Creating Company Database

```
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
);
```

~ Primary Key

For M so only 1 character is needed.

Both "foreign keys".

We can't actually make these foreign keys yet b/c the Employee table doesn't technically exist and the branch table doesn't exist yet, since we haven't created them yet.

~ so we hold off naming them

Creating the Branch Table?

```
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
);
```

~ "also foreign key"

now we can designate it as primary

~ TALK ABOUT LATER

defining a foreign key

table

name of column

Whenever we create a foreign key we put this.

now we can add the foreign keys?

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

now we need to the super-id and branch-id.

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

~ since the Employee table is created we can now set the foreign key,

references manager (emp-id)

- foreign key stores another primary key,

```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
);
```

P
Table

```sql
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
);
```

2

```sql
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
);
```

Now were inserting Information into the tables:

→ Circular - relationship ←
With foreign keys

Corporate

branch hasn't been created yet. -> branch_id = 1

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

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

↳ now branch values have been created

update to adjust

```
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

```
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);
```

Inserting data

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);
```

★ Displaying Employee Table ★

```
select * from employee
Limit 100;
```

Success
9 rows

Explore | SQL | Data | Chart | Export

emp_id	first_name	last_name	birth_day	sex	salary	super_id	branch_id
100	David	Wallace	1967-11-17	M	250000	NULL	1
101	Jan	Levinson	1961-05-11	F	110000	100	1
102	Michael	Scott	1964-03-15	M	75000	100	2
103	Angela	Martin	1971-06-25	F	63000	102	2
104	Kelly	Kapoor	1980-02-05	F	55000	102	2
105	Stanley	Hudson	1958-02-19	M	69000	102	2
106	Josh	Porter	1969-09-05	M	78000	100	3
107	Andy	Bernard	1973-07-22	M	65000	106	3
108	Jim	Halpert	1978-10-01	M	71000	106	3

```
Select * from works_with
Limit 100;
```

emp_id	client_id	total_sales
102	401	267000
102	406	15000
105	400	55000
105	404	33000
105	406	130000
107	403	5000
107	405	26000
108	402	22500
108	403	12000

```
Select * from Branch
Limit 100;
```

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

```
Select * from Client
Limit 100;
```

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

```
select * from branch_supplier
Limit 100;
```

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

More Basic Queries

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

emp_id	first_name	last_name	birth_day	sex	salary	super_id	branch_id
100	David	Wallace	1967-11-17	M	250000	NULL	1
101	Jan	Levinson	1961-05-11	F	110000	100	1
102	Michael	Scott	1964-03-15	M	75000	100	2
103	Angela	Martin	1971-06-25	F	63000	102	2
104	Kelly	Kapoor	1980-02-05	F	55000	102	2
105	Stanley	Hudson	1958-02-19	M	69000	102	2
106	Josh	Porter	1969-09-05	M	78000	100	3
107	Andy	Bernard	1973-07-22	M	65000	106	3
108	Jim	Halpert	1978-10-01	M	71000	106	3

```
-- Find all clients
SELECT *
FROM 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

```sql
-- Find all employees ordered by salary
SELECT *
from employee
ORDER BY salary ASC;
```

emp_id	first_name	last_name	birth_day	sex	salary	super_id	branch_id
104	Kelly	Kapoor	1980-02-05	F	55000	102	2
103	Angela	Martin	1971-06-25	F	63000	102	2
107	Andy	Bernard	1973-07-22	M	65000	106	3
105	Stanley	Hudson	1958-02-19	M	69000	102	2
108	Jim	Halpert	1978-10-01	M	71000	106	3
102	Michael	Scott	1964-03-15	M	75000	100	2
106	Josh	Porter	1969-09-05	M	78000	100	3
101	Jan	Levinson	1961-05-11	F	110000	100	1
100	David	Wallace	1967-11-17	M	250000	NULL	1

```sql
-- Find all employees ordered by salary
SELECT *
from employee
ORDER BY salary DESC;
```

emp_id	first_name	last_name	birth_day	sex	salary	super_id	branch_id
100	David	Wallace	1967-11-17	M	250000	NULL	1
101	Jan	Levinson	1961-05-11	F	110000	100	1
106	Josh	Porter	1969-09-05	M	78000	100	3
102	Michael	Scott	1964-03-15	M	75000	100	2
108	Jim	Halpert	1978-10-01	M	71000	106	3
105	Stanley	Hudson	1958-02-19	M	69000	102	2
107	Andy	Bernard	1973-07-22	M	65000	106	3
103	Angela	Martin	1971-06-25	F	63000	102	2
104	Kelly	Kapoor	1980-02-05	F	55000	102	2

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

emp_id	first_name	last_name	birth_day	sex	salary	super_id	branch_id
103	Angela	Martin	1971-06-25	F	63000	102	2
101	Jan	Levinson	1961-05-11	F	110000	100	1
104	Kelly	Kapoor	1980-02-05	F	55000	102	2
107	Andy	Bernard	1973-07-22	M	65000	106	3
100	David	Wallace	1967-11-17	M	250000	NULL	1
108	Jim	Halpert	1978-10-01	M	71000	106	3
106	Josh	Porter	1969-09-05	M	78000	100	3
102	Michael	Scott	1964-03-15	M	75000	100	2
105	Stanley	Hudson	1958-02-19	M	69000	102	2

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

emp_id	first_name	last_name	birth_day	sex	salary	super_id	branch_id
100	David	Wallace	1967-11-17	M	250000	NULL	1
101	Jan	Levinson	1961-05-11	F	110000	100	1
102	Michael	Scott	1964-03-15	M	75000	100	2
103	Angela	Martin	1971-06-25	F	63000	102	2
104	Kelly	Kapoor	1980-02-05	F	55000	102	2

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

first_name	last_name
David	Wallace
Jan	Levinson
Michael	Scott
Angela	Martin
Kelly	Kapoor
Stanley	Hudson
Josh	Porter
Andy	Bernard
Jim	Halpert

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

forename	surname
David	Wallace
Jan	Levinson
Michael	Scott
Angela	Martin
Kelly	Kapoor
Stanley	Hudson
Josh	Porter
Andy	Bernard
Jim	Halpert

Changes names of columns when we print!

- Calling the Columns forename and surname

```
-- Find out all the different genders
SELECT DISTINCT sex
FROM employee;
```

Will return different sexes

sex
M
F

★ DISTINCT ★

- What are the different values stored in a column

```
-- Find out all the different branch ids
SELECT DISTINCT branch_id
FROM employee;
```

branch_id
1
2
3

Functions

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

"Special SQL function"

COUNT(emp_id)
9

COUNT() function

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

~ Will count rows with Values in them!

COUNT(super_id)
8

Since Null under David Column

```
-- Find the number of female employees born after 1970
SELECT COUNT(emp_id)
FROM employee
WHERE SEX = 'F' AND birth_day > '1970-01-01';
```

COUNT(emp_id)
2

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

AVG(salary)
92888.8889

AVG() function

```
-- Find the average of all employee's salaries
SELECT AVG(salary)
FROM employee
WHERE SEX = 'M';
```

AVG(salary)
101333.3333

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

SUM(salary)
836000

SUM( ) function

Aggregation

- Use function to display data in more useful way!

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

Always use to "group"

COUNT(sex)	sex
6	M
3	F

GROUP BY

★ w/o , sex ★

```
SELECT COUNT(sex)
FROM employee
GROUP BY sex;
```

→

COUNT(sex)
6
3

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

print out

SUM(total_sales)	emp_id
282000	102
218000	105
31000	107
34500	108

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

★ Can add up total sales and group them by client_id ★

SUM(total_sales)	client_id
55000	400
267000	401
22500	402
17000	403
33000	404
26000	405
145000	406

How much each client spent!?

```
SELECT SUM(salary), branch_id
FROM employee
GROUP BY branch_id;
```

→

SUM(salary)	branch_id
360000	1
262000	2
214000	3

~ IASIP

Wildcards

Grab data that matches pattern!

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

- % and _
- and LIKE keyword, use with wildcards

- Find Something ★

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

~ selecting whole table

in the '' We include a pattern. Use two special characters

How it read % or _

client_id	client_name	branch_id
403	John Daly Law, LLC	3

-If any number of characters and then ends with 'LLC'

★ one % before the characters means it's look for that exact character at the end!

if the client_name is like this pattern, return it! if at end!

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

branch_id	supplier_name	supply_type
2	J.T. Forms & Labels	Custom Forms

% at beginning and end of characters, it will search the whole table and not just the end!

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

client_id	client_name	branch_id
400	Dunmore Highschool	2

Find Employee born in October

"reps one character"

```
Select *
FROM employee
WHERE birth_day LIKE '____-10%';
```

Year Month day
2020 10 23 ★

- 4 underscores, cause the year doesn't matter, then 10 (For October) then %. ★

Using _ Example

→ emp_id first_name last_name ....
108 Jim Halpert ....

Searching application - real life Example of Wildcards

Union ~ Combine results of multiple select statements into one!

one single SQL query

```
-- 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;
```

Changed name of returned column as this.

-return in single Column

Employee_Branch_Names
David
Jan
Michael
Angela
Kelly
Stanley
Josh
Andy
Jim
Corporate
Scranton
Stamford

```
SELECT first_name
FROM employee;

SELECT branch_name
FROM branch;
```

- We can combine these statements?

Rules for Unions

1. Have to have the same # of Columns that your setting in each select statement
2. Similar Data Type (Both strings)

```
SELECT employee.first_name, last_name
FROM employee
UNION
SELECT branch_name
FROM branch;
```

have 2 columns here instead of 1

→ Failed

```
-- Find a list of all clients & branch suppliers' names
SELECT client_name
FROM client
UNION
SELECT supplier_name
FROM branch_supplier;
```

client_name
Dunmore Highschool
Lackawana Country
FedEx
John Daly Law, LLC
Scranton Whitepages
Times Newspaper
Hammer Mill
J.T. Forms & Labels
Uni-ball
Patriot Paper
Stamford Lables

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

- Branches their associated with?

client_name	branch_id
Dunmore Highschool	2
Lackawana Country	2
FedEx	3
John Daly Law, LLC	3
Scranton Whitepages	2
Times Newspaper	3
FedEx	2
Hammer Mill	2
J.T. Forms & Labels	2
Uni-ball	2
Hammer Mill	3
Patriot Paper	3
Stamford Lables	3

```
-- Find a list of all money spent or earned by the company
SELECT salary
FROM employee
UNION
SELECT total_sales
FROM works_with;
```

salary
250000
110000
75000
63000
55000
69000
78000
65000
71000
267000
15000
33000
130000
. . .
. . . .
. . . .
. .

← if Ucs 17 rows, didn't print all of them

Simple Examples

Joins Combine rows from 2 or more tables based on a related column between them so joins can be really useful for combing information different tables into a single result which we can then use to you know obviously find out specific information that is stored in our database,

"if you want to combine information"

To Learn joins were going to insert ↴

```
INSERT INTO branch VALUES(4, "Buffalo", NULL, NULL);
```

The branch is then updated to include the Buffalo 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
4	Buffalo	NULL	NULL

new branch →

Join Example

```
-- Find all branches and the names of their managers
SELECT employee.emp_id, employee.first_name, branch.branch_name
FROM employee
JOIN branch
ON employee.emp_id = branch.mgr_id;
```

Grab info from Employee tab.

since were 'joining' we grab a column from the branch table"

↳ Want to combine as long the emp_id = mgr_id

on Both → Employee table and Branch table, they each have a column that stores employee_id (Emp_id & mgr_id)

- "Column that is shared"
- A join is used to combine rows based on related columns ?

Going to join the Employee table and Branch table together! on a specific Column

- Column that they have in column.

Results ↳

emp_id	first_name	branch_name
100	David	Corporate
102	Michael	Scranton
106	Josh	Stamford

~ Combined rows from branch table into the Employee table into one table.

* 4 Types of Join. *

1. General Join ~ refer to as a inner join. (JOIN) ~ Key word

   a inner join is going to combine rows from the employee table and the branch table whenever they have the shared column in column.

2. LEFT JOIN

   - all employees got included, Why, with a LEFT JOIN, we include all of the rows from the left table, so in the example, the left table is the employee table (included in FROM statement).

Example on next PAGE!

3. RIGTH JOIN

   - all rows from the right table (which is branches in our example)
   - opposed of LEFT JOIN

4. FULL OUTER JOIN

   - Can't doin mySQL ☹ Basically a Left and right join combined ?

LEFT JOIN

```
SELECT employee.emp_id, employee.first_name, branch.branch_name
FROM employee
LEFT JOIN branch
ON employee.emp_id = branch.mgr_id;
```

emp_id	first_name	branch_name
100	David	Corporate
101	Jan	NULL
102	Michael	Scranton
103	Angela	NULL
104	Kelly	NULL
105	Stanley	NULL
106	Josh	Stamford
107	Andy	NULL
108	Jim	NULL

- With a LEFT JOIN, we include all the rows from the left table. In our problem, the left table was the employee table (Table included in FROM statement)

We got all the employees, instead of just David, Michael, Josh
- all employees set included, not just branch managers

RIGHT JOIN

```
SELECT employee.emp_id, employee.first_name, branch.branch_name
FROM employee
RIGHT JOIN branch
ON employee.emp_id = branch.mgr_id;
```

~ opposite of LEFT JOIN

emp_id	first_name	branch_name
100	David	Corporate
102	Michael	Scranton
106	Josh	Stamford
NULL	NULL	Buffalo

← All of the rows from the right table!

Nested Queries

- Using multiple SELECT statements to get specific set of information
- More advanced query writing
- Very common !

```
-- Find names of all employees who have sold over
-- $30,000 to a single client
SELECT works_with.emp_id
FROM works_with
WHERE works_with.total_sales > 30000;
```

emp_id
102
105
105
105

Returned →

"select" I want the emp-id from the Works_with table where the total_sales are greater than $30,000 ?

Simple Nested Example

~ find name of all employee who have sold over $30,000 to a single client

```
SELECT employee.first_name, employee.last_name
FROM employee
WHERE employee.emp_id IN (
    SELECT works_with.emp_id
    FROM works_with
    WHERE works_with.total_sales > 30000
);
```

Formated → indented

~ nested query

"inner is ran first"

first_name	last_name
Michael	Scott
Stanley	Hudson

- remember the "IN" Keyword is going us a result if the employee id is in values that we specify inside of these parentheses.

I want to get the first and last name from the employee table where the employee ID is in the result of that query!

Another Example

```
-- Find all clients who are handled by the branch that Michael Scott manages
-- Assume you know Michael's ID
```

```
SELECT branch.branch_id
FROM branch
WHERE branch.mgr_id = 102;
```

branch_id
2

We want to get the branch_id from the branch table where the ~~mgr~~ mgr_id = 102

- Branch Michael Scott manages.

Know we want to get all the clients handle by the branch.

```
SELECT client.client_name
FROM client
WHERE client.branch_id = (
    SELECT branch.branch_id
    FROM branch
    WHERE branch.mgr_id = 102
);
```

~ I want the client name from the client table where the branch_id

client_name
Dunmore Highschool
Lackawana Country
Scranton Whitepages
FedEx

- All clients managed by the scranta branch?

```sql
SELECT client.client_name
FROM client
WHERE client.branch_id = (
    SELECT branch.branch_id
    FROM branch
    WHERE branch.mgr_id = 102
    LIMIT 1
);
```

client_name
Dunmore Highschool
Lackawana Country
Scranton Whitepages
FedEx

~ this statement isn't necessarily guaranteed to return only one value?
So if Michael Scott was the manager at like multiple branches it's possible this statement would return multiple values?

This will make sure we only get one returned

Some thing.

Use results from one query and use it in another!

```sql
-- Find all clients who are handles by the branch that Michael Scott manages
-- Assume you DON'T know Michael's ID
SELECT client.client_id, client.client_name
FROM client
WHERE client.branch_id = (SELECT branch.branch_id
                          FROM branch
                          WHERE branch.mgr_id = (SELECT employee.emp_id
                                                 FROM employee
                                                 WHERE employee.first_name = 'Michael' AND employee.last_name ='Scott'
                                                 LIMIT 1));
```

client_id	client_name
400	Dunmore Highschool
401	Lackawana Country
404	Scranton Whitepages
406	FedEx

Inner Inner

I want the emp-id from the Employee table, where name = to Michael Scott

→ return emp-id

102

Inner loop

I want the branch_id from the branch table, where mgr-id equals our inner inner loop which is 102?

- returned branch-id

2

I want the client_id and client_name from the client table where branch_id equals the inner loop, which value returned was 2?

```sql
-- Find the names of employees who work with clients handled by the scranton branch
SELECT employee.first_name, employee.last_name
FROM employee
WHERE employee.emp_id IN (
                          SELECT works_with.emp_id
                          FROM works_with
                          )
AND employee.branch_id = 2;
```

Conditional statement

returns emp-id from works-with table

emp-id
105
102
108
107
108
105
107
102
105

first_name	last_name
Michael	Scott
Stanley	Hudson

I want the first and last name from the Employee table where emp-id is In our inner nest And ~~Emp~~ branch id is equal to 2.

Another Example

```
-- Find the names of all clients who have spent more than 100,000 dollars
SELECT client.client_name
FROM client
WHERE client.client_id IN (
                    SELECT client_id
                    FROM (
                        SELECT SUM(works_with.total_sales) AS totals, client_id
                        FROM works_with
                        GROUP BY client_id) AS total_client_sales
                    WHERE totals > 100000

);
```

client_name
Lackawana Country
FedEx

I want the sum of total_sales from the works_with table As totals, and client_id from the works_with table, Group them by client_id

Returned

totals	client_id
55000	400
267000	401
22500	402
17000	403
33000	404
26000	405
145000	406

I want the client_id from inner nest written as total_client_sales, where total > 100,000

returned

total_client_sales
401
406

outer Nested Query

- I want client_name from the client table, where the client_id is In the nested loops, which is 401, 406 and we get their names!

On Delete

- Deleting entries in the database when they have foreign keys associated to them

Scenario, what would happen if Michael Scott was deleted from Employee table?

- We delete the employee with ID 102 (Michael Scott), what's going to happen to the manager ID (mgr_id)?
- The manager ID is supposed to be linking us to an actual row in the employee table but if we delete Michael Scott then all of a sudden 102 doesn't mean anything, Michael Scott is gone. His employee ID (emp_id) is no longer inside our employee table

There are two things we can do when this situation occurs:

1. First thing would be **on delete set null,** which is basically where if we delete one of these employees that means that the manager ID (mgr_id) that was associated to that employee is going to get set to **NULL**.

2. The second thing would be **on delete cascade** is essentially where if we delete the employee whose ID is stored in the manager ID (mgr_id) column then we're just going to delete the entire row in the database (in the table **Branch**).

ON DELETE SET NULL

```
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
);
```

If the **emp_id** in the **employee** table gets deleted I want to set the **mgr_id** equal to **NULL**

Example:

```
DELETE FROM employee
WHERE emp_id = 102;
```

Success

Rows affected: 1

What happened inside of the **branch** table?

```
SELECT * from branch;
```

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

- We can see that the **mgr_id** is now set to **NULL** and that's because we deleted Michael Scott (102 entry) so the manger ID which was storing that as a foreign key is just going to be set to NULL, that's because we defined it when we created the **branch** table.

Let's look at the employee table:

```
SELECT * from employee;
```

emp_id	first_name	last_name	birth_day	sex	salary	super_id	branch_id
100	David	Wallace	1967-11-17	M	250000	NULL	1
101	Jan	Levinson	1961-05-11	F	110000	100	1
103	Angela	Martin	1971-06-25	F	63000	NULL	2
104	Kelly	Kapoor	1980-02-05	F	55000	NULL	2
105	Stanley	Hudson	1958-02-19	M	69000	NULL	2
106	Josh	Porter	1969-09-05	M	78000	100	3
107	Andy	Bernard	1973-07-22	M	65000	106	3
108	Jim	Halpert	1978-10-01	M	71000	106	3

- **super_id** set to **NULL** in many of the rows
- when we created the **employee** table, the **super_id** also had **ON DELETE SET NULL**

ON DELETE SET 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
);
```

If the **branch_id** that's stored as the foreign key and the branch supplier table gets deleted then we're just going to delete the entire row in the database.

- If we were to delete **branch_id** 2 then all of the rows (in the **Branch Supplier table**) that had branch would just get deleted!

Example:

```
DELETE FROM branch
WHERE branch_id = 2;
```

Let's see what happened to **the Branch Supplier** table

```
SELECT * from branch_supplier;
```

branch_id	supplier_name	supply_type
3	Hammer Mill	Paper
3	Patriot Paper	Paper
3	Stamford Lables	Custom Forms
3	Uni-ball	Writing Utensils

- No longer no branch_id 2
- This is what **ON DELETE CASCADE** does

What situations where we might use them:

ON DELETE SET NULL

```
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
);
```

- In the branch table we used **ON DELETE SET NULL** because the **mgr_id** on the branch table is just a **foreign key** and not **a primary key**

- The **mgr_id** is absolutely essential for the branch table !

ON DELETE SET 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
);
```

Referring to the Branch Supplier Table, if the **branch_id** (foreign key here, which is also part of the primary key) which means the **branch_id** on the Branch Supplier table is absolutely crucial for this row in the database. If the **branch_id** disappears we can't set it to **NULL** because a primary key can't have a NULL value, we just have to delete the entire thing. That's why we use **ON DELETE CASCADE** instead of **ON DELETE SET NULL**.

Note:

If you have a situation like **branch_supplier** table, where a foreign key is also a primary key (or also a component of primary key) then it always has to be **ON DELETE CASCADE** otherwise you're going to run into trouble.

Triggers

- Block of SQL code we can write that we can write, which will define a certain action that should happen when a certain operation gets performed on the database

```
CREATE TABLE trigger_test (
    message VARCHAR(100)
);
```

We're going to have to use the Command (terminal) line because we're going to have to change the SQL delimiter

Might have to log in (in the terminal)
Password: password

```
(base) Devins-MBP-2:~ devinpowers$ mysql -u root -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 11
Server version: 8.0.20 MySQL Community Server - GPL

Copyright (c) 2000, 2020, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql>
```

Once were in here, were going to want to use the database (giraffe) that we created for this tutorial. (**use <insert database>**)

```
mysql> use giraffe
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
```

Examples

```
DELIMITER $$
CREATE
    TRIGGER my_trigger BEFORE INSERT
    ON employee
    FOR EACH ROW BEGIN
        INSERT INTO trigger_test VALUES('added new employee');
    END$$
DELIMITER ;
```

What does all this mean?

- Defining a trigger, giving it a name and saying before something gets inserted on the employee table, for each of the new items getting inserted, I want to insert into the trigger_test table the VALUES 'added new employee'.
- Useful for automating things

The reason we have to do it in the terminal, is because PopSQL you can't actually configure the delimiter, so the delimiter is something that's defined not on the like text editor level it's defined like over here.

In the Terminal:

```
mysql> DELIMITER $$
```

Now were going to paste in the actual part where we're creating the trigger

```
mysql> CREATE
    ->     TRIGGER my_trigger BEFORE INSERT
    ->     ON employee
    ->     FOR EACH ROW BEGIN
    ->         INSERT INTO trigger_test VALUES('added new employee');
    ->     END$$
```

Then finally were going to change the delimiter back to a semicolon

```
mysql> DELIMITER ;
```

So hopefully now this trigger is all set up inside of my sequel so one thing we can do to test it, is just to add in another employee (in PopSQL):

```
INSERT INTO employee
VALUES(109, 'Oscar', 'Martinez', '1968-02-19', 'M', 69000, 106, 3);
```

```
SELECT * FROM trigger_test;
```

message
added new employee

Looks like the trigger got set up correctly

Other cool things we can do with Triggers

```
DELIMITER $$
CREATE
    TRIGGER my_trigger BEFORE INSERT
    ON employee
    FOR EACH ROW BEGIN
        INSERT INTO trigger_test VALUES(NEW.first_name);
    END$$
DELIMITER ;
```

- Allowing us to access a particular attribute about the thing that we just inserted
- **NEW** is going to refer to the row that's getting inserted and then we can access specific columns from that row
- **NEW.first_name** will give us the first name of the employee that's getting inserted

We have to update the Trigger in the terminal again:

```
mysql> DELIMITER $$
```

Then paste all the code

```
mysql> CREATE
    ->     TRIGGER my_trigger1 BEFORE INSERT
    ->     ON employee
    ->     FOR EACH ROW BEGIN
    ->         INSERT INTO trigger_test VALUES(NEW.first_name);
    ->     END$$
```

Then change the delimiter back to a semicolon

```
mysql> DELIMITER ;
```

Let's add in Kevin now:

```
INSERT INTO employee
VALUES(110, 'Kevin', 'Malone', '1978-02-19', 'M', 69000, 106, 3);

SELECT * FROM trigger_test;
```

message
added new employee
added new employee
Kevin

More Complex Trigger

- Using conditional

```
DELIMITER $$
CREATE
    TRIGGER my_trigger BEFORE INSERT
    ON employee
    FOR EACH ROW BEGIN
        IF NEW.sex = 'M' THEN
            INSERT INTO trigger_test VALUES('added male employee');
        ELSEIF NEW.sex = 'F' THEN
            INSERT INTO trigger_test VALUES('added female');
        ELSE
            INSERT INTO trigger_test VALUES('added other employee');
        END IF;
    END$$
DELIMITER ;
```

```
mysql> DELIMITER $$
```

```
mysql> CREATE
    ->     TRIGGER my_trigger2 BEFORE INSERT
    ->     ON employee
    ->     FOR EACH ROW BEGIN
    ->         IF NEW.sex = 'M' THEN
    ->             INSERT INTO trigger_test VALUES('added male employee');
    ->         ELSEIF NEW.sex = 'F' THEN
    ->             INSERT INTO trigger_test VALUES('added female');
    ->         ELSE
    ->             INSERT INTO trigger_test VALUES('added other employee');
    ->         END IF;
    ->     END$$
```

Then change delimiter back to a semicolon

```
mysql> DELIMITER ;
```

Let's insert a new employee!

```
INSERT INTO employee
VALUES(111, 'Pam', 'Beesly', '1988-02-19', 'F', 69000, 106, 3);
```

```
SELECT * FROM trigger_test;
```

- All these triggers are going to compound on each other

message
added new employee
added new employee
Kevin
added new employee
Pam
added female

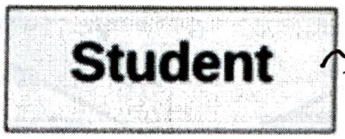
We can also drop a **Trigger**

```
mysql> DROP TRIGGER my_trigger;
```

-Now **my_trigger** is no longer going to be active

Er Diagrams Intro

ER: Enity Relationship

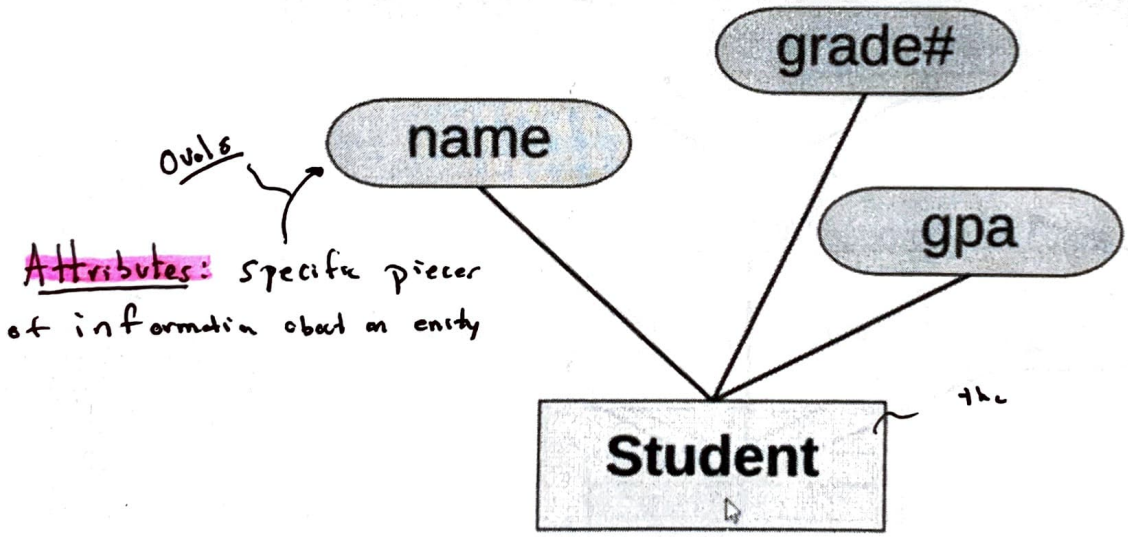


the sq corn reps the Enity!

Enity: An Object we want to model & store information about

Database schema

Working for a School Diagram

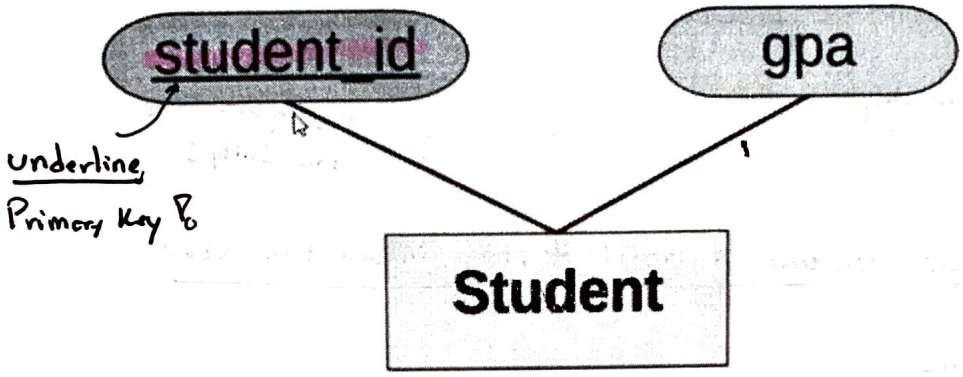


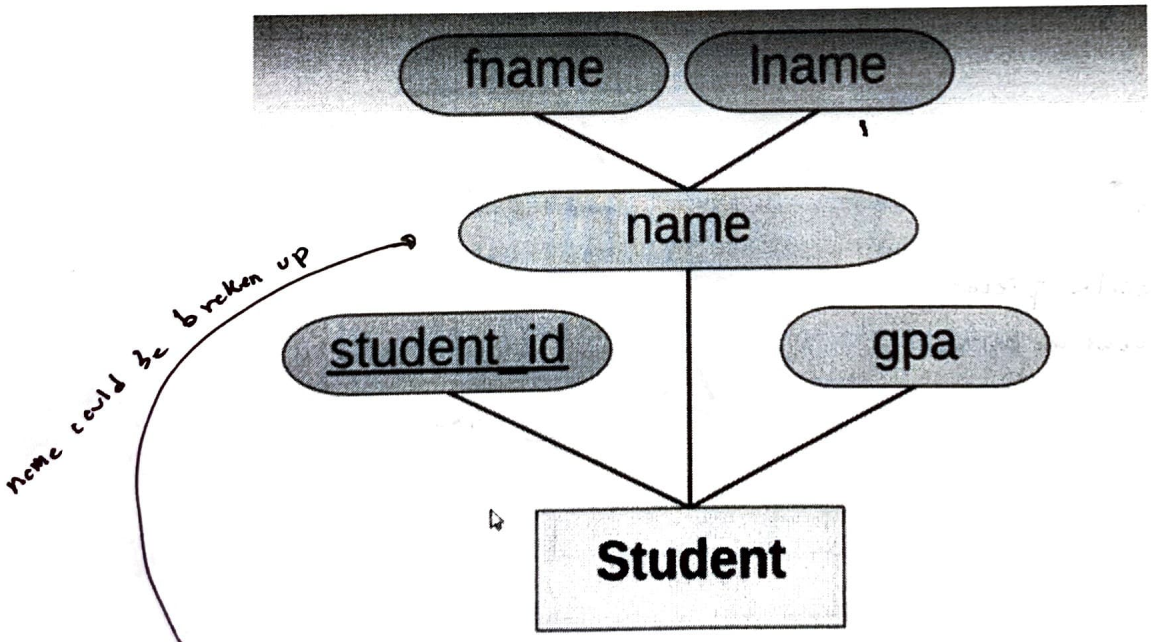
Ovals

Attributes: specific piecer of information about an enity

the

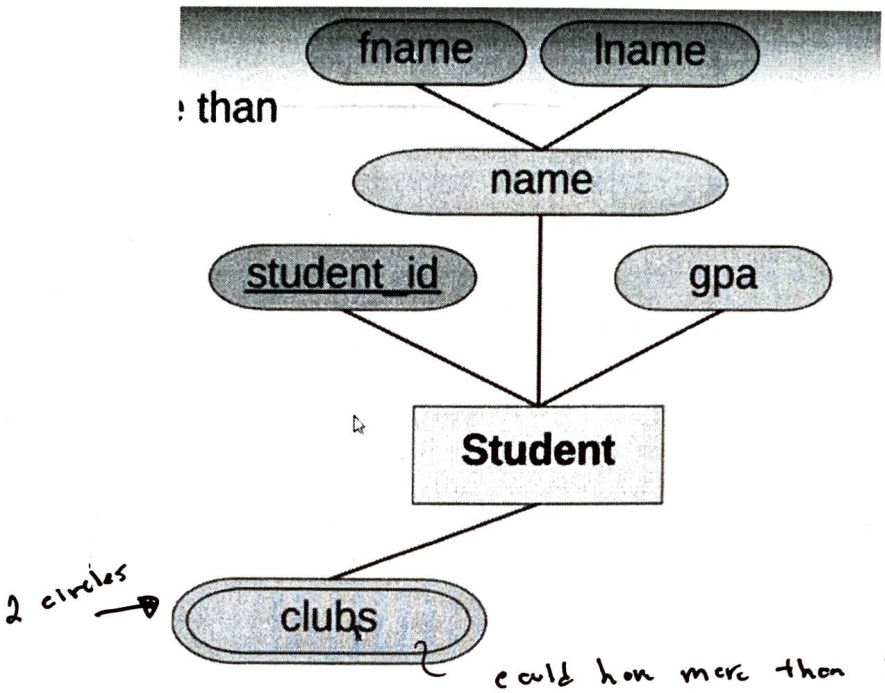
Primary Key: An attribute(s) that uniquely identifies an entry in the database table





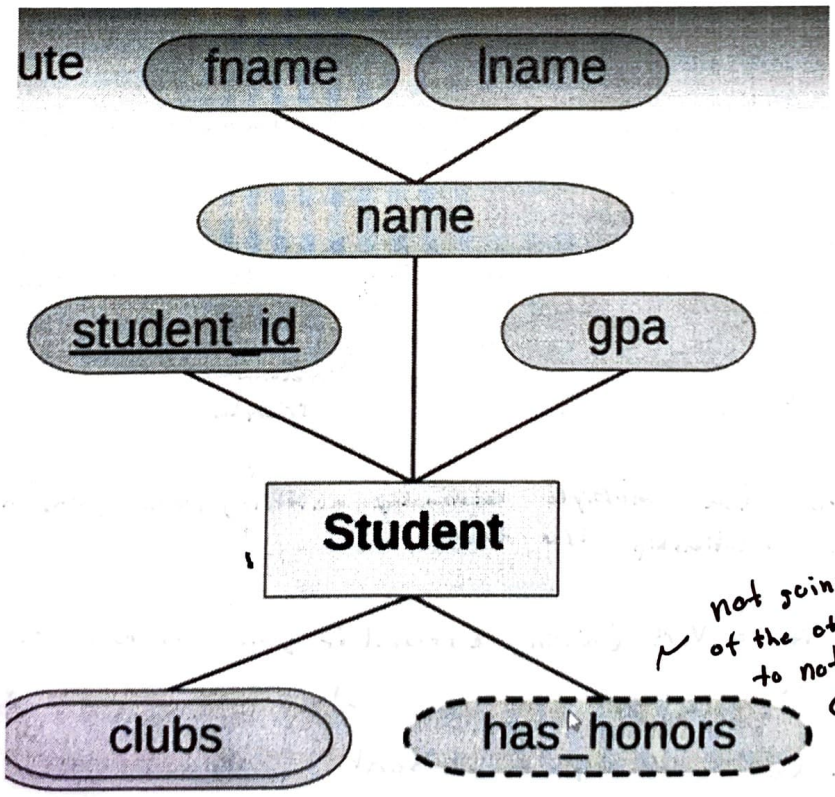
Composite Attributes: An attribute that can be broken up into sub-attributes

Multi-Valued Attribute - An attribute that can have more than one value.



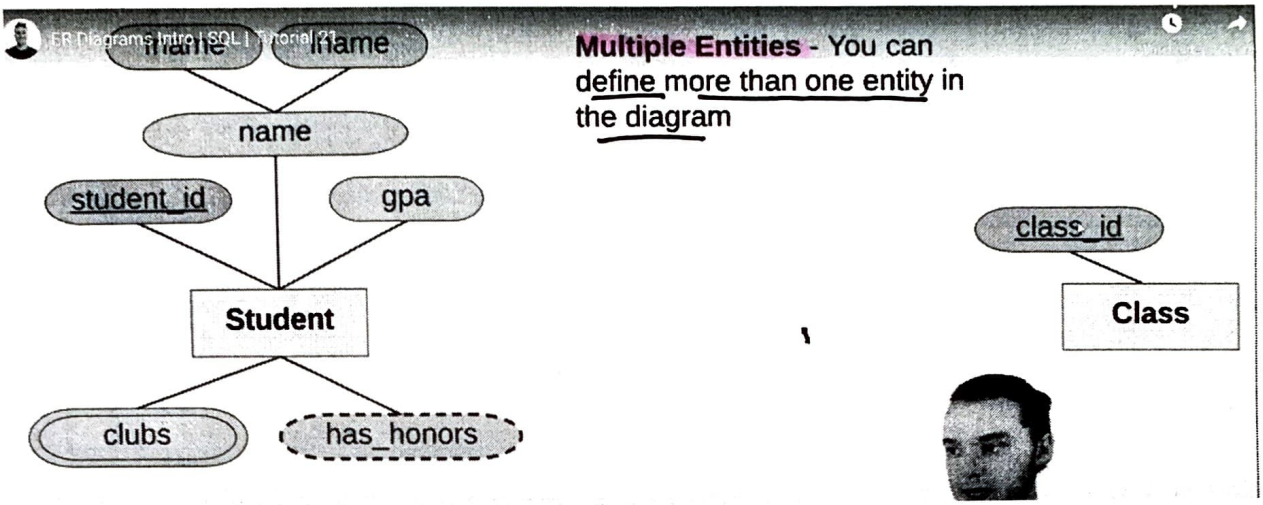
2 circles

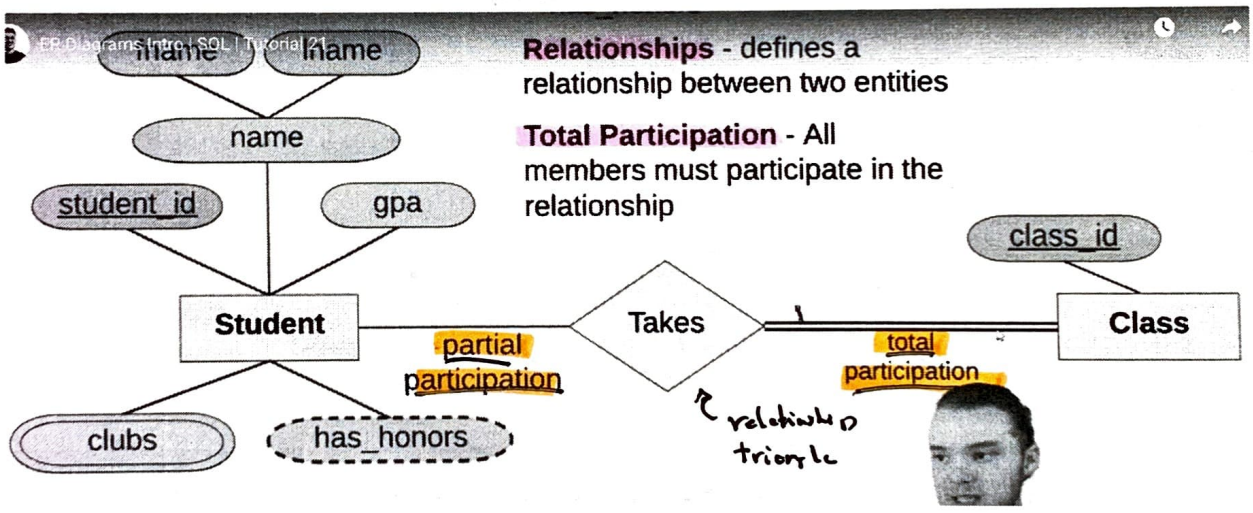
could have more than 1 clubs they belongs to.



not going to keep track of the attribute. Just a way to notate attributes that could be derived.

Derived Attribute - An attribute that can be derived from the other attributes?





- When we have multiple relationship entities, we're going to have to define relationship b/w them.

- relationship is a Verb (Noting a student is going to take a class)

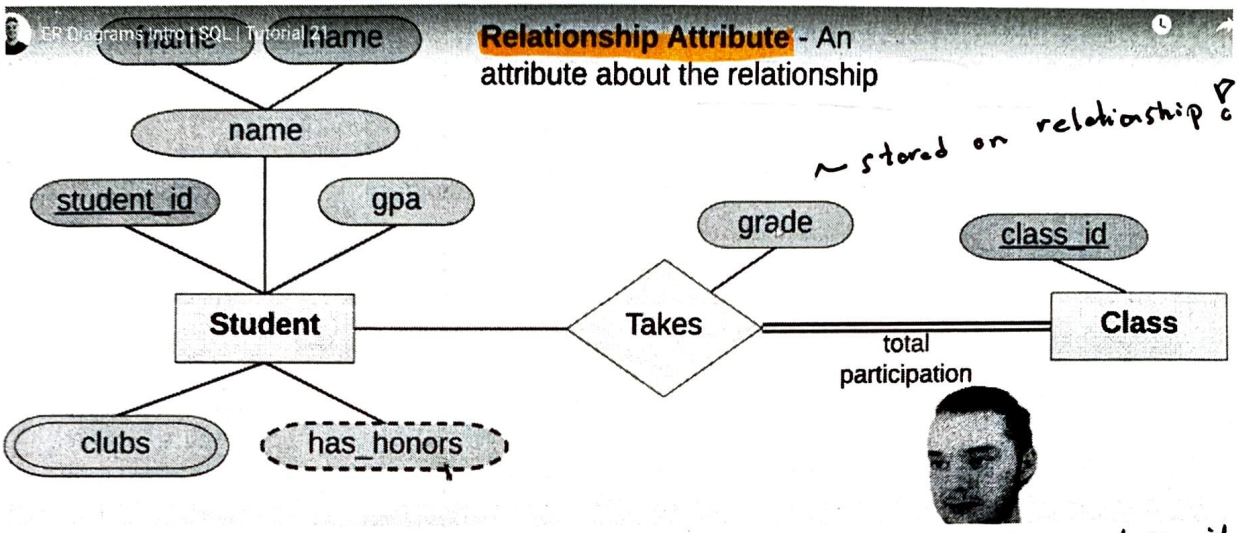
Student → is going to take a class] read it both ways!
A Class can be taken by a student

We can also define participation:

Single line = partial participation ~ indicates (not all students need to take a class)

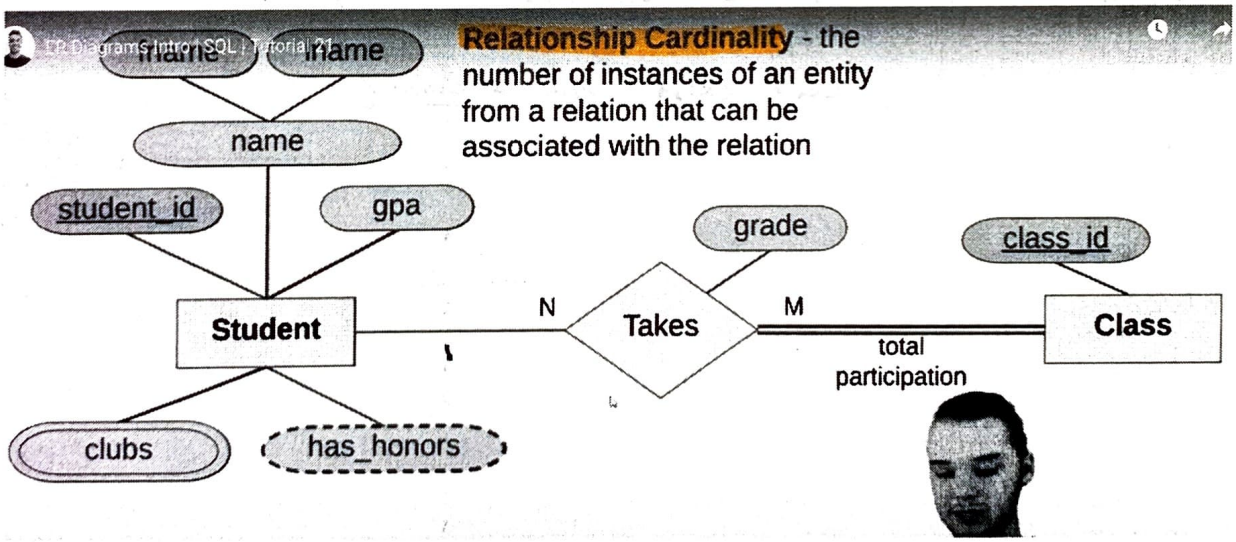
Double line it indicates total participation, which means it, all the classes need to be taken by a single student.

~ All classes have to have a student taking it.



~ stored on relationship?

- The only way I can get a grade from a class is if I can take it?



1 : 1
1 : N
N : M

N, M ~ refers to any #

a student can take any number of classes?

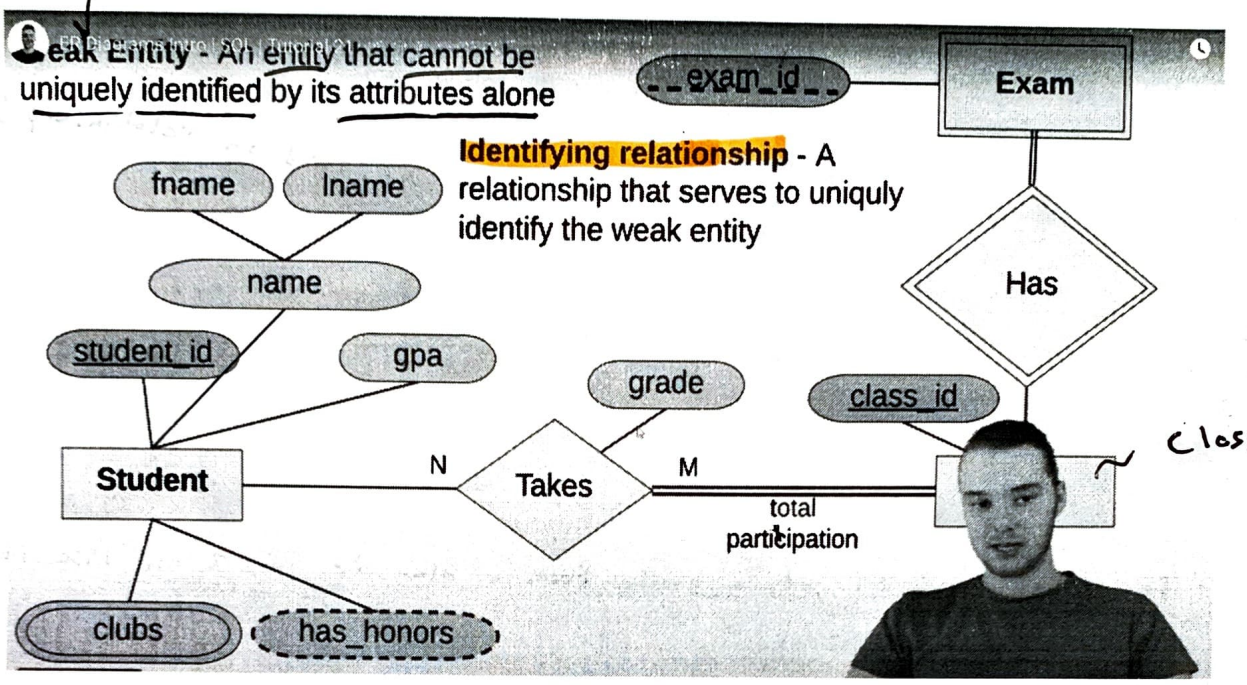
Relationship Cardinality

1 : 1 : a student can take 1 class and a class can be taken by 1 student

1 : N : A student can take 1 class and a class can be taken by any # of student
~ could reverse it?

N : M : A student can take any # of classes and a class can have any # of students taking it?

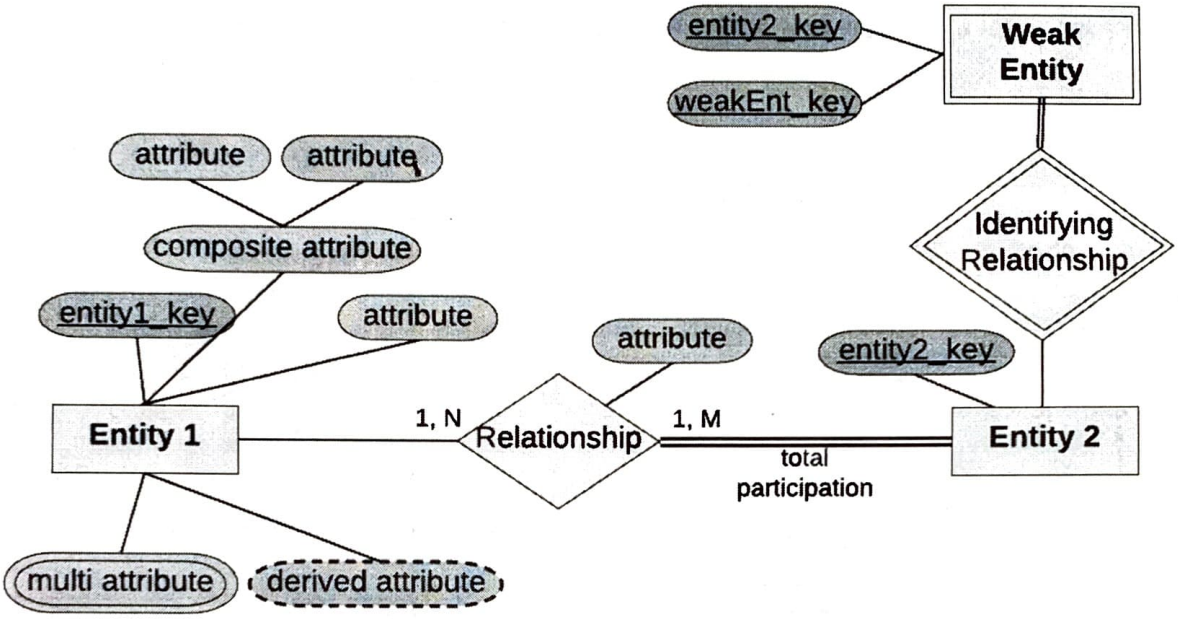
Week Entity



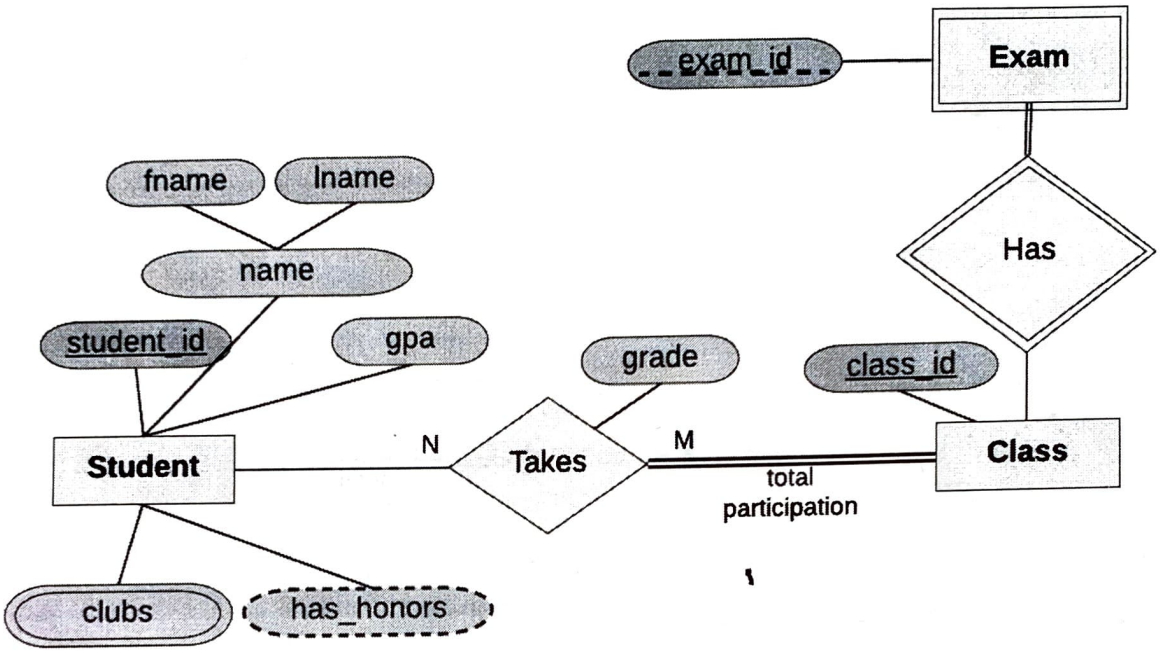
Week Entity: Going to rely on/depend on another entity

Example [Exam] ~ a class can have an Exam, Exam is like an entity, Exam might have an Exam-id, an Exam can't exist w/o a class ∇

ER Diagram Template



Student Diagram



Designing An Er Diagram

Company Data Storage 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.

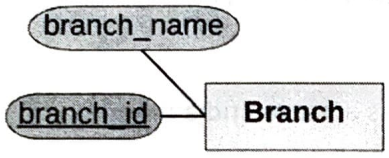
An employee can act as a supervisor for other employees at the branch, an employee may also act as the supervisor for employees at other branches. An employee can have at most one supervisor.

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.

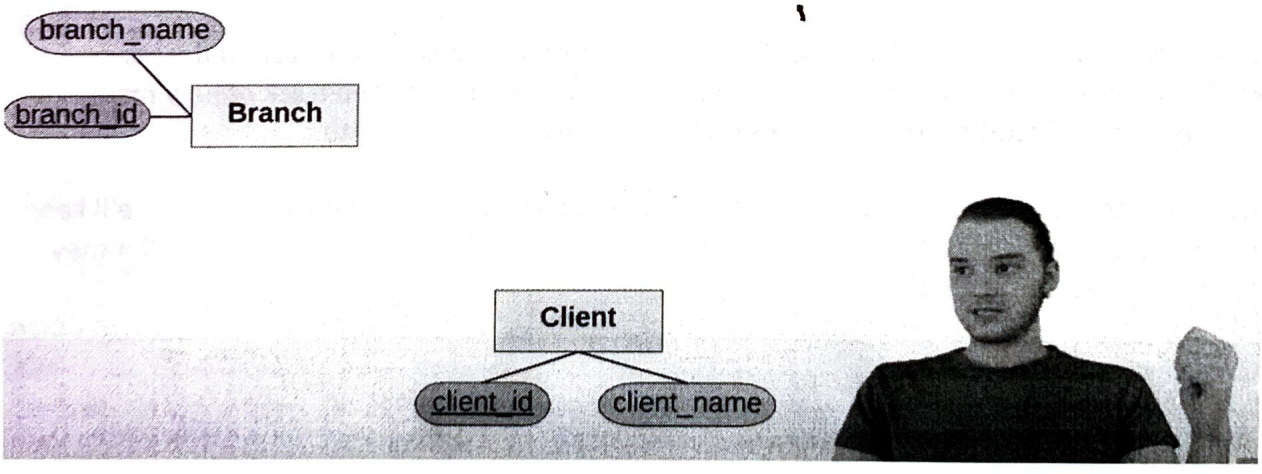
Employees can work with clients controlled by their branch to sell them stuff. If nescessary multiple employees can work with the same client. We'll want to keep track of how many dollars worth of stuff each employee sells to each client they work with.

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.

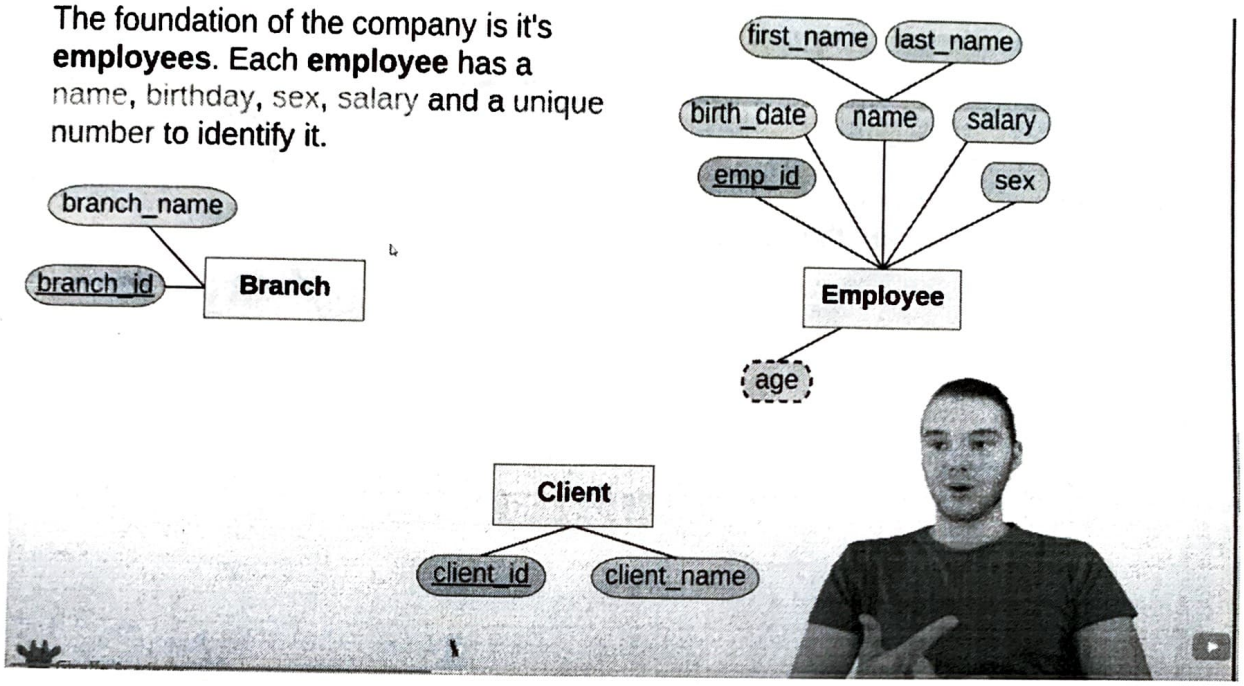
The company is organized into **branches**. Each branch has a unique number, and a name



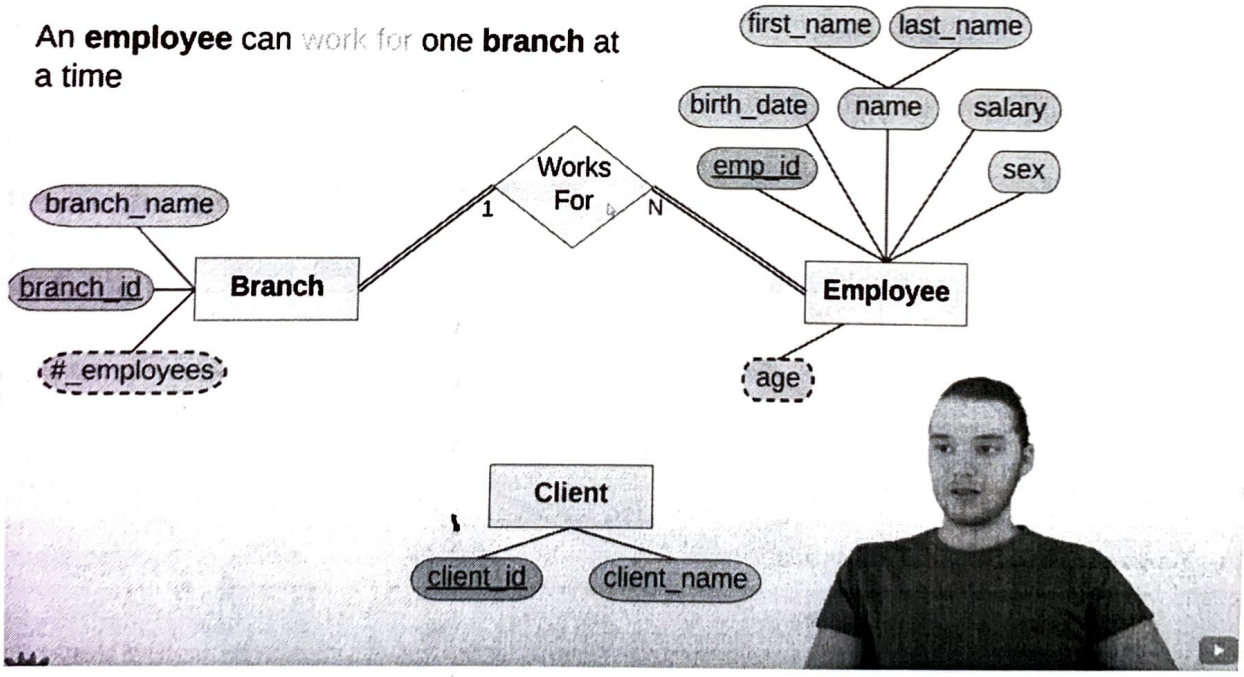
The company makes it's money by selling to **clients**. Each **client** has a name and a unique number to identify it.



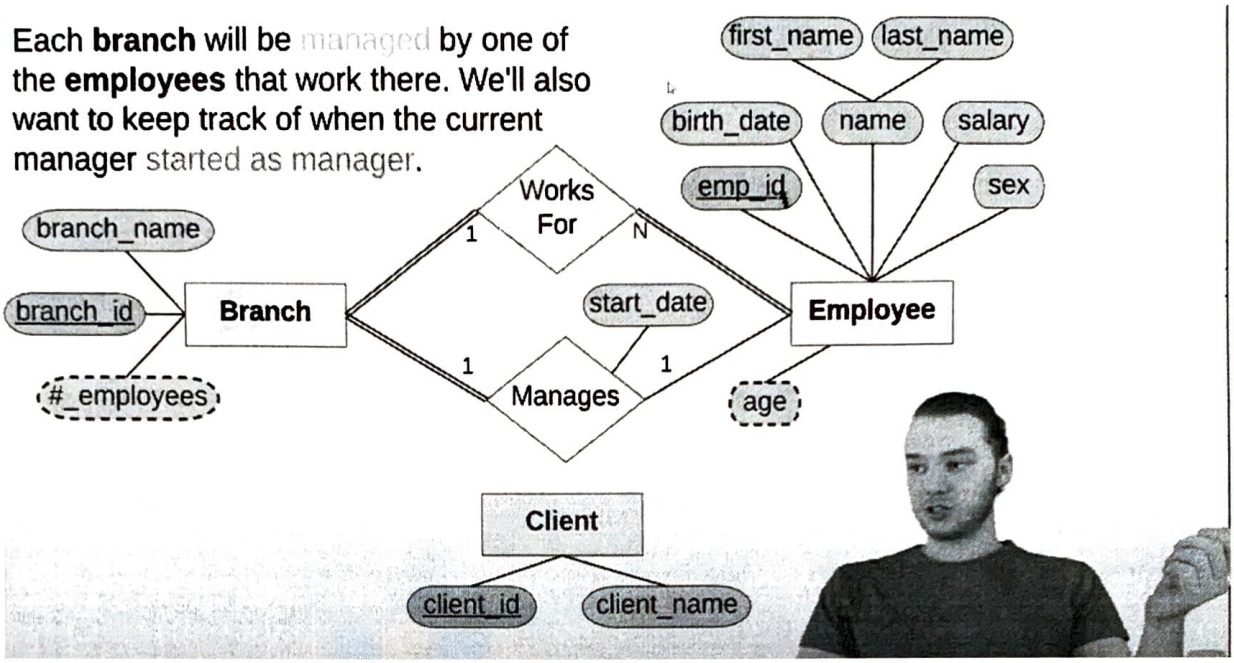
The foundation of the company is it's **employees**. Each **employee** has a name, birthday, sex, salary and a unique number to identify it.



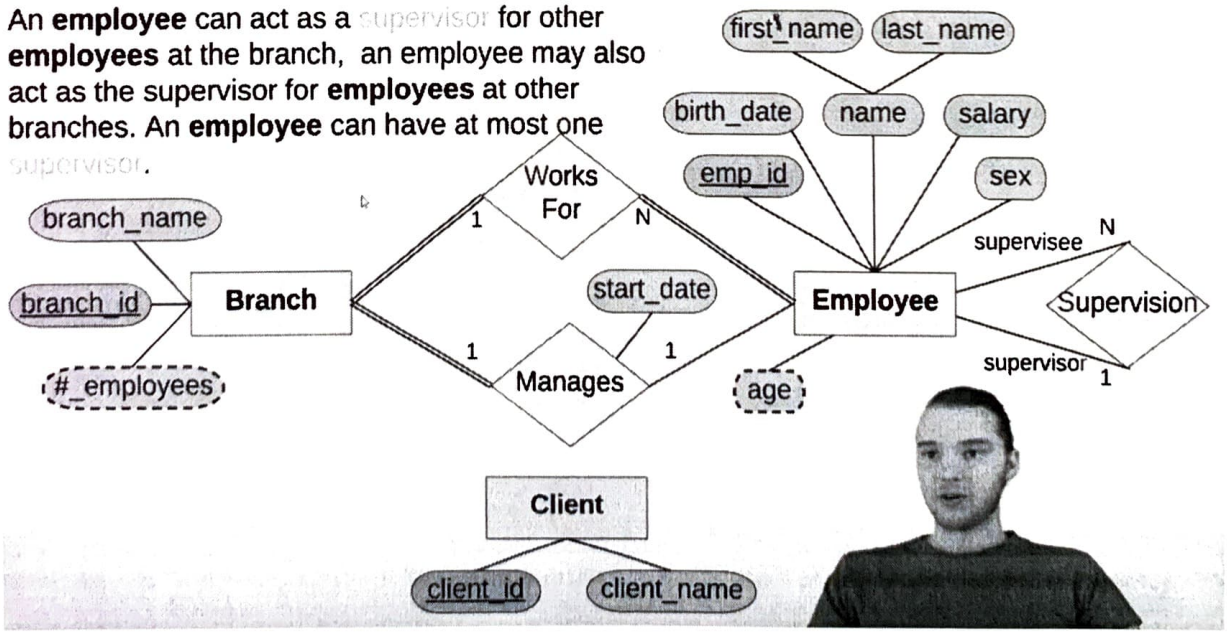
An **employee** can work for one **branch** at a time



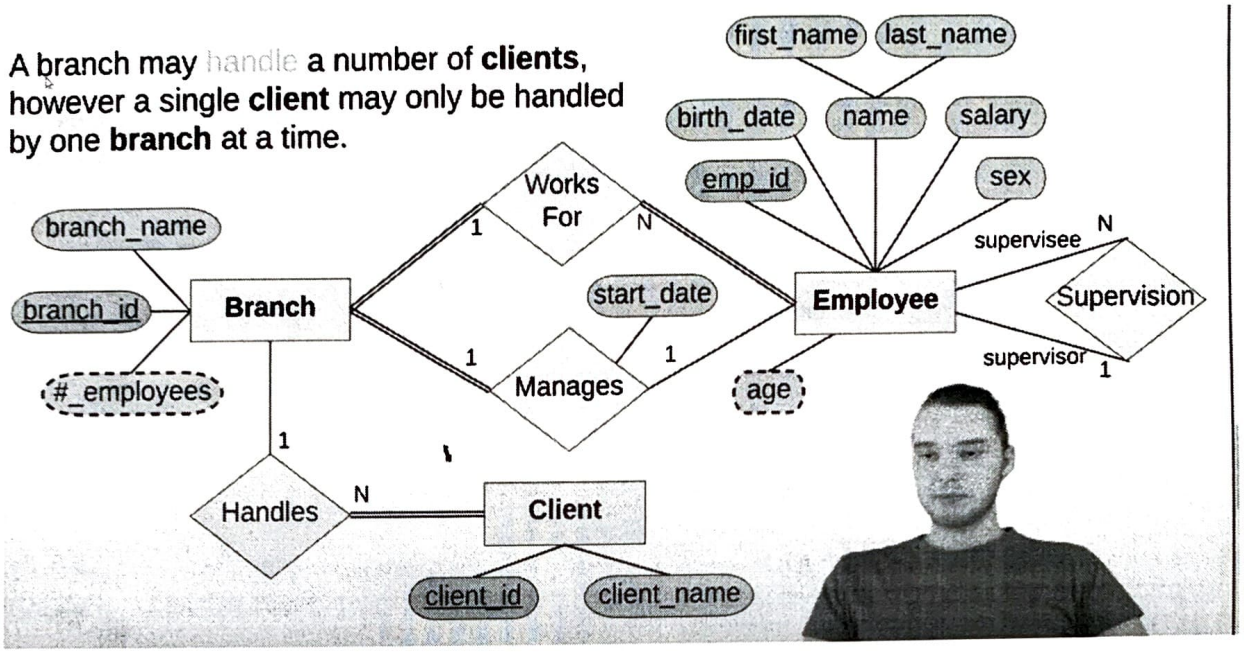
Each **branch** will be managed by one of the **employees** that work there. We'll also want to keep track of when the current **manager** started as manager.



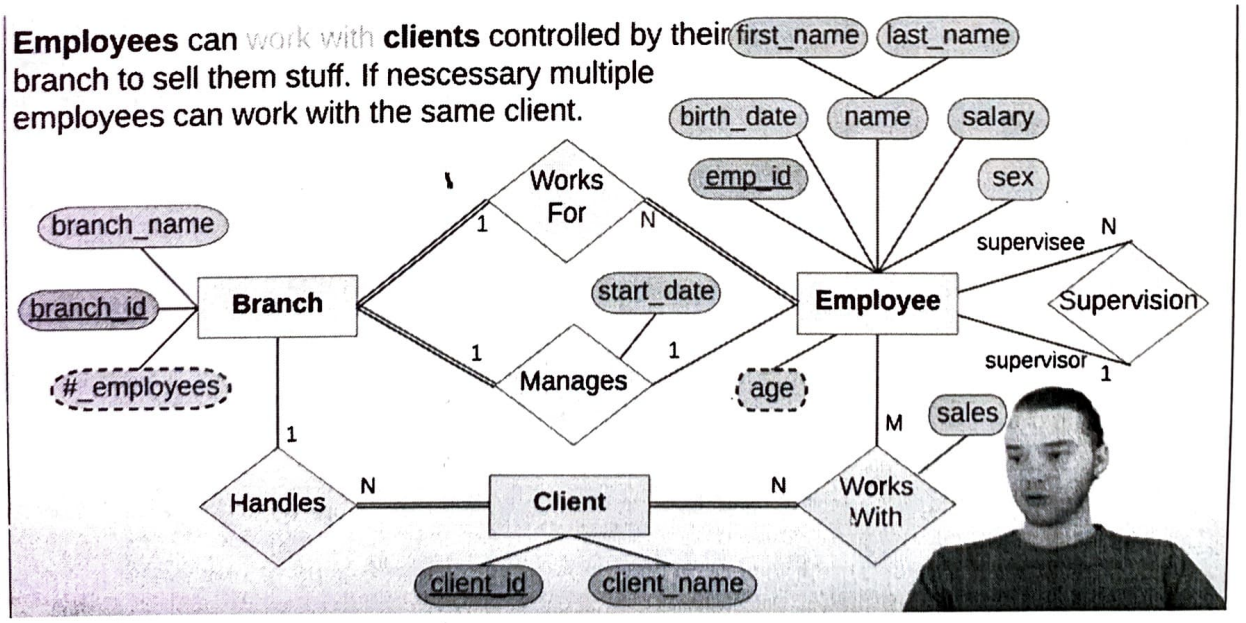
An **employee** can act as a supervisor for other **employees** at the branch, an employee may also act as the supervisor for **employees** at other branches. An **employee** can have at most one supervisor.

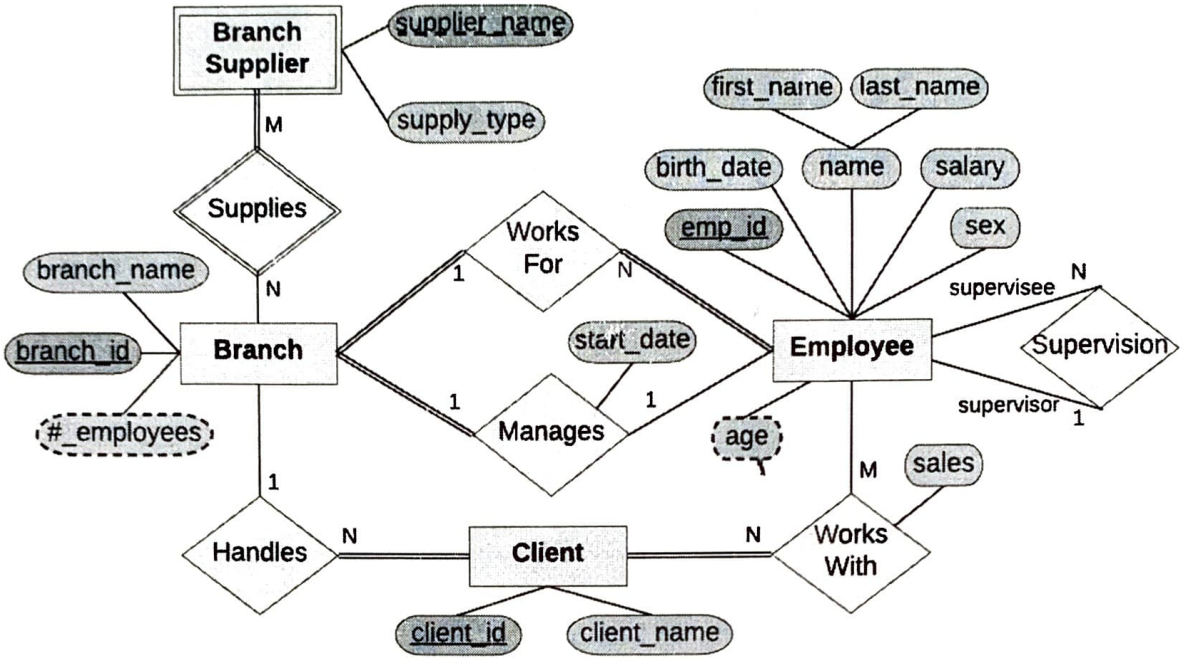


A branch may handle a number of **clients**, however a single **client** may only be handled by one **branch** at a time.



Employees can work with **clients** controlled by their branch to sell them stuff. If nescessary multiple employees can work with the same client.




Branch Supplier
supplier_name
supply_type
M
Supplies
N
branch_name
branch_id
Branch
#_employees
1
Handles
N
Client
client_id
client_name
1
Works For
N
1
Manages
start_date
1
first_name
last_name
birth_date
name
salary
emp_id
sex
supervisee
N
Employee
Supervision
supervisor
1
age
M
sales
Works With
N