# Databases

CRUD → { create, read, update, delete }


# Databases

## Relational

↳ Table
↳ cols + rows
↳ unique key

excel

## Non-Relational

↳ Not a table
↳ key-value
↳ document (JSON)
↳ graph, JSON, key-value

RDBMS ⇒ create + maintain relational database

eg. MySql, oracle.

Database →

↳ Primary key

↳ Foreign key

↳ relatⁿ b/w 2 tables

↳ relⁿ b/w rows of one table

↳ Composite key

↳ Combination of 2 elem that uniquely identifies a table row

# SQL — basics

↳ lang to interact w/ RDBMS

↳ CRUD - create, retrieve, update, delete

↳ security.

Hybrid lang — 4 languages

- DML — data manipulation lang
- DDL — data definition lang
- DCL — data control lang
- DQL — data query lang

## Data tpes

```
INT                     -- Whole Numbers
DECIMAL(10,4)           -- Decimal Numbers - Exact Value
VARCHAR(100)            -- String of text of length 1
BLOB                    -- Binary Large Object, Stores large data
DATE                    -- 'YYYY-MM-DD'
TIMESTAMP               -- 'YYYY-MM-DD HH:MM:SS' - used for recordin
```

## Create Table

```
CREATE TABLE student
(
    student_id INT  PRIMARY KEY,
    student_name VARCHAR(20),
    major VARCHAR(20)
);
```

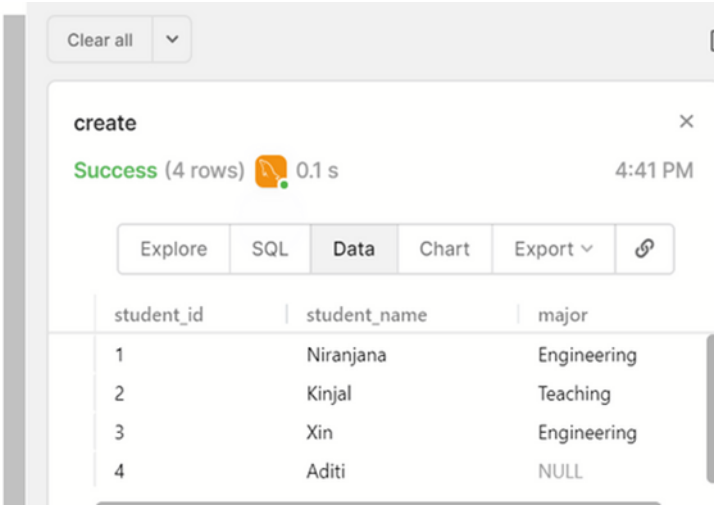
Steps :
- describe tab;e
- delete table
- alter table

## Create Table

```sql
-- Define a database schema
CREATE TABLE student (
    student_id INT PRIMARY KEY,
    student_name VARCHAR(20),
    major VARCHAR(20)
);
-- Describe the table
DESCRIBE student;

-- Delete the table
DROP TABLE student;

-- Adding a gpa table in decimal foramt with upto 2 decimal places
ALTER TABLE student ADD gpa DECIMAL(3,2);
DESCRIBE student;

-- Now, drop the table and check -
ALTER TABLE student DROP COLUMN gpa;
DESCRIBE student;
```

## Insert into a table

```sql
-- Define a database schema
CREATE TABLE student (
    student_id INT PRIMARY KEY,
    student_name VARCHAR(20),
    major VARCHAR(20)
);

-- inert value into the database
Insert INTO student VALUES(3,'Xin', 'Engineering');

-- insert when we  know only certain attributes
Insert INTO student(student_id, student_name) VALUES(4,'Aditi');

SELECT *    FROM student;
```




## Control the data enetered in the table - contraints

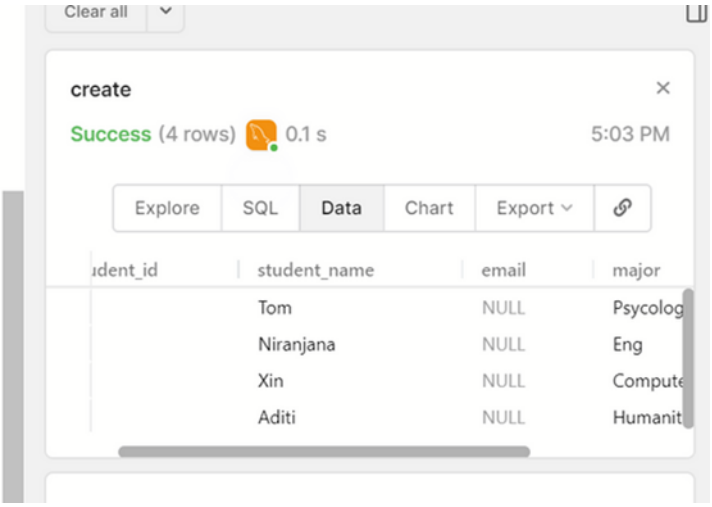
- not null , unique, primary key, defult, auto increment primary key

```sql
-- Define a database schema
-- Add specifiers for each entry - not nulll, unique
CREATE TABLE student (
    student_id INT PRIMARY KEY AUTO_INCREMENT,
    student_name VARCHAR(20) NOT NULL,
    email VARCHAR(20) UNIQUE,
    major VARCHAR(20) DEFAULT 'undecided'
);
```

# UPDATE AND DELETE

Update Engineering to Eng
Update student with student id = 3 to Computer Science
Update arts or undecided to humanities
Update student id 1 with name Tom and major Psycology

```sql
SELECT *    FROM student;

-- UPDATES

UPDATE student
SET major = 'Eng'
where major = 'Engineering';

UPDATE student
SET major = 'Computer Science'
where student_id = 3;

UPDATE student
SET major = 'Humanities'
where major = 'Arts' OR major = 'undecided';

UPDATE student
SET student_name = 'Tom', major = 'Psycology'
where student_id = 1;
```



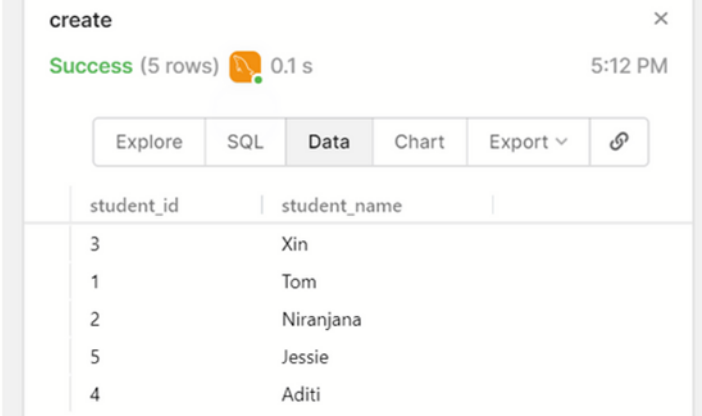

Delete specific rows
Delete all rows

```sql
-- DELETES

DELETE FROM student
WHERE student_id = 5;

-- delete all
Delete FROM student;
```

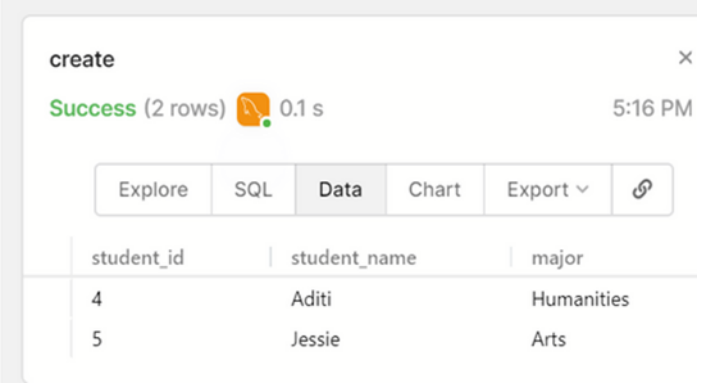
# QUERYING - SELECT

- Select student name and major from a the table
- Order the student name and student major by student name in descending order
- Order by name  and student id (respectively)
- Limit the number of rsults to 2

```sql
SELECT  student.student_id, student.student_name
FROM student
ORDER BY student_name DESC;
```



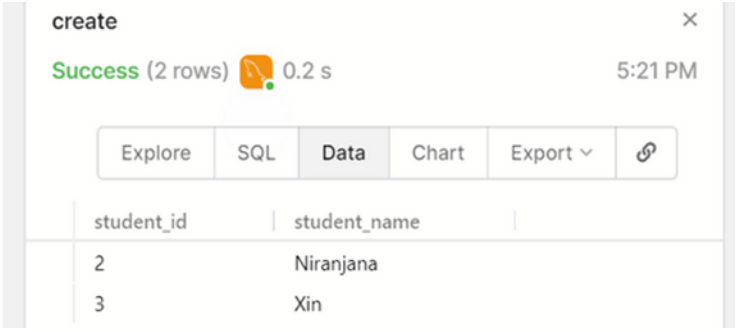
```sql
-- SELECT

SELECT  student.student_id, student.student_name, student.major
FROM student
ORDER BY student_name, student_id
LIMIT 2;
```




Update only when certain conditions are true

- all data from students when major is arts
- select only name and id where major is engineering or computer science

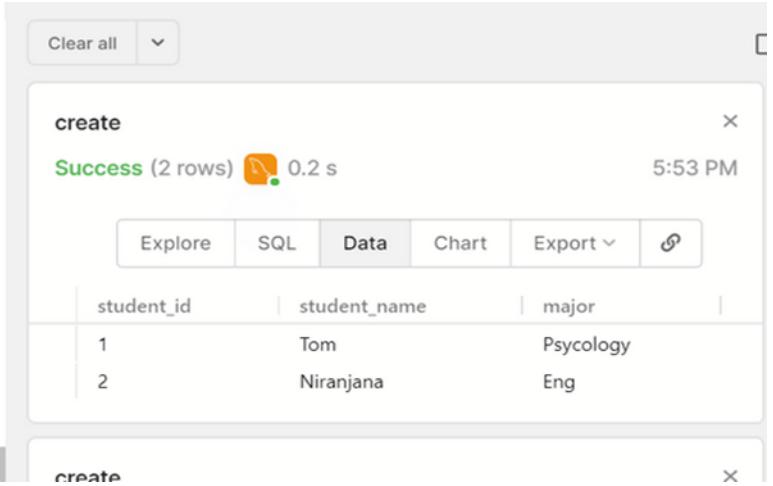
```sql
SELECT  student.student_id, student.student_name
FROM student
WHERE major = 'Computer Science' OR major = 'Eng';
```




Different comparison operators `<, >, <=,>=, =, <>, AND, OR`

- select name , major and id - where major is not arts and student is is less than or equal to 3
- select where major is comp sci or eng or humanities and student is id greater than 2

```sql
-- Comparison
-- > , < , <=, >=, AND, OR

SELECT student_id, student_name , major
FROM
student
WHERE
major <> 'Arts' AND
student_id <= 2;


SELECT student_id, student_name , major
FROM
student
WHERE
major IN ('Eng', 'Computer Science', 'Humanities') AND
student_id >2;
```



# Creating A More Complex Schema

## Company Database

### Employee

| emp_id | first_name | last_name | birth_date | sex | salary | super_id | branch_id |
|---|---|---|---|---|---|---|---|
| 100 | David | Wallace | 1967-11-17 | M | 250,000 | NULL | 1 |
| 101 | Jan | Levinson | 1961-05-11 | F | 110,000 | 100 | 1 |
| 102 | Michael | Scott | 1964-03-15 | M | 75,000 | 100 | 2 |
| 103 | Angela | Martin | 1971-06-25 | F | 63,000 | 102 | 2 |
| 104 | Kelly | Kapoor | 1980-02-05 | F | 55,000 | 102 | 2 |
| 105 | Stanley | Hudson | 1958-02-19 | M | 69,000 | 102 | 2 |
| 106 | Josh | Porter | 1969-09-05 | M | 78,000 | 100 | 3 |
| 107 | Andy | Bernard | 1973-07-22 | M | 65,000 | 106 | 3 |
| 108 | Jim | Halpert | 1978-10-01 | M | 71,000 | 106 | 3 |

### Branch

| branch_id | branch_name | mgr_id | mgr_start_date |
|---|---|---|---|
| 1 | Corporate | 100 | 2006-02-09 |
| 2 | Scranton | 102 | 1992-04-06 |
| 3 | Stamford | 106 | 1998-02-13 |

### Client

| client_id | client_name | branch_id |
|---|---|---|
| 400 | Dunmore Highschool | 2 |
| 401 | Lackawana Country | 2 |
| 402 | FedEx | 3 |
| 403 | John Daly Law, LLC | 3 |
| 404 | Scranton Whitepages | 2 |
| 405 | Times Newspaper | 3 |
| 406 | FedEx | 2 |

### Works_With

| emp_id | client_id | total_sales |
|---|---|---|
| 105 | 400 | 55,000 |
| 102 | 401 | 267,000 |
| 108 | 402 | 22,500 |
| 107 | 403 | 5,000 |
| 108 | 403 | 12,000 |
| 105 | 404 | 33,000 |
| 107 | 405 | 26,000 |
| 102 | 406 | 15,000 |
| 105 | 406 | 130,000 |

### Branch Supplier

| branch_id | supplier_name | supply_type |
|---|---|---|
| 2 | Hammer Mill | Paper |
| 2 | Uni-ball | Writing Utensils |
| 3 | Patriot Paper | Paper |
| 2 | J.T. Forms & Labels | Custom Forms |
| 3 | Uni-ball | Writing Utensils |
| 3 | Hammer Mill | Paper |
| 3 | Stamford Lables | Custom Forms |

my

First -
- Create employee table without mentioning that uper id and branch id are foreign keys
- create branch table and mention that manager id is a foreign key that refernces employee table, specifically employee id
- Then alter the original employee table to make branch id and super id foregin keys
- create a client table with foreign key as branch id that references the branch table , specifically branch id
- create works_with table that has two primary keys emp_id and client_id, with them being cross referenced as foreign keys on their original table
- create table branch supplier that has a composite key of branch id and supplier name with branch id being cross referenced as foreign key

```sql
CREATE TABLE employee (
  emp_id INT PRIMARY KEY,
  first_name VARCHAR(40),
  last_name VARCHAR(40),
  birth_day DATE,
  sex VARCHAR(1),
  salary INT,
  super_id INT,
  branch_id INT
);

CREATE TABLE branch (
  branch_id INT PRIMARY KEY,
  branch_name VARCHAR(40),
  mgr_id INT,
  mgr_start_date DATE,
  FOREIGN KEY(mgr_id) REFERENCES employee(emp_id) ON DELETE SET NULL
);

ALTER TABLE employee
ADD FOREIGN KEY(branch_id)
REFERENCES branch(branch_id)
ON DELETE SET NULL;

ALTER TABLE employee
ADD FOREIGN KEY(super_id)
REFERENCES employee(emp_id)
ON DELETE SET NULL;
```

```sql
CREATE TABLE client (
  client_id INT PRIMARY KEY,
  client_name VARCHAR(40),
  branch_id INT,
  FOREIGN KEY(branch_id) REFERENCES branch(branch_id) ON DELETE SET NULL
);

CREATE TABLE works_with (
  emp_id INT,
  client_id INT,
  total_sales INT,
  PRIMARY KEY(emp_id, client_id),
  FOREIGN KEY(emp_id) REFERENCES employee(emp_id) ON DELETE CASCADE,
  FOREIGN KEY(client_id) REFERENCES client(client_id) ON DELETE CASCADE
);

CREATE TABLE branch_supplier (
  branch_id INT,
  supplier_name VARCHAR(40),
  supply_type VARCHAR(40),
  PRIMARY KEY(branch_id, supplier_name),
  FOREIGN KEY(branch_id) REFERENCES branch(branch_id) ON DELETE CASCADE
);
```

Insert data- to be done in a specific order
- first insert corporate manager into empolyee branch with null as branch id
- inset the corresponding branch value
- update teh empolyee table with the branch id of the branch manager

- now add the locat branch managers in the same order
- add the employees of the branch

- add supplier data, client data and works with data sequentially

```sql
-- Corporate
INSERT INTO employee VALUES(100, 'David', 'Wallace', '1967-11-17', 'M', 250000, NULL, NULL);

INSERT INTO branch VALUES(1, 'Corporate', 100, '2006-02-09');

UPDATE employee
SET branch_id = 1
WHERE emp_id = 100;

INSERT INTO employee VALUES(101, 'Jan', 'Levinson', '1961-05-11', 'F', 110000, 100, 1);

-- Scranton
INSERT INTO employee VALUES(102, 'Michael', 'Scott', '1964-03-15', 'M', 75000, 100, NULL);

INSERT INTO branch VALUES(2, 'Scranton', 102, '1992-04-06');

UPDATE employee
SET branch_id = 2
WHERE emp_id = 102;

INSERT INTO employee VALUES(103, 'Angela', 'Martin', '1971-06-25', 'F', 63000, 102, 2);
INSERT INTO employee VALUES(104, 'Kelly', 'Kapoor', '1980-02-05', 'F', 55000, 102, 2);
INSERT INTO employee VALUES(105, 'Stanley', 'Hudson', '1958-02-19', 'M', 69000, 102, 2);

-- Stamford
INSERT INTO employee VALUES(106, 'Josh', 'Porter', '1969-09-05', 'M', 78000, 100, NULL);

INSERT INTO branch VALUES(3, 'Stamford', 106, '1998-02-13');

UPDATE employee
SET branch_id = 3
WHERE emp_id = 106;

INSERT INTO employee VALUES(107, 'Andy', 'Bernard', '1973-07-22', 'M', 65000, 106, 3);
INSERT INTO employee VALUES(108, 'Jim', 'Halpert', '1978-10-01', 'M', 71000, 106, 3);
```

```sql
-- BRANCH SUPPLIER
INSERT INTO branch_supplier VALUES(2, 'Hammer Mill', 'Paper');
INSERT INTO branch_supplier VALUES(2, 'Uni-ball', 'Writing Utensils');
INSERT INTO branch_supplier VALUES(3, 'Patriot Paper', 'Paper');
INSERT INTO branch_supplier VALUES(2, 'J.T. Forms & Labels', 'Custom Forms');
INSERT INTO branch_supplier VALUES(3, 'Uni-ball', 'Writing Utensils');
INSERT INTO branch_supplier VALUES(3, 'Hammer Mill', 'Paper');
INSERT INTO branch_supplier VALUES(3, 'Stamford Lables', 'Custom Forms');

-- CLIENT
INSERT INTO client VALUES(400, 'Dunmore Highschool', 2);
INSERT INTO client VALUES(401, 'Lackawana Country', 2);
INSERT INTO client VALUES(402, 'FedEx', 3);
INSERT INTO client VALUES(403, 'John Daly Law, LLC', 3);
INSERT INTO client VALUES(404, 'Scranton Whitepages', 2);
INSERT INTO client VALUES(405, 'Times Newspaper', 3);
INSERT INTO client VALUES(406, 'FedEx', 2);

-- WORKS_WITH
INSERT INTO works_with VALUES(105, 400, 55000);
INSERT INTO works_with VALUES(102, 401, 267000);
INSERT INTO works_with VALUES(108, 402, 22500);
INSERT INTO works_with VALUES(107, 403, 5000);
INSERT INTO works_with VALUES(108, 403, 12000);
INSERT INTO works_with VALUES(105, 404, 33000);
INSERT INTO works_with VALUES(107, 405, 26000);
INSERT INTO works_with VALUES(102, 406, 15000);
INSERT INTO works_with VALUES(105, 406, 130000);
```

Simple Queries to Explore

- Find all employees
- Find all clients
- Find all employees ordered by salary
- Find all employees ordered by sex then name
- Find the first 5 employees in the table
- Find the first and last names of all employees
- Find the forename and surnames names of all employees
- Find out all the different genders
- Find all male employees
- Find all employees at branch 2
- Find all employee's id's and names who were born after 1969
- Find all female employees at branch 2
- Find all employees who are female & born after 1969 or who make over 80000
- Find all employees born between 1970 and 1975
- Find all employees named Jim, Michael, Johnny or David

```sql
-- Find all employees
SELECT *
FROM employee;

-- Find all clients
SELECT *
FROM clients;

-- Find all employees ordered by salary
SELECT *
from employee
ORDER BY salary ASC/DESC;

-- Find all employees ordered by sex then name
SELECT *
from employee
ORDER BY sex, name;

-- Find the first 5 employees in the table
SELECT *
from employee
LIMIT 5;

-- Find the first and last names of all employees
SELECT first_name, employee.last_name
FROM employee;

-- Find the forename and surnames names of all employees
SELECT first_name AS forename, employee.last_name AS surname
FROM employee;
```

---

```sql
WHERE branch_id = 2;

-- Find all employee's id's and names who were born after 1969
SELECT emp_id, first_name, last_name
FROM employee
WHERE birth_day >= 1970-01-01;

-- Find all female employees at branch 2
SELECT *
FROM employee
WHERE branch_id = 2 AND sex = 'F';

-- Find all employees who are female & born after 1969 or who make over 80000
SELECT *
FROM employee
WHERE (birth_day >= '1970-01-01' AND sex = 'F') OR salary > 80000;

-- Find all employees born between 1970 and 1975
SELECT *
FROM employee
WHERE birth_day BETWEEN '1970-01-01' AND '1975-01-01';

-- Find all employees named Jim, Michael, Johnny or David
SELECT *
FROM employee
WHERE first_name IN ('Jim', 'Michael', 'Johnny', 'David');
```

UNION operator
- same number of columns and similar datatypes

JOINS
- inner join (JOIN)
- left join (LEFT JOIN)
- right join
- full outer join

- Find a list of employee and branch names
- Find a list of all clients & branch suppliers' names

```sql
-- Find a list of employee and branch names
SELECT employee.first_name AS Employee_Branch_Names
FROM employee
UNION
SELECT branch.branch_name
FROM branch;

-- Find a list of all clients & branch suppliers' names
SELECT client.client_name AS Non-Employee_Entities,
client.branch_id AS Branch_ID
FROM client
UNION
SELECT branch_supplier.supplier_name, branch_supplier.branch_id
FROM branch_supplier;
```

# NESTED QUERIES

ON delete

specifically used for foreign keys
- set null
- cascade

TRIGGERS

- Triggers can be created for insert update or delete to automates certain tasks
- BEFORE OR AFTER  -- create, delete etc.
- create trigger MY_TRIGGER (before/after) (CRUD)

- Simple triggers
- Triggers that can access info from the new/changed element
- Triggers with ELSE IF statements

syntax

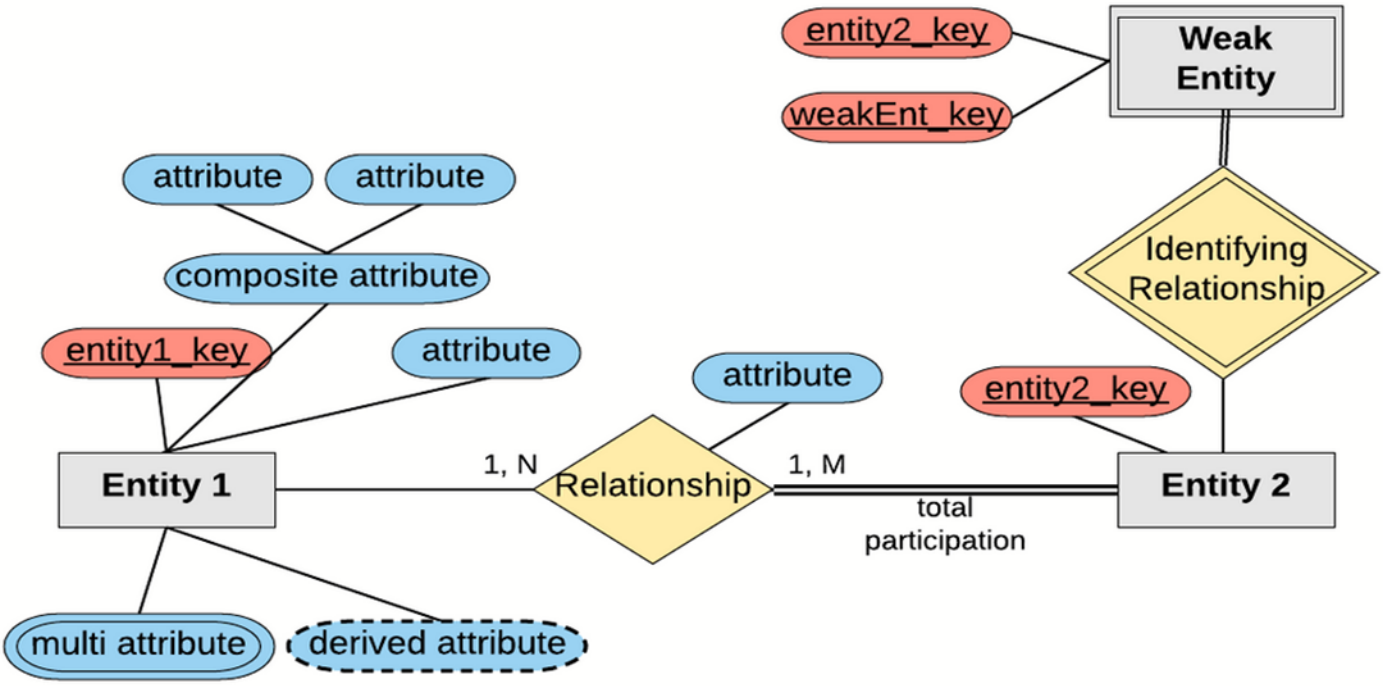
```
CREATE
    TRIGGER `event_name` BEFORE/AFTER
INSERT/UPDATE/DELETE
    ON `database`.`table`
    FOR EACH ROW BEGIN
  -- trigger body
  -- this code is applied to every
  -- inserted/updated/deleted row
    END;
```

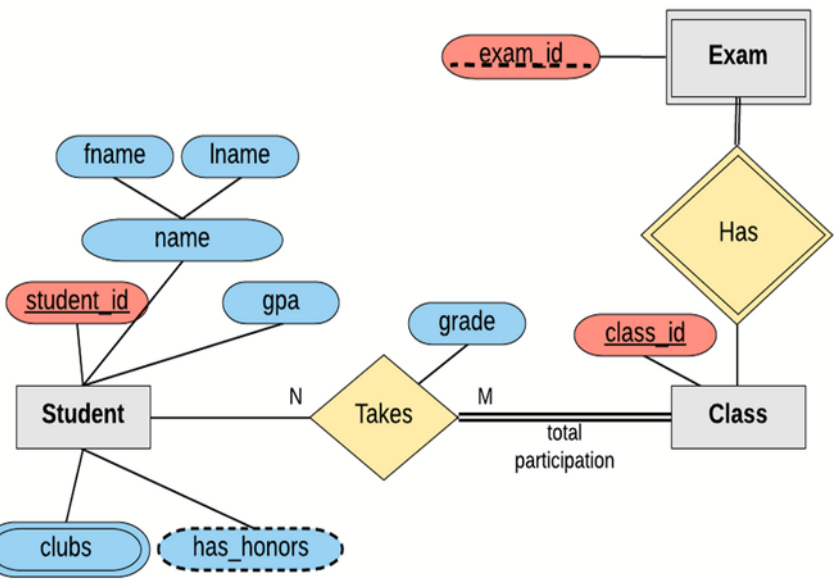
```
9
10  CREATE TABLE trigger_test (
11      message VARCHAR(100)
12  );
13
14
15
16
17  DELIMITER $$
18  CREATE
19      TRIGGER my_trigger BEFORE INSERT
20      ON employee
21      FOR EACH ROW BEGIN
22          INSERT INTO trigger_test VALUES('added new employee');
23      END$$
24  DELIMITER ;
25  INSERT INTO employee
26  VALUES(109, 'Oscar', 'Martinez', '1968-02-19', 'M', 69000, 106, 3);
27
28
29  DELIMITER $$
30  CREATE
31      TRIGGER my_trigger BEFORE INSERT
32      ON employee
33      FOR EACH ROW BEGIN
34          INSERT INTO trigger_test VALUES(NEW.first_name);
35      END$$
36  DELIMITER ;
37  INSERT INTO employee
38  VALUES(110, 'Kevin', 'Malone', '1978-02-19', 'M', 69000, 106, 3);
39
40  DELIMITER $$
41  CREATE
42      TRIGGER my_trigger BEFORE INSERT
43      ON employee
44      FOR EACH ROW BEGIN
45          IF NEW.sex = 'M' THEN
46              INSERT INTO trigger_test VALUES('added male employee');
47          ELSEIF NEW.sex = 'F' THEN
48              INSERT INTO trigger_test VALUES('added female');
49          ELSE
50              INSERT INTO trigger_test VALUES('added other employee');
51          END IF;
52      END$$
53  DELIMITER ;
54  INSERT INTO employee
55  VALUES(111, 'Pam', 'Beesly', '1988-02-19', 'F', 69000, 106, 3);
56
57
58  DROP TRIGGER my_trigger;
```

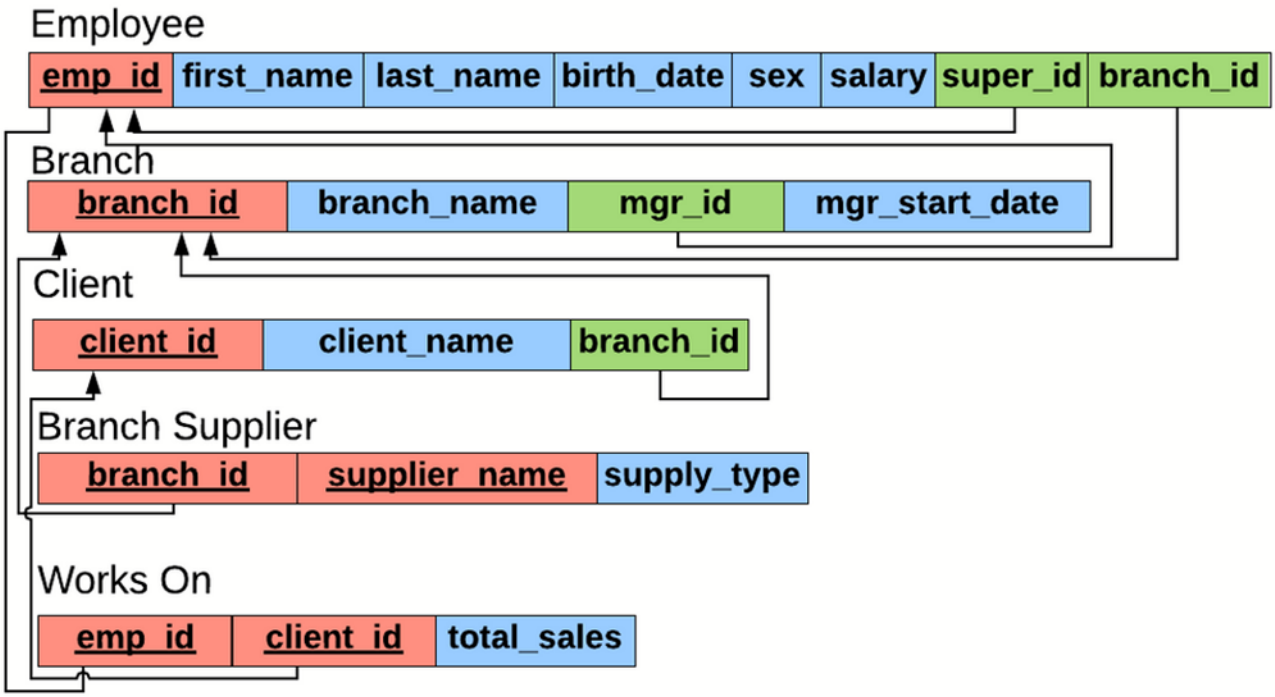
# ENTITY RELATIONSHIP DIGARAMS ( ER DIAGRAMS)

## ENTITY
- primary attributes
- composite attributes
- multi- vallued attribute
- derived attribute

## MULTIPLE ENTITIES
- reltionship between entities must be defined betweeen entities
- realtionship attribute - depeneds on the defined relationship between entities
- realtionship cardinality(1:1 , 1:N, N:M, M:1)




Employee
emp_id
first_name
last_name
birth_date
sex
salary
super_id
branch_id
Branch
branch_id
branch_name
mgr_id
mgr_start_date
Client
client_id
client_name
branch_id
Branch Supplier
branch_id
supplier_name
supply_type
Works On
emp_id
client_id
total_sales

# Designing an ER Diagram

Requirements:

The company is organized into branches. Each branch has a unique number, a name, and a particular employee who manages it.

The company makes it's money by selling to clients. Each client has a name and a unique number to identify it.

The foundation of the company is it's employees. Each employee has a name, birthday, sex, salary and a unique number.

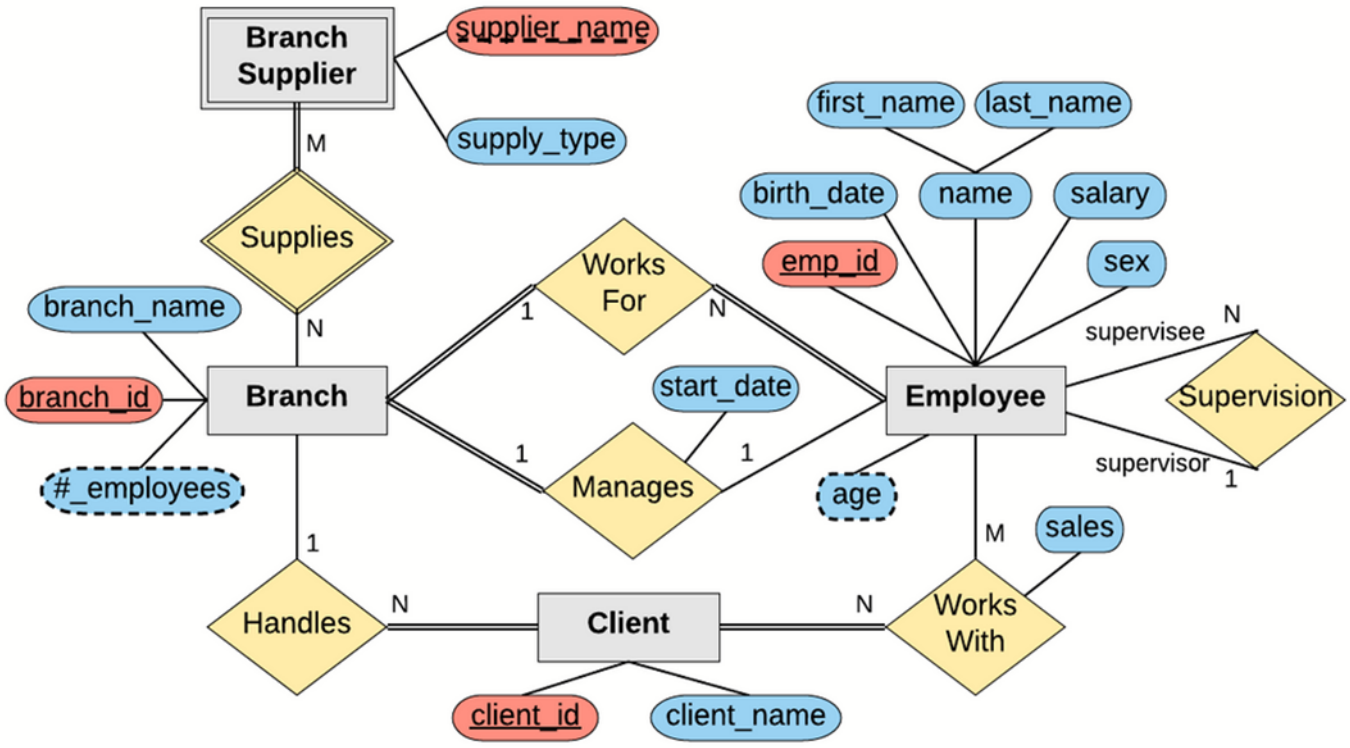
An employee can work for one branch at a time, and each branch will be managed by one of the employees that work there. We'll also want to keep track of when the current manager started as manager.

An employee can act as a supervisor for other employees at the branch, an employee may also act as the supervisor for employees at other branches. An employee can have at most one supervisor.

A branch may handle a number of clients, with each client having a name and a unique number to identify it. A single client may only be handled by one branch at a time.

Employees can work with clients controlled by their branch to sell them stuff. If nescessary multiple employees can work with the same client. We'll want to keep track of how many dollars worth of stuff each employee sells to each client they work with.

Many branches will need to work with suppliers to buy inventory. For each supplier we'll keep track of their name and the type of product they're selling the branch. A single supplier may supply products to multiple branches.



## SQL functions

- count , avg, sum
- group by ( aggregation)

- Find the number of employees
- Find the average of all employee's salaries
- Find the sum of all employee's salaries
- Find out how many males and females there are
- Find the total sales of each salesman
- Find the total amount of money spent by each client

```sql
-- functions in SQL

-- Find the number of employees
SELECT COUNT(super_id)
FROM employee;

-- Find the average of all employee's salaries
SELECT AVG(salary)
FROM employee;

-- Find the sum of all employee's salaries
SELECT SUM(salary)
FROM employee;

-- Find out how many males and females there are
SELECT COUNT(sex), sex
FROM employee
GROUP BY sex

-- Find the total sales of each salesman
SELECT SUM(total_sales), emp_id
FROM works_with
GROUP BY client_id;

-- Find the total amount of money spent by each client
SELECT SUM(total_sales), client_id
FROM works_with
GROUP BY client_id;
```

## Wild cards

- keyword - LIKE ( _ , %)

- Find any client's who are an LLC
- Find any branch suppliers who are in the label business
- Find any employee born on the 10th day of the month
- Find any clients who are schools

```sql
-- % = any # characters, _ = one character

-- Find any client's who are an LLC
SELECT *
FROM client
WHERE client_name LIKE '%LLC';

-- Find any branch suppliers who are in the label business
SELECT *
FROM branch_supplier
WHERE supplier_name LIKE '% Label%';

-- Find any employee born on the 10th day of the month
SELECT *
FROM employee
WHERE birth_day LIKE '_____10%';

-- Find any clients who are schools
SELECT *
FROM client
WHERE client_name LIKE '%Highschool%';
```