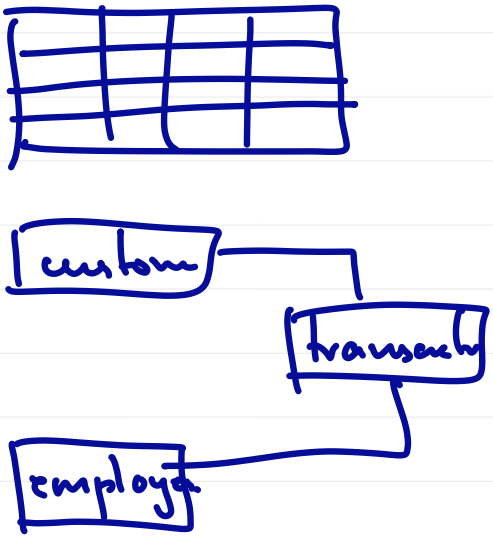
MySQL

Relational DB

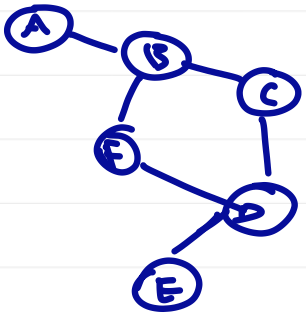
SQL



Non-Relational DB

NoSQL

{
—:—
—:—
}



DBMS → Database Management System.

↳ Workspace to write SQL Statements & work with our database.

MySQL Oracle PostgreSQL ----

Installation:

↳ My SQL.com → community downloads e.t.c.

↓

mini installer

server, workbench.↵

MySQL → Port No :- 3307

root - Password :- CHaran@123

Databases:

↳ Any collection of related information

Queries

↳ Requests made to DBMS to perform a particular tasks.

Database - Structure:

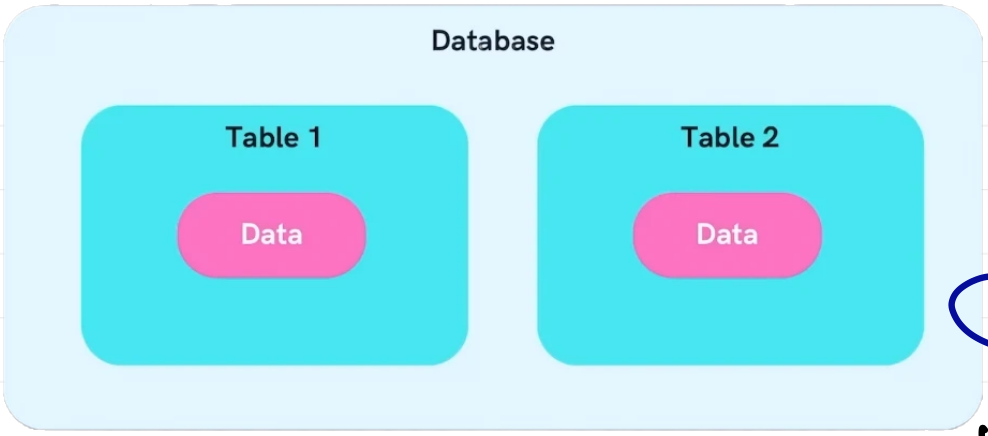


Table:

attributes.

RollNo	Name	Class	DOB	Gender	City	Marks
1	Nanda	X	1995-06-06	M	Agra	551
2	Saurabh	XII	1993-05-07	M	Mumbai	462
3	Sonal	XI	1994-05-06	F	Delhi	400
4	Trisla	XII	1995-08-08	F	Mumbai	450
5	Store	XII	1995-10-08	M	Delhi	369
6	Marisla	XI	1994-12-12	F	Dubai	250
7	Neha	X	1995-12-08	F	Moscow	377
8	Nishant	X	1995-06-12	M	Moscow	489

→ row1, row2 } tuples.

col1 col2 .

Columns → Structure

Rows → Data.

SQL- Commands:

1) Creating & dropping DB:

```
CREATE DATABASE db_name;

DROP DATABASE db_name;
```

2) Selecting database

↳ USE db_name;

3) Create table: → For creating col's.

```
CREATE TABLE table_name (
    column_name1 datatype constraint,
    column_name2 datatype constraint,
    column_name2 datatype constraint
);
```

varchar(50) → no. of chars

```
CREATE TABLE student (
    id INT PRIMARY KEY,
    name VARCHAR(50),
    age INT NOT NULL
);
```

↓ Example.

→ create table student(id int, name varchar(50));

4) Insert data into Tables:-

omitted when column order & no. of col's = tuple values len & order.

-> insert into student (id,name) values (10,"Harith"), (20,"Vani");

for order

id	name
10	Harith
20	Vani.

(:id,name) (10,harith)

2 2

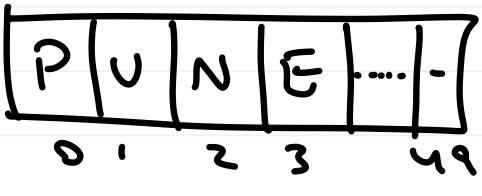
(so no need of this)

-> Datatypes in SQL:

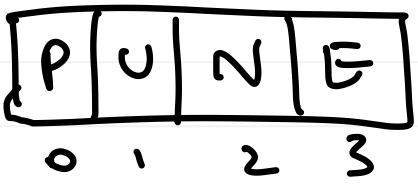
DATATYPE	DESCRIPTION	USAGE
CHAR	string(0-255), can store characters of fixed length	CHAR(50)
VARCHAR	string(0-255), can store characters up to given length	VARCHAR(50)
BLOB	string(0-65535), can store binary large object	BLOB(1000)
INT	integer(-2,147,483,648 to 2,147,483,647)	INT
TINYINT	integer(-128 to 127)	TINYINT
BIGINT	integer(-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807)	BIGINT
BIT	can store x-bit values. x can range from 1 to 64	BIT(2)
FLOAT	Decimal number - with precision to 23 digits	FLOAT
DOUBLE	Decimal number - with 24 to 53 digits	DOUBLE
BOOLEAN	Boolean values 0 or 1	BOOLEAN
DATE	date in format of YYYY-MM-DD ranging from 1000-01-01 to 9999-12-31	DATE
YEAR	year in 4 digits format ranging from 1901 to 2155	YEAR

char(50) vs varchar(50)

"PUNE"



fixed length.



stores upto given len.

BIT(1) → 0 or 1

BIT(2) → 00/01/10/11 } can store only these

→ SQL Datatypes:-

Numeric

↓

INT
FLOAT
DOUBLE
TINYINT } +ve or -ve

(But in some cases like age, salary → +ve)

→ TINYINT UNSIGNED (0 to 255)

→ TINYINT (-128 to 127) (Here mem reserved for -ve numbers is given to +ve)

Types of SQL Commands

DDL (Data Definition Language) : create, alter, rename, truncate & drop

DQL (Data Query Language) : select

DML (Data Manipulation Language) : , insert, update & delete

DCL (Data Control Language) : grant & revoke permission to users

TCL (Transaction Control Language) : start transaction, commit, rollback e

☆ create database if not exists student;

drop database if exists student;

conditional queries → to reduce errors.

→ SELECT Command; → or write required col's or with conditions

→ select * from students;

(all) ↳ print's table.

```
CREATE TABLE employee(
  id INT PRIMARY KEY,
  name VARCHAR(100),
  salary INT
);

INSERT INTO employee
(id, name, salary)
VALUES
(1, "adam", 25000),
(2, "bob", 30000),
(3, "casey", 40000);
```

→ Try using this format for SQL Queries.

→ Keys:-

1) Primary key → (unique) → NOT NULL

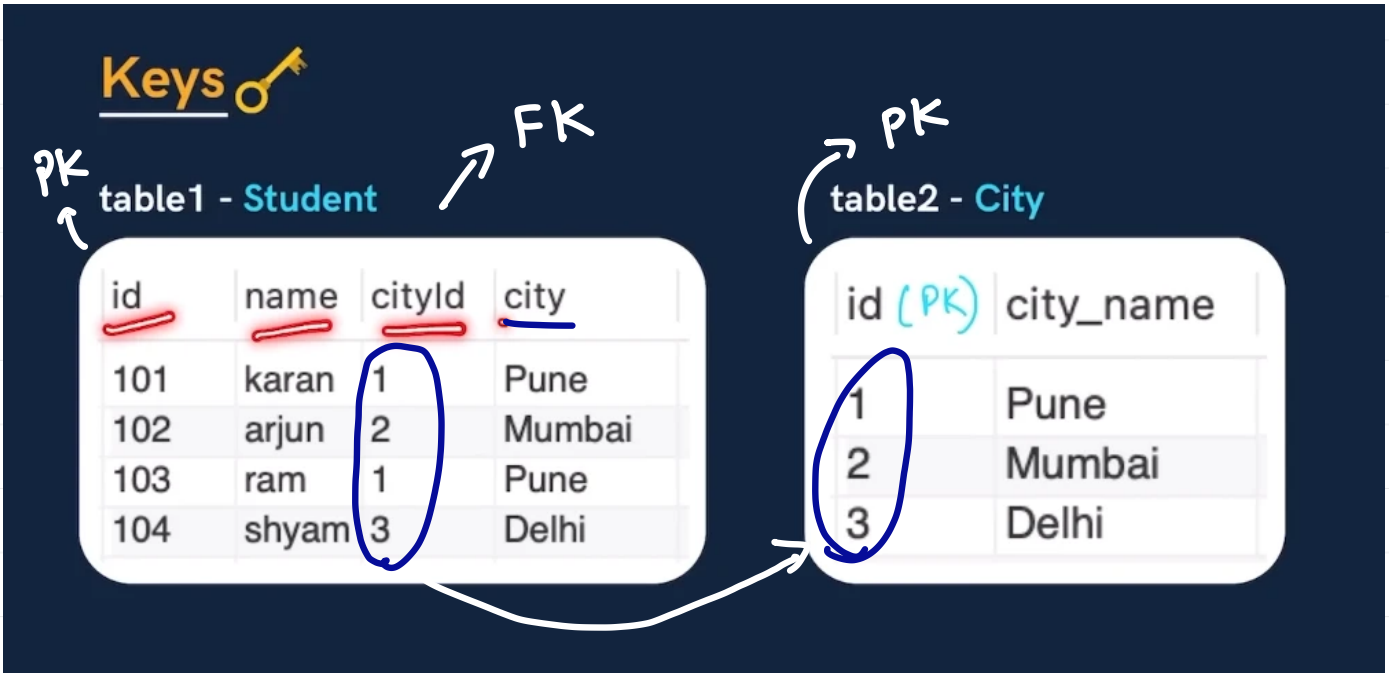
☆ (A table has only 1 PK)

↳ Used for uniquely identify each row.

2) Foreign key:

↳ A column (or set of columns) - in a table that refers to the PK

→ There can be multiple FK's. (duplicate & NULL can be)



→ Constraints:. (at time of creation)

↳ These specify values for data in a table.

1) NOT NULL

↳ col's cannot be a null value.

```
col1 int NOT NULL
```

2) UNIQUE

↳ all values in column are different.

```
col1 int UNIQUE
```

3) Primary key

↳ makes a col unique & not null but used only once.

-> id int primary key

(or)

```
-> create table students (
        id int unique not null,
        primary key (id)
   );
```

☆ There can be a combination used as a PK.

```
-> create table student (
        id int,
        name varchar(50),
        primary key (id, name)
   );
```

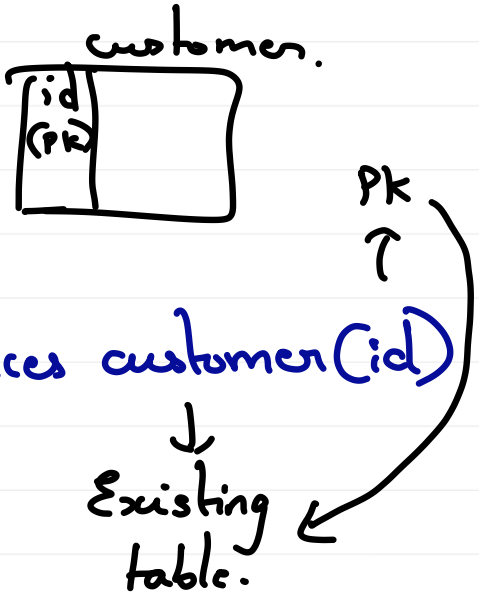
combination -> PK.

id -> can have duplicate
name -> can have duplicate

(id, name) -> can't have duplicate

4) Foreign key:-

↳ prevent actions that would destroy links between tables.




```
-> CREATE TABLE temp (
      cust_id int,
      Foreign key (cust_id) references customer(id)
   );
```

cust_id → col -> FK.

customer(id) → Existing table.

5) DEFAULT

↳ sets the default value of a column.

-> salary int default 25000

↓

def-value.

6) CHECK

↳ It can limit values allowed in a column.

```
-> create table city (
      id int primary key,
      city varchar(50),
      age int,
      CONSTRAINT age_check CHECK (age >= 18
                  AND city = "Delhi")
   );
```

(or)

```
-> create table newtab(
        age int check(age>=18)
   );
```

Create this sample table

```
CREATE DATABASE college;
USE college;

CREATE TABLE student (
   rollno INT PRIMARY KEY,
   name VARCHAR(50),
   marks INT NOT NULL,
   grade VARCHAR(1),
   city VARCHAR(20)
);
```

Insert this data

```
INSERT INTO student
(rollno, name, marks, grade, city)
VALUES
(101, "anil", 78, "C", "Pune"),
(102, "bhumika", 93, "A", "Mumbai"),
(103, "chetan", 85, "B", "Mumbai"),
(104, "dhruv", 96, "A", "Delhi"),
(105, "emanuel", 12, "F", "Delhi"),
(106, "farah", 82, "B", "Delhi");
```

rollno	name	marks	grade	city
101	anil	78	C	Pune
102	bhumika	93	A	Mumbai
103	chetan	85	B	Mumbai
104	dhruv	96	A	Delhi
105	emanuel	12	F	Delhi
106	farah	82	B	Delhi

-> SELECT in detail:

-> (DISTINCT)

-> select distinct city from student;

-> Where clause → condition.

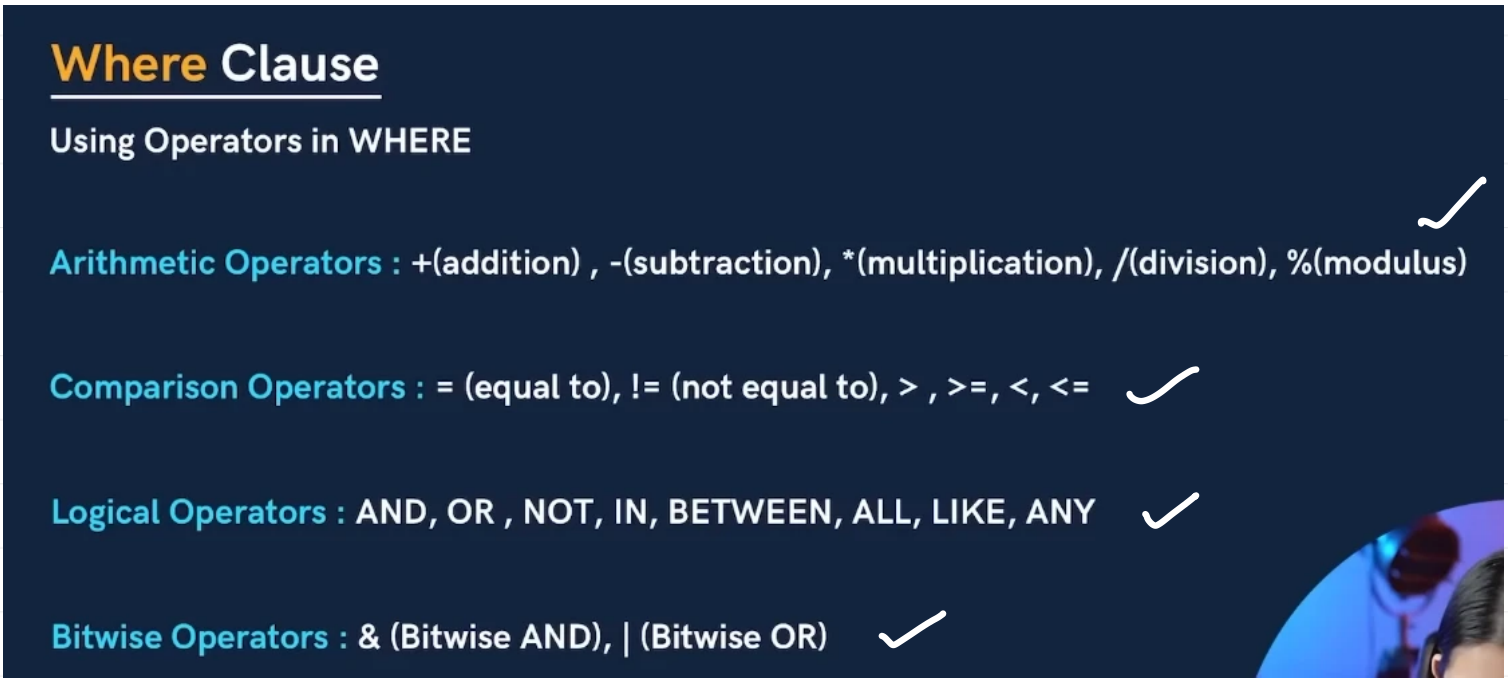
↳ to define some conditions.

-> select col1, col2 from table_name } syntax.
where conditions;

-> select * from student where city="mumbai"

-> select * from student where marks > 80;

↳ ☆ can also use AND, OR



Arithmetic:

+ -> --- marks + 10 >= 100;

Logical Operators:

BETWEEN ⇒ ... marks between 80 AND 90;

inclusive.

IN → ... city in ("Delhi", "Mumbai");

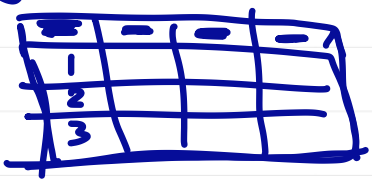
NOT → ... city not in ("Delhi", Mumbai");

offset?

→ **Limit clause**

↳ sets an upper limit of no. of rows to be shown or returned.

→ select * from student limit 3;



↳ only 3 students.

→ **Order by clause** ↗ default

↳ to sort in ascending (ASC) ↑ or descending order (DESC) ↓

→ select * from student order by city ASC;

Aggregate functions:

↳ Performs a calculation on a set of values, & return a single value.

1) COUNT() → count of no. of values.

2) MAX() → max in a set of values.
(for str's → Letter order max)

3) MIN() → min " " "

4) SUM() → Add all values in a column.

5) AVG() → Average " " " "

. . . .

Generally used with select.

Group By Clause:

↳ It Groups rows that have same values into summary rows.

→ It collects data from multiple records & groups the result by one or more column

Q) Count number of students in each city?

→ select city, count(name) from student group by city;

We have to select multiple col's if we group by multiple col's.

Practice Qs

Write the Query to find avg marks in each city in ascending order.

→ select city, avg(marks) from student
group by city order by city Asc;

Practice Qs

For the given table, find the total payment according to each payment method.

customer_id	customer	mode	city
101	Olivia Barrett	Netbanking	Portland
102	Ethan Sinclair	Credit Card	Miami
103	Maya Hernandez	Credit Card	Seattle
104	Liam Donovan	Netbanking	Denver
105	Sophia Nguyen	Credit Card	New Orleans
106	Caleb Foster	Debit Card	Minneapolis
107	Ava Patel	Debit Card	Phoenix
108	Lucas Carter	Netbanking	Boston
109	Isabella Martinez	Netbanking	Nashville
110	Jackson Brooks	Credit Card	Boston

→ payment table

→ select mode, count(customer)
from payment
group by mode;

Having Clause:

↳ ~ where, but applies conditions to groups.

Q) Count number of students in each city where max marks cross 90.

→ select city, count (students)
from student
group by city
Having marks > 90;

applied on group.

☆ General - order:-

General Order

```
SELECT column(s)
FROM table_name
WHERE condition
GROUP BY column(s)
HAVING condition
ORDER BY column(s) ASC;
```

→ on rows

→ on col's / groups

→ Table - Related Queries:-

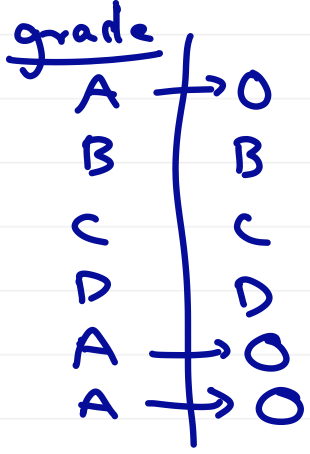
1) Update
↳ to update existing rows

→ update table_name
set col1 = val2, col2 = val2
where condition;

} Syntax

→ update student
set grade = "O"
where grade = "A";

→



!!! when doing update it says error code: 1175: Safe mode.

to avoid this, Run ↗ off.

→ set SQL_SAFE_UPDATES = 0;

Q) Increase marks of every student by 1?

→ update student
set marks = marks + 1;

2) DELETE:

↳ to delete existing rows

Delete from table_name
where _condition;

} Syntax.

→ if we don't write this entire data in tb can be erased.

→ delete from student
where marks < 33;

Revisiting Foreign key

dept ← teacher (FK)

(PK)

id	name
101	Science
102	English
103	Hindi

↓ Parent table.

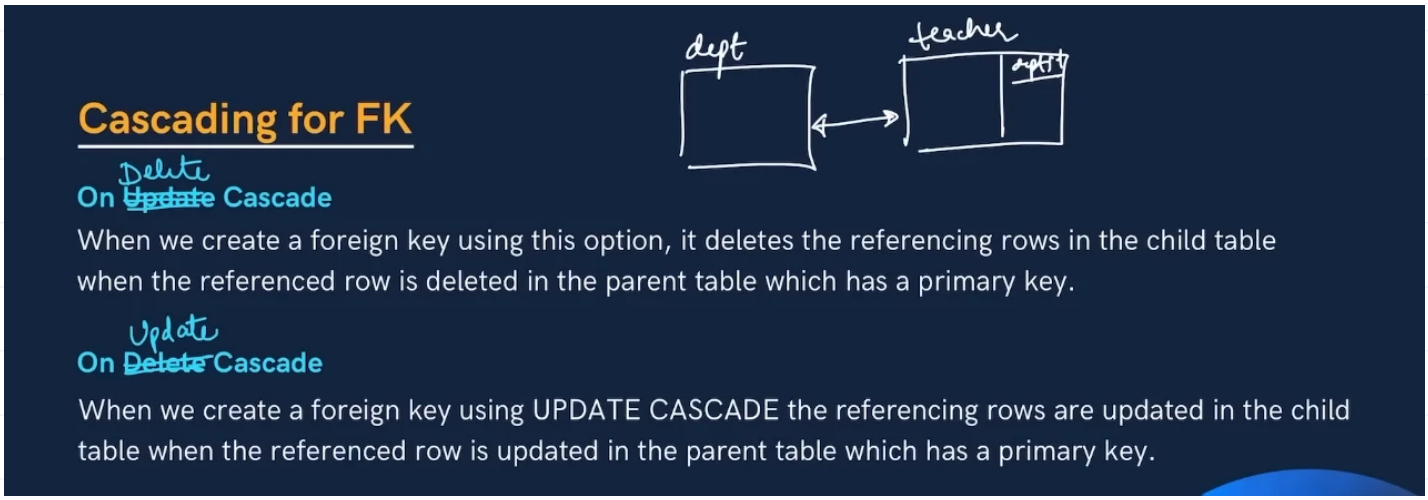
FK

id	name	dept_id
101	Adam	101
102	Bob	103
103	Casey	102
104	Donald	102.

↓ child table.

```
Foreign key (dept_id)
REFERENCES dept (id)
```

Cascading for FK (Define at time of tb creation)




```sql
CREATE TABLE teacher (
  id INT PRIMARY KEY,
  name VARCHAR(50),
  dept_id INT,
  FOREIGN KEY (dept_id) REFERENCES dept(id)
  ON UPDATE CASCADE
  ON DELETE CASCADE
);
```

If a change occurs in dept then it automatically cascades into teacher.

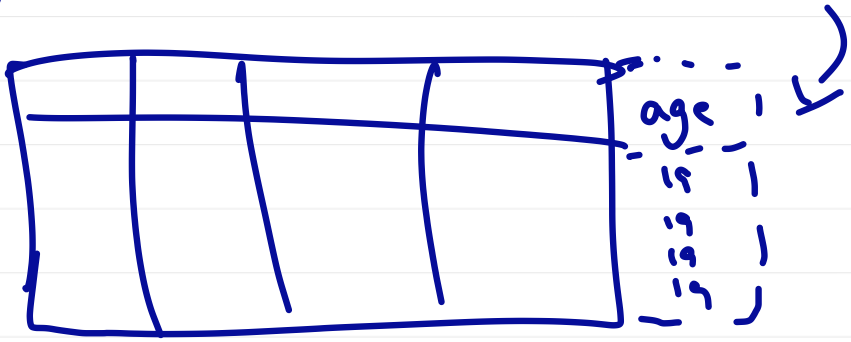
Table Related Queries

Alter: ↳ to change the schema of existing tb.
↓
design (columns, datatypes, constraints)

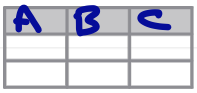
1) Add Column (adds a column to existing tb)

```
alter table table_name
add column column_name datatype constraint;
```

→
```
alter table student
  add column age int not null default 19;
```



→
```
alter table student
  add column D int after B;
```



specifies direction.

2) Drop Column (deletes a col in existing table)

```
alter table table_name
drop column column_name;
```

3) **Rename table** : (to rename table name)

```
alter table table_name
rename to new_table_name;
```

4) **Change column** : (redefining a column)

```
alter table table_name
change column old_name new_name
             new_datatype new_constraint;
```

☆ You need to specify new_datatype & new_constraint even though you want to rename the column.

5) **Modify Column** : (modifies a column)

```
Alter table table_name
modify col_name new_datatype new_constraint;
```

Truncate : (delete's table data)

```
truncate table table_name;
```

Joins

→ Joins are used to combine rows from two or more tables, based on a related column between them.

employee

id	name
101	
102	

salary

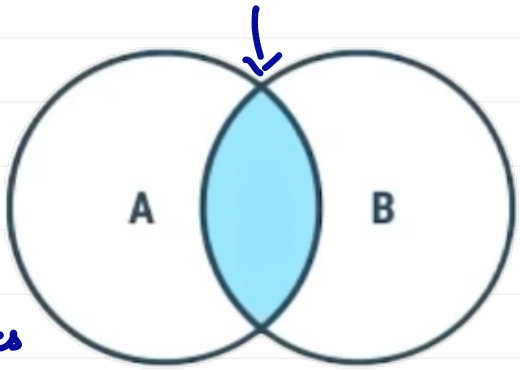
id	salary
102	
103	

same col. → so we can relate name & salary.

Types of Joins :

1) Inner Join

Returns records that have matching values in both tables




```
SELECT *
FROM student
INNER JOIN course
ON student.student_id = course.student_id;
```

id id

Inner Join

→ A ∩ B

Example

A ↗ B ↗

student

id	name
101	adam
102	bob
103	casey

course

id	course
102	english
105	math
103	science
107	computer science

↓ 102, 103.

↓ ↙

id	name	id	course
102	bob	102	english
103	casey	103	science

↓

Now if you only want id, name, course

write student.id, name, course.course.

↓ written cuz both tables have id's.

↓ We reference this cuz we don't select from course.

☆ **Alias**: (alternate names)

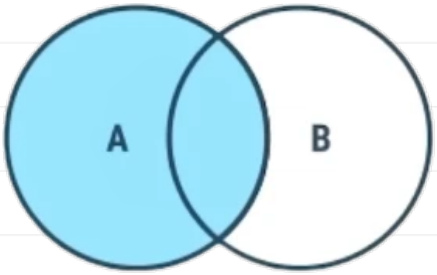
```
select *
from student as s
inner join course as c
on s.id = c.id;
```

s, c ← aliases.

Outer Joins

2) Left Join

Returns all records from left table & the matched records from the right table.



Left Join

Example

student id

~~student_id~~	name
101	adam
102	bob
103	casey

course id

~~student_id~~	course
102	english
105	math
103	science
107	computer science

Table A

Table B

```
SELECT *
FROM student as s
LEFT JOIN course as c
ON s.student_id = c.student_id;
```

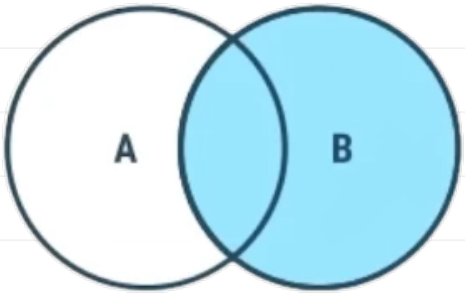
id (s.student_id) id (c.student_id)

id	name	id	course
101	adam	null	null
102	bob	102	english
103	casey	103	science

→ null is written cuz this row doesn't have a match.

3) Right Join :

Returns all records from the right table & matched records from the left table.



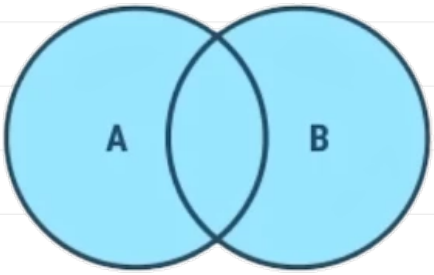
id	name	id	course
102	bob	102	english
103	casey	103	science
NULL	NULL	105	math
NULL	NULL	107	computer science

A B.

Table A → Table B.

```
SELECT *
FROM student as a
RIGHT JOIN course as b
ON a.id = b.id;
```

4) Full Join :



Returns all records when there is a match in either L or R table.

☆ There is no full join syntax in MySQL except in ORACLE e.t.c.

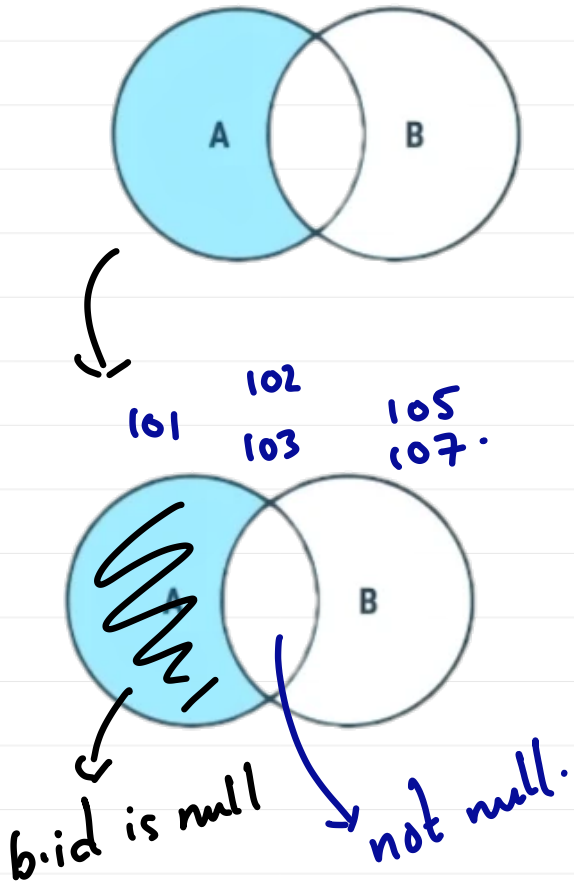
Full Join → Left join
UNION
Right join.

id	name	id	course
101	adam	NULL	NULL
102	bob	102	english
103	casey	103	science
NULL	NULL	105	math
NULL	NULL	107	computer science

```
select *
from student as s
left join course as c
on s.id=c.id
union
select *
from student as s
right join course as c
on s.id=c.id;
```

5) Exclusive joins

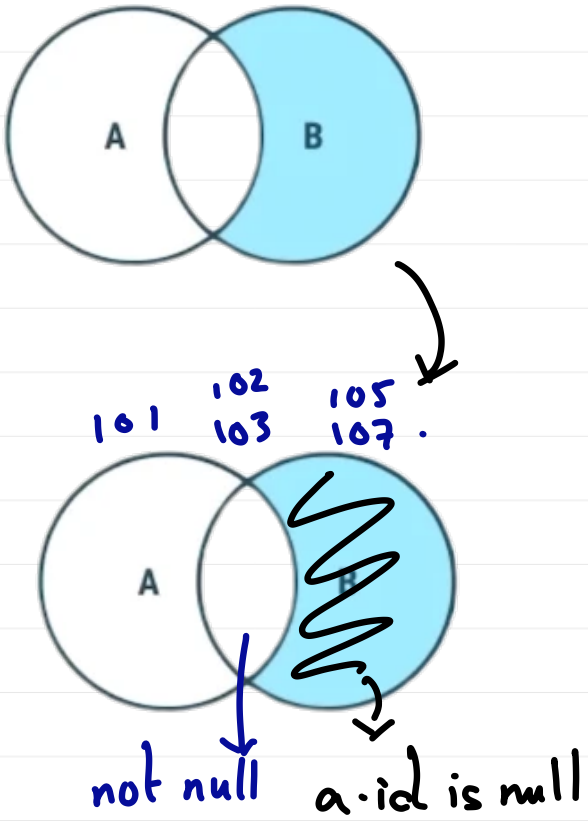
Left Exclusive Join




```
select *
from student as s
left join course as c
on s.id=c.id
where c.id is null;
```

id	name	id	course
101	adam	NULL	NULL

Right Exclusive Join.

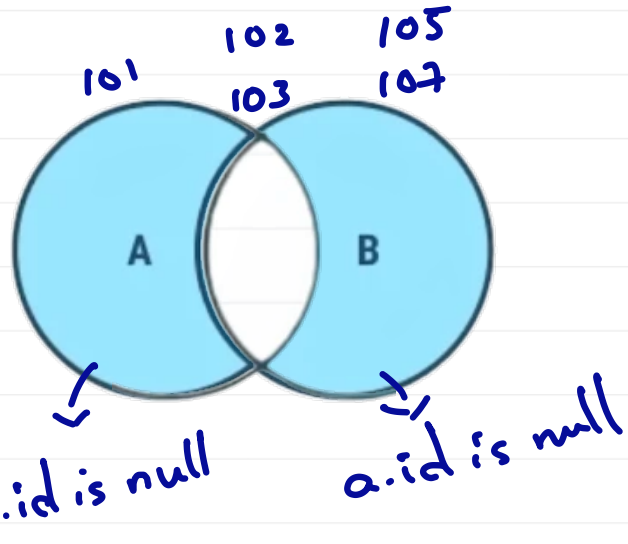



```
select *
from student as s
right join course as c
on s.id=c.id
where s.id is null;
```

id	name	id	course
NULL	NULL	105	math
NULL	NULL	107	computer science

Exclusive join:

↳ left exclusive join
union
right exclusive join.




```sql
-- exclusive join
select *
from student as s
left join course as c
on s.id=c.id
where c.id is null
union
select *
from student as s
right join course as c
on s.id=c.id
where s.id is null;
```

id	name	id	course
101	adam	NULL	NULL
NULL	NULL	105	math
NULL	NULL	107	computer science

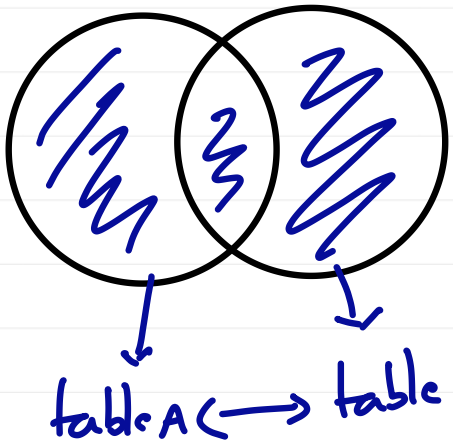
6) Self Join:

→ It is a regular join but the table is joined with itself.

(executes right to left)

Union : (used mainly for full join)

It is used to combine the result-set of 2 or more select stmt's. It gives unique records.



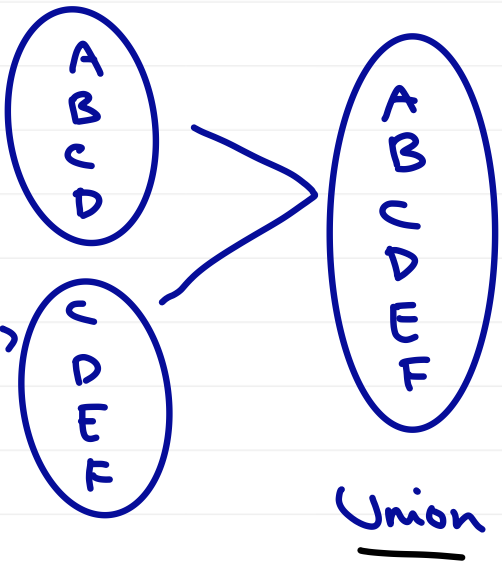
To use it:

1) Every select should have same no. of columns
2) columns must have similar datatypes.
3) columns in every select should be in same order.

Syntax:

```
Select col(s) from table A
union
select col(s) from table B
```

Ex:

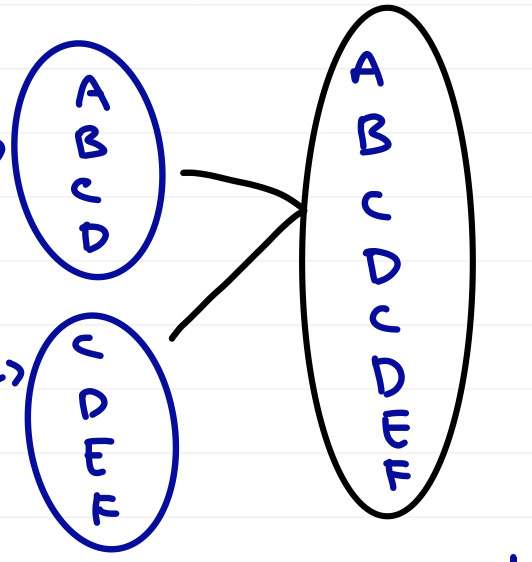


Union all :

→ This allows duplicates.

Syntax:

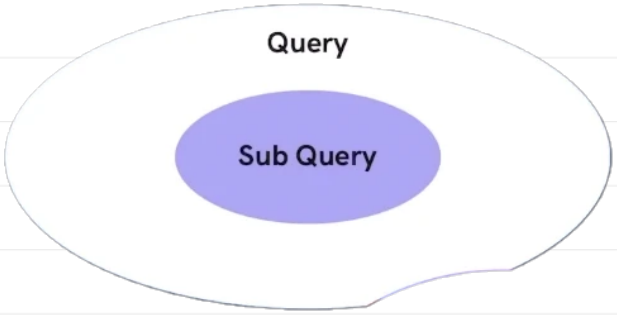
```
Select col(s) from table A
union all
elect col(s) from table B
```



Union all

Sub Queries

→ A Subquery is a query within another SQL Query.



3 Ways to write Sub-queries

select | from | where **

Where :

```
select column(s)
from table_name
where col_name operator
(subquery);
```

} Syntax

Example - 1:

student

rollno	name	marks
101	anil	78
102	bhumika	93
103	chetan	85
104	dhruv	96
105	emanuel	92
106	farah	82

Example

Get names of all students who scored more than class average.

Step 1. Find the avg of class
Step 2. Find the names of students with marks > avg

```
-> select name, marks
   from student
   where marks > (select avg(marks)
                  from students);
```

for taking care of dynamic changes.

Example

Find the names of all students with even roll numbers.

Step 1. Find the even roll numbers
Step 2. Find the names of students with even roll no

```
-> select name, roll no
   from student
   where roll no in (
         select roll no
         from student
         where roll no % 2 = 0
   );
```

☆ When subquerying or selecting from multiple values use **in**

From

Example with FROM

Find the max marks from the students of Delhi

Step 1. Find the students of Delhi
Step 2. Find their max marks using the sublist in step 1
& details

student

rollno	name	marks	city
101	anil	78	Pune
102	bhumika	93	Mumbai
103	chetan	85	Mumbai
104	dhruv	96	Delhi
105	emanuel	92	Delhi
106	farah	82	Delhi

☆ Use Subquery in from when we need to select from subset of the table.

```
-> select max(marks)
   FROM (
        select *
        from student
        where city = "Delhi"
   ) as temp;
```

↑

Note: whenever we use a subquery in from we use should an alias.

(or)

```
-> select max(marks)
   from student
   where city = "Delhi";
```

Select (not that imp)

this subquery should only return 1 row

```
select (select max(marks) from student), name
from student;
```

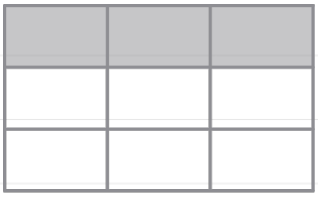
Views

→ a view is a virtual table based on the result-set of an SQL statement.

→ mainly used to avoid having multiple queries on original data or table.

→ create view view1 as
select rollno, name, marks from student;

view1



You can use queries on this view.

→ drop view view1; (delete's a view)

Auto_Increment : (to get incrementing default values for a column)

→ create table student (
id int primary key AUTO_INCREMENT = 1000 (starting value)
name varchar(50)
);

default = 1

→ Used only on PK. (in some case → UK)

→ Only one col per table can have auto_increment.

☆ (To change auto-increment step size

↳ SET @@auto_increment_increment = 5;

default = 1

Wild cards :

→ used to search wild card characters/patterns.

& substitute them.

~ regex matching.

1) Search all rows with first name starting with "s".

```
SELECT * FROM employees
WHERE first_name LIKE "s%";
```

(%) → matches any no of random chars.

2) " " " " " ending with "r".

```
SELECT * FROM employees
WHERE last_name LIKE "%r";
```

"sp%" → spare
sponge

"_ook" → cook
book
look.
↓
a random
char

"_a%" → janitor
manager

"__ook" → spook
brook

"____-__-03" → all dates with day=03
yyyy mm

Indexes

(B-Tree DS)

→ indexes are used to find values within a specific column more quickly.

MySQL

100 101 102 103 104 105 . . .
* * * * . . . .

→ Searches Sequentially
↓
Not effective ✗

Pro's

Select takes less time
O(log n)

Con's

Update takes more time.
(CUD)
writes.

1) Show - indexes:

→ show indexes from customers;

☆ (Even if we don't create any index it will show 1 index → for primary key)

→ So if you have a PK in a table.
↳ index.

2) Create single - column Index:

→ create index last_name_index ↗ index name.
on customers (last_name);
↳ column to have the indexing.

3) Multi Column - Index

→ create index last_name_first_name_idx
on customers (last_name, first_name);

Order of columns is important. (L → R)

→ this index can be used in queries as.

.. where last_name = "Sharma" ✓

.. where last_name = "Sharma" AND first_name = "Ann" ✓

.. where first_name = "Amit" ✗

(cuz, if we don't search the first specified column we can't get into the filtered or sorted block)

* You can use leftmost one or more, but not skip ahead. (Not good with OR)

4) Drop index:

-> Drop index idx_name on students;

(or)

-> Alter table students
drop index idx_name;

Rounding-off values:

1) CEIL(number)

↳ rounds up to nearest whole number.

CEIL(4.3) → 5

CEIL(-4.3) → -4 (In -ve smaller num is greater)

2) FLOOR(number)

↳ rounds down to nearest whole num

FLOOR(4.7) → 4

FLOOR(-4.7) → -5

3) TRUNCATE(number, decimal_places)

↳ Removes digits after decimal point without rounding.

TRUNCATE(4.789,1) → 4.7.

TRUNCATE(4.789,0) → 4

TRUNCATE(-4.789,0) → -4

4) ROUND(number, decimal-places)

↳ Rounds off digits to the required decimal places.

ROUND(123.4567,2) → 123.46

ROUND(123.4515,2) → 123.45

Questions:

1) To round off average of population in city table.

→ select floor(Avg(population)) from city;

2)

Samantha was tasked with calculating the average monthly salaries for all employees in the **EMPLOYEES** table, but did not realize her keyboard's 0 key was broken until after completing the calculation. She wants your help finding the difference between her miscalculation (using salaries with any zeros removed), and the actual average salary.

Write a query calculating the amount of error (i.e.: $actual - miscalculated$ average monthly salaries), and round it up to the next integer.

Sample Input

ID	Name	Salary
1	Kristeen	1420
2	Ashley	2006
3	Julia	2210
4	Maria	3000

Sample Output

```
2061
```

→ select ceil(avg(salary) - avg(cast(
replace(salary, '0', '') as double) from employees;

replace → returns str.
'0' → old char
'' → new char.
↳ change to double using cast.

3)

We define an employee's total earnings to be their monthly $salary \times months$ worked, and the maximum total earnings to be the maximum total earnings for any employee in the **Employee** table. Write a query to find the maximum total earnings for all employees as well as the total number of employees who have maximum total earnings. Then print these values as 2 space-separated integers.

employee_id	name	months	salary
12228	Rose	15	1968
33645	Angela	1	3443
45692	Frank	17	1608
56118	Patrick	7	1345
59725	Lisa	11	2330
74197	Kimberly	16	4372
78454	Bonnie	8	1771
83565	Michael	6	2017
98607	Todd	5	3396
99989	Joe	9	3573

→ select max(months * salary), count(*)
from employee
where months * salary = (select max(months * salary) from employee);

4) Query the sum of Northern Latitudes (LAT_N) from **STATION** having values greater than 38.7880 and less than 137.2345. Truncate your answer to 4 decimal places.

STATION

Field	Type
ID	NUMBER
CITY	VARCHAR2(21)
STATE	VARCHAR2(2)
LAT_N	NUMBER
LONG_W	NUMBER

→ select truncate(sum(lat_n), 4)
from (
select * from station
where lat_n > 38.7880 AND lat_n < 137.2345
) as temp;

(or)

lat_n between 38.7880 and 137.2345

5) Query the Western Longitude (LONG_W) for the largest Northern Latitude (LAT_N) in **STATION** that is less than 137.2345. Round your answer to 4 decimal places.

→ select round(long_w, 4)
from station
where lat_n = (select max(lat_n) from station
where lat_n < 137.2345);

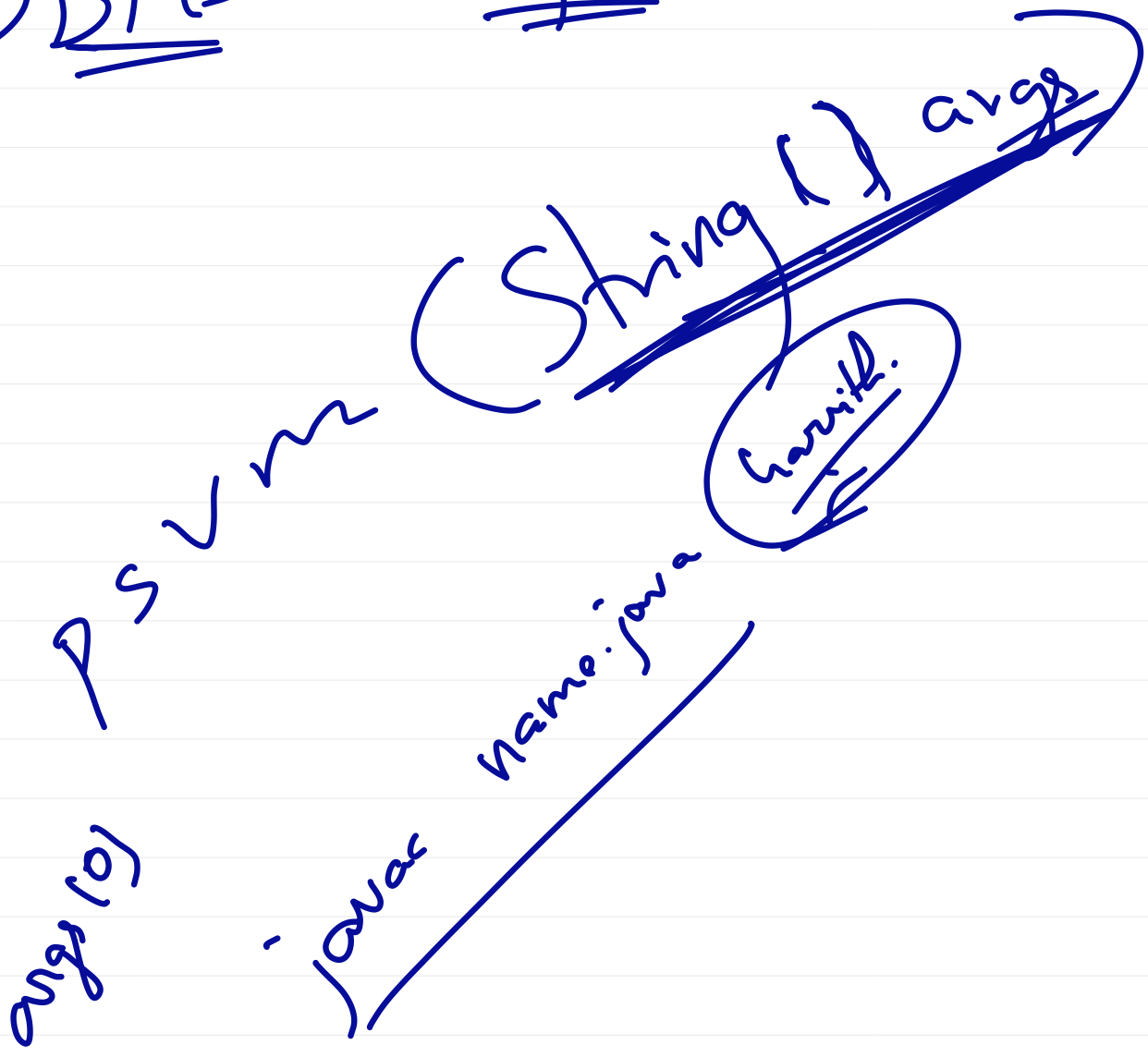
1) Stored procedures.

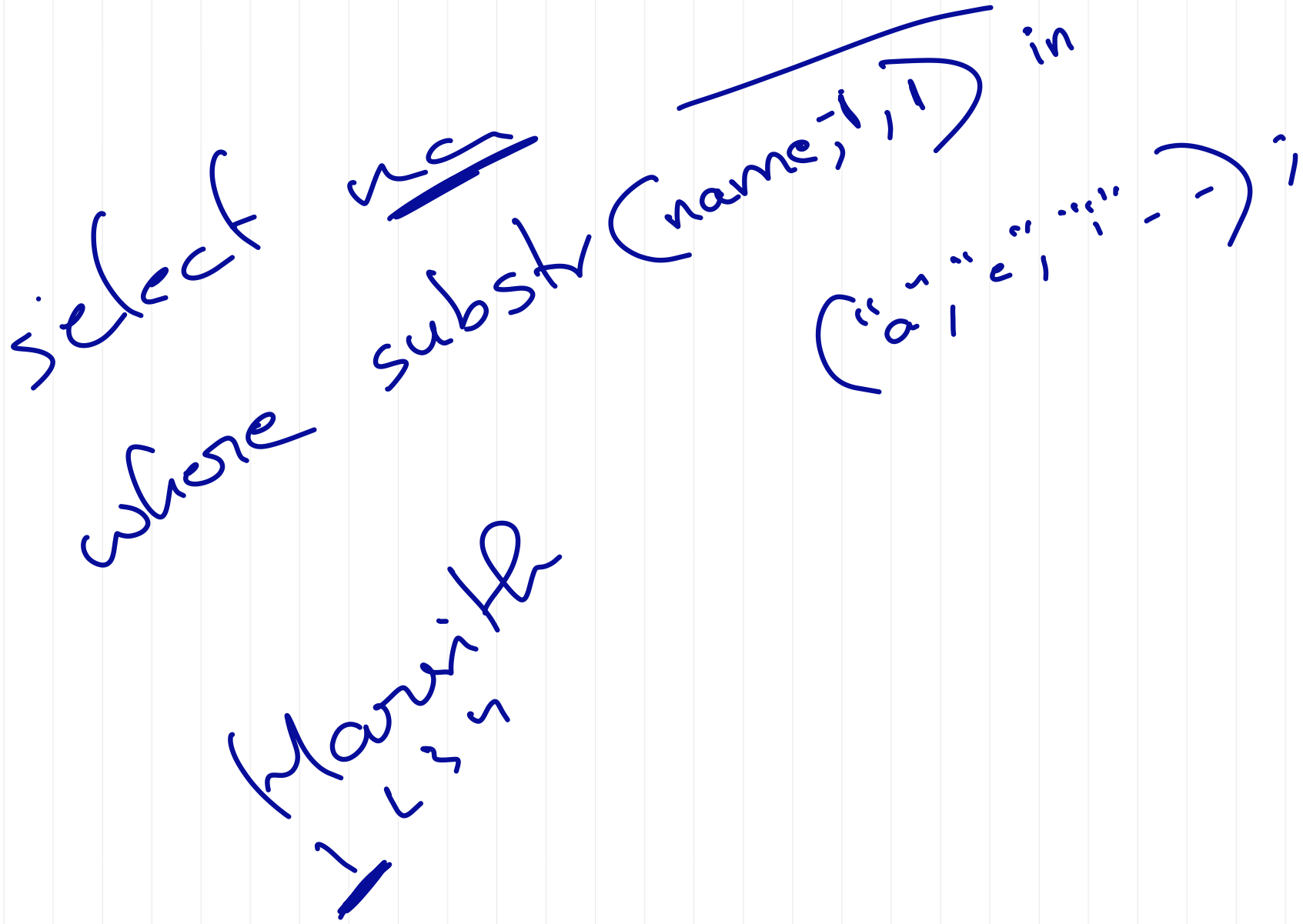
2) Triggers.

3) Aggregat functions

4) Cursors

DBMS - Topics.





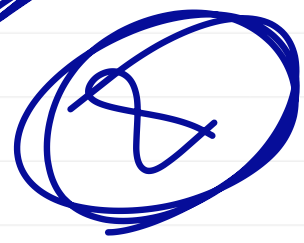
Month
Year
date

YYYY-MM-DD

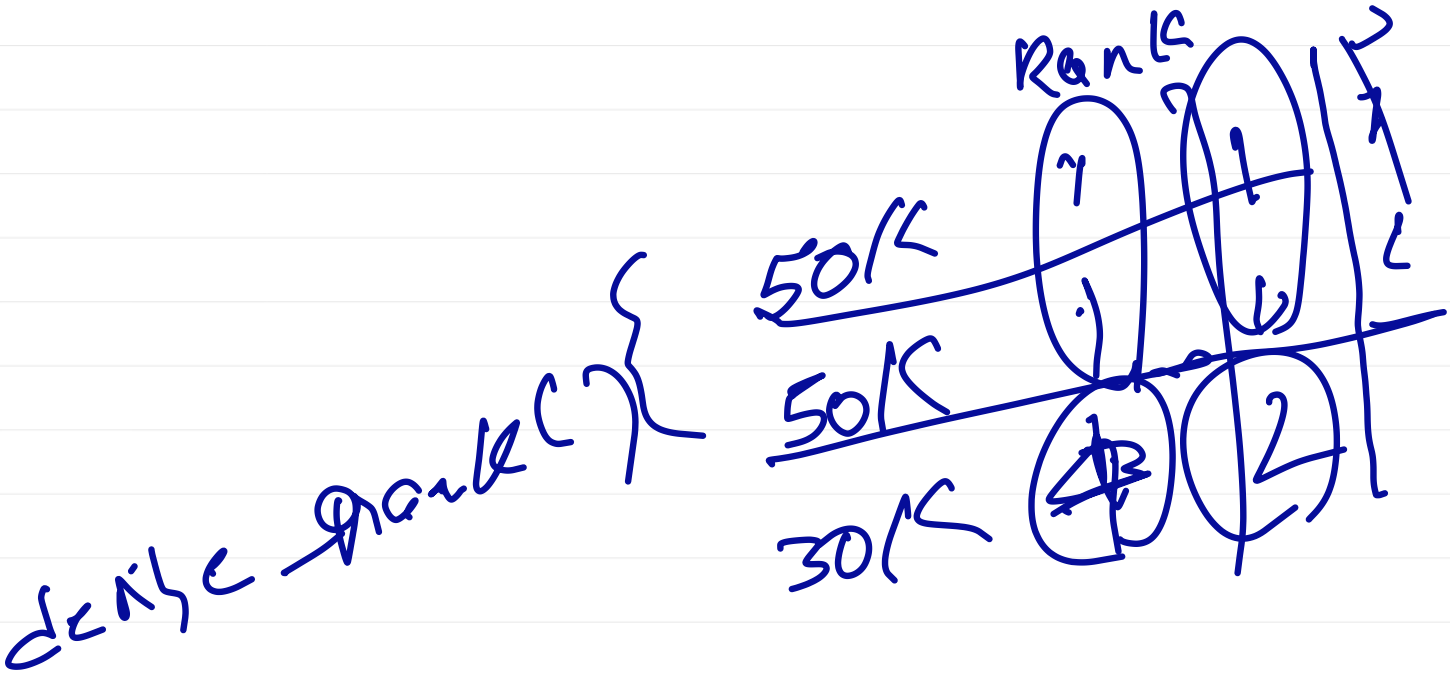
Hour
Minute
Sec

HH-MM-SS

Logical AND




1
2
3


Rank
50K
50K
30K
dense Rank()