SQL Notes

ReadMe--

* Indicates definition

! Indicates important non definition information

----

1 Introduction:

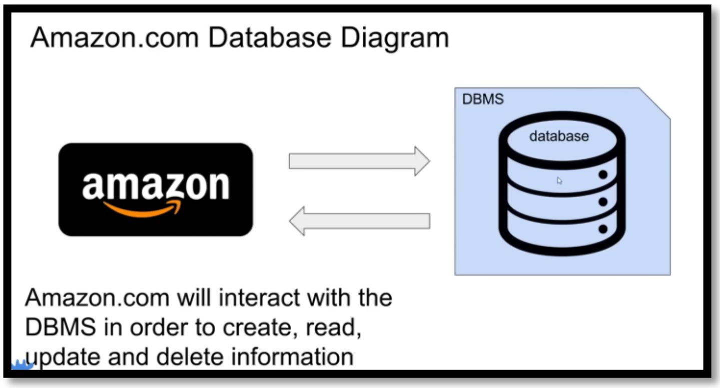
*** Database**:

- Any collection of related information (i.e. phone book, to do list etc). That can be stored (Paper, computer, in your mind etc)

*** Database Management Systems (DBMS)**:

-A special software program that helps users create and maintain a database

-Advantages: Easier to manage large amounts of information, security handling, backups, importing/exporting data, ability to execute more than one program or task simultaneously (concurrency), integrates with other applications (ie programming languages). Example:


Amazon.com Database Diagram

Amazon.com will interact with the DBMS in order to create, read, update and delete information

*** C.R.U.D (C**reate **R**ead **U**pdate **D**elete**)**:

-Represents the four main operations when working with DBMS (**C**reate **R**ead **U**pdate **D**elete).

! Two types of Databases:

1. ***Relational Databases (SQL)**:

- Organize data into one or more tables (each table has columns and rows and unique key identifiers)

***Relational Database Management System (RDBMS)**:

-Help users create and maintain a relational database (mySQL, Oracle etc)

***Structured Query Language (SQL)**:

-Standard language for interacting with RDBMS

-Used to perform C.R.U.D operations

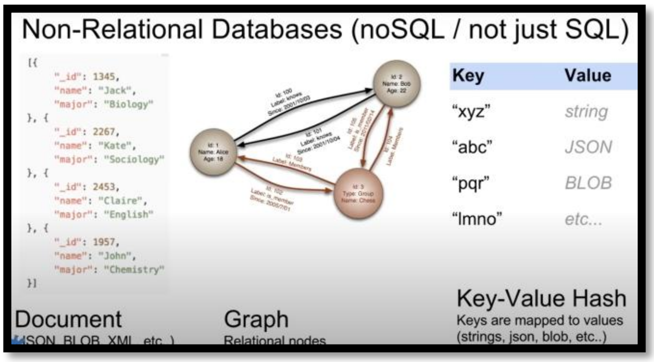
-Used to perform other tasks such as backups, security and management

-Used to define table structures

2. * **Non-Relational Databases (noSQL/ not just SQL)**:

-Organized data that is not in the form of a traditional table (key-value stores, documents, graphs, flexible tables).

Example:



Non-Relational Databases (noSQL / not just SQL)

```
[{
    "_id": 1345,
    "name": "Jack",
    "major": "Biology"
}, {
    "_id": 2267,
    "name": "Kate",
    "major": "Sociology"
}, {
    "_id": 2453,
    "name": "Claire",
    "major": "English"
}, {
    "_id": 1957,
    "name": "John",
    "major": "Chemistry"
}]
```

Key	Value
"xyz"	*string*
"abc"	*JSON*
"pqr"	*BLOB*
"lmno"	*etc...*

Id: 1 Name: Alice Age: 18
Id: 2 Name: Bob Age: 22
Id: 3 Type: Group Name: Chess
Id: 100 Label: knows Since: 2001/10/03
Id: 101 Label: knows Since: 2001/10/04
Id: 102 Label: is_member Since: 2005/7/01
Id: 103 Label: Members
Id: 104 Label: Members
Id: 105 Label: is_member Since: 2011/02/14

Document

JSON, BLOB, XML, etc..)

Graph

Relational nodes

Key-Value Hash

Keys are mapped to values (strings, json, blob, etc..)

***Relational Database Management System (RDBMS)**:

- Help users create and maintain non-relational database (mongoDB, DynamoDB, firebase, etc)

-Implementation specific (no standard language)

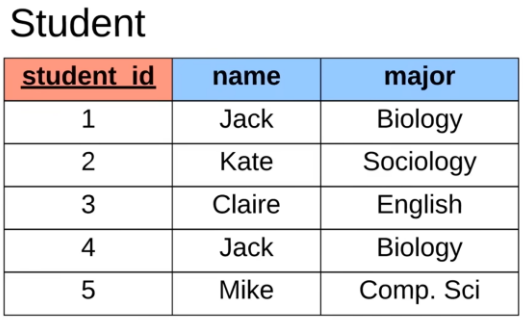
***Database Queries**:

-Requests made to the DBMS for specific information

-Reason as database grows and becomes more complex in structure it becomes difficult to extract specific pieces of information. (Google search is a type of query)

2 Tables and Keys

Table example:



Student

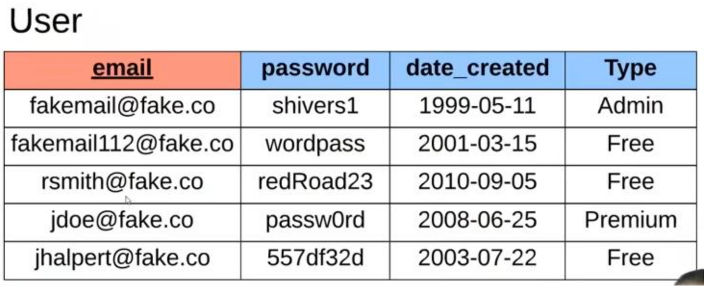
student_id	name	major
1	Jack	Biology
2	Kate	Sociology
3	Claire	English
4	Jack	Biology
5	Mike	Comp. Sci

***Entry:** Is a row

***Primary key:** Attribute of an entry that is uniquely defined (in this case student_id)

(You can have surrogate keys which are artificially made identifiers and natural key where we use unique identifiers that relate to entries for example social security numbers)

Other Table example

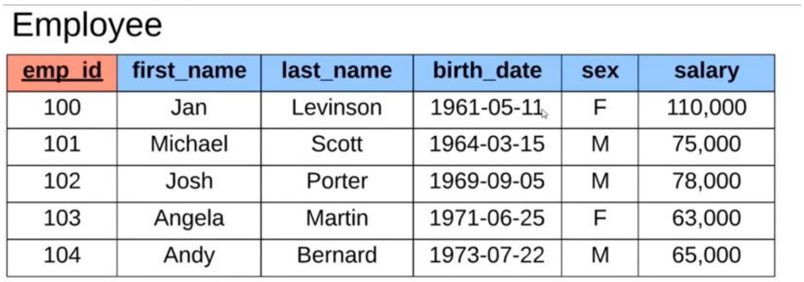


User

email	password	date_created	Type
fakemail@fake.co	shivers1	1999-05-11	Admin
fakemail112@fake.co	wordpass	2001-03-15	Free
rsmith@fake.co	redRoad23	2010-09-05	Free
jdoe@fake.co	passw0rd	2008-06-25	Premium
jhalpert@fake.co	557df32d	2003-07-22	Free

Other Table Example

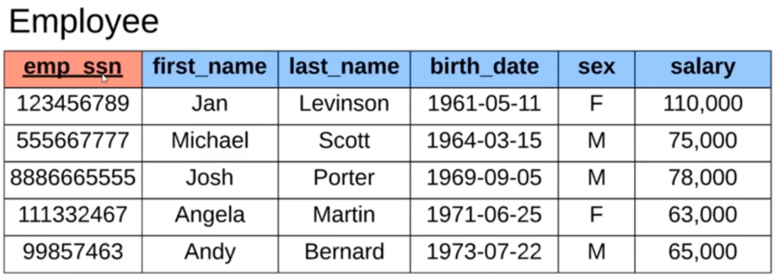
Example of Table using surrogate keys:



Employee

emp_id	first_name	last_name	birth_date	sex	salary
100	Jan	Levinson	1961-05-11	F	110,000
101	Michael	Scott	1964-03-15	M	75,000
102	Josh	Porter	1969-09-05	M	78,000
103	Angela	Martin	1971-06-25	F	63,000
104	Andy	Bernard	1973-07-22	M	65,000

Example of Table using natural keys:

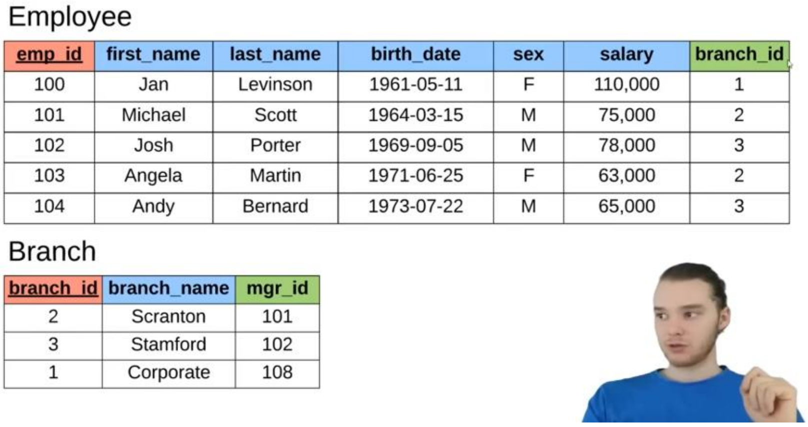


Employee

emp_ssn	first_name	last_name	birth_date	sex	salary
123456789	Jan	Levinson	1961-05-11	F	110,000
555667777	Michael	Scott	1964-03-15	M	75,000
8886665555	Josh	Porter	1969-09-05	M	78,000
111332467	Angela	Martin	1971-06-25	F	63,000
99857463	Andy	Bernard	1973-07-22	M	65,000

***Foreign key:** Attribute that links the entry to a different table.

Example (branch_id):



Employee

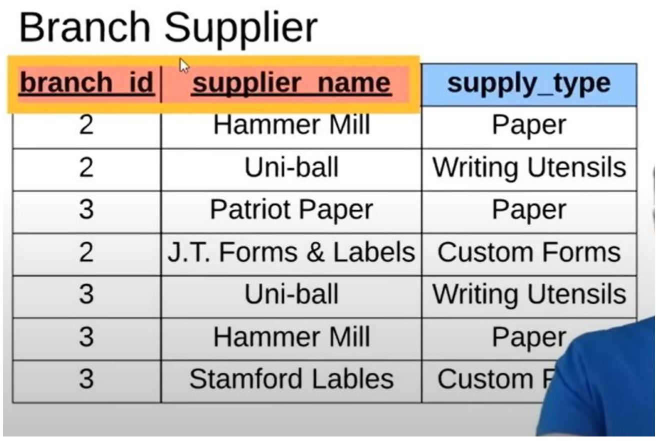
emp_id	first_name	last_name	birth_date	sex	salary	branch_id
100	Jan	Levinson	1961-05-11	F	110,000	1
101	Michael	Scott	1964-03-15	M	75,000	2
102	Josh	Porter	1969-09-05	M	78,000	3
103	Angela	Martin	1971-06-25	F	63,000	2
104	Andy	Bernard	1973-07-22	M	65,000	3

Branch

branch_id	branch_name	mgr_id
2	Scranton	101
3	Stamford	102
1	Corporate	108

***Composite key:** When two columns define a unique identifier.

Example (branch_id and supplier_name):



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 F

3 SQL Basics

***Structured Query Languages (SQL)**:

Recall:

-Standard language for interacting with RDBMS

-Used to perform C.R.U.D operations

-Used to define table structures

-Create and manage databases

-Used to perform other tasks such as backups, security and management

-It is a hybrid language (Data Query Language, Data Definition Language, Data Control Language, Data Manipulation Language)

! Note, that not all Relational Database Management System (RDBMS) follow the SQL standard exactly the same. (Concepts are the same but implementations vary)

***Queries**:

-Set of instructions given to a RDBMS that tells the RDBMS what information you will want it to retrieve for you.

4 SQL Commands

Create a Database:

mysql> create database database_name;

Most common SQL datatypes:
INT --whole numbers

DECIMAL(M,N) --Decimal Numbers (exact value)

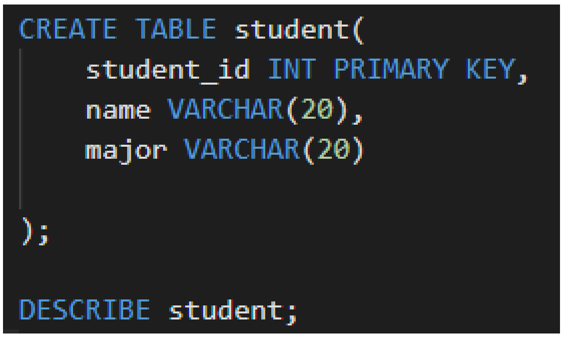
VARCHAR(1) --string of text of length 1

BLOB --Binary Large Object, Stores Large data

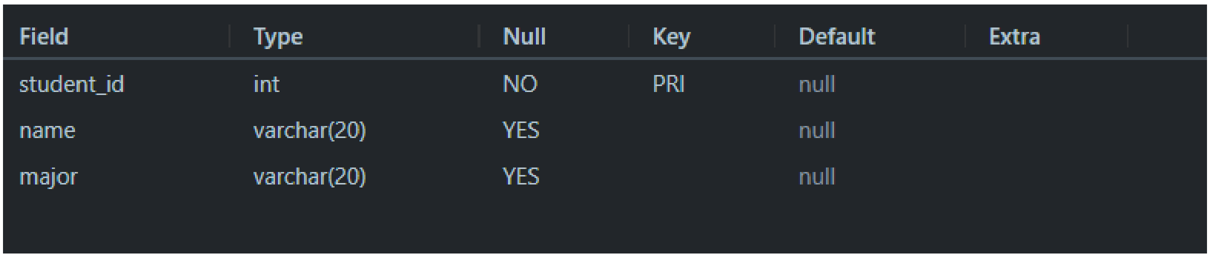
DATE --'YYYY-MM-DD'

TIMESTAMP --'YYYY-MM-DD HH:MM:SS'

Create and describe Tables:

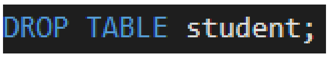


```
CREATE TABLE student(
    student_id INT PRIMARY KEY,
    name VARCHAR(20),
    major VARCHAR(20)

);

DESCRIBE student;
```



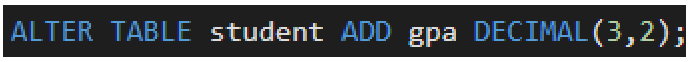
Field	Type	Null	Key	Default	Extra
student_id	int	NO	PRI	null	
name	varchar(20)	YES		null	
major	varchar(20)	YES		null	

<u>Delete tables</u>:

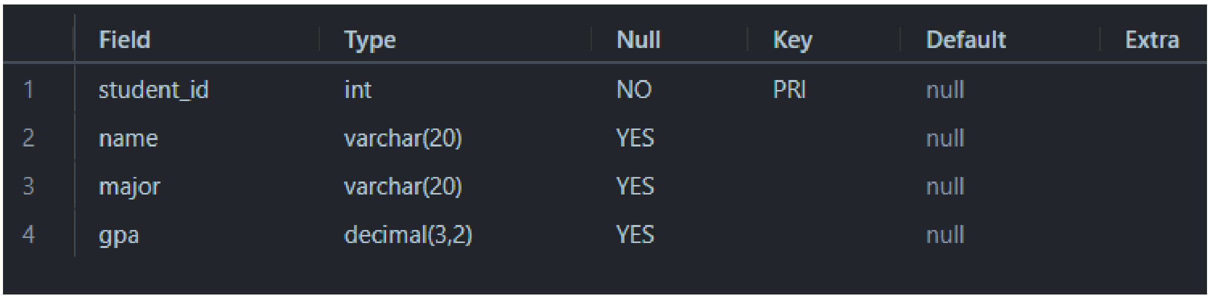


```
DROP TABLE student;
```

<u>Alter Tables</u>:

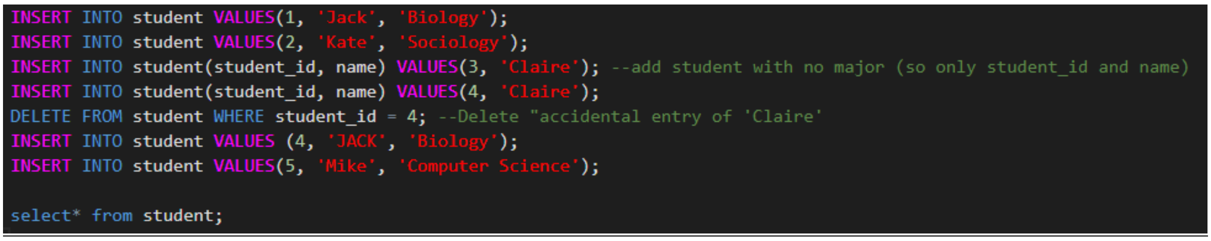


```
ALTER TABLE student ADD gpa DECIMAL(3,2);
```

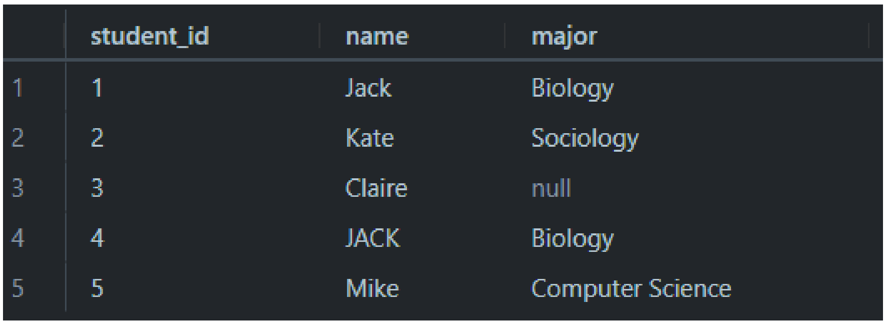


	Field	Type	Null	Key	Default	Extra
1	student_id	int	NO	PRI	null	
2	name	varchar(20)	YES		null	
3	major	varchar(20)	YES		null	
4	gpa	decimal(3,2)	YES		null	

<u>Inserting Data and viewing all (*) data in table</u>:



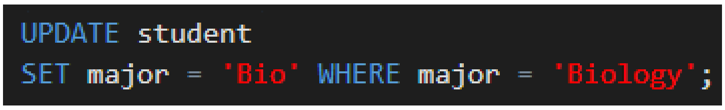
```
INSERT INTO student VALUES(1, 'Jack', 'Biology');
INSERT INTO student VALUES(2, 'Kate', 'Sociology');
INSERT INTO student(student_id, name) VALUES(3, 'Claire'); --add student with no major (so only student_id and name)
INSERT INTO student(student_id, name) VALUES(4, 'Claire');
DELETE FROM student WHERE student_id = 4; --Delete "accidental entry of 'Claire'
INSERT INTO student VALUES (4, 'JACK', 'Biology');
INSERT INTO student VALUES(5, 'Mike', 'Computer Science');

select* from student;
```



	student_id	name	major
1	1	Jack	Biology
2	2	Kate	Sociology
3	3	Claire	null
4	4	JACK	Biology
5	5	Mike	Computer Science

<u>Update Tables</u>:

Example where changing biology to bio



```
UPDATE student
SET major = 'Bio' WHERE major = 'Biology';
```

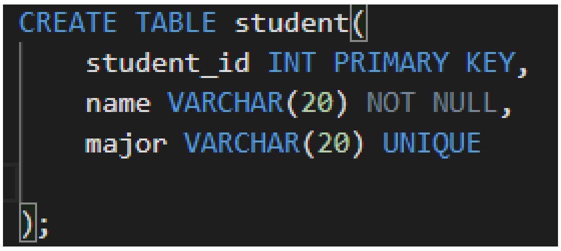
<u>Other useful commands</u>:

NOT NULL -- Ensures that no entries in this column can be NULL

UNIQUE -- Ensures that there are no duplicate instances in a column

*note that a primary key inherently possesses NOT NULL and UNIQUE constraints

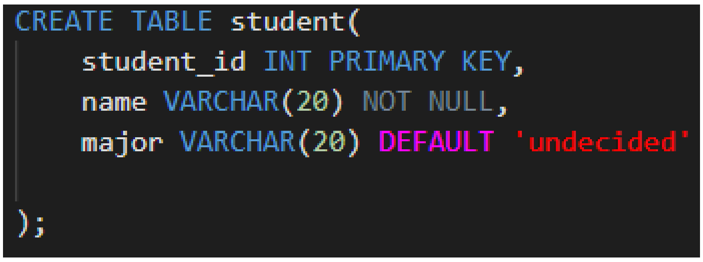
Implementation example:



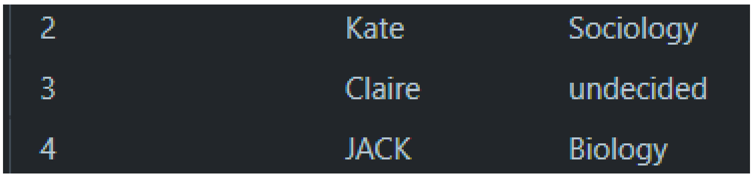
```
CREATE TABLE student(
    student_id INT PRIMARY KEY,
    name VARCHAR(20) NOT NULL,
    major VARCHAR(20) UNIQUE

);
```

<u>Setting default values</u>:

Implementation example:



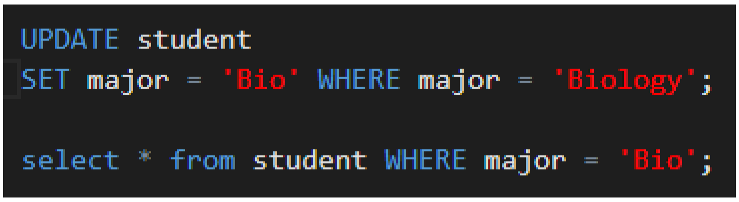
```
CREATE TABLE student(
    student_id INT PRIMARY KEY,
    name VARCHAR(20) NOT NULL,
    major VARCHAR(20) DEFAULT 'undecided'

);
```



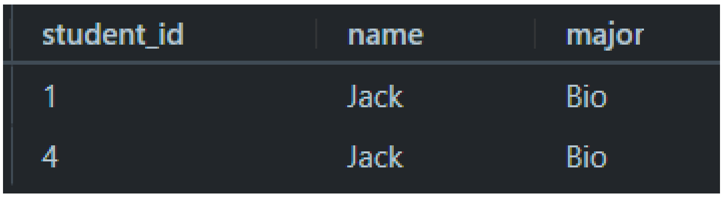
2	Kate	Sociology
3	Claire	undecided
4	JACK	Biology

Updating rows:

Implementation example:

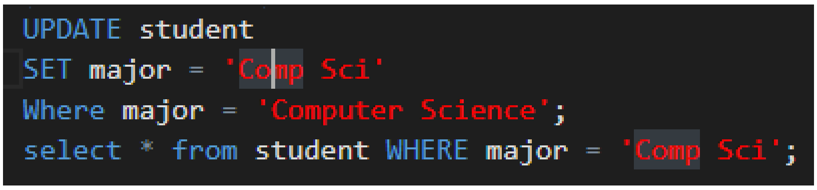


```
UPDATE student
SET major = 'Bio' WHERE major = 'Biology';

select * from student WHERE major = 'Bio';
```

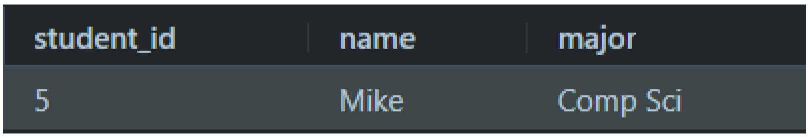


student_id	name	major
1	Jack	Bio
4	Jack	Bio

Other example:

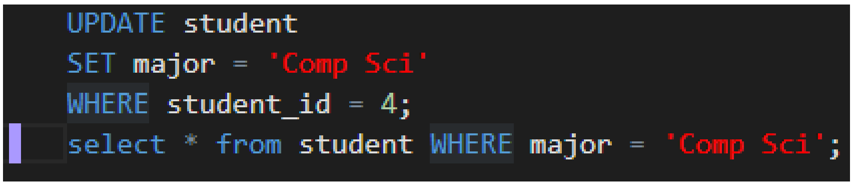


```
UPDATE student
SET major = 'Comp Sci'
Where major = 'Computer Science';
select * from student WHERE major = 'Comp Sci';
```

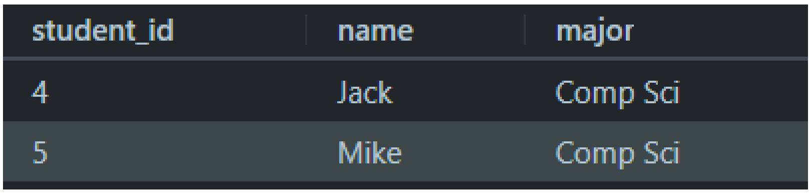


student_id	name	major
5	Mike	Comp Sci

Another Example:

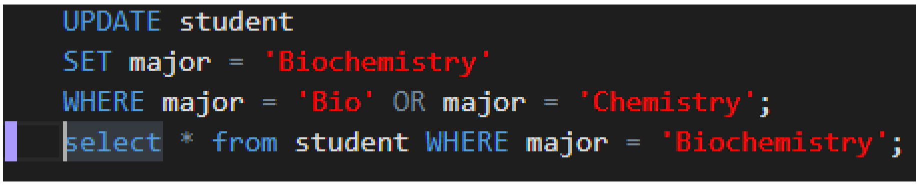


```
UPDATE student
SET major = 'Comp Sci'
WHERE student_id = 4;
select * from student WHERE major = 'Comp Sci';
```

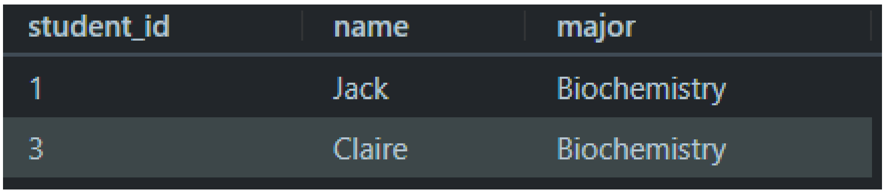


student_id	name	major
4	Jack	Comp Sci
5	Mike	Comp Sci

Other example:

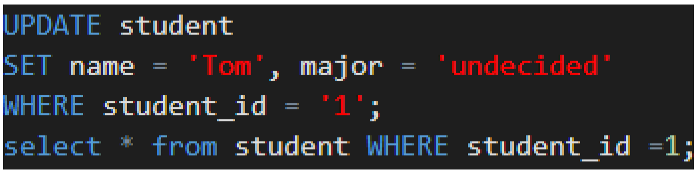


```sql
UPDATE student
SET major = 'Biochemistry'
WHERE major = 'Bio' OR major = 'Chemistry';
select * from student WHERE major = 'Biochemistry';
```

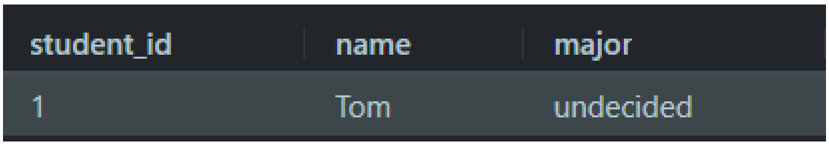


student_id	name	major
1	Jack	Biochemistry
3	Claire	Biochemistry

Other example:

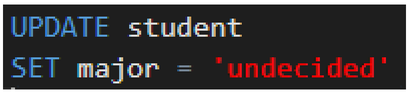


```sql
UPDATE student
SET name = 'Tom', major = 'undecided'
WHERE student_id = '1';
select * from student WHERE student_id =1;
```



student_id	name	major
1	Tom	undecided

*Note a query like this:

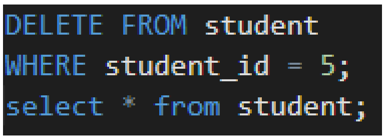


```sql
UPDATE student
SET major = 'undecided'
```

Will set all the majors to ‘undecided.

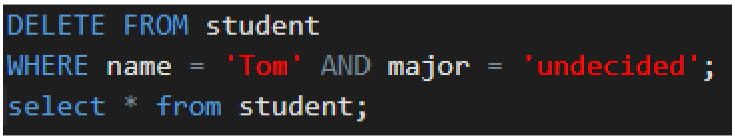
<u>Deleting rows</u>:

Delete student entry that has student id of 5…



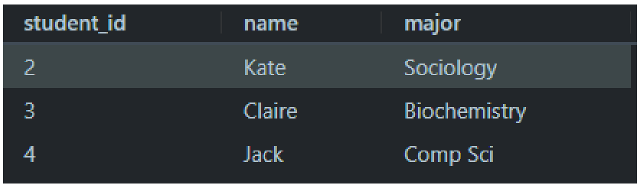
```sql
DELETE FROM student
WHERE student_id = 5;
select * from student;
```

Other example:



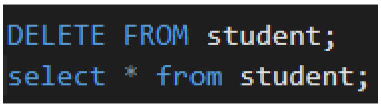
```sql
DELETE FROM student
WHERE name = 'Tom' AND major = 'undecided';
select * from student;
```

Selecting all students we have..

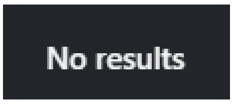


student_id	name	major
2	Kate	Sociology
3	Claire	Biochemistry
4	Jack	Comp Sci

Deleting all rows...



```
DELETE FROM student;
select * from student;
```

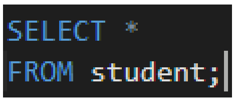


No results

5 Basic Queries

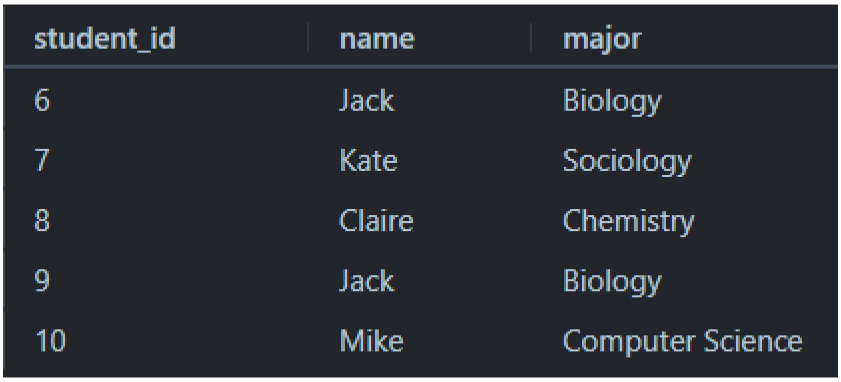
Working with this table:

Selecting all columns:



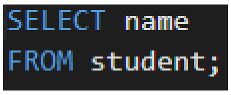
```
SELECT *
FROM student;
```

...

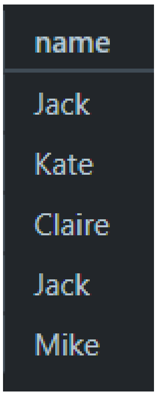


student_id	name	major
6	Jack	Biology
7	Kate	Sociology
8	Claire	Chemistry
9	Jack	Biology
10	Mike	Computer Science

Selecting only one column:

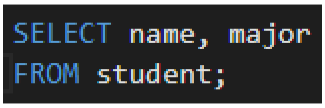


```
SELECT name
FROM student;
```



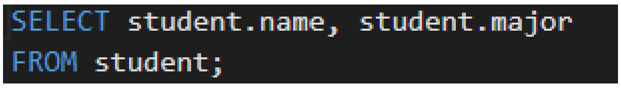
name
Jack
Kate
Claire
Jack
Mike

Selecting multiple columns:

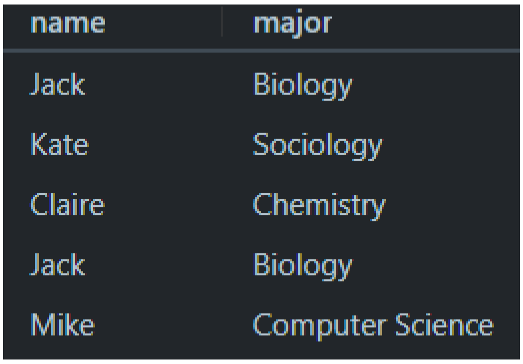


```
SELECT name, major
FROM student;
```

Or

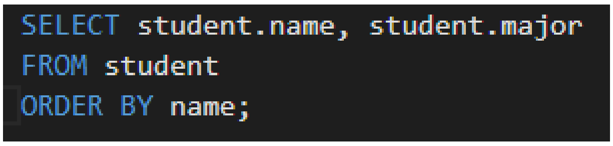


```
SELECT student.name, student.major
FROM student;
```

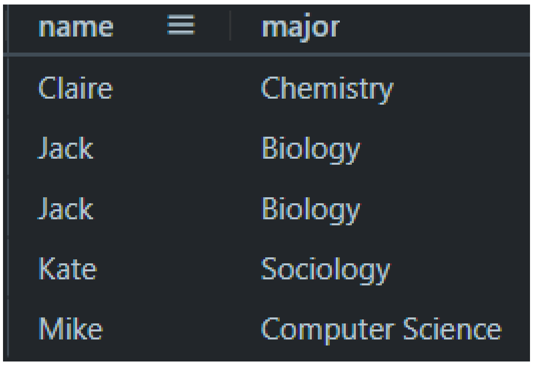


name	major
Jack	Biology
Kate	Sociology
Claire	Chemistry
Jack	Biology
Mike	Computer Science

Ordering by column (by default in Ascending order):

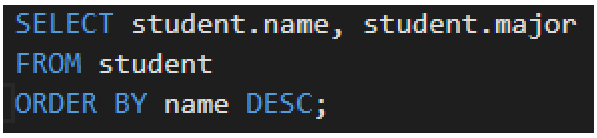


```
SELECT student.name, student.major
FROM student
ORDER BY name;
```

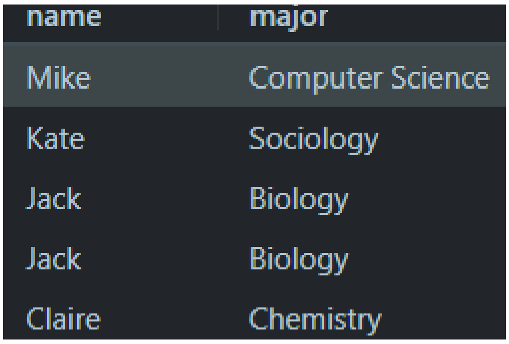


name	major
Claire	Chemistry
Jack	Biology
Jack	Biology
Kate	Sociology
Mike	Computer Science

Ordering by column in descending order:

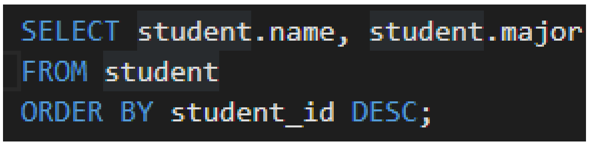


```
SELECT student.name, student.major
FROM student
ORDER BY name DESC;
```

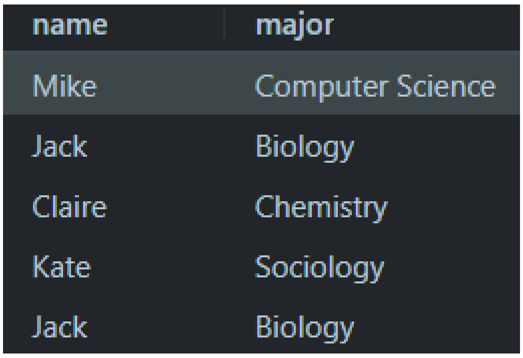


name	major
Mike	Computer Science
Kate	Sociology
Jack	Biology
Jack	Biology
Claire	Chemistry

Order by column that is not selected:

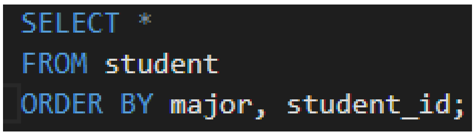


```sql
SELECT student.name, student.major
FROM student
ORDER BY student_id DESC;
```

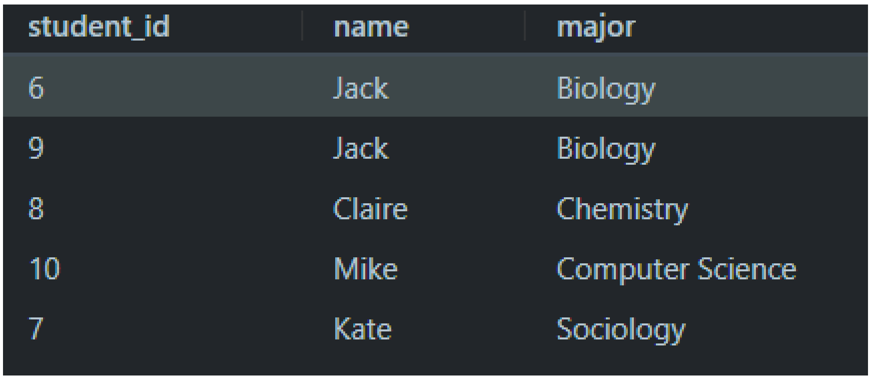


name	major
Mike	Computer Science
Jack	Biology
Claire	Chemistry
Kate	Sociology
Jack	Biology

Sub ordering example:

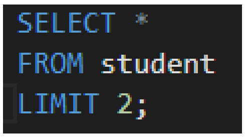


```sql
SELECT *
FROM student
ORDER BY major, student_id;
```

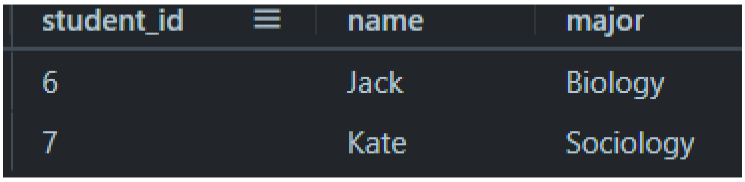


student_id	name	major
6	Jack	Biology
9	Jack	Biology
8	Claire	Chemistry
10	Mike	Computer Science
7	Kate	Sociology

LIMIT the number of entries shown:

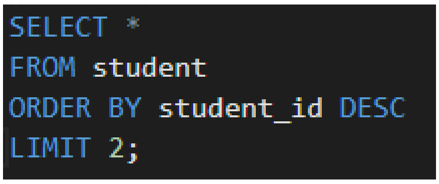


```sql
SELECT *
FROM student
LIMIT 2;
```

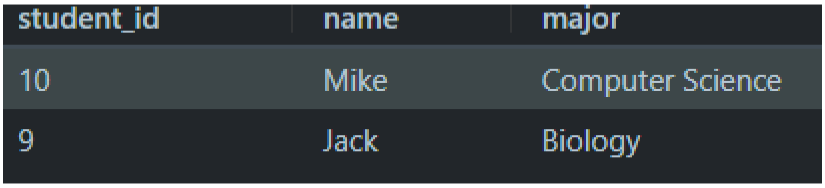


student_id	name	major
6	Jack	Biology
7	Kate	Sociology

LIMIT and order the columns:

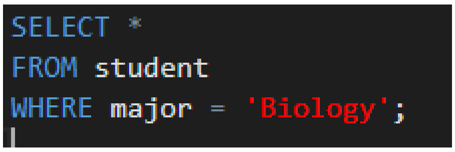


```sql
SELECT *
FROM student
ORDER BY student_id DESC
LIMIT 2;
```

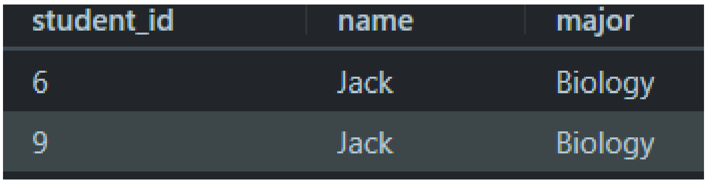


student_id	name	major
10	Mike	Computer Science
9	Jack	Biology

Filtering example:

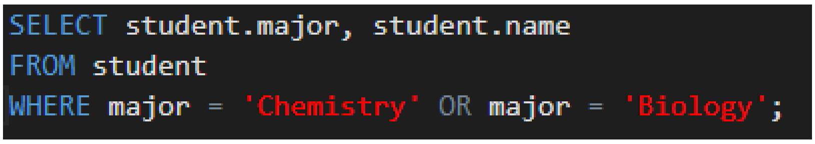


```
SELECT *
FROM student
WHERE major = 'Biology';
```

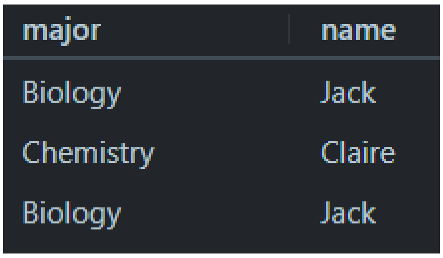


student_id	name	major
6	Jack	Biology
9	Jack	Biology

Other filtering example:



```
SELECT student.major, student.name
FROM student
WHERE major = 'Chemistry' OR major = 'Biology';
```

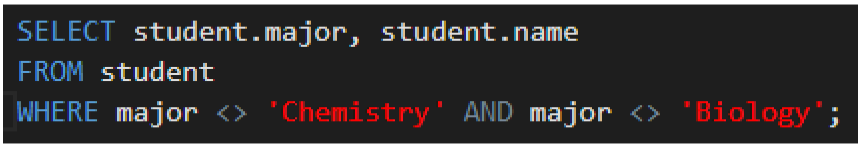


major	name
Biology	Jack
Chemistry	Claire
Biology	Jack

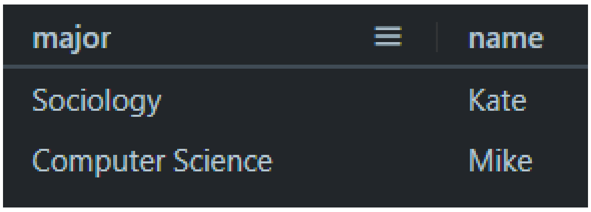
! Comparison operators:

<, >, <=, >=, =, <>, AND, OR *note <> is NOT operator

Example of <> implementation:

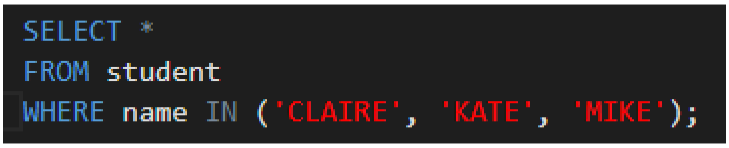


```
SELECT student.major, student.name
FROM student
WHERE major <> 'Chemistry' AND major <> 'Biology';
```

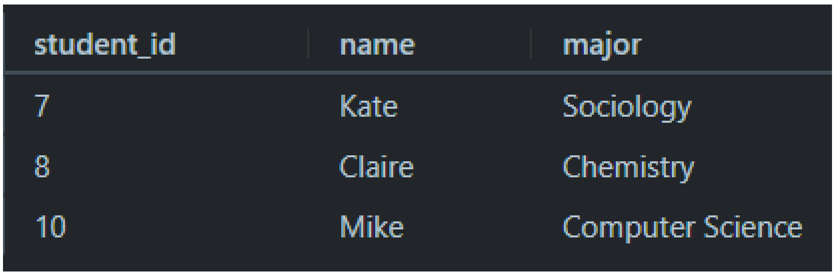


major	name
Sociology	Kate
Computer Science	Mike

Example where we select students that have certain names:



```
SELECT *
FROM student
WHERE name IN ('CLAIRE', 'KATE', 'MIKE');
```

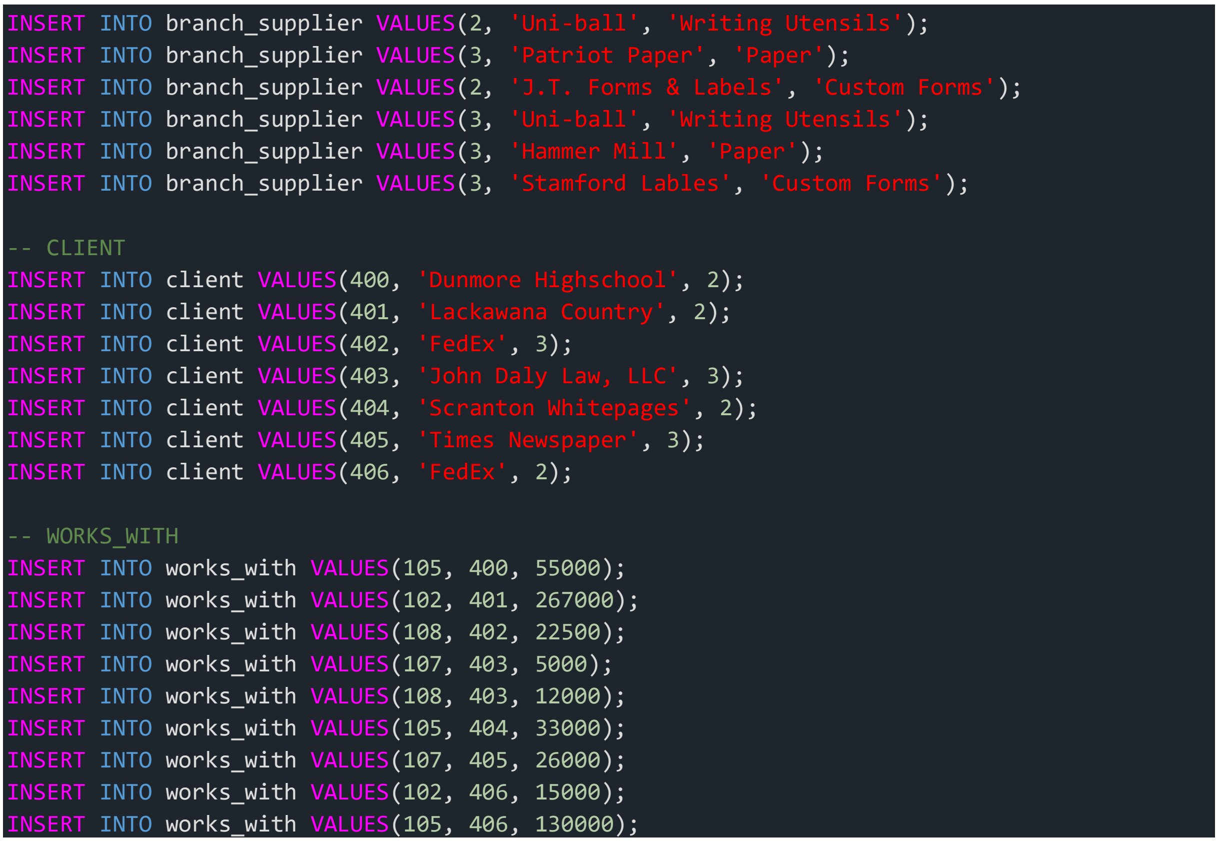


student_id	name	major
7	Kate	Sociology
8	Claire	Chemistry
10	Mike	Computer Science

6 Company Database Intro

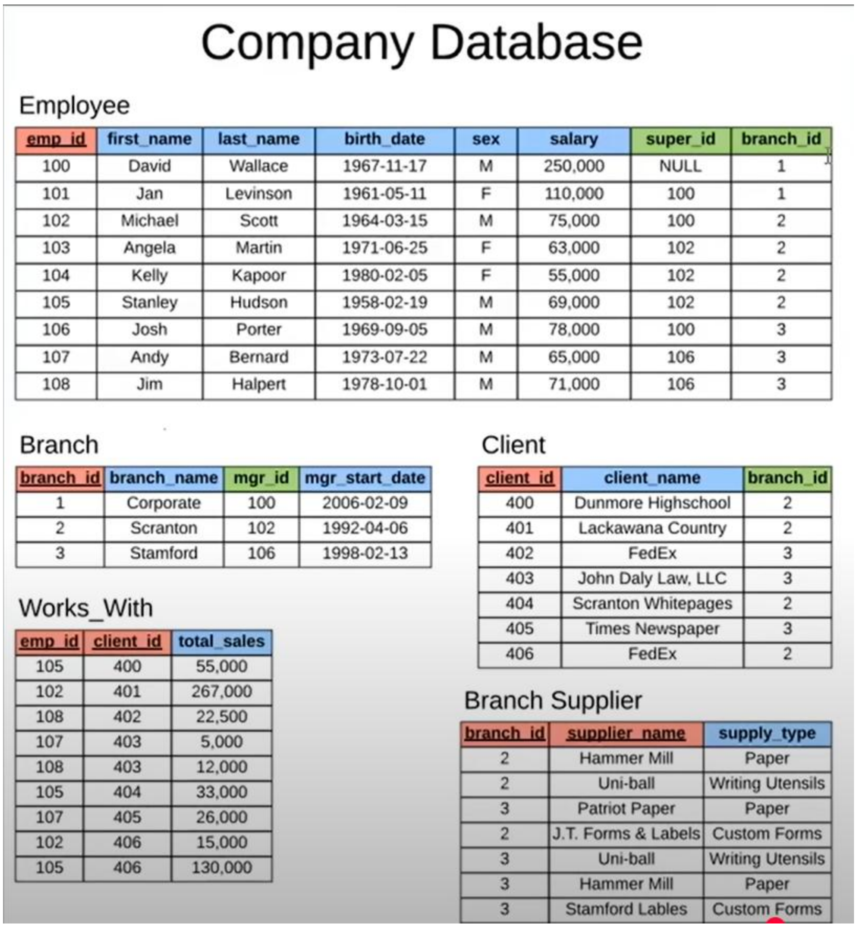
```
-- Create company database

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
);

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
);
--ADD branch id in employee table as a forreign key
ALTER TABLE employee
ADD FOREIGN KEY(branch_id)
REFERENCES branch(branch_id)
ON DELETE SET NULL;

--ADD supervisor ids (super_id) as a forreign key in employee table
ALTER TABLE employee
ADD FOREIGN KEY(super_id)
REFERENCES employee(emp_id)
ON DELETE SET NULL;

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 (
```

```sql
  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 INFO in tables

-- 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);


-- 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);
```

RESULT:



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

Querying the company database:

```
--Querying the company database----------------------------

--Find all employees
select * from employee;

--Find all clients
select * from client
limit 100;

--Find all employees ordered by salary
select * from employee
ORDER BY salary DESC;

-- Find all employees by sex then name
select * from employee
ORDER BY sex DESC, first_name ASC, last_name DESC
limit 100;

-- Find the first 5 employees in the table
select * from employee
limit 5;

--Find first and last names of all employees
SELECT first_name, last_name
FROM employee
LIMIT 50;

--Find the forenames and surnames of all employees
SELECT employee.first_name AS forename, employee.last_name AS surname
FROM employee
LIMIT 50;

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

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

! New Command:

SELECT DISTINCT –Selects unique instances of specified column

7 Functions

```
--------FUNCTIONS---------

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

-- Count number of employees that have supervisors
SELECT COUNT(super_id)
FROM employee;
```

```
--Find number of females employees born after 1970
SELECT COUNT(emp_id)
FROM employee
WHERE sex = 'F' AND birth_day >= '1971-01-01'
LIMIT 100;


--Find average salary of all employees who are male

SELECT AVG(salary)
FROM employee
WHERE sex = 'M';

--Find sum of all employee salary
SELECT SUM(salary)
FROM employee;

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

--Find total sales of each salesman //added joins to this
SELECT works_with.emp_id, employee.first_name, SUM(works_with.total_sales) AS total_sales
FROM works_with
JOIN employee ON works_with.emp_id = employee.emp_id
GROUP BY works_with.emp_id, employee.first_name;

--How much mony has each client spent
SELECT works_with.client_id, client.client_name, SUM(works_with.total_sales) AS total_sales
FROM works_with
JOIN client ON works_with.client_id = client.client_id
GROUP BY works_with.client_id, client.client_name;
```

8 Wildcards

% = any number of characters, _ = one character

Implementation:

```
-----WILD CARDS-----

--- Find any clients who are an LLC
SELECT * FROM client
WHERE client_name LIKE '%LLC'; --if the client name has any number of chaacters then "LLC"
at the end then return it;

--Find any branch suppliers thats in the label business
SELECT *
FROM branch_supplier
WHERE supplier_name LIKE '% label%' OR supplier_name LIKE '% lables%' ;

--Find any employee born in october
```

```
select * from employee;
SELECT *
FROM employee
WHERE birth_day LIKE '____-02%';

--Find any client who are schools

select * from client;
SELECT *
FROM client
WHERE client_name LIKE '%school%' ;
```

9 Unions

*Used to combine results of multiple select statements into one

```
----UNIONS-----

--Find a list of employee and baranch names
SELECT employee.first_name AS All_Names
FROM employee
UNION
SELECT branch.branch_name
From branch
UNION
SELECT client.client_name
FROM client;

--Find a list of all clients and branch supplier names and branch ids
SELECT client.client_name, client.branch_id
FROM client
UNION
SELECT branch.branch_name, branch.branch_id
From branch;

--Find a list of all money spent or earned by the company
SELECT employee.salary
FROM employee
UNION
SELECT works_with.total_sales
FROM works_with;
--essentially it just puts what ever is first above what ever is selected next into one
table (bottom join)
```

10 Joins

*Used to combine rows from two or more tables based on related columns between them.

```
--------JOINS--------
INSERT INTO branch VALUES(4, 'Buffalo', NULL, NULL); --inserted for example branch with no
mgr_id
```

```
-- Find all branches and corrisponding names of managers

SELECT employee.emp_id, employee.first_name, branch.branch_name
FROM employee
JOIN branch -- inner join //combines rows fom employee table and branch table with repect
to shared column
ON employee.emp_id = branch.mgr_id; --columns that are in common

SELECT employee.emp_id, employee.first_name, branch.branch_name
FROM employee
LEFT JOIN branch -- left join combines all rows fom employee table and adds to it on the
right branch table.
ON employee.emp_id = branch.mgr_id; --columns that are in common

SELECT employee.emp_id, employee.first_name, branch.branch_name
FROM employee
RIGHT JOIN branch -- Right join includes all rows fom branch table and adds employee table
column to the right of it.
ON employee.emp_id = branch.mgr_id; --columns that are in common

--Full outer join combines both left and right join logic by grabbing all employees and all
branches no matter if they met a certain condition
-- (ie employee.emp_id = branch.mgr_id)
-- Not function in MySQL
```

11 Nested Queries

*Involves using multiple select statements to get specific information

```
-------NESTED QUERIES--------

--Find names of all employees who have sold over 30000 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
);

--FIND all clients who are handled by the branch that
--Michael Scott manages
--Assume we know his ID

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

SELECT client.client_name
FROM client
WHERE client.branch_id = (
    SELECT branch.branch_id
    FROM branch
    WHERE branch.mgr_id = 102
    LIMIT 1 --ensures its limited to one output
);
```

12 On Delete

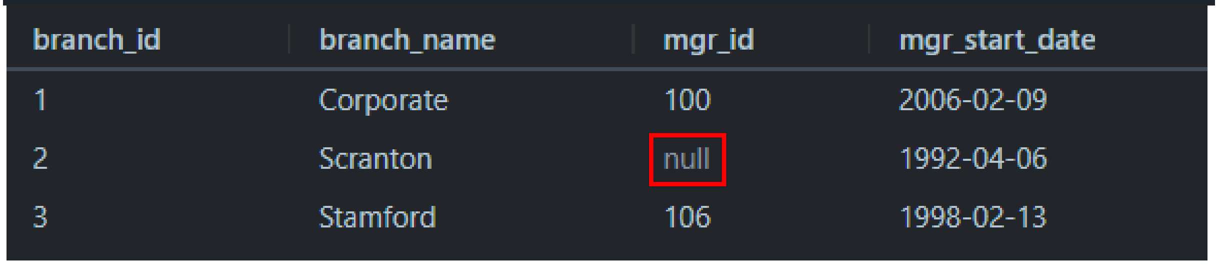
Think of a case where we Michael Scott gets fired and now we have to delete his info from our database. We know that there are several tables that contain his id for example employee, branch and works_with table. Thus, we must use a method that appropriately deletes his name while updating all the dependent entries. There are two ways to approach this:

1. **ON DELETE SET NULL**

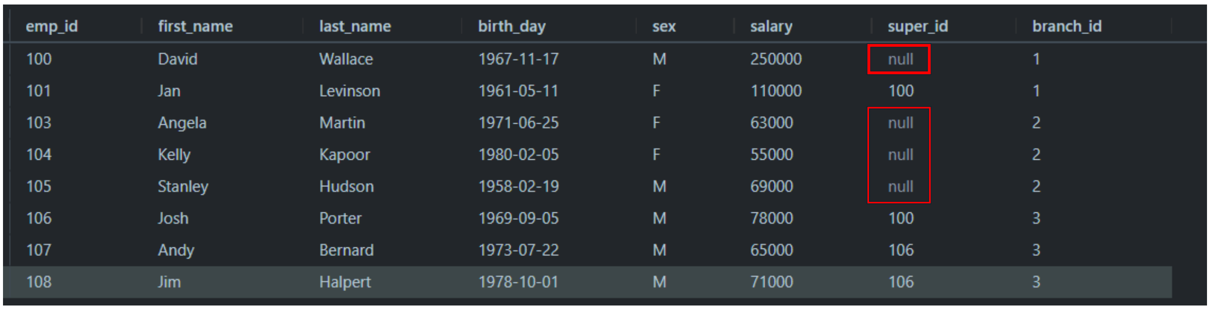
- If an employee is deleted the entries associated with that employee is set to null

Example:

```
--recall this table we created earlier...
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 employee
gets deleted then the mgr_id is set to NULL
);

DELETE FROM employee
WHERE emp_id = 102;

select * from branch;
select * from employee;
```



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

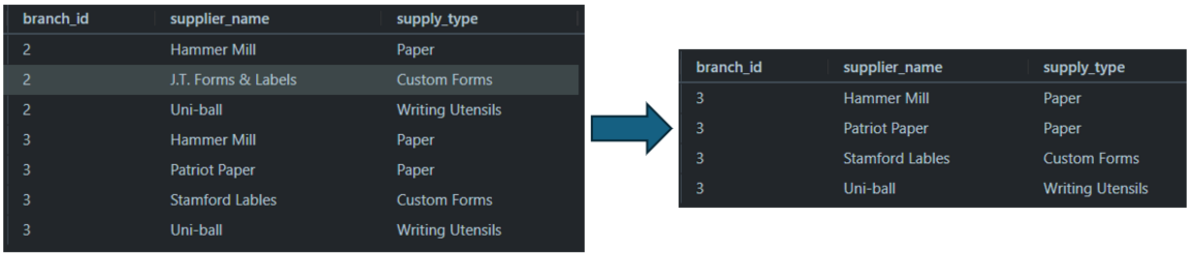


emp_id	first_name	last_name	birth_day	sex	salary	super_id	branch_id
100	David	Wallace	1967-11-17	M	250000	null	1
101	Jan	Levinson	1961-05-11	F	110000	100	1
103	Angela	Martin	1971-06-25	F	63000	null	2
104	Kelly	Kapoor	1980-02-05	F	55000	null	2
105	Stanley	Hudson	1958-02-19	M	69000	null	2
106	Josh	Porter	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

2. **ON DELETE CASCADE**

- -If an employee is deleted the entries associated with that employee is deleted from the database

Example:



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

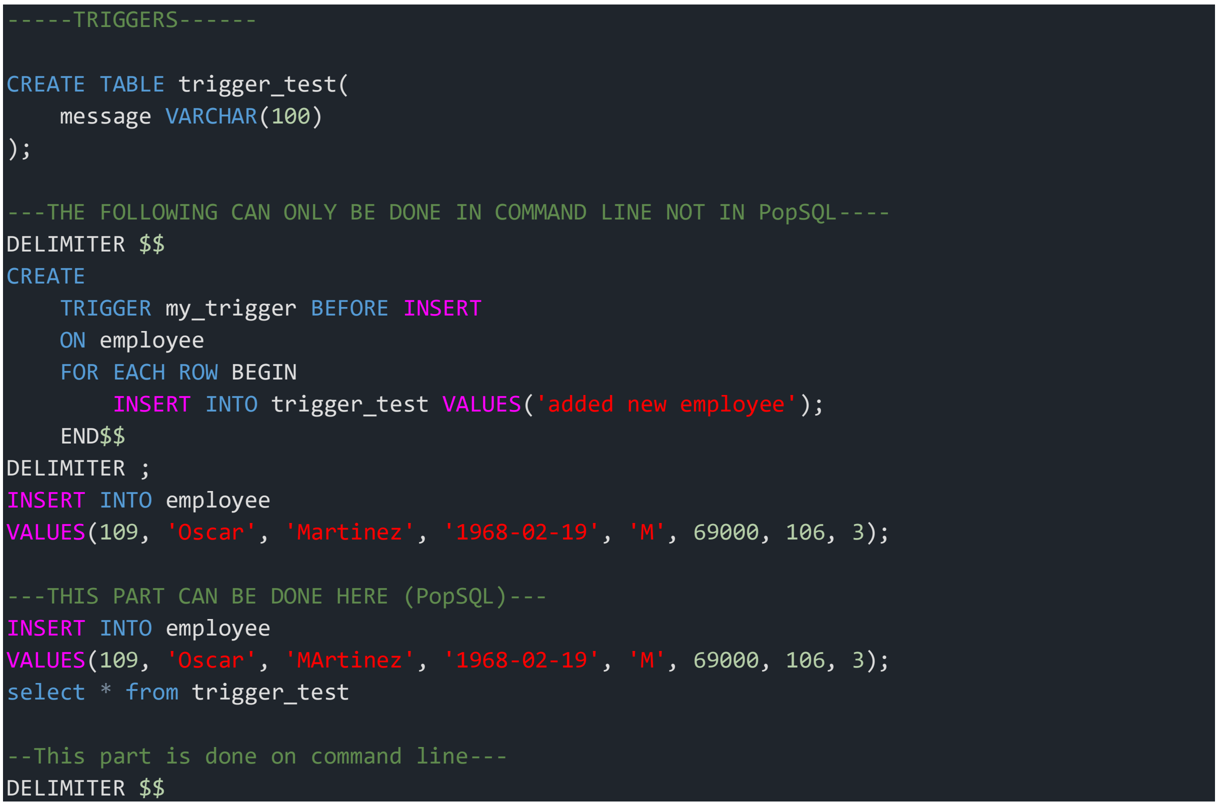
branch_id	supplier_name	supply_type
3	Hammer Mill	Paper
3	Patriot Paper	Paper
3	Stamford Lables	Custom Forms
3	Uni-ball	Writing Utensils

! NOTE: Its probably best to use **ON DELETE CASCADE** when the entry being deleted is a primary on a different table since a primary key cannot be set to null. Usually if it's just a foreign key on other tables **ON DELETE SET NULL** is fine.

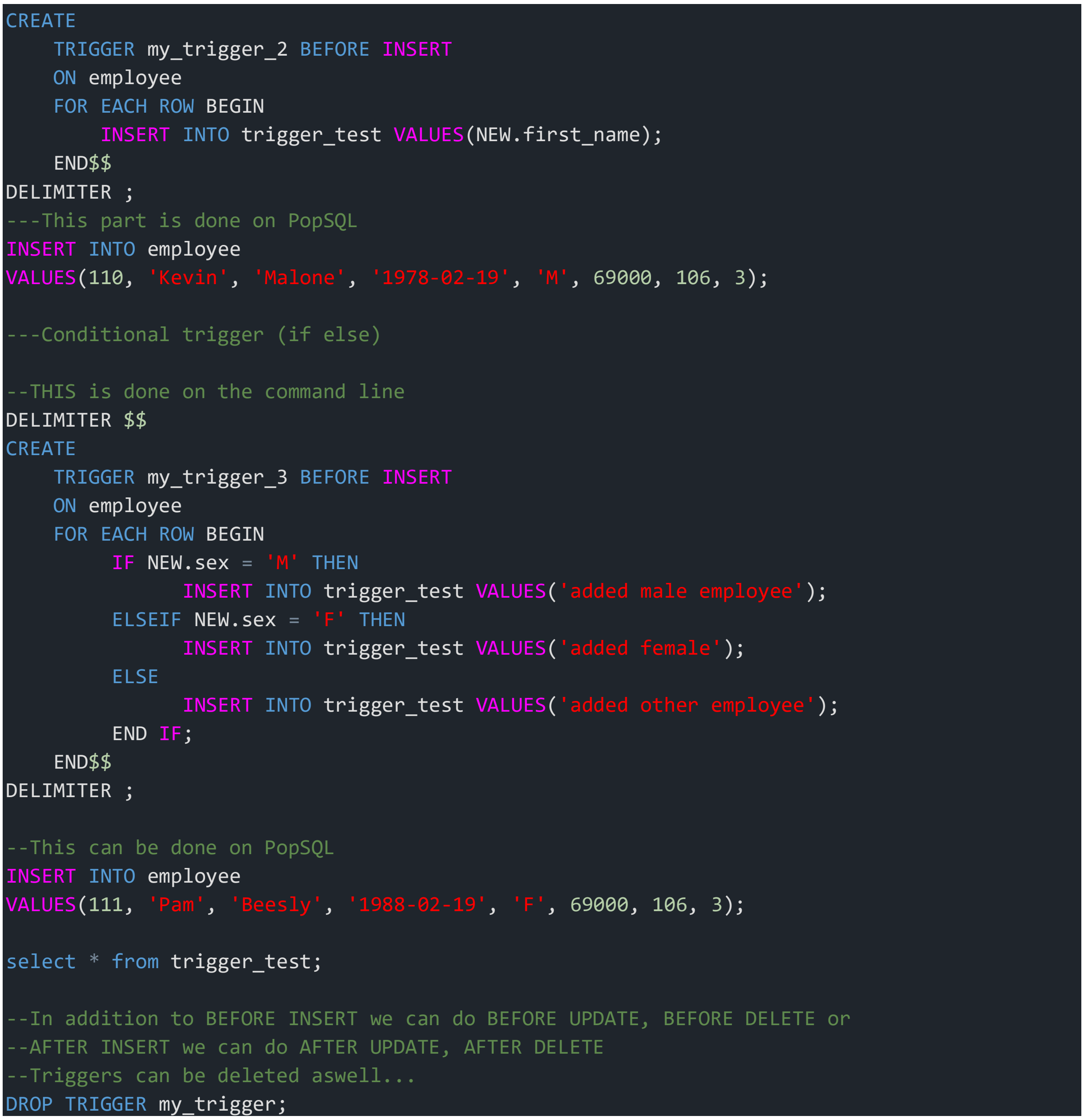
13 Triggers

*Is a block of SQL code which we can write which will define a certain action that should happen when a certain operation gets performed on the database.

*Triggers has to be defined on the command line (can't use PopSQL) since we have to change the SQL delimiter



```
-----TRIGGERS------

CREATE TABLE trigger_test(
    message VARCHAR(100)
);

---THE FOLLOWING CAN ONLY BE DONE IN COMMAND LINE NOT IN PopSQL----
DELIMITER $$
CREATE
    TRIGGER my_trigger BEFORE INSERT
    ON employee
    FOR EACH ROW BEGIN
        INSERT INTO trigger_test VALUES('added new employee');
    END$$
DELIMITER ;
INSERT INTO employee
VALUES(109, 'Oscar', 'Martinez', '1968-02-19', 'M', 69000, 106, 3);

---THIS PART CAN BE DONE HERE (PopSQL)---
INSERT INTO employee
VALUES(109, 'Oscar', 'MArtinez', '1968-02-19', 'M', 69000, 106, 3);
select * from trigger_test

--This part is done on command line---
DELIMITER $$
```



```
CREATE
    TRIGGER my_trigger_2 BEFORE INSERT
    ON employee
    FOR EACH ROW BEGIN
        INSERT INTO trigger_test VALUES(NEW.first_name);
    END$$
DELIMITER ;
---This part is done on PopSQL
INSERT INTO employee
VALUES(110, 'Kevin', 'Malone', '1978-02-19', 'M', 69000, 106, 3);

---Conditional trigger (if else)

--THIS is done on the command line
DELIMITER $$
CREATE
    TRIGGER my_trigger_3 BEFORE INSERT
    ON employee
    FOR EACH ROW BEGIN
         IF NEW.sex = 'M' THEN
               INSERT INTO trigger_test VALUES('added male employee');
         ELSEIF NEW.sex = 'F' THEN
               INSERT INTO trigger_test VALUES('added female');
         ELSE
               INSERT INTO trigger_test VALUES('added other employee');
         END IF;
    END$$
DELIMITER ;

--This can be done on PopSQL
INSERT INTO employee
VALUES(111, 'Pam', 'Beesly', '1988-02-19', 'F', 69000, 106, 3);

select * from trigger_test;

--In addition to BEFORE INSERT we can do BEFORE UPDATE, BEFORE DELETE or
--AFTER INSERT we can do AFTER UPDATE, AFTER DELETE
--Triggers can be deleted aswell...
DROP TRIGGER my_trigger;
```

14 ER Diagrams Introduction

*__Entity__: An object we want to model and store info about.

*__Primary Key__: An attribute(s) that uniquely identify an entry in the database table. Indicated by underlining the name.

*__Attributes__: Specific pieces of info about an entity

*__Composite Attribute__: An attribute that can be broken up into sub-attributes

*__Multi-Valued Attribute__: An attribute that can have more than one value

*__Derived Attribute__: An attribute that can be derived from other attributes

*__Relationships__: Defines a relationship between two entities

*__Relationship Attribute__: Attribute about the relationship (stored on relationship)

*__Relationship Cardinality__: Number of instances of an entity from a relation that can be associated with the relation. Different types:

1:1 One to one relationships: E.g. Class can only have one subject, and one subject can only have one class

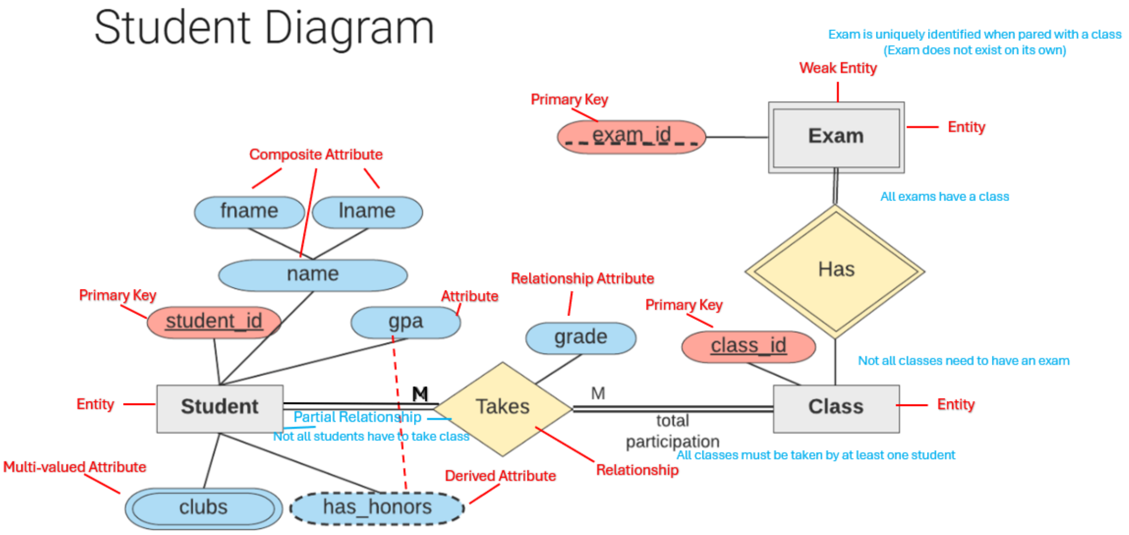
1:N One to many relationships: E.g. Class can have many exams, but exam has one class

N:M: Many to many relationships: E.g. Many students can take many classes

*__Weak Entity__: An entity that cannot be uniquely identified by its attributes alone

*__Identifying relationship__: A relationship that serves to uniquely identify weak entity

EXAMPLE:



15 Designing an ER Diagram

Given information:

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 its money by selling to clients. Each client has a name and a unique number to identify it.

The foundation of the company is its 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.

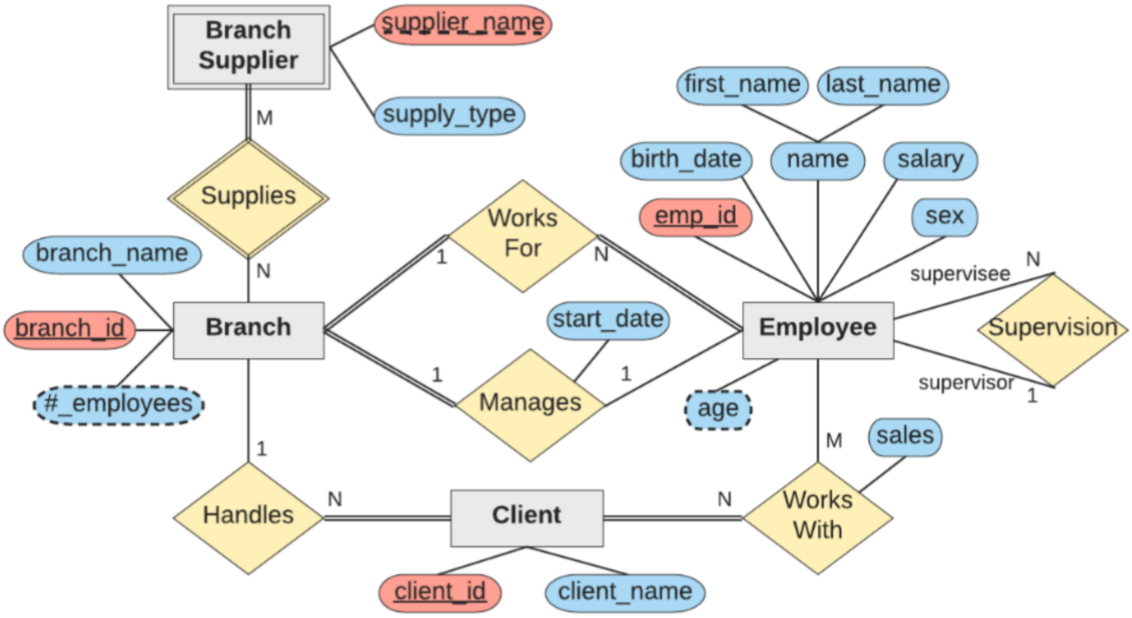
A branch may handle a number of clients, with each client having a name and a unique number to identify it. A single client may only be handled by one branch at a time.

Employees can work with clients controlled by their branch to sell them stuff. If necessary 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.

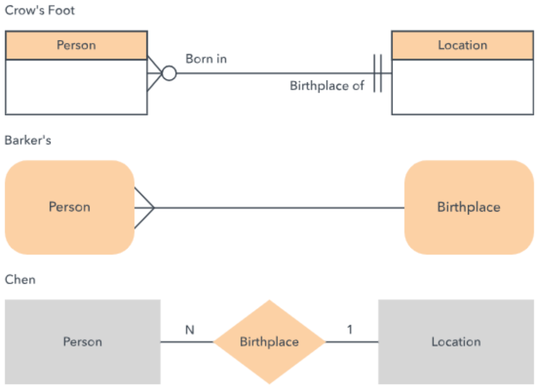
Many branches will need to work with suppliers to buy inventory. For each supplier we'll keep track of their name and the type of product they're selling the branch. A single supplier may supply products to multiple branches.

Approach: Just go through each line convert it to ER diagram. Then Find the relationships between all the entities. Result:

Company ER Diagram



There are different conventions for ER diagrams. For example:

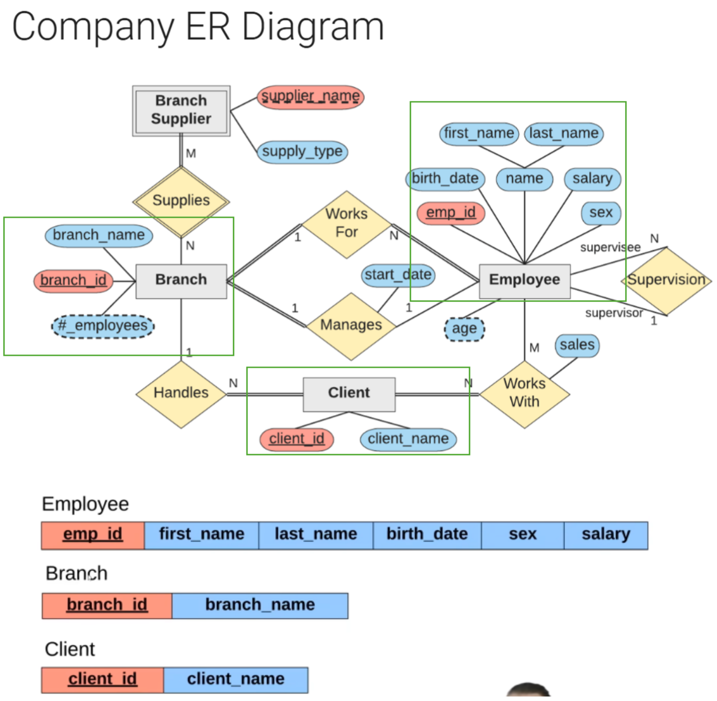


I believe we used Chen convention. I recommend looking at the following website that goes over the different conventions. They all follow the same logic but not all companies would use the same. (For example, I prefer Crow's foot as its arguably more clean

https://www.lucidchart.com/pages/er-diagrams

16 Converting ER Diagrams to Schemas

STEP 1: Mapping of regular entity types



Company ER Diagram

Employee

emp_id	first_name	last_name	birth_date	sex	salary

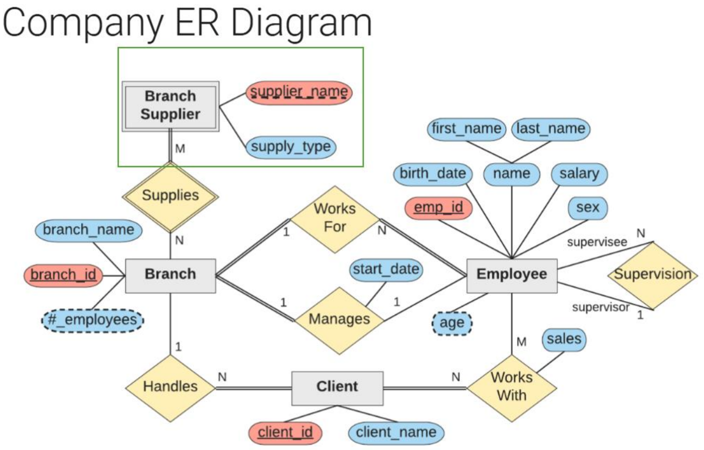
Branch

branch_id	branch_name

Client

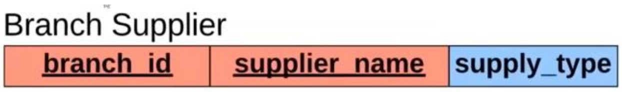
client_id	client_name

STEP 2: Mapping of weak entity types



Company ER Diagram

! Note: the primary key of the new relation should be the partial key of the weak entity (supplier_name) plus the partial key of the primary key of its owner (branch_id).

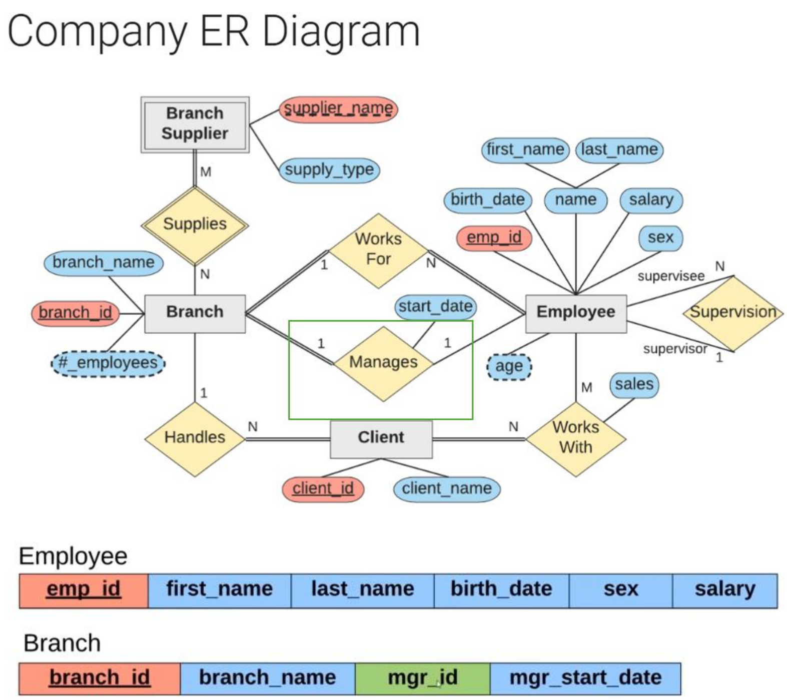


Branch Supplier

branch_id	supplier_name	supply_type

STEP 3: Mapping of Binary 1:1 Relationships

! Note: Include one side of the relationship as a foreign key in the other. Favor the total participation.



Company ER Diagram

Employee

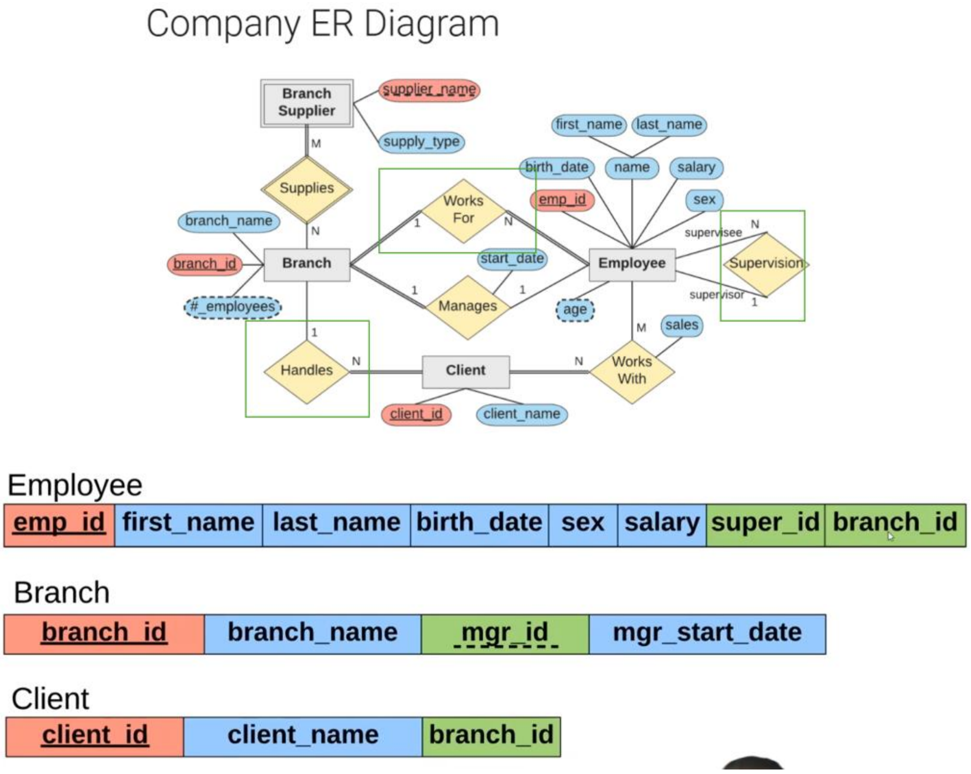
emp_id	first_name	last_name	birth_date	sex	salary

Branch

branch_id	branch_name	mgr_id	mgr_start_date

STEP 4: Mapping of Binary 1:N Relationships

! Note: Include the 1 sides primary key as foreign key on the N side relation (table



Company ER Diagram

Employee

emp_id	first_name	last_name	birth_date	sex	salary	super_id	branch_id

Branch

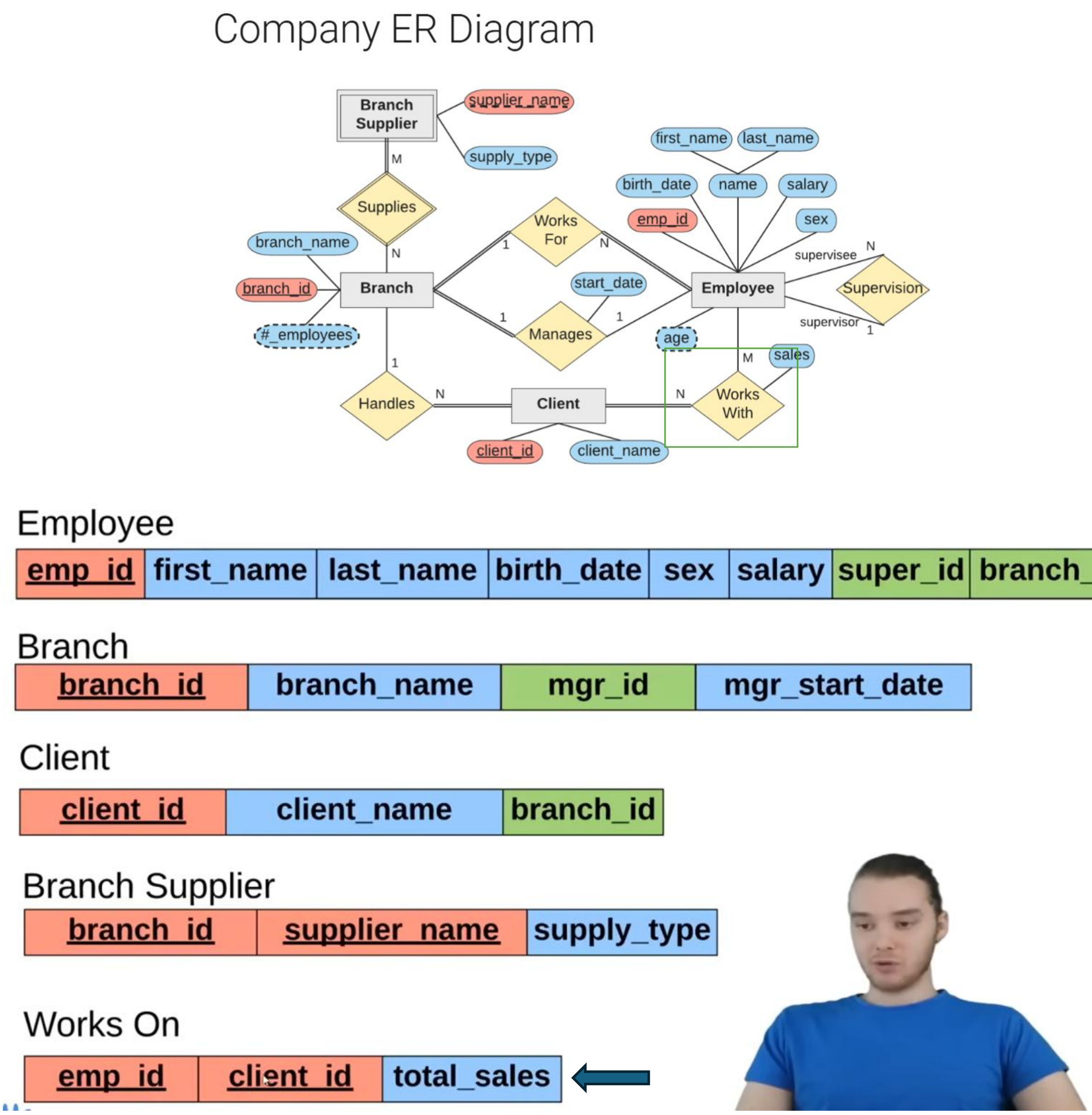
branch_id	branch_name	mgr_id	mgr_start_date

Client

client_id	client_name	branch_id

STEP 5: Mapping of Binary M:N Relationships

! Note: Create new relation table whose primary key is combination of both entities primary key also include relationship attributes



Company ER Diagram

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

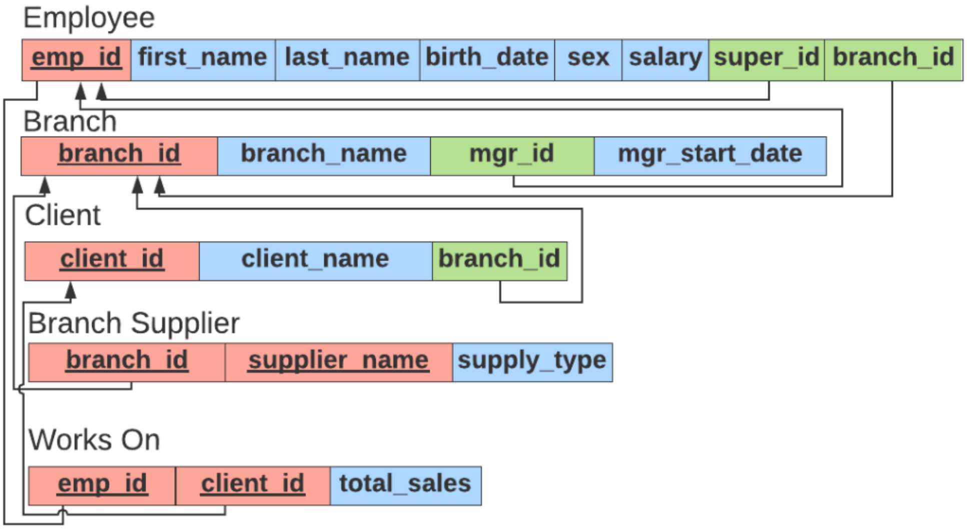
Branch Supplier

branch_id	supplier_name	supply_type

Works On

emp_id	client_id	total_sales

Result:



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

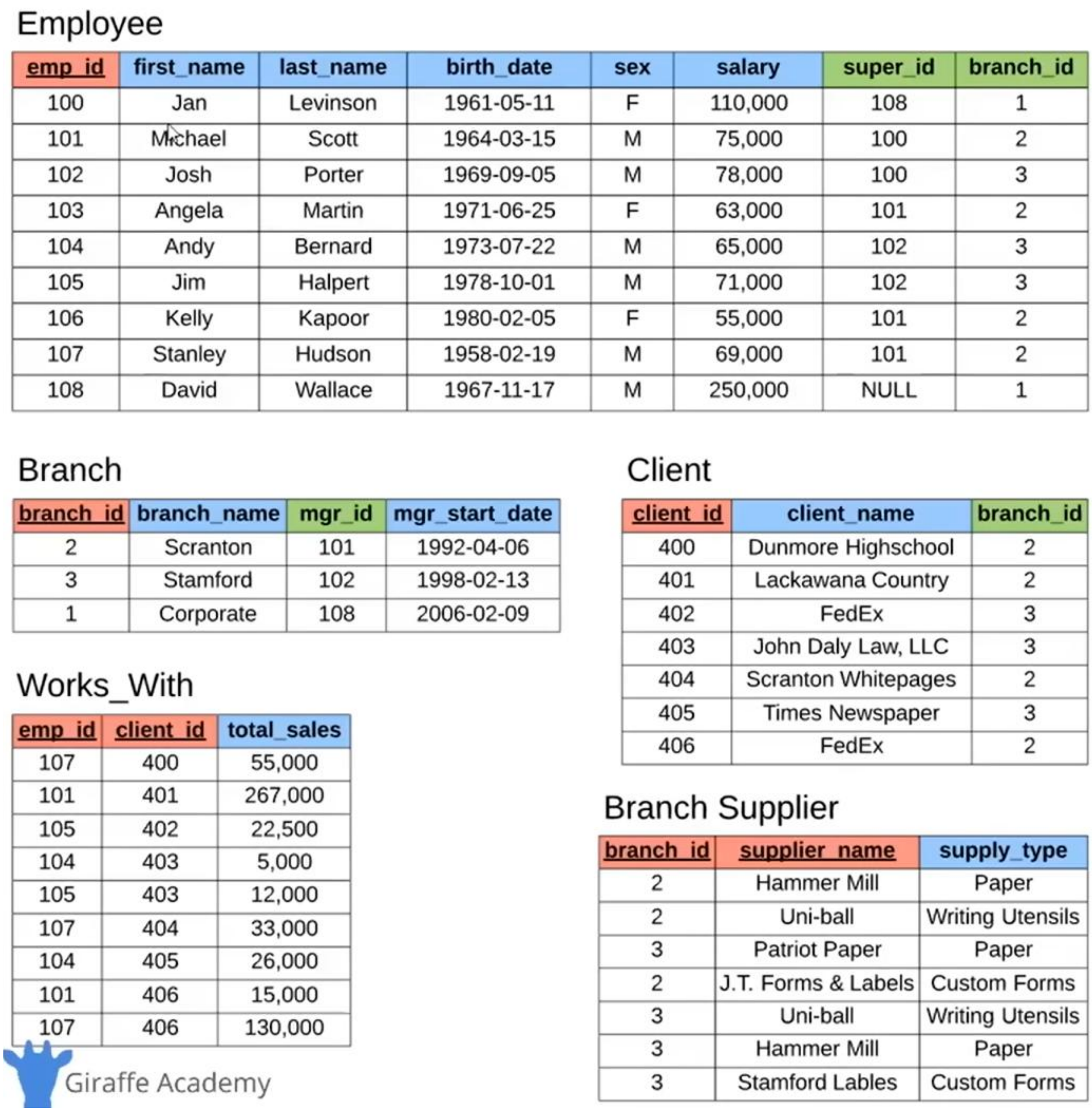
Branch Supplier

branch_id	supplier_name	supply_type

Works On

emp_id	client_id	total_sales

STEP 6: Create the database



Employee

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

Branch

branch_id	branch_name	mgr_id	mgr_start_date
2	Scranton	101	1992-04-06
3	Stamford	102	1998-02-13
1	Corporate	108	2006-02-09

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
107	400	55,000
101	401	267,000
105	402	22,500
104	403	5,000
105	403	12,000
107	404	33,000
104	405	26,000
101	406	15,000
107	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