MySQL

DDL → CREATE, ALTER, DROP, RENAME,TRUNCATE

DML → INSERT, UPDATE, DELETE

TCL → COMMIT, ROLLBACK

DQL → SELECT (extracting the data)

Describe: It describe the structure of the table in the database.

SYNTAX: **DESC emp;**

- New column, change the datatype, rename column name, table name, add constraints
- Database name can't be changed in workbench (exception).

Adding new column (By using ALTER command)

1. **Adding column to a table**
   ALTER TABLE emp ADD address VARCHAR(20);
2. **Adding column to a table after a specific column**
   ALTER TABLE emp ADD doj DATE AFTER empid;
3. **Adding column to a table before a specific column**

   ALTER TABLE emp ADD f_name VARCHAR(20) BEFORE empid; → NOT IN MYSQL

   ALTER TABLE emp ADD f_name VARCHAR(20) FIRST;
4. **BY DEFAULT, it add new column in the end**
   ALTER TABLE emp ADD incentive INT;

⇨ **Rename column name (By using ALTER, RENAME command)**
ALTER TABLE emp RENAME COLUMN salary TO sal;

⇨ **Changing datatype of column**
ALTER TABLE emp MODIFY f_name CHAR(15);

⇨ **Rename table name**
RENAME TABLE emp TO emp1;

⇨ **Drop column from the table**
ALTER TABLE emp1 DROP COLUMN address;

⇨ **Adding primary to a table**
ALTER TABLE emp1
ADD PRIMARY KEY(empid);

⇨ **Dropping primary key from the table**
(There is only 1 PK in a table)
ALTER TABLE emp1
DROP PRIMARY KEY;

- **PRIMARY KEY = NOT NULL + UNIQUE**

⇨ **DDL → CREATE, ALTER, RENAME, TRUNCATE, DROP**

- ***DROP TABLE t1;*** → it complete vanishes the table along with the data.
- ***DROP TABLE table1, table2;*** → Drop multiple table in one go.

⇨ **DML → INSERT, UPDATE, DELETE**

- **Drop multiple columns in one query**
  ALTER TABLE emp1 DROP doj, DROP sal;
- **Create new table**
  CREATE TABLE emp(
  empid INT, sal INT, ince INT);
- **Insert data in the table**
  INSERT INTO emp VALUES (1,34555,800);
  INSERT INTO emp VALUES (1,45000, null);
  INSERT INTO emp (empid, sal) VALUES (2, 35000)
  INSERT INTO emp VALUES (3,15200,900),(4,45200,800);
- **Alter table by adding new column**
  ALTER TABLE emp ADD deptid INT;
- **Updating values of new column**
  UPDATE emp SET ince=300; → *(*it updates every inces = 300 in the table)
  UPDATE emp SET ince=300 WHERE empid=2; → *(*it updates ince=300 only where empid=2)
- **In case of duplicates rows**
  UPDATE emp SET ince=100
  WHERE empid=1 AND sal=4500;

  UPDATE emp SET ince=1000 WHERE ince=NULL;
  (this query run but nothing will change in the table because NULL != NULL)

UPDATE emp SET ince=1000 WHERE ince IS NULL; → **(Correct query to update null values)**

DDL commands are auto commit.
ROLLBACK & COMMIT is only works for DML commands only if SET AUTOCOMMIT=0;

⇨ **Difference b/w DELETE & TRUNCATE**

1. **TRUNCATE :**
   a. DDL command (autocommit)
      It also works like DML
   b. Where clause can't allowed.
   c. Can't rollback the data
   d. Truncate deletes the data permanently.

DROP → DDL , vanish everything (structure+data), autocommit.
TRUNCATE → DDL, vanish only data, autocommit, can't use where clause.
DELETE →DML, vanish data only, can rollback the data, can use WHERE clause.

- **TRUNCATE is comparatively faster than DROP because it can't use where clause.**

⇨ **DQL → SELECT (extracting the data)**

- **To use particular schemas**
  USE hr;

- **To show all tables in the database**
  SHOW TABLES;
  (let tables :
  1. Employees
  2. Departments
  3. Locations
  4. Job_history
  5.

- **To show all data in a table**
  *SELECT * FROM employees;*

⇨ **Operators :**
Arithmatical → + - * / %
Conditional→ if, case
Comparison→ <, >, <>, !=
Relational→ >=, <=
Boolean→
Logical → and , or, not
IN
IS
BETWEEN
LIKE

- **Data Dictionary is like Python Libraries in python.**

- **DUAL :** Data dictionary which helps to create pseudo columns.
  SELECT 1+90; → 91
  SELECT 1+90 FROM DUAL; → standard
  SELECT 1=1 FROM DUAL; → 1 (True)
  SELECT 1=NULL FROM DUAL → *error*
  SELECT 1+'chandan' FROM DUAL → 1
  SELECT 'chandan' + 'chandan' FROM DUAL → 0 (because strings can't be added)
  SELECT 1=1 OR 1=NULL FROM DUAL → 1
  SELECT 1=1 AND 1=NULL FROM DUAL → null

- **To extract all data of those employees whose department_id = 90**
  *SELECT * FROM employees*
  WHERE department_id=90;

- **Alias** → nick name
  - To shorten the name
  - To hide the information

  SELECT employee_id empid, *first_name* f_name, *salary* s *FROM employees;*

⇨ **Alternate syntaxes to use alias**
SELECT
employee_id empid,
employee_id emp_id,
employee_id "emp id",
employee_id AS "emp#$%^id"
FROM employees;

SELECT
employee_id empid,
first_name, salary,
salary+5000 "Net Salary" ← **Computational columns**
FROM employees;

- **When we have to store the output of the query**
  CREATE TABLE output AS
  SELECT
  employee_id empid,
  first_name, salary,
  salary+5000
  FROM employees;

- **To view the output of the query which is saved earlier**
  *SELECT * FROM output;*

- **If we have to save the output of the query in another schema**
  CREATE TABLE schemaname.output AS
  SELECT
  employee_id empid,
  first_name, salary,
  salary+5000
  FROM employees;

- **Write a query to fetch all the data of department_id is 50 or 80 and salary > 8000.**
  SELECT employee_id, department_id, salary
  FROM employees
  WHERE (department_id =50 OR department_id = 80) AND salary>8000; → **21 rows**

  SELECT employee_id, department_id, salary
  FROM employees
  WHERE department_id IN (50,80)
  AND salary>8000; → **21 rows**

⇨ **Write a query for all those employees who get commissioned and increase their salary by 1000.**
SELECT employee_id, salary, salary+1000 net
FROM employees
WHERE commission_pct IS NOT NULL; → **35 rows**

⇨ **Write a query for all those employees who doesn't get commissioned and increase their salary by 1000.**
SELECT employee_id, salary, salary+1000 net
FROM employees
WHERE commission_pct IS NULL; → **72 rows**

⇨ **Write a query for net salary by using their commission_pct.**
*SELECT employee_id, salary, commission_pct, salary+(salary*commission_pct) net*
FROM employees; → **Wrong query (we have to use Function for right answer)**

⇨ **Write a query for all those employees whose salary is between 7000 and 12000.**
SELECT employee_id, salary
FROM employees
WHERE salary >7000 AND salary<12000; → **35 rows(7000 & 12000 excluded)**

SELECT employee_id, salary
FROM employees
WHERE salary BETWEEN 7000 AND 12000; →
SELECT employee_id, salary
FROM employees
WHERE salary >7000 OR salary<12000; → **41 rows (7000 & 12000 included)**

⇨ **Write a query for all those employees whose job_id are either FI_ACCOUNT or ST_MAN.**
SELECT employee_id, job_id
FROM employees
WHERE job_id = 'FI_ACCOUNT' OR job_id = 'ST_MAN'; → **10 rows**

SELECT employee_id, job_id
FROM employees
WHERE job_id IN ('FI_ACCOUNT', 'ST_MAN'); → **10 rows**

SELECT employee_id, job_id
FROM employees
WHERE job_id IN ('FI_ACCOUNT', 'ST_MAN', 'ST_CLERK', 'IT_PROG'); → **35 rows (more preferred way)**

- **LIKE operator** → same kind of pattern/ similar
  (% → 0 or >0 char)
  (_ → 1 char)

  *SELECT * FROM employees*
  WHERE first_name LIKE 'S%'; → **(all names must start with S)**

  *SELECT * FROM employees*
  WHERE first_name LIKE '%S'; → **(all names must end with S)**

- **Write a query where all names start or ends with S.**
  *SELECT * FROM employees*
  WHERE first_name LIKE 'S% OR
  first_name LIKE '%S'; → **20 rows**

⇨ **In MySQL,**
Chandan, chandan, CHANdan → All are same.

*SELECT * FROM employees*
WHERE BINARY(first_name) = 'James';
(Here, binary makes it case sensitive)

- **Write a query for first_name in which first_name's 2nd last character is c.**
  SELECT employee_id, first_name
  FROM employees
  WHERE first_name LIKE '%c_'; → **5 rows**

- **Either your first name start with v or last name ends with g.**
  SELECT employee_id, first_name, last_name
  FROM employees
  WHERE first_name LIKE 'V%' OR
  last_name LIKE '%g'; → **8 rows**

- **Arrange the data in ascending or descending form using ORDER BY clause (ORDER BY clause is the last clause of any query)**
  SELECT employee_id, first_name, last_name, salary
  FROM employees
  ORDER BY salary DESC;

(order by works on the columns which are in select statement)

- **Arrange the data in ascending order based on 2nd column if there is any duplicity then prefer salary column.**
  SELECT employee_id, first_name, last_name, salary FROM employees
  ORDER BY first_name, salary;

- **Select all the employees whose name is John and which John is getting highest salary.**
  SELECT employee_id, first_name, last_name, salary FROM employees
  WHERE first_name = 'John'
  ORDER BY first_name, salary DESC;

- **Select only starting 3 rows**
  SELECT employee_id, first_name, last_name, salary
  FROM employees LIMIT 3;

- **Pickup 3 rows from the top but leave the 1st row.**
  SELECT employee_id, first_name, last_name, salary
  FROM employees LIMIT 1,3;

- **Display employees information whose getting 5th highest salary**
  SELECT employee_id, first_name, salary
  FROM employees
  ORDER BY salary DESC
  LIMIT 4,1;

- **If there is any duplicity in salary , then query will be use DISTINCT clause.**
  SELECT DISTINCT salary
  FROM employees
  ORDER BY salary DESC
  LIMIT 4,1;

⇨ **Sequence of clauses for writing a query:-**
1. **SELECT**
2. **FROM**
3. **LIMIT**
4. **WHERE**
5. **ORDER BY**

⇨ **Execution sequence** :-
1. **FROM**
2. **WHERE (**alias can't be used in where)
3. **SELECT**
4. **DISTINCT**
5. **ORDER BY (**alias can be used in order by)
6. **LIMIT**

----------FUNCTIONS----------

⇨ **FUNCTION :** Set of codes which performs specifics tasks again & again.

⇨ **Functions are of two types:**
1. Built-In function
2. User defined function

⇨ **Again Built-In function has two categories:**
1. **Single Row**
   - **a.** 1 row = 1 answer
   - **b.** Datatypes
   - **c.** String functions
   - **d.** Numeric functions
   - **e.** Date functions

2. **Multiple Row**
   - **a.** Multiple rows = 1 answer

⇨ **STRING FUNCTIONS:**
(UPPER, LOWER, CONCAT, TRIM, INITCAP(This is not in MySQL)

SELECT UPPER(first_name),
LOWER(last_name)
FROM employees;

*SELECT * FROM employees*
WHERE UPPER(first_name) = 'LEX';

- **TRIM ( )**
  SELECT ' Chandan ',
  TRIM(' Chandan ')
  FROM DUAL;

- **SUBSTR ( )**
  SELECT SUBSTR('Chandan', 2, 3);

	SUBSTR('Chandan', 2, 3)
▶	han

- **Separate the first name and last name 'Chandan Singh'**
  SELECT 'Chandan Singh',
  SUBSTR('Chadnan Singh', 1, 7) f_name,
  SUBSTR('Chandan Singh',9) l_name;

	Chandan Singh	f_name	l_name
▶	Chandan Singh	Chadnan	Singh

- **INSTR( ) → it returns the position value.**
SELECT first_name, INSTR(first_name, 'e')
FROM employees;

	first_name	INSTR(first_name, 'e')
▶	Steven	3
	Neena	2
	Lex	2

- **General query to separate first name and last name? using SUBSTR () and INSTR()**
SELECT 'Chandan#Singh' full_name,
SUBSTR('Chadnan#Singh',1,INSTR('Chandan#Singh','#')) f_name,
SUBSTR('Chandan#Singh',INSTR('Chandan#Singh','#')) l_name;

	full_name	f_name	l_name
▶	Chandan#Singh	Chadnan#	#Singh

SELECT 'Chandan#Singh' full_name,
SUBSTR('Chadnan#Singh',1,INSTR('Chandan#Singh','#')-1) f_name,
SUBSTR('Chandan#Singh',INSTR('Chandan#Singh','#')+1) l_name;

	full_name	f_name	l_name
▶	Chandan#Singh	Chadnan	Singh

- **Convert the first letter into capital and rest all are in small letters.**
SELECT 'chandan',
CONCAT(UPPER(SUBSTR('chandan',1,1)),
SUBSTR('chandan',2));

	chandan	CONCAT(UPPER(SUBSTR('chandan',1,1)), SUBSTR('chandan',2))
▶	chandan	Chandan

- **CONCAT ( )**
SELECT
CONCAT(first_name,' you works in ',
department_id, ' department_id')
FROM employees;

	CONCAT(first_name,' you works in ', department_id, ' department_id')
▶	Steven you works in 90 department_id
	Neena you works in 90 department_id
	Lex you works in 90 department_id

- **REPLACE ( )**
REPLACE('Blablckbl', 'Bl', 'J'); **→ case sensitive exception**

	REPLACE('BlaBlckBl', 'Bl', 'J')
▶	JaJckJ

- **LEFT () and RIGHT ()**
SELECT 'Chandan Singh' full_name,
LEFT('Chandan Singh',7) f_name,
RIGHT('Chandan Singh',5) l_name;

SELECT 'Chandan Singh',
LEFT('Chandan Singh',instr('Chandan Singh', ' ')-1) f_name,
RIGHT('Chandan Singh',
LENGTH('Chandan Singh')-
INSTR('Chandan Singh',' ')) l_name;

	full_name	f_name	l_name
▶	Chandan Singh	Chandan	Singh

⇨ **NUMERIC functions**
1. **Round**
2. **Truncate**
3. **Floor**
4. **Ceil**

- **ROUND ()**
SELECT ROUND(34.67),
ROUND(34.67,0),
ROUND(34.67,1);

	ROUND(34.67)	ROUND(34.67,0)	ROUND(34.67,1)
▶	35	35	34.7

SELECT ROUND(34.67,-1),
ROUND(34.67,-2),
ROUND(89.67,-2);

	ROUND(34.67,-1)	ROUND(34.67,-2)	ROUND(89.67,-2)
▶	30	0	100

- **FLOOR ()**
SELECT FLOOR(67.89)
FROM DUAL;

	FLOOR(67.89)
▶	67

- **CEIL ()**
SELECT CEIL(67.89)
FROM DUAL;

	CEIL(67.89)
▶	68

- **TRUNCATE() :** It is similar to FLOOR function but it takes 2nd parameter.
SELECT TRUNCATE(67.23, 1),
TRUNCATE(67.58,1) FROM DUAL;

	TRUNCATE(67.23, 1)	TRUNCATE(67.58,1)
▶	67.2	67.5

- **DATE Function:**

1. Current_date ()
2. Datediff ()
3. Adddate()

⇨ **CURRENT_DATE ()**

SELECT
CURRENT_DATE(),CURDATE(),
CURRENT_TIME(),CURTIME();

	CURRENT_DATE()	CURDATE()	CURRENT_TIME()	CURTIME()
▶	2022-08-05	2022-08-05	10:27:52	10:27:52

SELECT LAST_DAY('2022-02-24'),
LAST_DAY(CURRENT_DATE()),
NOW();

	LAST_DAY('2022-02-24')	LAST_DAY(CURRENT_DATE())	NOW()
▶	2022-02-28	2022-08-31	2022-08-05 10:32:50

- **DATEDIFF Function:**

SELECT hire_date, CURRENT_DATE(),
DATEDIFF(CURRENT_DATE(), hire_date)/365 years,
FLOOR(DATEDIFF(CURRENT_DATE(),
hire_date)/365) floor_years,
*(DATEDIFF(CURRENT_DATE(), hire_date)/365)*12 months*
FROM employees;

	hire_date	CURRENT_DATE()	years	floor_years	months
▶	1987-06-17	2022-08-05	35.1589	35	421.9068
	1989-09-21	2022-08-05	32.8932	32	394.7178
	1993-01-13	2022-08-05	29.5781	29	354.9370

- **Modify datatype of a column:**

First convert the csv file into xls format ,then import it into MySQL. And then using *ALTER* and *MODIFY* command, we can change the datatypes.

BEFORE:

	Field	Type	Null	Key	Default	Extra
▶	sno	int	YES		NULL	
	dob	text	YES		NULL	

AFTER:

ALTER TABLE book1
MODIFY dob DATE;

	Field	Type	Null	Key	Default	Extra
▶	sno	int	YES		NULL	
	dob	date	YES		NULL	

- **ADDDATE(), SUBDATE()**

SELECT CURRENT_DATE(),
ADDDATE(CURRENT_DATE(),3) next_date,
SUBDATE(CURRENT_DATE(),3) previous ;

	CURRENT_DATE()	next_date	previous
▶	2022-08-08	2022-08-11	2022-08-05

- **PERIOD_DIFF( )**

SELECT PERIOD_DIFF(202208,202201);

	PERIOD_DIFF(202208,202201)
▶	7

Note: date format must be "yyyymm" otherwise it won't work.

- **DATE format**

SELECT hire_date,
DATE_FORMAT(hire_date, '%d%m%y') d1,
DATE_FORMAT(hire_date, '%D%M%Y') d2,
DATE_FORMAT(hire_date, '%D-%M-%Y') d3
FROM employees;

	hire_date	d1	d2	d3
▶	1987-06-17	170687	17thJune1987	17th-June-1987
	1989-09-21	210989	21stSeptember1989	21st-September-1989
	1993-01-13	130193	13thJanuary1993	13th-January-1993

⇨ **Pickup only date from this ('1999-09-01 12:23:03')**

SELECT ('1999-09-01 12:23:03'),
DATE('1999-09-01 12:23:03') date,
YEAR('1999-09-01 12:23:03') year,
MONTH('1999-09-01 12:23:03') month,
MONTHNAME('1999-09-01 12:23:03')
monthname,
TIME('1999-09-01 12:23:03') time
FROM DUAL;

	1999-09-01 12:23:03	date	year	month	monthname	time
▶	1999-09-01 12:23:03	1999-09-01	1999	9	September	12:23:03

SELECT DATE_FORMAT(CURRENT_DATE(),
'%D %M %Y %W %w');

	DATE_FORMAT(CURRENT_DATE(), '%D %M %Y %W %w')
▶	8th August 2022 Monday 1

⇨ **Display all the names which must starts with the vowels.**

SELECT first_name
FROM employees
WHERE LOWER(LEFT(first_name,1))
IN ('a','e','i','o','u'); **→ 16 rows**

⇨ **Show location id and cities of US or UK whose city name starts from 'S' but not from 'South'.**

SELECT location_id, city, country_id
FROM locations
WHERE country_id IN ('US', 'UK')
AND city like 'S%' AND city NOT LIKE 'South%';

	location_id	city	country_id
▶	2600	Stretford	UK
	1700	Seattle	US
*	NULL	NULL	NULL

→ 2 rows

⇨ **Cnditional clauses;**

1. **IF(column_name, if TRUE retuen 1, else 0)**
   SELECT salary,
   IF(salary>5000,1,0)
   FROM employees;

	salary	IF(salary>5000,1,0)
▶	26000.00	1
	17000.00	1
	17000.00	1

2. **IFNULL()**
   SELECT salary, commission_pct,
   *salary+(salary*IFNULL(commission_pct,0)) net*
   from employees;

	salary	commission_pct	net
▶	26000.00	NULL	26000.0000
	17000.00	NULL	17000.0000
	17000.00	NULL	17000.0000

3. **NULLIF( )** (very rarely used)
   SELECT sal, incentive,
   NULLIF(sal, incentive)
   FROM nn;

	sal	incentive	NULLIF(sal, incentive)
▶	2000	2000	NULL
	2000	2000	NULL
	3000	3000	NULL
	2000	200	2000
	500	20	500
	3000	300	3000

4. **Extract the employees details whose net salary >8000.**
   SELECT salary, commission_pct,
   *salary+(salary*IFNULL(commission_pct,0)) net_sal*
   FROM employees
   WHERE
   *salary+(salary*IFNULL(commission_pct,0)) > 8000;* → **42 rows**

   Using derived table
   *SELECT * FROM (*
   SELECT salary, commission_pct,
   *salary+(salary*IFNULL(commission_pct,0)) net_sal*
   FROM employees) t1
   WHERE net_sal >8000; → **42 rows**

	salary	commission_pct	net_sal
	11000.00	NULL	11000.0000
	8200.00	NULL	8200.0000
	14000.00	0.40	19600.0000
	13500.00	0.30	17550.0000

- **CASE**

⇨ **Display all the details if salary>15001 then grade A, if salary is b/w 7001 and 15000 then grade B, if salary is b/w 0 and 7000, then grade C.**
SELECT salary,
CASE WHEN salary > 15001 THEN 'A'
WHEN salary BETWEEN 7001 AND 15000 THEN 'B'
WHEN salary BETWEEN 0 AND 7000 THEN 'C'
END AS output
FROM employees;

	salary	output
▶	26000.00	A
	17000.00	A

- **COALESCE ( )**
  If first condition is true then return true value, otherwise zero.
  SELECT commission_pct,
  COALESCE(commission_pct,0)
  FROM employees;

	commission_pct	COALESCE(commission_pct,0)
	NULL	0.00
	NULL	0.00
	0.40	0.40
	0.30	0.30

⇨ **Create empty table without data but structure is retained as it is.**
CREATE TABLE new_table AS
*SELECT * FROM employees*
WHERE 1=2;

- **<u>Multiple Row Function</u>**
  (Multiple Rows(Aggregate) = 1 answer)

⇨ **Sum(), avg(), max(), min()**
SELECT SUM(salary),
AVG(salary),
MAX(salary),
MIN(salary)
FROM employees;

	SUM(salary)	AVG(salary)	MAX(salary)	MIN(salary)
▶	693400.00	6480.373832	26000.00	2100.00

⇨ **Count(*)**
It counts all the rows whether the row is null or not.
*SELECT * FROM mul;*
SELECT COUNT(*) FROM mul;

	sno	sal
▶	1	777
	2	NULL
	NULL	NULL
	1	777
	2	NULL
	NULL	NULL
	7	419

	COUNT(*)
▶	7

⇨ *SELECT COUNT(*), COUNT(salary), COUNT(commission_pct)*
FROM employees;

	COUNT(*)	COUNT(salary)	COUNT(commission_pct)
▶	107	107	35

⇨ **Sum of salaries of department id 80 and 90 separately.**
SELECT department_id,SUM(salary)
FROM employees
WHERE department_id IN (80,90)
GROUP BY department_id
ORDER BY department_id; **(more optimised if condition is known)**

SELECT department_id,
SUM(salary)
FROM employees
GROUP BY department_id
*HAVING department_id IN (80,90);***(less optimised)**

	department_id	SUM(salary)
▶	80	304500.00
	90	60000.00

⇨ **Sum and average of salary according to designation of company**
SELECT job_id, SUM(salary),
AVG(salary)
FROM employees
GROUP BY job_id; → **19 rows**

	job_id	SUM(salary)	AVG(salary)
▶	AC_ACCOUNT	8300.00	8300.000000
	AC_MGR	12000.00	12000.000000
	AD_ASST	4400.00	4400.000000

⇨ **Count number of employees hired in particular year**
SELECT YEAR(hire_date), COUNT(*) no_of_emp
FROM employees
GROUP BY YEAR(hire_date)
ORDER BY no_of_emp DESC; → **12 rows**

	YEAR(hire_date)	no_of_emp
▶	1997	28
	1998	23
	1999	18

⇨ **Count number of employees hired in particular year whose sum of salary > 80000**
SELECT YEAR(hire_date),SUM(salary),
COUNT(*) no_of_emp
FROM employees
GROUP BY YEAR(hire_date)
HAVING SUM(salary) > 80000
ORDER BY no_of_emp DESC;

	YEAR(hire_date)	SUM(salary)	no_of_emp
▶	1997	180900.00	28
	1998	112100.00	23
	1999	88900.00	18
	1996	86000.00	10

With aggregate functions, must use HAVING clause instead of WHERE clause because WHERE clause only works with simple functions.
HAVING clause is specially meant for aggregate function.

⇨ **Display all duplicates based on their first names.**
SELECT first_name , COUNT(first_name)
FROM employees
GROUP BY first_name
HAVING COUNT(first_name)>1; → **13 rows**

	first_name	COUNT(first_name)
▶	Steven	2
	Alexander	2
	David	3

⇨ **Display count of employees who is hired in a particular in a particular month.**
SELECT YEAR(hire_date),
DATE_FORMAT(hire_date, '%M') monthname,
COUNT(*)
FROM employees
GROUP BY YEAR(hire_date), MONTH(hire_date)
ORDER BY YEAR(hire_date),
MONTH(hire_date); → **56 rows**

	YEAR(hire_date)	monthname	COUNT(*)
▶	1987	June	1
	1987	September	1
	1989	September	1

⇨