionally, sorting can be done based on the columns that appear in the first SELECT query. For this reason, it is **recommended to sort this kind of queries using column positions**.

Example

Get distinct ids of actors who acted in Sherlock Holmes (id=6) or Avengers Endgame(id=15). Sort ids in the descending order.

SQL

```sql
SELECT actor_id
FROM cast
WHERE movie_id=6
UNION
SELECT actor_id
FROM cast
WHERE movie_id=15
ORDER BY 1 DESC;
```

Try it Yourself!

1. Get all the movie ids in which actor Robert Downey Jr. (id=6) is casted & not Chris Evans(id=22). Sort the ids in the descending order.

Pagination in Set Operations

Similar to ORDER BY clause, LIMIT and OFFSET clauses are used at the end of the list of queries.

Example

Get the first 5 ids of actors who acted in Sherlock Holmes (id=6) or Avengers Endgame(id=15). Sort ids in the descending order.

SQL

```sql
SELECT actor_id
FROM cast
WHERE movie_id=6
```

```
WHERE movie_id=6
UNION
SELECT actor_id
FROM cast
WHERE movie_id=15
ORDER BY 1 DESC
LIMIT 5;
```

Try it Yourself!

1. Get the first 5 unique movie ids in which either actor Robert Downey Jr. (id=6) or Ryan Reynolds(id=7) is casted. Sort ids in the descending order.

Clauses

Clauses	How to Use It	Functionality
CREATE TABLE	CREATE TABLE table_name ...	Creates a new table
INSERT	INSERT INTO table_name ...	Used to insert new data in the table
SELECT	SELECT col1, col2 ..	Retrieves the selected columns
SELECT	SELECT * FROM ...	Retrieves all the columns from a table
FROM	FROM table_name	FROM clause specifies the table(s) in which the required data columns are located
WHERE	WHERE col > 5	Retrieves only specific rows based on the given conditions
UPDATE, SET	UPDATE table_name SET column1 = value1;	Updates the value of a column of all the rows (or only specific rows using WHERE clause)
DELETE	DELETE FROM table_name	Deletes all the rows from the table
DROP	DROP TABLE table_name	Deletes the table from the database
ALTER	ALTER TABLE table_name ...	Used to add, delete or modify columns in a table
ORDER BY	ORDER BY col1 ASC/DESC..	Sorts the table based on the column(s) in the ascending or descending orders
DISTINCT	SELECT DISTINCT col, ...	Gets the unique values of given column(s)
LIMIT	LIMIT 10	Limits the number of rows in the output to the mentioned number
OFFSET	OFFSET 5	Specifies the position (from nth row) from where the chunk of the results are to be retrieved
GROUP BY	GROUP BY col ...	Groups the rows that have same values in the given columns
HAVING	HAVING col > 20	Filters the resultant rows after the application of GROUP BY clause

Clauses	How to Use It	Functionality
CASE	CASE WHEN condition1 THEN value1 WHEN .. ELSE .. END	Returns a corresponding value when the first condition is met

Operators

Operators	How to Use It	Functionality
<>	WHERE col <> 5	Filters the rows where the given column is not equal to 5. Similarly, other comparison operators (=,>, <,>=,<=) are also used.
LIKE	WHERE col LIKE '%Apple%'	Retrieves the rows where the given column has 'apple' within the text
AND	WHERE col1 > 5 AND col2 < 3	Retrieves the rows that satisfy all the given conditions
OR	WHERE col1 > 5 OR col2 < 3	Retrieves the rows that satisfy at least one condition
NOT	WHERE NOT col = 'apple'	Retrieves the rows if the condition(s) is NOT TRUE
IN	WHERE col IN ('Apple', 'Microsoft')	Retrieves the rows if the column value is present in the given values
BETWEEN	WHERE col BETWEEN 3 AND 5	Retrieves the rows if the column value is present between (and including) the given values

Functions

Functions	How to Use It	Functionality
COUNT	SELECT COUNT(col) ...	Counts the number of values in the given column
SUM	SELECT SUM(col) ...	Adds all the values of given column

Functions	How to Use It	Functionality
MIN	SELECT MIN(col) ...	Gets the minimum value of given column
MAX	SELECT MAX(col) ...	Gets the maximum value of given column
AVG	SELECT AVG(col) ...	Gets the average of the values present in the given column
strftime()	strftime("%Y", col) ...	Extracts the year from the column value in string format. Similarly, we can extract month, day, week of the day and many.
CAST()	CAST(col AS datatype) ...	Converts the value to the given datatype
FLOOR()	FLOOR(col)	Rounds a number to the nearest integer below its current value
CEIL()	CEIL (col)	Rounds a number to the nearest integer above its current value
ROUND()	ROUND(col)	Rounds a number to a specified number of decimal places
UPPER()	UPPER(col)	Converts a string to upper case
LOWER()	Lower(col)	Converts a string to lower case

Modelling Databases: Part 1

To model a database, we have to first understand the business requirements at conceptual level, which is later translated into a relational database.

For understanding the business requirements at a conceptual level, we use

Entity Relationship Model (ER Model) .

Core Concepts in ER Model

Entity

John

Emma

Real world objects/concepts are called **entities** in ER Model.

Attributes of an Entity

name: John

age: 29

name: Emma

age: 25

Properties of real world objects/concepts are represented as **attributes** of an entity in ER model.

Key Attribute

aadhaar_no: XXXX

name: John

age: 29

imei_no: 86670XXX

brand: Redmi

The attribute that uniquely identifies each entity is called **key attribute**.

Entity Type

aadhaar_no: XXXX

name: John

age: 29

aadhaar_no: XXXX

name: Emma

age: 25

Person

Entity Type is a **collection of entities** that have the same attributes (not values).

Relationships

Association among the entities is called a **relationship**.

Example:

- Person **has a** passport.
- Person can **have many** cars.
- Each student can **register for many** courses, and a course can **have many** students.

Types of relationships

- One-to-One Relationship
- One-to-Many or Many-to-One Relationship
- Many-to-Many Relationship

One-to-One Relationship


1
has
1
PASSPORT

An entity is related to **only one entity**, and vice versa.

Example

- A person can have **only one** passport.
- similarly, a passport belongs to **one and only one** person.

One-to-Many Relationship

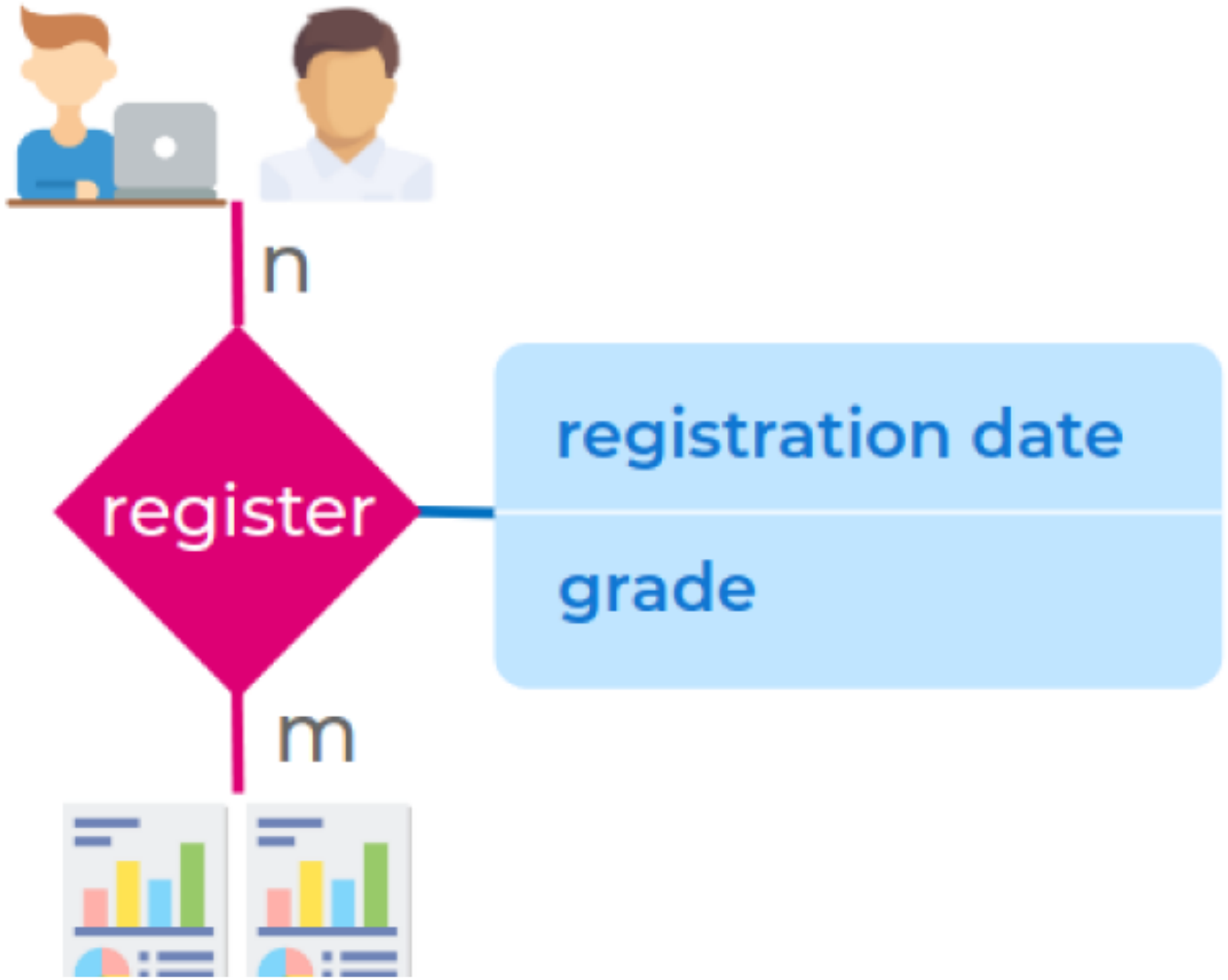

1
has
n

An entity is related to **many other** entities.

Example

- A person **can have many** cars. But a car belongs to **only one** person.

Many-to-Many Relationship


n
register
registration date
grade
m

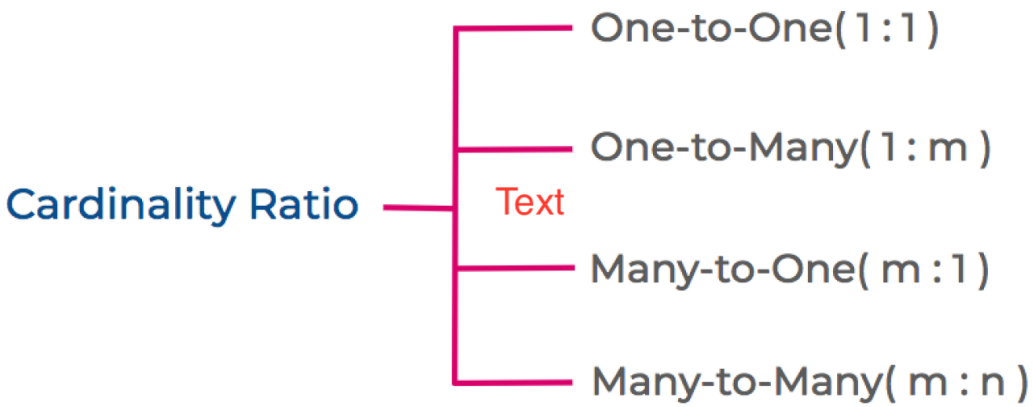
Multiple entities are related to multiple entities.

Example

- Each student can register to multiple courses.
- similarly each course is taken by multiple students.

Cardinality Ratio

Cardinality in DBMS defines the maximum number of times an instance in one entity can relate to instances of another entity.



Submit Feedback

Applying ER Model Concepts

In the previous cheatsheet, we have understood the **core concepts of ER Model** — ***entity types, relationships and attributes***. Now, let's build an ER model for a real-world scenario.

E-commerce Application

In a typical e-commerce application,

- ***Customer*** has only one ***cart***. A ***cart*** belongs to only one ***customer***
- ***Customer*** can add ***products*** to ***cart***
- ***Cart*** contains multiple ***products***
- ***Customer*** can save multiple ***addresses*** in the application for further use like selecting delivery address

Let’s apply the concepts of ER Model to this e-commerce scenario.

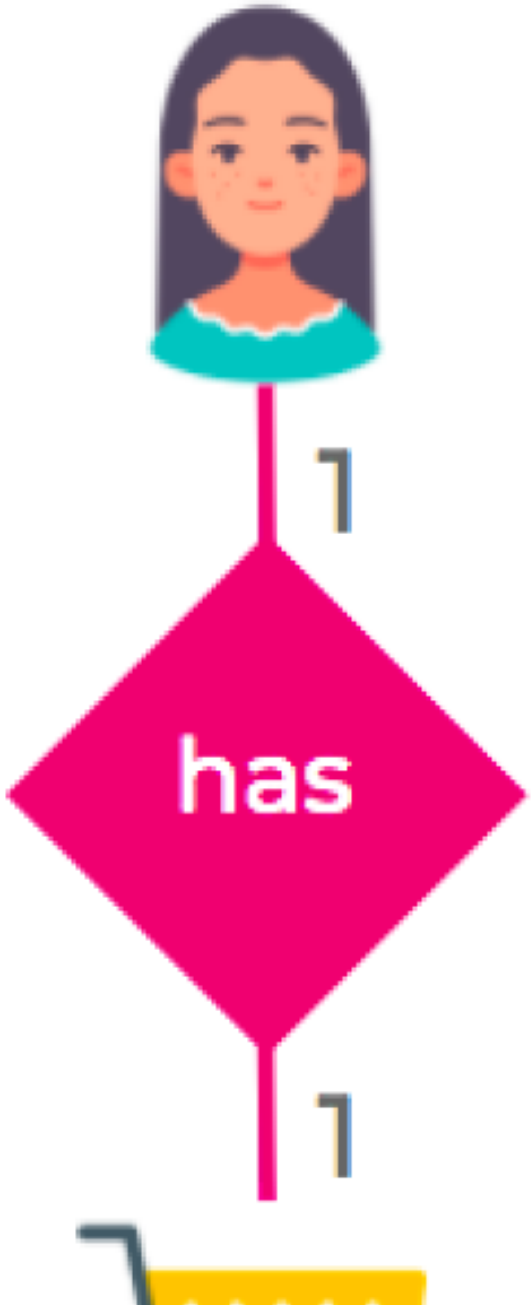
Entity types

- Customer
- Product
- Cart
- Address

Relationships

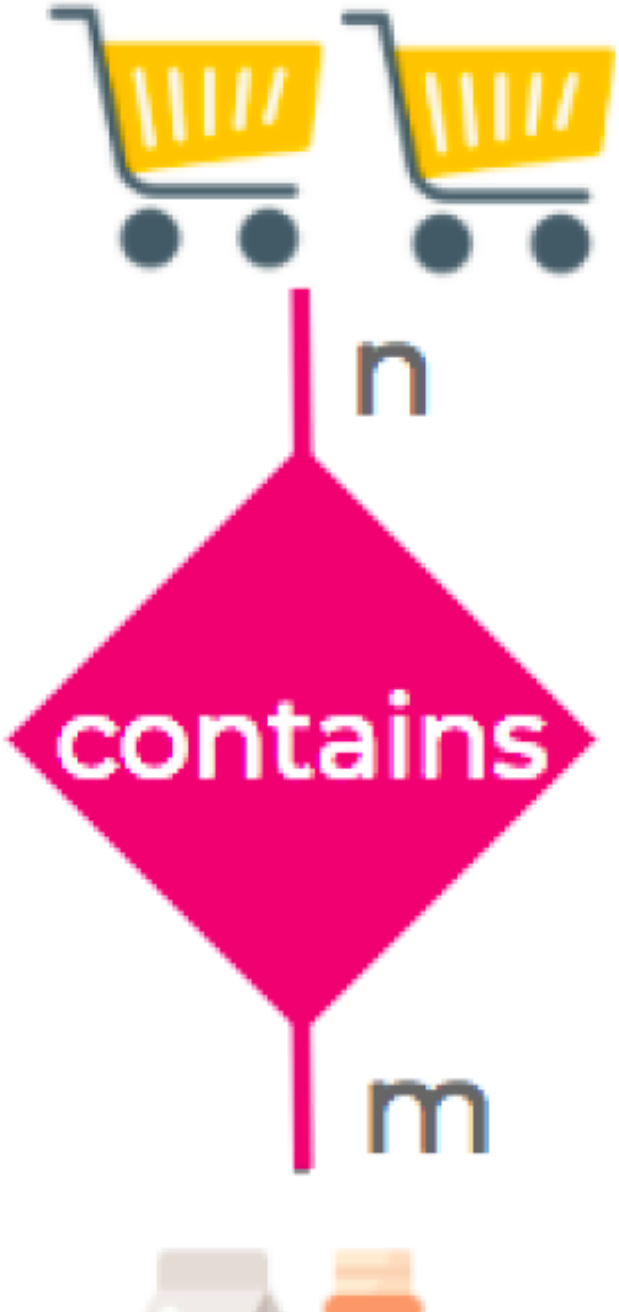
Let's understand the relationships in the e-commerce use-case.

Relation Between Cart and Customer


1
has
1

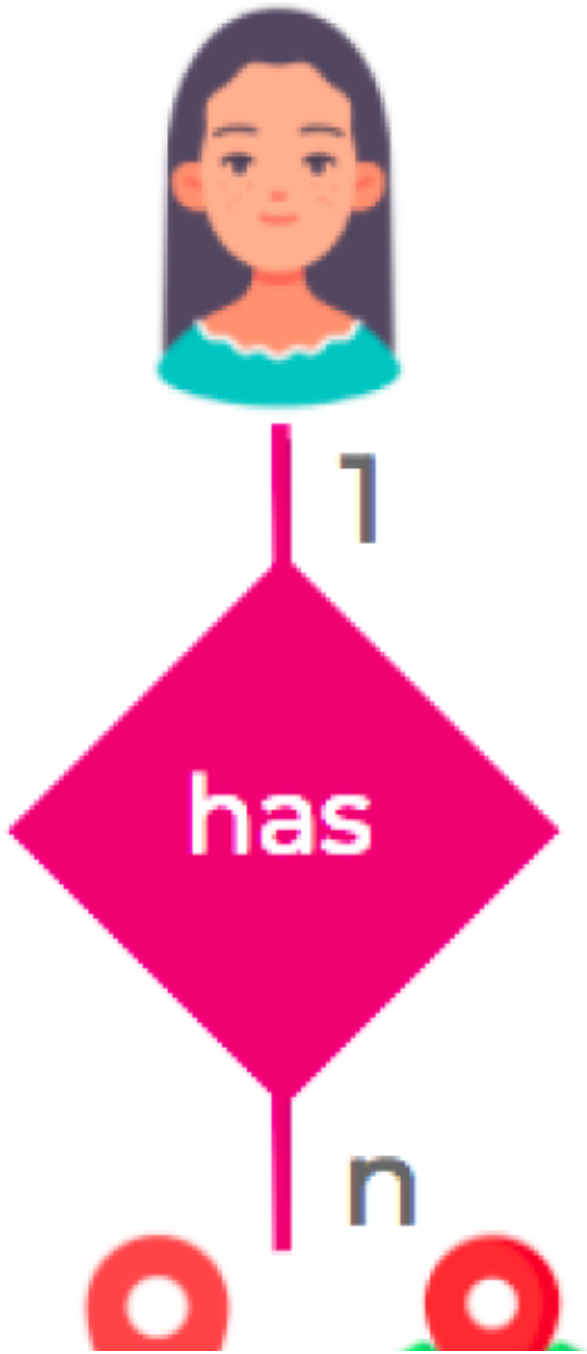
- A customer has **only one** cart.
- A cart is related to **only one** customer.
- Hence, the relation between customer and cart entities is **One-to-One relation**.

Relation Between Cart and Products


n
contains
m

- A cart can have **many** products.
- A product can be in **many** carts.
- Therefore, the relation between cart and product is **Many-to-Many relation**.

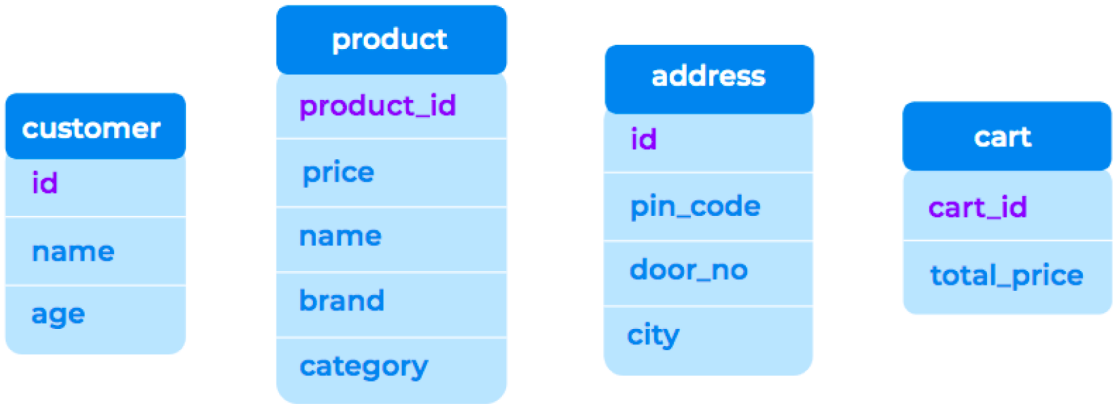
Relation Between Customer and Address


1
has
n

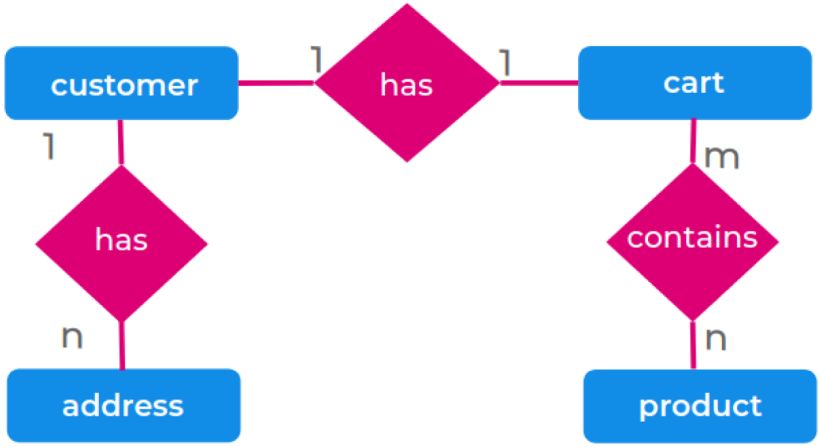
- A customer can have **multiple** addresses.
- An address is related to **only one** customer.
- Hence, the relation between customer and address is **One-to-Many relation**.

Attributes

Following are the attributes for the entity types in the e-commerce scenario.
Here, attributes like id, product_id, etc., are **key attributes** as they **uniquely identify each entity** in the entity type.



ER Model of e-commerce application



Submit Feedback

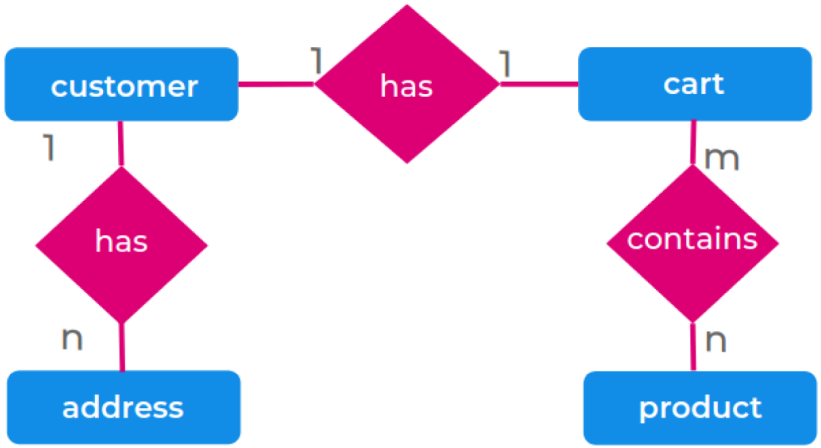
ER Model to Relational Database

In the previous cheatsheet, we've learnt to build an ER model for a given scenario. Now, let's convert this ER model to Relational Database.
Let's consider the same e-commerce application.

E-commerce Application

In a typical e-commerce application,

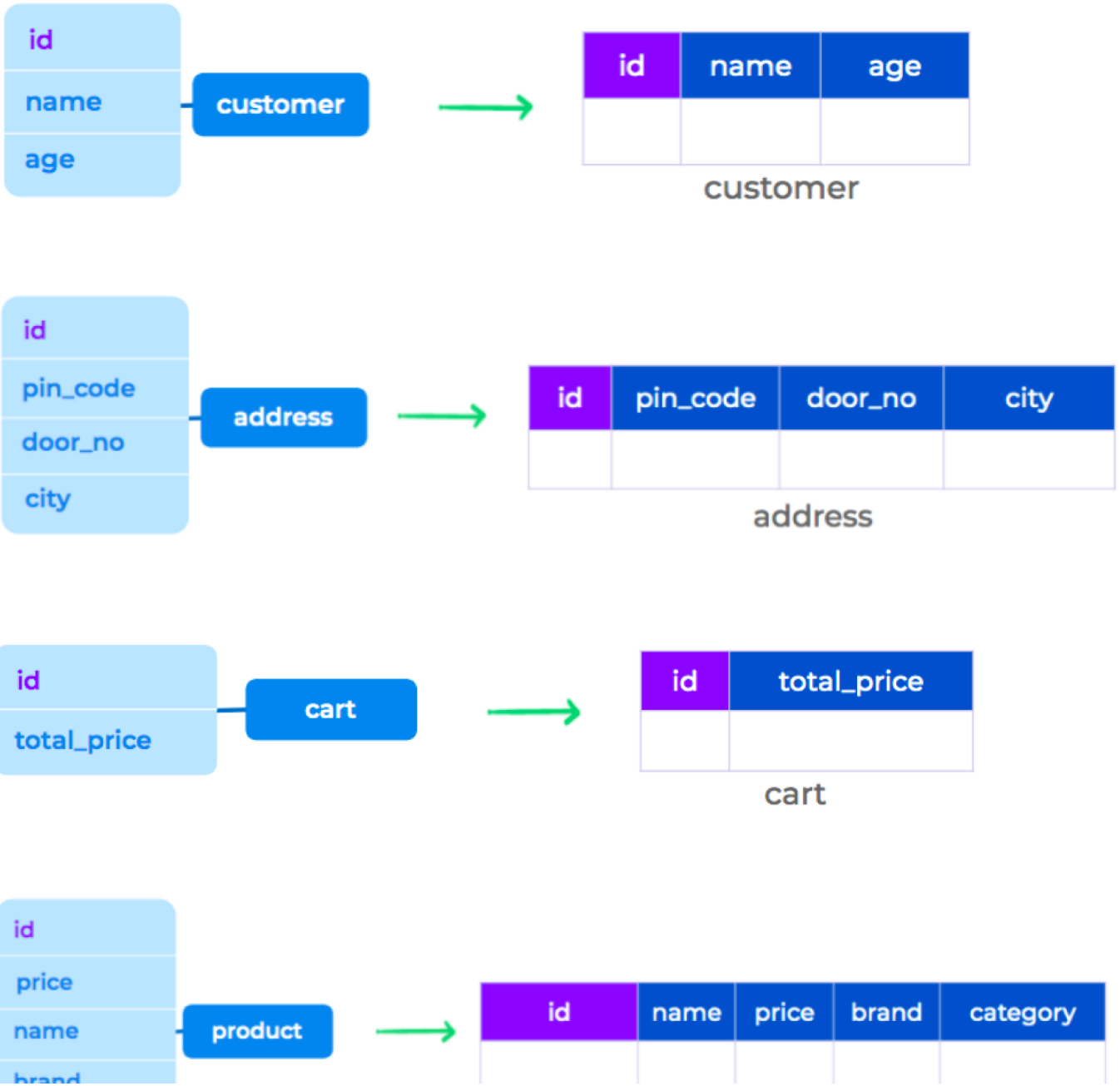
- ***Customer*** has only one ***cart***. A ***cart*** belongs to only one ***customer***
- ***Customer*** can add products to ***cart***
- ***Cart*** contains multiple ***products***
- ***Customer*** can save multiple ***addresses*** in the application for further use like selecting delivery address



Entity Type to Table

Primary key: A minimal set of attributes (columns) in a table that uniquely identifies rows in a table.

In the following tables, ***all the ids are primary keys*** as they ***uniquely identify each row*** in the table.


id
name
age
customer
id
name
age
customer
id
pin_code
door_no
city
address
id
pin_code
door_no
city
address
id
total_price
cart
id
total_price
cart
id
price
name
product
id
name
price
brand
category

category

product

Relationships

Relation Between Customer and Address - One to Many Relationship

- A customer can have multiple addresses.
- An address is related to only one customer.

We store the primary key of a customer in the address table to denote that the addresses are related to a particular customer.

This new column/s in the table that refer to the primary key of another table is called **Foreign Key**.

id	pin_code	door_no	...	customer_id
1001	517130	6-1	...	1
1002	615670	6-13	...	1

address

PK
FK
Unique FK

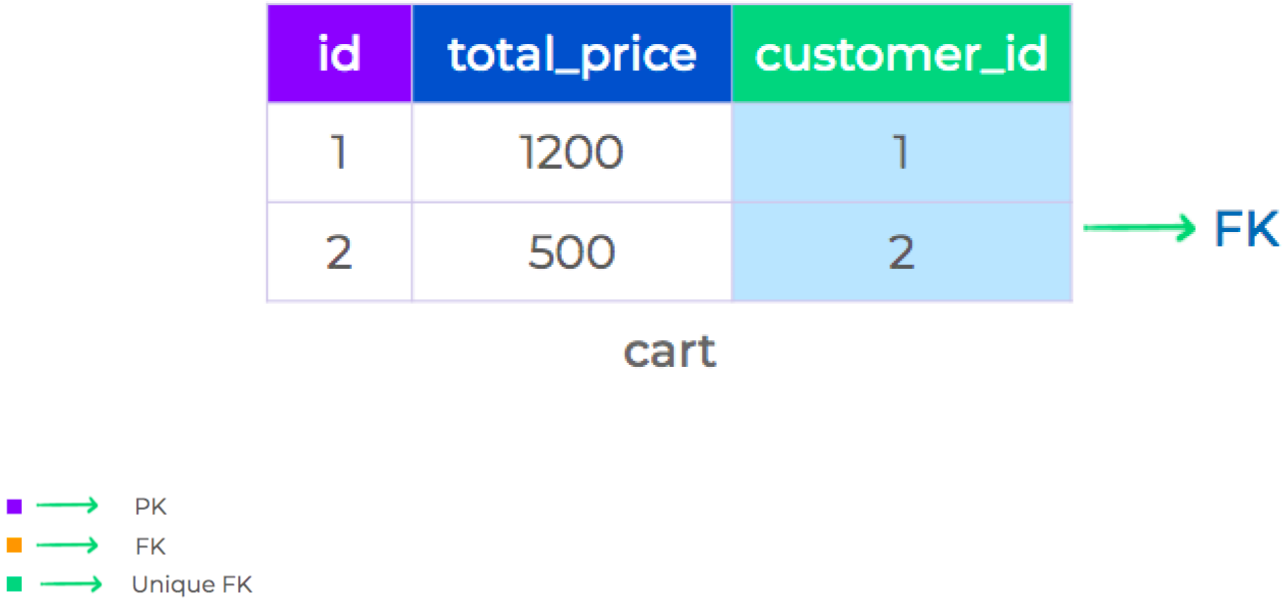
Here,

`customer_id` is the foreign key that stores `id` (primary key) of customers.

Relation Between Cart and Customer - One to One Relationship

- A customer has only one cart.
- A cart is related to only one customer.

This is similar to one-to-many relationship. But, we need to ensure that ***only one cart is associated to a customer***

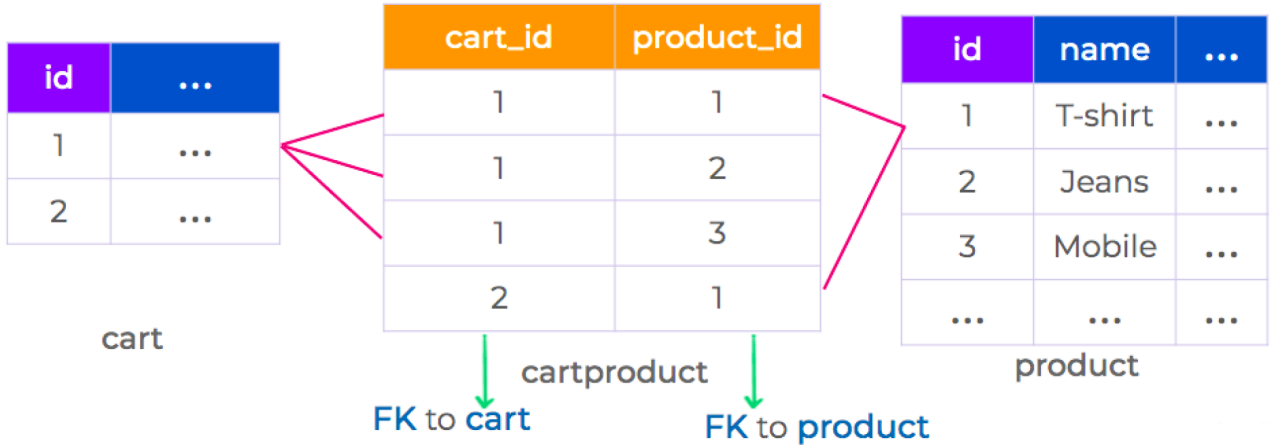


Relation Between Cart and Products - Many to Many Relationship

- A cart can have many products.
- A product can be in many carts.

Here, we cannot store either the primary key of a product in the cart table or vice versa.

To store the relationship between the cart and product tables, we use a **Junction Table**.



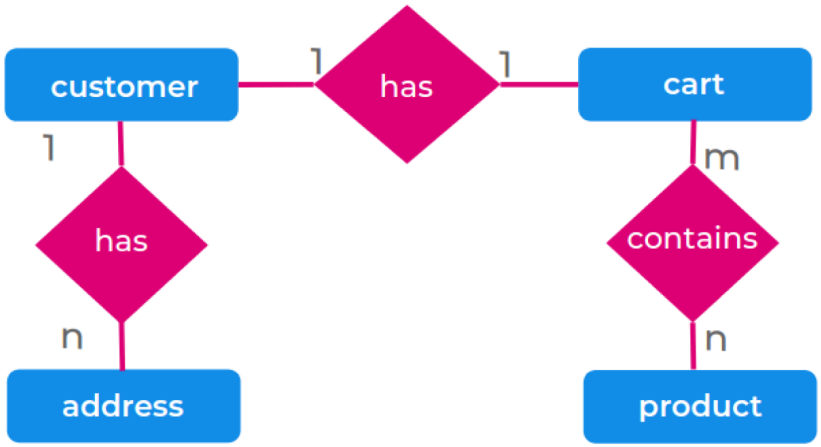
Note

We store the properties of a the relationship in the junction table. For example, quantity of each product in the cart should be stored in the junction table `cart_product`

E-commerce Usecase: ER Model to Relational Database

Following ER model is represented as the below tables in the relational database.

ER Model



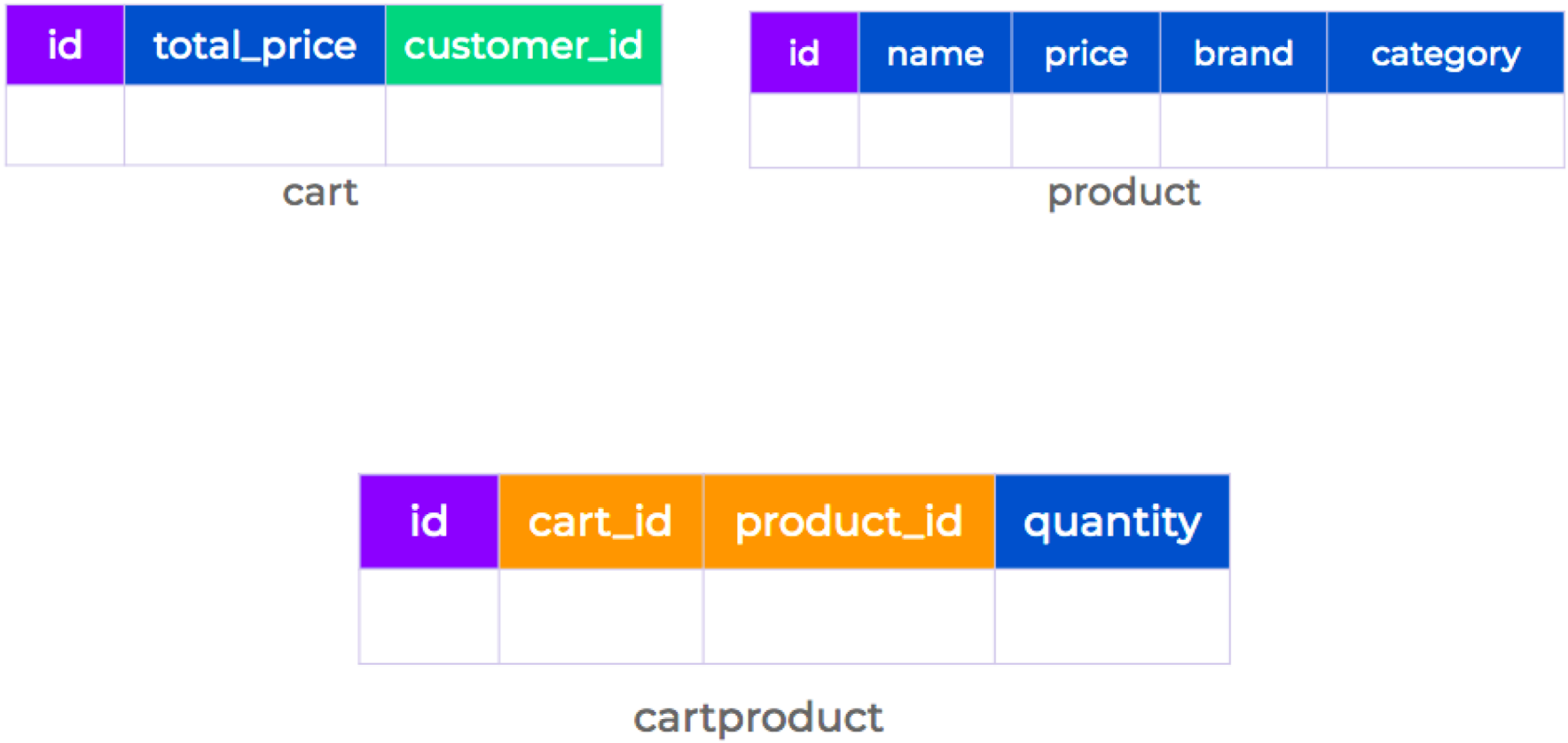
Relational Database

id	name	age

customer

id	pin_code	door_no	city	customer_id

address


id
total_price
customer_id
cart
id
name
price
brand
category
product
id
cart_id
product_id
quantity
cartproduct

Submit Feedback

Creating a Relational Database

In the previous sessions, we've explored how to represent an ER model in the form of tables in a relational database.

Now, let's create tables to store the data in the database by defining all the columns and relationships between the tables.

Consider the **e-commerce scenario**. The tables, columns and the relations between them are s follows.

id	name	age

customer

id	pin_code	door_no	city	customer_id

address

id	total_price	customer_id

cart

id	name	price	brand	category

product

id	cart_id	product_id	quantity

cartproduct

Primary Key

Following syntax creates a table with

`c1` as the primary key.

Syntax

```SQL
CREATE TABLE table_name (
    c1 t1 NOT NULL PRIMARY KEY,
...
    cn tn,
);
```

Foreign Key

In case of foreign key, we just create a foreign key constraint.

Syntax

```SQL
CREATE TABLE table2(
  c1 t1 NOT NULL PRIMARY KEY,
  c2 t2,
  FOREIGN KEY(c2) REFERENCES table1(c3) ON DELETE CASCADE
);
```

Understanding

```SQL
FOREIGN KEY(c2) REFERENCES table1(c3) [] []
```

Above part of the foreign key constraint ensure that foreign key can only contain values that are in the referenced primary key.

```SQL
ON DELETE CASCADE
```

Ensure that if a row in

`table1` is deleted, then all its related rows in `table2` will also be deleted.

Note

To enable foreign key constraints in SQLite, use `PRAGMA foreign_keys = ON;` By default it is enabled in our platform!

Creating Tables in Relational Database

Customer Table

SQL

```sql
CREATE TABLE customer (
   id INTEGER NOT NULL PRIMARY KEY,
   name VARCHAR(250),
   age INT
);
```

Product Table

SQL

```sql
CREATE TABLE product (
  id INTEGER NOT NULL PRIMARY KEY,
  name VARCHAR(250),
  price INT,
  brand VARCHAR(250),
  category VARCHAR(250)
);
```

Address Table

SQL

```sql
CREATE TABLE address(
  id INTEGER NOT NULL PRIMARY KEY,
  pin_code INTEGER,
  door_no VARCHAR(250),
  city VARCHAR(250),
  customer_id INTEGER,
  FOREIGN KEY (customer_id) REFERENCES customer(id) ON DELETE CASCADE
);
```

Cart Table

SQL

```sql
CREATE TABLE cart(
  id INTEGER NOT NULL PRIMARY KEY,
  customer_id INTEGER NOT NULL UNIQUE,
  total_price INTEGER,
  FOREIGN KEY (customer_id) REFERENCES customer(id) ON DELETE CASCADE
);
```

Cart Product Table (Junction Table)

SQL

```sql
CREATE TABLE cart_product(
  id INTEGER NOT NULL PRIMARY KEY,
  cart_id INTEGER,
  product_id INTEGER,
  quantity INTEGER,
  FOREIGN KEY (cart_id) REFERENCES cart(id) ON DELETE CASCADE,
  FOREIGN KEY (product_id) REFERENCES product(id) ON DELETE CASCADE
);
```

MARKED AS COMPLETE

JOINS

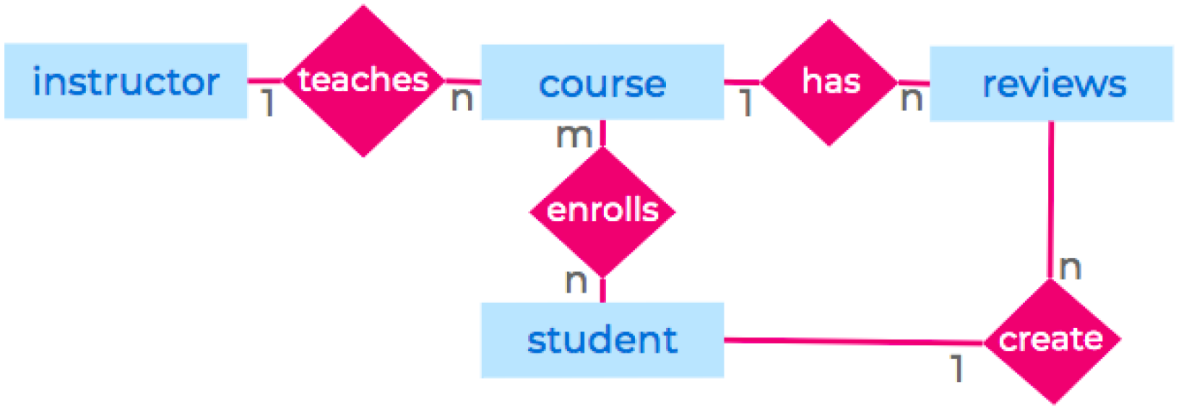
So far, we have learnt to analyse the data that is present in a single table. But in the real-world scenarios, often, the data is distributed in multiple tables. To fetch meaningful insights, we have to bring the data together by combining the tables.

We use JOIN clause to combine rows from two or more tables, based on a related column between them. There are various types of joins, namely Natural join, Inner Join, Full Join, Cross Join, Left join, Right join.

Let's learn about them in detail using the following database.

Database

Here, the database stores the data of students, courses, course reviews, instructors, etc., of an e-learning platform.



Refer the tables in the code playground for a better understanding of the database.

Natural JOIN

`NATURAL JOIN` combines the tables based on the common columns.

Syntax

```sql
SELECT *
FROM table1
   NATURAL JOIN table2;
```

Example

1. Fetch the details of courses that are being taught by "Alex".

Solving this problem involves querying on data stored in two tables, i.e., `course` & `instructor` . Both the tables have common column `instructor_id` . Hence, we use Natural Join.

```sql
SELECT course.name,
  instructor.full_name
FROM course
  NATURAL JOIN instructor
WHERE instructor.full_name = "Alex";
```

Output

name	full_name
Cyber Security	Alex

Try it Yourself!

Question 1:

Get the details of the instructor who is teaching "Cyber Security".

Expected Output:

full_name	gender
Alex	M

Question 2:

Get student full name and their scores in "Machine Learning" (course with id=11).

Expected Output:

full_name	score
Varun	80
Sandhya	90

INNER JOIN

`INNER JOIN` combines rows from both the tables if they meet a specified condition.

Syntax

SQL

```sql
SELECT *
FROM table1
   INNER JOIN table2
ON table1.c1 = table2.c2;
```

Note

We can use any comparison operator in the condition.

Example

Get the reviews of course “Cyber Security” (course with id=15)

```sql
SELECT student.full_name,
   review.content,
   review.created_at
FROM student
   INNER JOIN review
ON student.id = review.student_id
WHERE review.course_id = 15;
```

Output

full_name	content	created_at
Ajay	Good explanation	2021-01-19
Ajay	Cyber Security is awesome	2021-01-20

Try it Yourself!

Question 1:

Get the details of students who enrolled for "Machine Learning" (course with id=11).

Expected Output:

full_name	age	gender
Varun	16	M
Sandhya	19	F

Question 2:

Get the reviews given by "Varun" (student with id = 1)

Expected Output:

course_id	content	created_at
11	Great course	2021-01-19

LEFT JOIN

In

`LEFT JOIN` , for each row in the left table, matched rows from the right table are combined. If there is no match, NULL values are assigned to the right half of the rows in the temporary table.

Syntax

SQL

```sql
SELECT *
FROM table1
   LEFT JOIN table2
ON table1.c1 = tabl2.c2;
```

Example

Fetch the full_name of students who have not enrolled for any course

SQL

```sql
SELECT student.full_name
FROM student
   LEFT JOIN student_course
```

```
4  ON student.id = student_course.student_id
5  WHERE student_course.id IS NULL;
```

Output

full_name
Afrin

Try it Yourself!

Question 1:

Get the course details that doesn't have any students.

Expected Output:

name
Linux

Question 2:

Get the instructors details who is not assigned for any course.

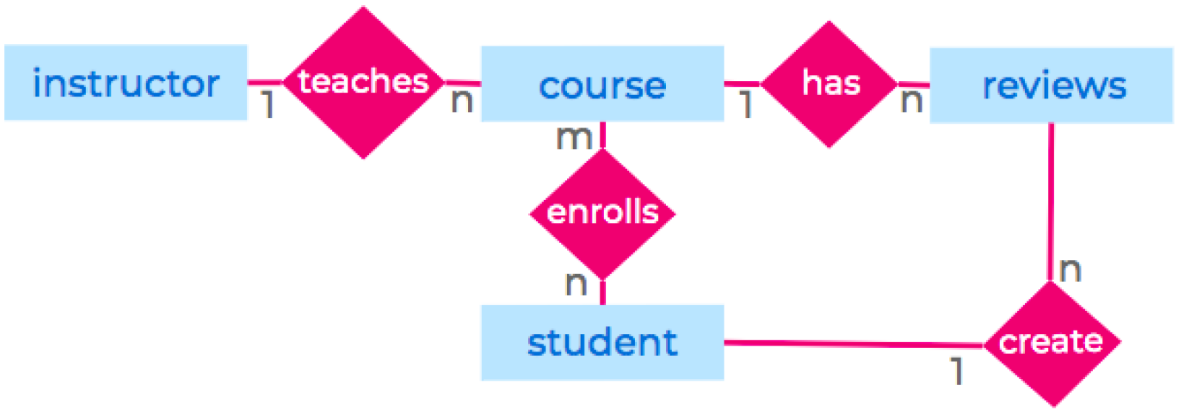
Expected Output:

full_name	gender
Bentlee	M

Querying with Joins

Database

Here, the database stores the data of students, courses, course reviews, instructors, etc., of an e-learning platform.



Refer the tables in the code playground for a better understanding of the database.

Joins on Multiple Tables

We can also perform join on a combined table.

Example

Fetch all the students who enrolled for the courses taught by the instructor “Arun” (id = 102)

```sql
SELECT T.name AS course_name,
   student.full_name
FROM (course
   INNER JOIN student_course
ON course.id = student_course.course_id) AS T
   INNER JOIN student
ON T.student_id = student.id
WHERE course.instructor_id = 102;
```

Output

course_name	full_name
Machine Learning	Varun
Machine Learning	Sandya

Note

Best Practices

1. Use

   `ALIAS` to name the combined table.

2. Use alias table names to refer the columns in the combined table.

Try it Yourself!

Question 1:

Fetch the name of the students who gave reviews to the "Machine Learning" course.

Expected Output:

full_name
Varun

Question 2:

Fetch the course names in which "Varun" has registered.

Expected Output:

course_name
Machine Learning

Let's learn about the Right Join, Full Join and Cross Join in the upcoming cheatsheet.

Using joins with other clauses

We can apply

`WHERE` , `ORDER BY` , `HAVING` , `GROUP BY` , `LIMIT` , `OFFSET` and other clauses (which are used for retrieving data tables) on the temporary joined table as well.

Example:

Get the name of the student who scored highest in "Machine Learning" course.

```sql
SELECT student.full_name
FROM (course
   INNER JOIN student_course
ON course.id = student_course.course_id) AS T
   INNER JOIN student
ON T.student_id = student.id
WHERE course.name = "Machine Learning"
ORDER BY student_course.score DESC
LIMIT 1;
```

Output

full_name
Sandhya

Try it Yourself!

Question 1:

Get all the courses taken by the student with id=1 and his respective scores in each course

Expected Output

name	score
Machine learning	80

Question 2:

Get all the students who registered for at least one course.

Expected Output

full_name
Varun
Ajay
Sandhya

Using joins with aggregations

We can apply

`WHERE` , `ORDER BY` , `HAVING` , `GROUP BY` , `LIMIT` , `OFFSET` and other clauses (which are used for retrieving data tables) on the temporary joined table as well.

- Get the highest score in each course.

SQL

```sql
SELECT
  course.name AS course_name,
  MAX(score) AS highest_score
FROM
  course
  LEFT JOIN student_course
ON course.id = student_course.course_id
GROUP BY
  course.id;
```

Output

course_name	highest_score
Machine Learning	90
Cyber Security	60
Linux	

Try it Yourself!

Question 1:

Get the course name and the average score for each course.

Expected Output

name	avg_score
Machine Learning	85
Cyber Security	60
Linux	

Question 2:

Get the number of students in each course .

Expected Output

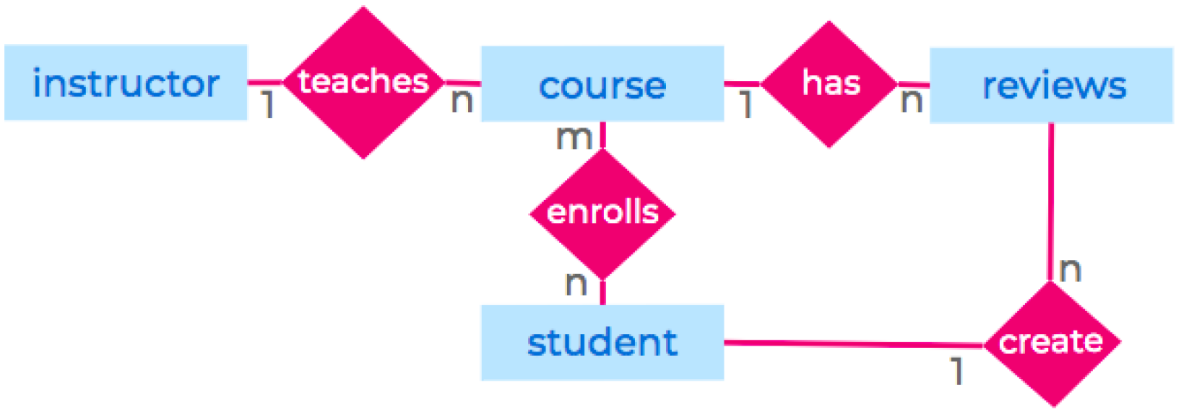
name	no_of_students
Machine learning	2
Cyber Security	1

name	no_of_students
linux	0

JOINS Cont'd

Database

The database stores the data of students, courses, course reviews, instructors, etc., of an e-learning platform.



Refer the tables in the code palyground for a better understanding of the database.

RIGHT JOIN

RIGHT JOIN or RIGHT OUTER JOIN is vice versa of LEFT JOIN.

I.e., in

`RIGHT JOIN` , for each row in the right table, matched rows from the left table are combined. If there is no match, NULL values are assigned to the left half of the rows in the temporary table.

Syntax

```SQL
SELECT *
FROM table1
   RIGHT JOIN table2
ON table1.c1 = table2.c2;
```

Which is similar to

```SQL
SELECT *
FROM table2
   LEFT JOIN table1
ON table1.c1 = table2.c2;
```

Example

Following query performs RIGHT JOIN on course and instructor tables

```SQL
SELECT course.name,
    instructor.full_name
FROM course
   RIGHT JOIN instructor
ON course.instructor_id = instructor.instructor_id;
```

Note

Right Join is not supported in some dbms(SQLite).

FULL JOIN

FULL JOIN or FULL OUTER JOIN is the result of both RIGHT JOIN and LEFT JOIN

Syntax

SQL

```sql
SELECT *
FROM table1
   FULL JOIN table2
ON c1 = c2;
```

Example

Following query performs FULL JOIN ON course and instructor tables

SQL

```sql
SELECT course.name,
    instructor.full_name
FROM course
   FULL JOIN instructor
ON course.instructor_id = instructor.instructor_id;
```

Note

FULL JOIN is not supported in some dbms(SQLite).

CROSS JOIN

In CROSS JOIN, each row from the first table is combined with all rows in the second table.
Cross Join is also called as CARTESIAN JOIN

Syntax

```SQL
SELECT *
FROM table1
   CROSS JOIN table2;
```

Example

Following query performs CROSS JOIN on course and instructor tables

```SQL
SELECT course.name AS course_name,
    instructor.full_name AS instructor_name
FROM course
   CROSS JOIN instructor;
```

Output

course_name	instructor_name
Machine Learning	Alex
Machine Learning	Arun
Machine Learning	Bentlee
Cyber Security	Alex
...	...

SELF JOIN

So far, we have learnt to combine different tables. We can also combine a table with itself. This kind of join is called SELF-JOIN.

Syntax

```sql
SELECT t1.c1,
   t2.c2
FROM table1 AS t1
   JOIN table1 AS t2
ON t1.c1 = t2.cn;
```

Note

We can use any JOIN clause in self-join.

Example

Get student pairs who registered for common course.

```sql
SELECT sc1.student_id AS student_id1,
  sc2.student_id AS student_id2, sc1.course_id
FROM
   student_course AS sc1
   INNER JOIN student_course sc2 ON sc1.course_id = sc2.course_id
WHERE
    sc1.student_id < sc2.student_id;
```

Output

student_id1	student_id2	course_id
1	3	11

JOINS Summary

Join Type	Use Case
Natural Join	Joins based on common columns
Inner Join	Joins based on a given condition
Left Join	All rows from left table & matched rows from right table
Right Join	All rows from right table & matched rows from left table
Full Join	All rows from both the tables
Cross Join	All possible combinations

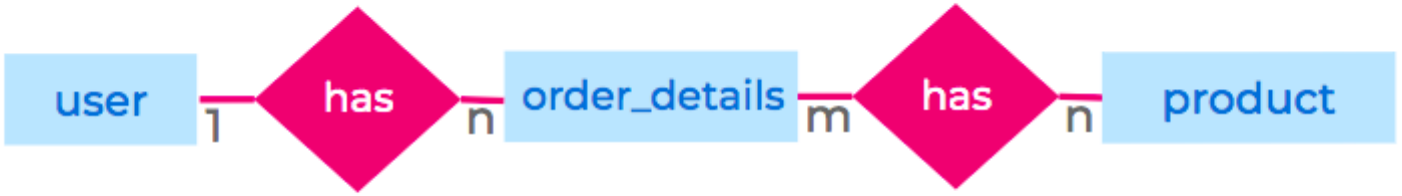
Views

Database:

The database stores the sample data of an e-commerce aplication.

Here, the database consists of

`user` , `order_details` and `product` tables that store the information of products, orders placed, and the products on the platform.



Refer the tables in the code playground for a better understanding of the database.

View

A view can simply be considered as a name to a SQL Query

Create View

To create a view in the database, use the

`CREATE VIEW` statement.

Example

Create

`user_base_details` view with id, name, age, gender and pincode.

SQL

```sql
CREATE VIEW user_base_details AS
SELECT id, name, age, gender, pincode
FROM user;
```

Note

In general, views are read only.

We cannot perform write operations like updating, deleting & inserting rows in the base tables through views.

Try it Yourself!

Create

`order_with_products` view with order_id, product_id, no_of_units, name, price_per_unit, rating, category, brand.

Querying Using View

We can use its name instead of writing the original query to get the data.

SQL

```sql
SELECT *
FROM user_base_details;
```

Output

id	name	age	gender	pincode
1	Sai	40	Male	400068
2	Boult	20	Male	30154
3	Sri	20	Female	700009
...	...	...	...	...

We can use same operations which are used on tables like WHERE clause, Ordering results, etc.

If we try to retrive data which is not defined in the view it raises an

`error` .

Example

SQL

```sql
SELECT name, address
FROM user_base_details
WHERE gender = "Male";
ORDER BY age ASC;
```

Output

SQL

```
Error: no such column:address
```

Try it Yourself!

From the

`order_with_products` view created above, get the name and no_of_units ordered in order_id = 802.

Expected Output

name	no_of_units
Oneplus 8 Pro	1
Gorilla Glass	1

List All Available Views

In SQLite, to list all the available views, we use the folowing query.

SQL

```sql
SELECT
  name
FROM
  sqlite_master
WHERE
  TYPE = 'view';
```

Output

name
order_with_products
user_base_details

Delete View

To remove a view from a database, use the

`DROP VIEW` statement.

Syntax

```sql
DROP VIEW view_name;
```

Example

Delete

`user_base_details` view from the database.

```sql
DROP VIEW user_base_details;
```

Advantages

- Views are used to write **complex queries** that involves **multiple joins**, **group by**, etc., and can be used whenever needed.
- **Restrict access** to the data such that a user can only see limited data instead of a complete table.

Subqueries

We can write nested queries, i.e., a query inside another query.

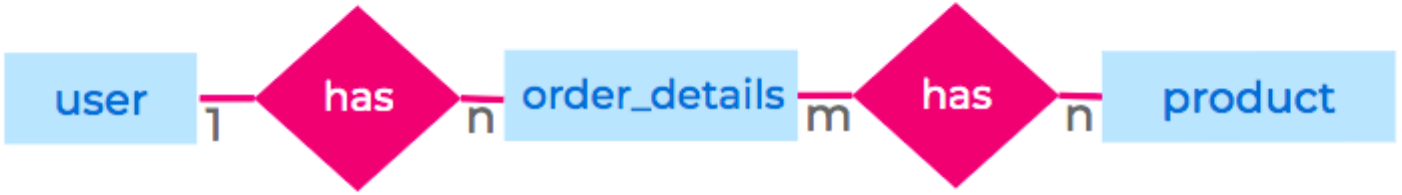
Let's understand the scenarios where subqueries can be used with the following database.

Database:

The database stores the sample data of an e-commerce aplication.

Here, the database consists of

`user` , `order_details` and `product` tables that store the information of products, orders placed, and the products on the platform.



Refer the tables in the code playground for a better understanding of the database.

Examples

Example 1:

Get the rating variance of products in the "WATCH" category. Rating variance is the difference between average rating and rating of a product.

Here, we need to write an expression to subtract rating of each product from the average rating as following.

```sql
SELECT name,
(average_rating - rating) AS rating_variance
...
```

Replace average_rating with a query which computes the average.

```sql

SELECT
    name,
    (
        SELECT AVG(rating)
        FROM product
        WHERE category = "WATCH"
    ) - rating AS rating_variance
FROM product
WHERE category = "WATCH";
```

Output

name	rating_variance
Analog-Digital	-0.766666666666667
Fastfit Watch	-0.366666666666666
Fastrack M01	0.333333333333334
...	...

Example 2:

Fetch all the products whose ratings is greater than average rating of all products.

Output

SQL

```sql

SELECT *
FROM product
WHERE rating > (
        SELECT AVG(rating)
        FROM product
    );
```

Expected Output

product_id	name	price_per_unit	rating	category	brand
202	Biotique Almond Soap	34	4.5	SOAP	BIPTIQUE
203	Boat Stone Speaker	1999	4.3	SPEAKER	BOAT
...	...	...	...	...	...

Example 3:

Fetch all the order_ids in which order consists of mobile (product_ids : 291, 292, 293, 294, 296) and not ear phones (product_ids : 227, 228, 229, 232, 233).

SQL

```sql
SELECT
  order_id
FROM
  order_details
WHERE
  order_id IN (
    SELECT
      order_id
```

```
9     FROM
10      order_product
```

Expand

Output

order_id
801
802
806
807

Possible Mistakes

In SELECT Clause

A subquery in the SELECT clause can have only one column.

Query

SQL

```sql
SELECT name, (
   SELECT AVG(rating), MAX(rating)
      FROM product
      WHERE category = "WATCH"
   ) - rating AS rating_variance
FROM product
WHERE category = "WATCH";
```

Output

SQL

```
Error:
sub-select returns 2 columns - expected 1
```

In WHERE Clause

Query

In WHERE clause, a subquery can have only one column.

SQL

```
 SELECT
   order_id, total_amount
FROM order_details
WHERE total_amount > (
   SELECT total_amount, order_id
   FROM order_details
);
```

Output

SQL

```
Error: Row value misused
```

Try it Yourself!

Question 1

Get the rating variance of products in the "MOBILE" category. Rating variance is the difference between average rating and rating of a product.

Rating variance is the difference between average rating and rating of a product

Expected Output Format:

name	rating_variance
Oneplus 8 Pro	-0.040000000000000924
Oneplus 8t Pro	0.2599999999999989
...	...

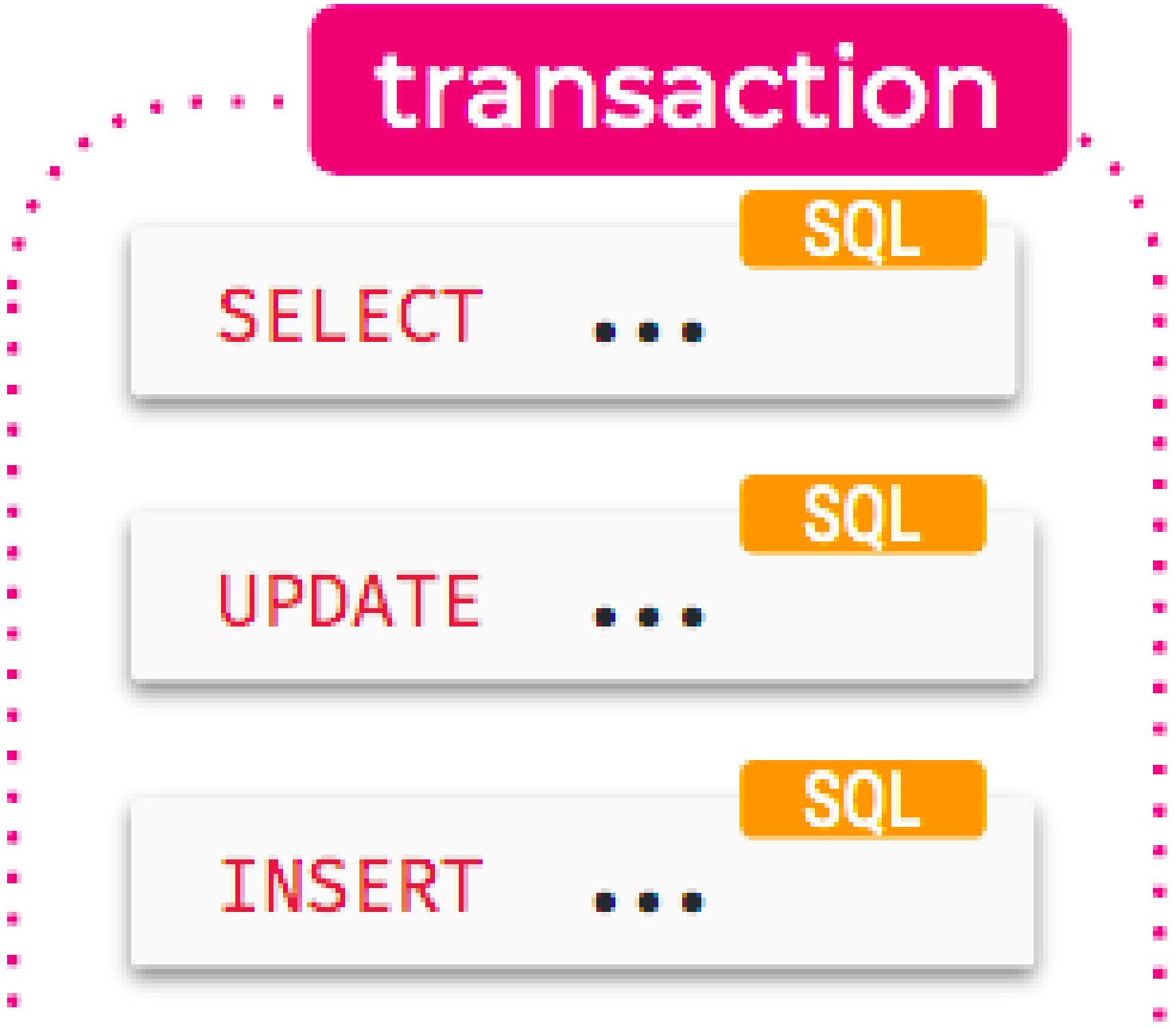
Question 2

Get all the products from the "MOBILE" category, where rating is greater than average rating.

Expected Output Format:

name	rating
Oneplus 8 Pro	4.5
Mi 10T	4.5
Samsung S21 Ultra	4.7
...	...

Transactions


transaction
SQL
SELECT ...
SQL
UPDATE ...
SQL
INSERT ...

A transaction is a logical group of one or more SQL statements.

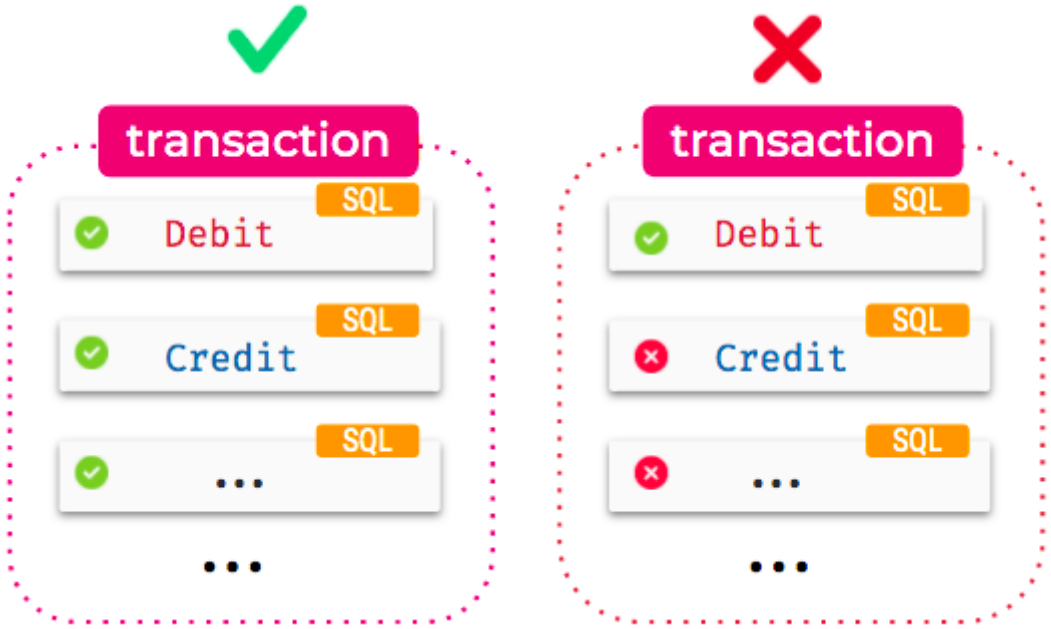
Transactions are used in various scenarios such as banking, ecommerce, social networks, booking tickets, etc.

A transaction has four important properties.

- Atomicity
- Consistency
- Isolation
- Durability

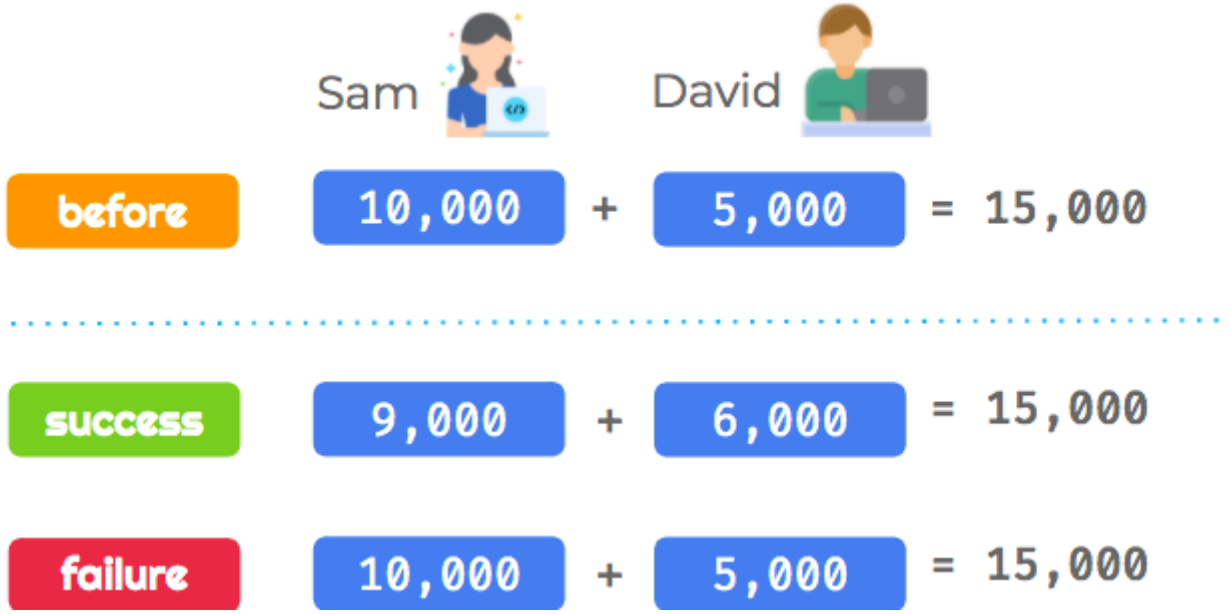
Atomicity

Either all SQL statements or none are applied to the database.



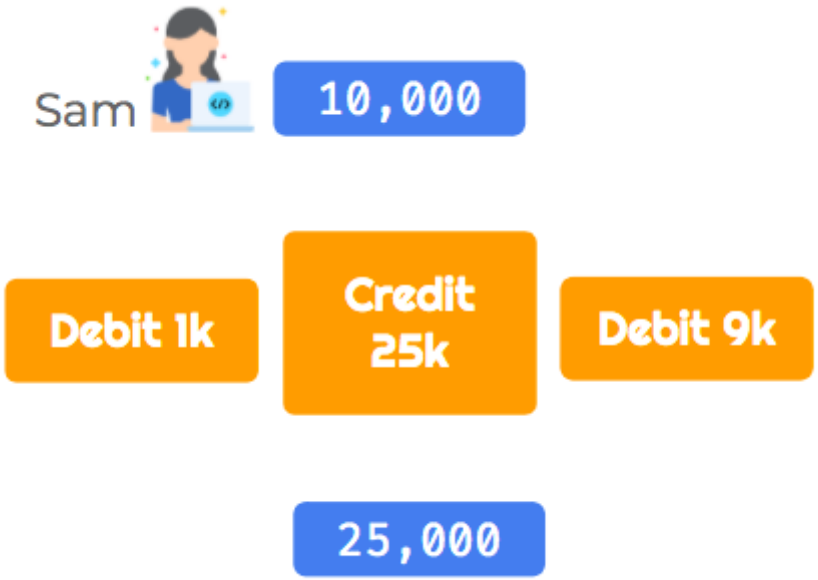
Consistency

Transactions always leave the database in a consistent state.



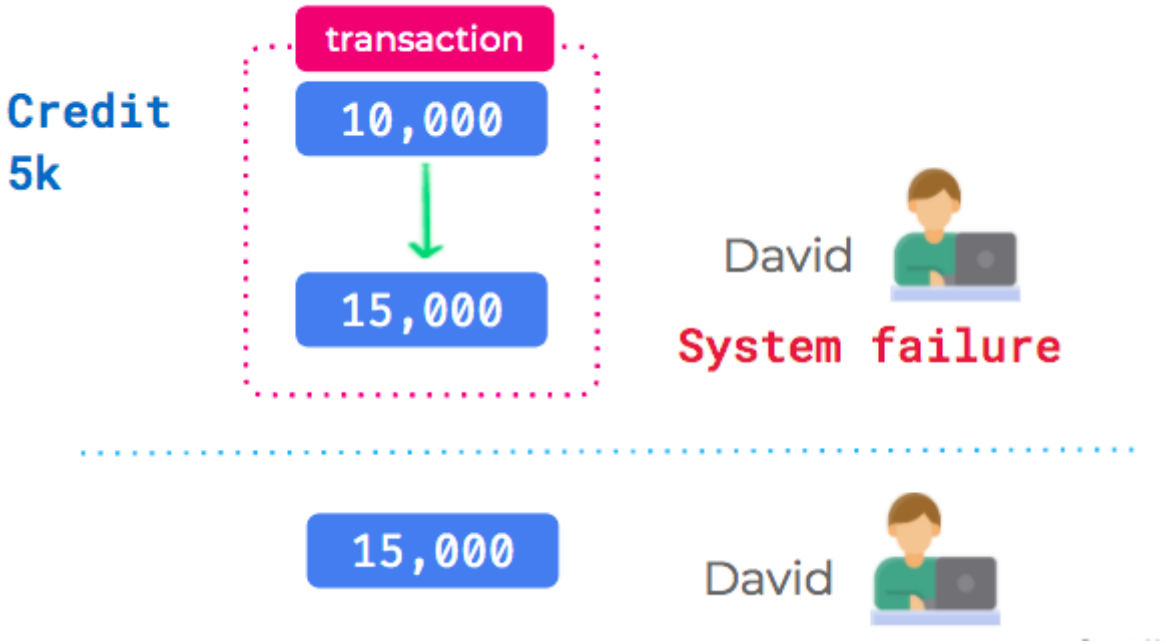
Isolation

Multiple transaction can occur at the same time without adversely affecting the other.



Durability

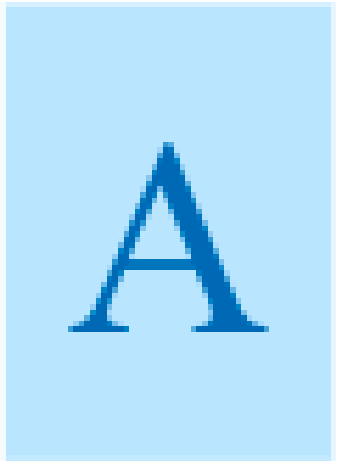
Changes of a successful transaction persist even after a system crash.



These four properties are commonly acronymed as ACID.

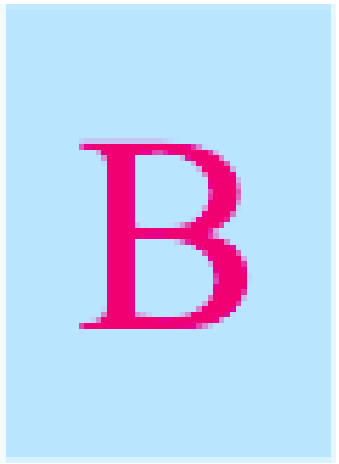
Atomicity **C**onsistency **I**solation **D**urable

Indexes



ab... 02

az... 23

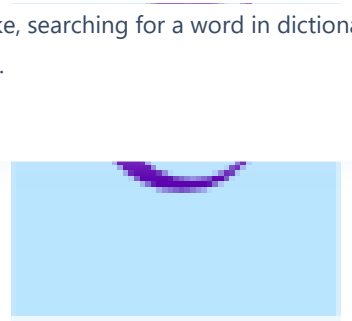


ba... 24

bz... 32

ca... 33

In scenarios like, searching for a word in dictionary, we use index to easily search for the word. Similarly, in databases, we maintain indexes to speed up the search for data in a table.



cz...

43

Submit Feedback