Introduction to database

Any sort of information that is stored is called data.

Example

Messages & multimedia.

Database:-

An organised collection of data

DBMS

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

Database ⟶ DBMS ⟶ web application server
Data

Advantages

Security

Ease of use → simple way to use

Durability → accessing

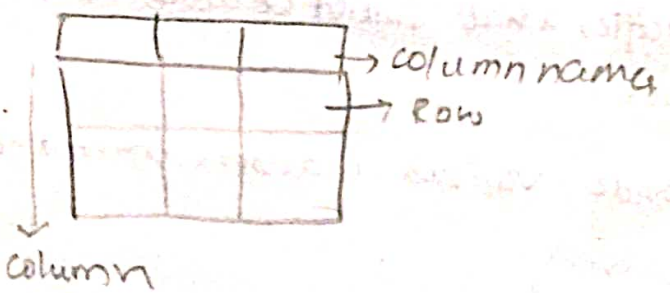
performance

Types of database

→ Relational
→ Analytical
→ key value
→ column family
→ Graph
→ Document

Relational database

Data Organised in the form of data.



Non-relational database

all remaining other than relational comes under non-relational databases

Relational DBMSs

A relational DBMS designed specifically for relational databases. Relational databases organize the data in form of tables

Example

Oracle, postgresql

Non-relational DBMS

A non-relational DBMS specifically for non-relational databases that can store the data in non-tabular form

Example

Oracle, Redis

frequently used datatypes

Integers, Date, float, Time, Varchar, Datetime, Text, Boolean

Introduction to SQL

SQL from a developer point of view RDMS using would be same.

RDMS ← Tables ← Database ← RDMS

To work with RDMS ⟵ ⟶ will be using SQL.

SQL is used to perform operations of RDMS.

↓

structured Query language ⟶ request

↳ formal syntax.

User specifies what should be done (✓)

how(x)

SQL provide various clauses (commands) to perform operations.

C — create
R — read
U — update
D — delete

} operations

SQL - Create operation

Insert clause can be used to create new row in a table.

SQL - Read operation

Select clause can be used to access rows in a table selectively

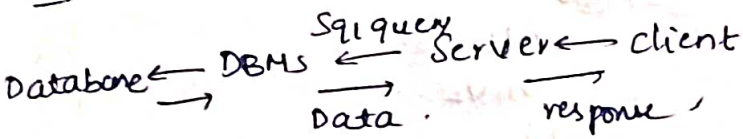
SQL - Update operation

update clause can be used to update the existing table

SQL - delete operation

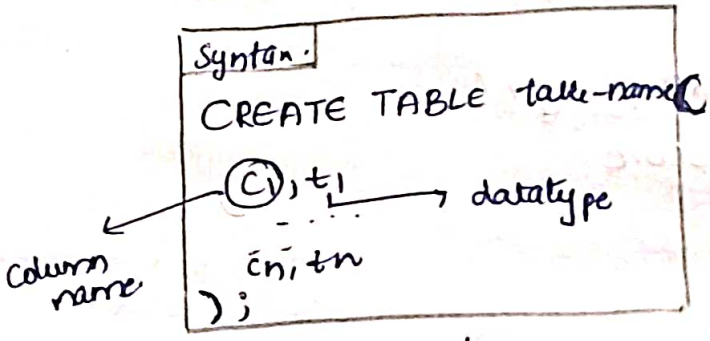
Delete clause can be used to delete existing row in a table

Application flow



Create table using SQL

Creates a new table in the database.



Datatypes in SQL

	Syntax	
Integer	Integer	0, 123, -333 etc
String	VARCHAR	"Book", "423"
text	TEXT	"we can do it"
Float	FLOAT	24.3
Date	DATE	2020-12-17
Time	TIME	12:00:00
DATETIME	DATETIME	2020-12-17 12:00:00
Boolean	BOOLEAN	True, false.

Example

```
CREATE Table player(
 name VARCHAR,
 age INT,
 score INT(3)x
);
```

Pragma

PRAGMA TABLE-INFO command returns a table information

Syntax

```
PRAGMA TABLE_INFO
  ( PLAYER );
```

↓ Table name.

Note

If the table name given in the query do not Exist and do not give any result

Inserting Data using SQL

Inserts a new row in the given table

Syntax

```
INSERT INTO table-name(
  c1, c2 ...
)
VALUES(
  v1, v2 ...
);
```

c1 → column

v1 → value.

for example

```
INSERT INTO player (
  name, age, score
)
VALUES (
Dhoni, 37, 19.2
);
```

Retrieve Data

We will using select clause.

Syntax

```
SELECT c1, c2 ...
FROM table-name;
```

→ Example → select name, age, score FROM players;

INSERT Multiple rows

```
INSERT INTO PLAYER (
  name, age, score
)
VALUES
  ('Sam', 39, 35),
  ('John', 40, 35);
```

← Example ←

Syntax

```
INSERT INTO table-name(
c1, c2, ...
)
VALUES
  (v1, v2, ... vn),
  (v1, v2 ... vn)
);
```

Selecting columns

select name, age
FROM Player

* Here only name and age will be there in output

Retrieving all columns

Retreive all columns from the given table

Syntax
SELECT *
from table-name;

Selecting rows

Retrieves only selected rows from a table where condition states that only condition satisfy only that row filter

Syntax:
SELECT *
from table-name
WHERE CONDITION; — imp.

Example → SELECT *
FROM PLAYER
WHERE NAME = "RAM";

update Data all ROWS

Updates value at column C1 to value V1 for all rows in the given table.

Example
UPDATE player
SET score = 0;

Syntax:
UPDATE table-name
SET C1 = V1;

column name — which value.

Update data of specific rows

Updates values at column C1 to Value V1 for the rows which satisfy the condition in the given table.

Syntax
UPDATE table_name;
SET C1 = V1
WHERE condition

Example → UPDATE player
set score = 90 — X
WHERE name = "RAM"; ✓

different.

Delete data

Delete all the rows which match the given condition in the given table

Syntax
DELETE FROM TABLE_Name
Where condition;

Delete all rows

Deletes all the rows from the given table

Syntax
DELETE FROM TABLE_NAME

Drop table

Drop a table from the database.

Syntax
DROP TABLE TABLENAME

(*) case insensitive

Alter table

Add a column to the table with the given datatype

Syntax.
ALTER TABLE TABLE_NAME
Add (C) (A) ;
column datatype

Altertable Rename column

Rename a column in the table.

Syntax
ALTER TABLE table_name
RENAME column c1 to c2

Altertable DROP Column

Delete the given column c1 from the table

Syntax
ALTER Table Table_name
DROP Column c1 ;

Querying with

In applications like Amazon (or) flipkart the data is fetched by making wide range of queries

Example

products in Amazon and flipkart

Sql provides various operators to perform wide range of queries on DBMS

Two categories

→ comparision operators
→ logical operators.

Comparision operators

=	>=	IN
<> (Not equal)	<	Between
>	<=	
Like.		

→ how do we use this in dBMS syntax →

Syntax

Select * from table-name
where C operator V

WHERE category = "FOOD";
operator

WHERE category <> "FOOD";
Not equal to operator

String operation

↳ sequence of characters.

Like → operator.

like operator is used to perform queries on strings.

⊙ Retreives all the rows that match the given pattern.

Syntax.

Select *
from table-name.
where C like pattern;

Example

Exact match
starts with.
ends with
contains
pattern
matching

We write patterns using the following wildcard characters.

% → percentile

_ → underscore

% → represents '0' or more characters.

Shirt% → matches all the strings which have at begging

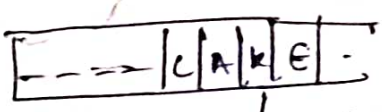
- Yes → shirt ✓
  Shirt with checks ✓
- No → Blackshirt (x)

→ Where name Like Shirt%

→ Shirt_ matches all the strings which have shirt at begging after shirt only one character.

- Shirts ✓ (Yes)
- No → shirt x
  Shirt with checks x

→ %cake → name end with cake.



end with cake.

→ %Jeans% → name contains Jeans

- Yes → (one jeans, colour Jean, Jeanspant)
- Shirt x

Querying with SQL part-2

Logical operators

AND

Retrieves all the rows which satisfy both condition1 and condition2

Syntax:

```
select *
from table-name
where c1 AND c2
```

OR operator

Retrives all the rows which satisfy atleast one of the given conditions.

Syntax.

```
Select *
from tablename
where c1 OR c2;
```

NOT operator

Not is used to negate a condition (opposite of does not satisfy) in where clause.

Syntax

```
Select *
FROM table-name
where
  NOT condition
```

Example

```
select *
from product
where
Not brand like "Denim"
```

using Multiple logical operators

for Example:-

Redmi brand and rating above 4 or oneplus brand

```
select *
FROM product
where
brand = "Redmi" and rating > 4
or brand = "oneplus";
```

precedence

① Brand = "Redmi"

② And rating > 4

③ OR brand = "oneplus"

(2 1 3) (1 1 2)

NOT	high
AND	↓
OR	low

Querying with SQL part-3

To precise the lengthy operator we will using IN operator.

In

Retrieve rows
where value of C1 is in
the given values

Syntax

```
Select *
From product
where C1 in (V1,V2,V3);
```

C1 → column name
(V1,V2,V3) → list of value

Get products whose brand
belong to
puma OR
roadster OR
levis OR
Nike

```
Select *.
From product
where brand in
("puma", "roadster", "Nike");
```

Between operator

Retrieve all the rows where value of C1 lies in between V1 and V2

C1 >= V1 and C2 <= V2

Syntax

```
select *
from table_name
where C1 between V1 and V2
```

→ Example—

```
Select *
from product
where price between
1000 and 5000;
```

In

check if a particular value Exist in
set of values

between

check if a particular value exist in the
given range

Ordering of Results

ORDER By clause is used to order rows (in arrange in Lexical order)

Syntax
Select *
From table name
ORDER By (1) Asc - Ascending order
DESC → descending order
column name

default - Ascending

Pagination

And fetch and the next chunk only when the user asks it

Previous → 1 → 2 → 3 → 4 → 5 → 6 → next

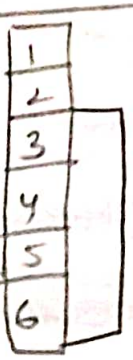
we use **limit** and **offset** clauses to select a chunk of results

limit → number of rows to fetch
offset → where the rows are to be selected
Rows are selected offset +1 (start)
limit + offset (end)

offset = 1
start
end = limit

limit - 4
offset - 2

offset +1
= 2+1 = 3 (start)
offset + limit
4+2 = 6 (end)



Syntax
Select *
from table_name
limit L
offset O;

Example
→ Select *
FRom product
ORDER By rating desc
Limit 5
offset 0;

Distinct results

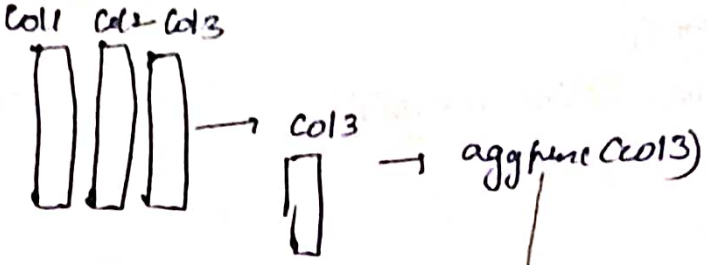
get unique brand names

Syntax
Select distinct brand
from product
ORDER by brand

→ only unique no repetition →

Aggregation

Combining multiple values into single value is called aggregation



- Sum
- Avg
- Min
- MAX
- Count

Sum → syntax →

Syntax

```
SELECT
Sum(Score)
FROM
Table_name
```

→

```
Select
operation (Column name)
FROM
Table_name
```

using where clause.

```
Select
Sum(Score)
from player
where name = "Ram";
```

→ only ram score will be summed.

Count

(Null values are not considered in count)

```
Select
count(*)
from
  player
```

→ how many all rows that are counted.

Multiple aggregate

```
Select
Sum(Score),
Avg(Score),
from player;
```

where clause

```
Select
Count(*)
from player
```

(or Count(1) or count(name) are same)

To avoid duplicate
① use distinct name
② count()

```
select
count (distinct name)
```

Alias

As

You can provide an alternate temporary name to the columns

```
SELECT C1 as A1, C2 AS a21 ...
FROM Table-name;
```

↓

changed as

```
Select name AS Player-name
Fromplayer
```

Group By

→ we can group the required rows
→ perform aggregation on each group

It is not practically possible to write a query for every player in the database as there would be numerous players.

Group By

Group rows of each player together

perform aggregation

Syntax

```
Select
name  sum(score)
from player
Group By name;
```

same.

Explanation

David 96
Lokesh 99
Stark 75
David 9
Lokesh 2
Stark 1

→

David 96
David 9
sum

Stark 75
Stark 1

Lokesh 99
Lokesh 87

David 105
Stark 76
Lokesh 186

David	105
Stark	76
Lokesh	186

Multiple rows

```
select name, year, sum(score)
from player
Group By name, year;
```

filter

```
Select name, year, sum(score)
from player
where score >= 50.
Group By name, year;
```

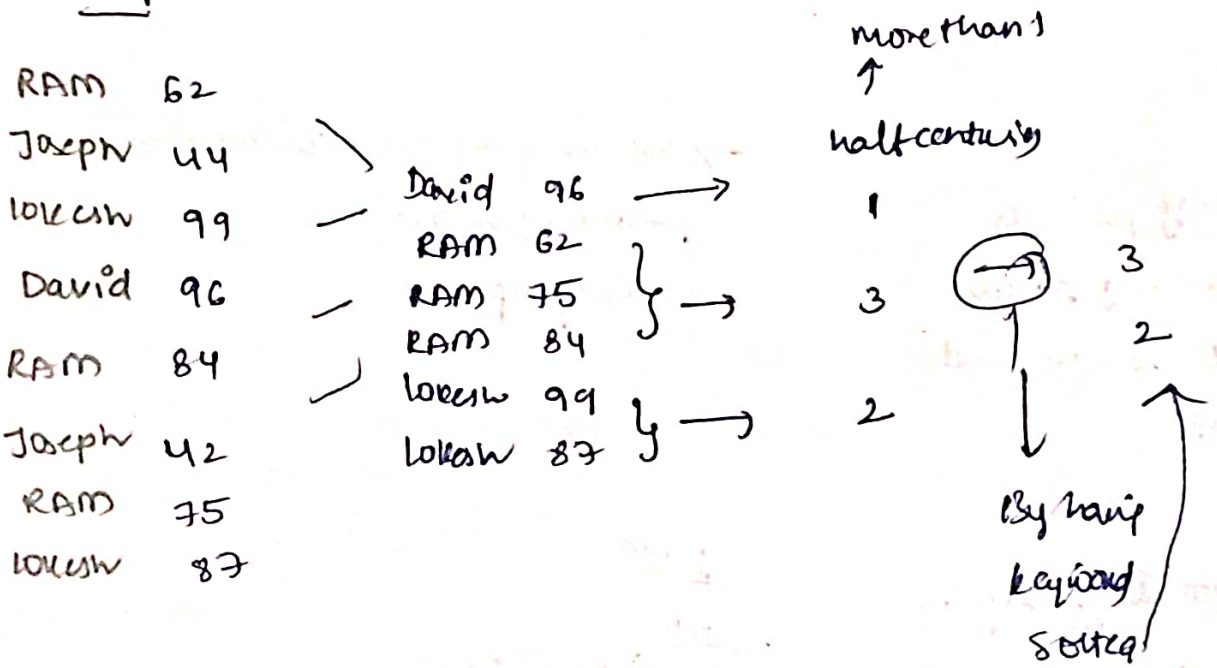
→ score is greater 50 are filtered

Group By having

Get all the players who scored more than 1 half-century

So we need a new clause to filter the groups formed by the Group By clause we will be using having clause for such Operations

Example



syntax

```
select name
count() As half centuries
from player
where score >=50
Group By name
Having half-centuries >1;
```

Where

Where is used to filter rows

Performed before grouping

Having

Having is used to filter groups

Performed after grouping

Expressions in Querying

Syntax

① select *
from Movie
where (collection_in_cr - budget_in_cr) ≥ 500;
└→ Expression

② update movie
set rating = rating/2;

SQL functions

SQL provides date functions which can be used in various clauses to
Exact month year from data field.

Strftime(format, field_name)
↓
Example

Strftime("%.Y", field_name)
└ Exact year.

Example — function

SELECT strftime('%.m', release_date) as month,
count(*) as total_movies,
month

FROM movie
where strftime('%.Y', release_date) = '2010'
Group By strftime('%.m', release_date);

CAST function

Change the datatype

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);

Different formats

%d Day of the month
%H Hour
%m Month
%J Day of the year
%M Minute

Other common functions

CAST - change one datatype to another
UPPER - convert to uppercase
Lower - convert to lowercase
Round - To perform round off value.
ceil → calculate ceil value

Case clause

we use case statements for conditions in query

Syntax

Select C1, C2
case
when condition 1 then V1
when condition 2 then V2
else value
end as Cn
from table

→ Example

Profit	Tax
<100 crores	10%
100 to 500 crore	15%
>500 crores	18%

Select id, name, Collection in cr - budget in cr as profit,
case
when ≤100 ( ) * 0.10
11 ≤500 (") * 0.15
else 11 * 0.18

end as tax_amount
from movie;

we can use case in various clauses like select, where, having, order by and group by

SQL set operators

The SQL set operation is used to combine two or more SQL queries

common set operations

Intersect - Common
Minus - only one
union - Both
union all - doesnot eliminate duplicate results

Intersect

```
Select actor-id
from cast
where movie-id=6
intersect
select actor-id
from cast
where movie-id=15
```

Minus/except

```
sclect actor-id
from CAST
where movie-id=6
except
sclect actor-id
from CAST
where movie-id=15;
```

union

```
sclect actor-id
from CAST
where movie-id=6
union
sclect actor-id
from CAST
where movie-id=15;
```

UnionAll

```
sclect actor-id
from CAST
where movie-id=6
union all
sclect actor-id
from CAST
where movie-id=15;
```

Basic Rule

```
Select
C1, C2 ... Cn
from table1
set operation
Select
d1, d2, d3 ... dn
from t2
```

Modelling database

→ understand business requirements at conceptual level.

→ Translate them to relational database.

To understand business requirements at a conceptual level we use the entity relationship model (ER Model)

ER Model

Core concepts

→ Entity → (objects / concepts) → attributes of entity → property

→ Entity type

→ Relationships

Key attribute

The attribute that uniquely identifies each entity

entity type

Collection of entities and contain same attributes

entity Relationship model

Relationship

Association among entities.

There are three types of relationships

→ one to one

→ one to many (or) many to one

→ many to many.

one-one

An entity is related to only one entity and viceversa.

Ex :- A person has only one passport.

One to many

An entity is related to many other entities.

Eg

John has many cars

opposite it many to one

Many-Many

multiple entities are related to multiple entities

Cardinality ratio

Here n:m it called cardinality ratio

- one-one - 1:1
- one-many - 1:m
- many to one - m:1
- many to many - m:n

e-commerce application

1) customer has only one cart
2) A cart belong to one customer
3) customer can add products to cart
4) cart contain multiple products
5) customer can save multiple address

entity types

Customer
product
Cart
Address

Attributes

Customer
id
name
age

} attributes

① Customer → has → Cart

Cart → belong → Customer

Customer → add → P1 P2 P3

Cart → contain → P1 P2 P3

product
product_id
price
name
brand
category

address
id
pin_code
door_no
city

Cart
Cart_id
total_price

Customer (1) has (1) Cart

Customer (1) has (n) address

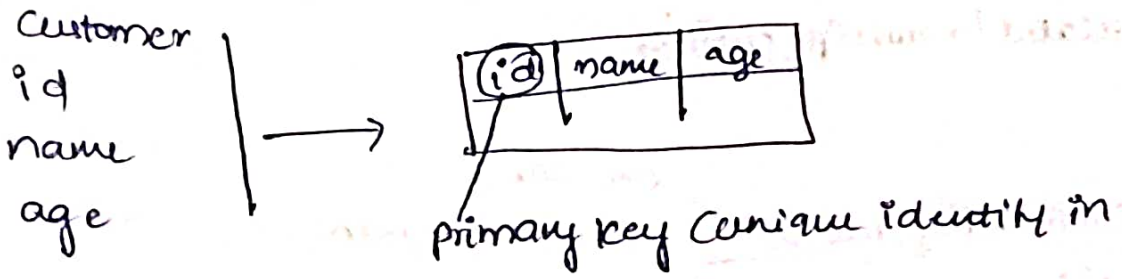
Cart (m) contain (n) product

ER Model to relational database

entity types - tables

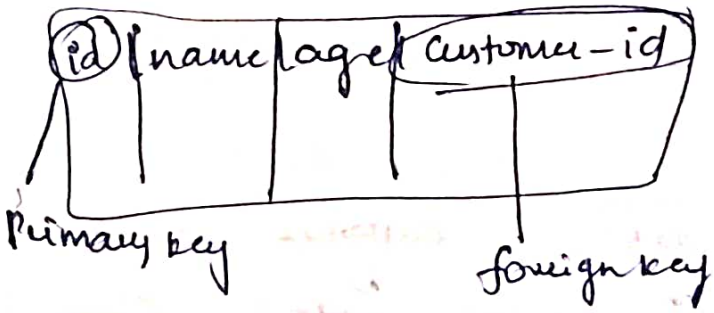
Attributes - columns

Key attribute - primary key.



foreign key

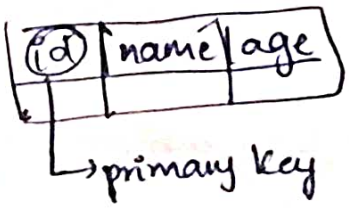
The new column that refer to primary key



we store the properties related to many-many in junction table

Modelling database part-2

customer



primary key syntax

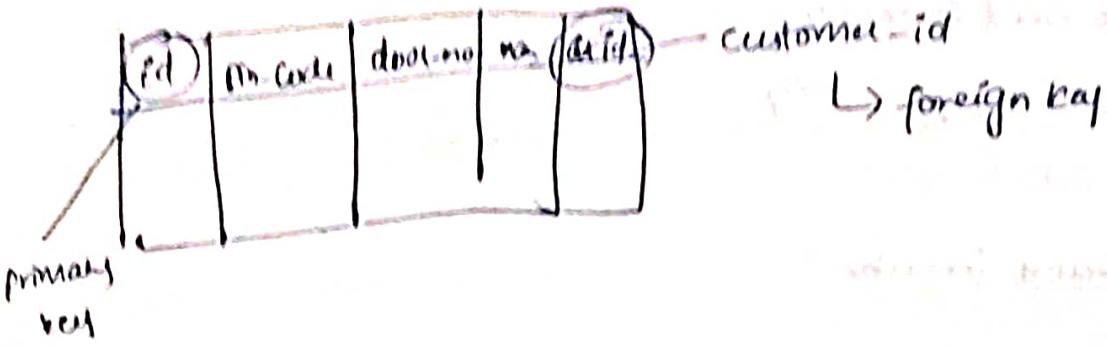
```
Create table tablename (
  clm1 Not Null primary key
  ...
);
```

→ ex →

```
Create table customer
id integer not null primary
name varchar()
age integer
);
```

Address table

In the address table we need to create a foreign key to customer table



Syntax

```
CREATE TABLE table2(
c1 t1 Notnull primary key,
c2 t2,
-foreignkey (c2) references table1(c3) on delete cascade
);
```

→ Allow valid values

→ if row in table1 is deleted then all its related rows in table2 will also deleted

Enabling foreign key

```
pragma foreign_key = ON;
```

Creating table using foreign key

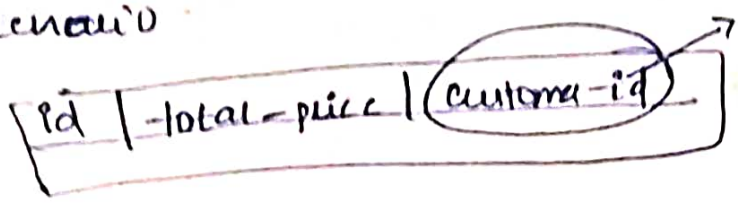
```
Create table address(
id Integer Not Null primary key,
pin-code integer,
door-no Varchar(50),
city (Varchar),
Customer-id integer,
foreign key(Customer-id) References customer(id) on
    Delete Cascade
);
```

Creating table

Cart table

In cart table we need to make customer-id column accept only unique values

we use unique constraint in such scenario



Ex

```
Create table cart(
 id integer Not Null primary key,
 total-price integer,
 Customer-id integer Not null unique,
 foreign key customer-id) references customer(id) on
 delete cascade))
```

Inserting data

Customer-table

```
Insert into
  customer (id, name, age)
values
  (1, 'John', 29);
  (2, 'Emma', 24);
```

Address table

```
insert into address
(id, pincode, doorno, city)
customer-id)
values
(1001, 517130, '6-1', 'hyderabad',
(1002, 615670, 6-3, 'Bang', 42)
```

Querying

One to many relationships (we need all addresses of customers)

```
Select *
-from Customer
JOIN address on customer.id = address.customer_id
where
 Customername = 'John';
```

One-one relationship

```
Select *
from customer
Join cart ON Customer.id = Cart.customer_id
where
Customer.name = 'John';
```

Many-Many Relationship

```
select *
from cart
Join cart-product ON Cart.id = Cart_product.Cart_id
Join product ON product.id = Cartproduct.product_id
where
Cart.customer_id = 1;
```

Joins

Join operation combines existing tables into single temporary table

Based on how the table are combined there are 6 types of joins

- Natural join
- Left join
- Right join
- Cross join
- Inner join
- full join

NATURAL JOIN

Natural join combine the tables based on common columns

id	name	duration	instructor-id
11	ML	96	102
15	Cyber	60	101
19	Linux	30	NULL

Table 1

instructor-id	name	Gender
101	Alen	M
102	Arun	M
105	Dexter	M

Table 2

common

11	ML	Arun	M	90	102
15	Cyber	Alen	M	60	101

Syntax

```
select *
FROM table 1
NATURAL JOIN table 2;
```

Natural join

Query

fetch the course that are being by Alen

↓

```
Select course-name,
instructor-full-name
from course
Natural Join instructor
```

Inner join

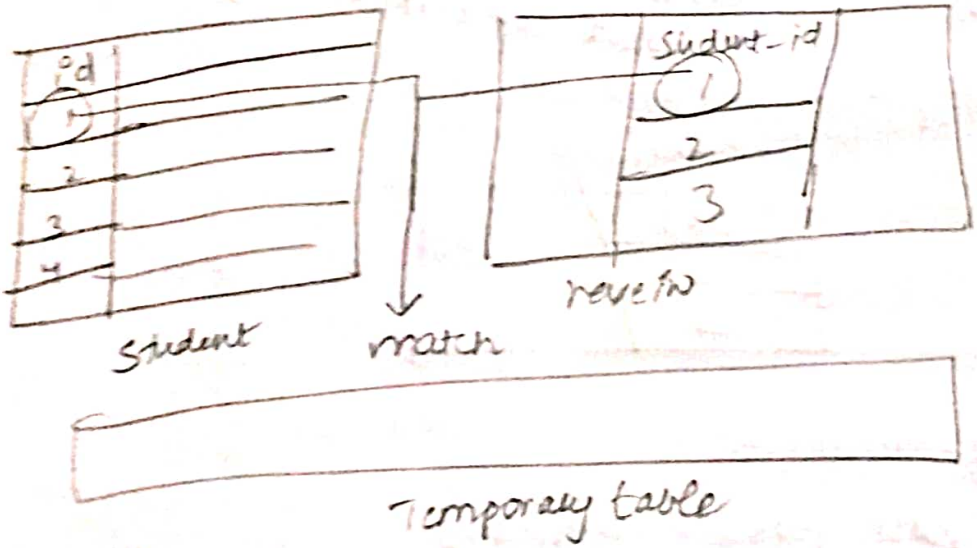
Inner join combines rows from both the tables if they meet a specific condition.

Syntax

```
select *
from table 1
inner join table 2
on table1.c1 = table2.c2;
   └ condition
```

→

```
Select *
from student
inner join review
on student.id = review.student_id;
```



Inner join Query

```
Select *
from student
INNER JOIN review on student_id = review.student_id
where
review.course_id = 15; → only cyber security course.
```

Left join

for common in the left table matched rows from the right table are considered

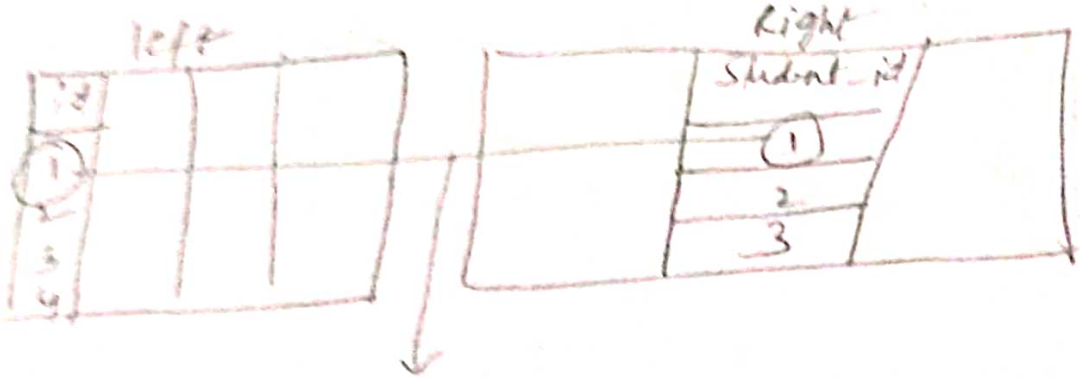
no match null values are returned to right hand of rows in temporary table.

Syntax

```
Select *
from table1
left join table 2
on table.c1 = table2.c2
```

→ Example →

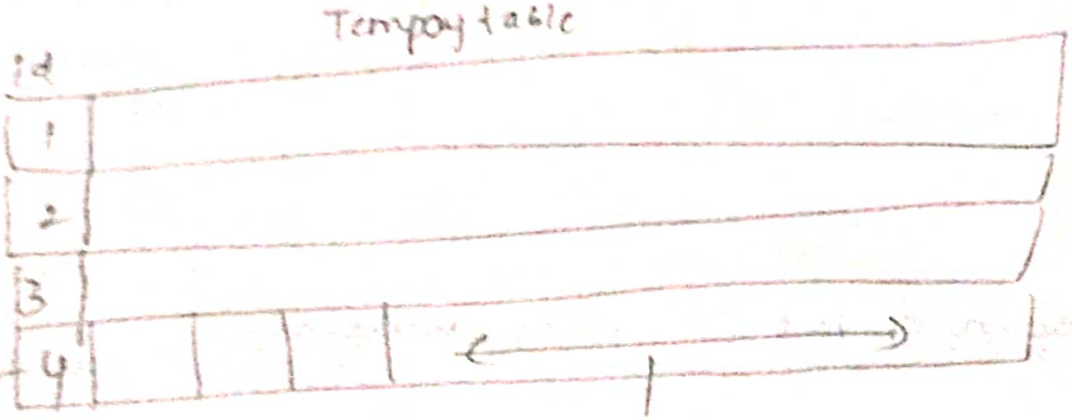
```
Select *
from student
left join student_course
on student.id =
student_course.student_id;
```



match

↓

Temporary table



Null

(Right side values are null)

Query

```
where Student_id= NULL
```

→

Right join

Syntax

```
select *
from table1
Right join table2
ON table.c1 = table2.c2;
```

Exact opposite of left join

full join

full join (or full outer join is the result of both right join and left join

Syntax

```
select *
from table2
Full join table2
 on table.c1 = table2.c2;
```

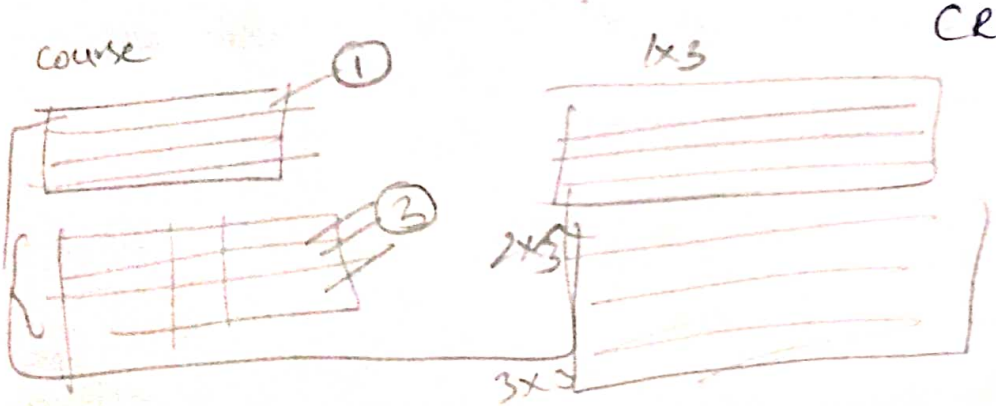
Cross join (Cartesian join)

In cross join, each row from first table is combined with all rows in second table.

```
Select *
from table1 Cross join table2;
```

→ example →

```
select course_name,
instructor.full_name
from course
Cross join instructor;
```



Self join

combine itself

Syntax

```
Select t1.c1,
t2.c2
from table As t1
Join table As t2
on t1.c1=t2.cn;
```

Views and subqueries

View

A view can simply be considered as a name to a sql query.

~~user_base_details~~

Syntax

```
CREATE VIEW user-base-details AS
Select id, name, age, gender, pincode from user;
```

Querying

```
Select *
from user_base_details
```

→ view can be used as table.

Using other clause in view

```
Select *
from user-base-details
where gender = "Male"
ORDER By age ASC;
```

using Views in Querying

```
Create View order_with_products AS
 Select *
from order_product
 INNER Join product
ON order_product.product_id = product.product_id;
```

List All Available Views

```
 Select name
from sqlite_master
where type='view';
```

Delete View

```
Drop view view_name;
```

Subquery

```
Select                          ——→ Outer query
name,
( select AvgRating
   from product                 }  Inner query
    where category = "watch"
)- rating AS rating_variance
fom product
 where Category = 'watch';
```

Clauses can be used

where, order by

Transaction and Indexes

A logical group of one or more SQL statements

A transaction has four important properties.

→ Atomicity

all SQL statement will occur or not occur.

→ Consistency

→ Isolation

Not to affect other transaction

→ durability

Changes of a successful transaction persist even after a system.

Indexing

Similarly in database we maintain indexes to speed up the search of data in table.