YouTube Channel: freeCodeCamp.org

SQL Tutorial - Full Database Course for Beginners

https://www.youtube.com/watch?v=HXV3zeQKqGY

David Corzo

2020 December 14

Índice general

Capítulo 1

Introduction, What is a Database?

1.1. What is SQL?

- SQL is a language used to interact with relational database management systems.
- A relational database management system is basically just a software application to create and manage different databases.

1.2. What is a database?

- Sometimes databases are abbreviated as DB.
- A database is any collection of related information:
  - Phone book.
  - Shoping list.
  - Todo list.
  - Your 5 best friends.
  - Facebook's User base.
- Database can be stored in different ways.
  - On paper.
  - In your mind.
  - On a computer.
  - This PowerPoint.
  - Comments section.

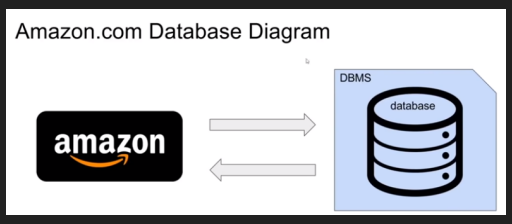
1.3. Computers with databases

- Storing a collection of related information on a computer is extremely useful, computers are great for this.
- A database can be stored anywhere, but there are better ways of storing databases than others. Computers are great at keeping track of large amounts of information.
- Take this example:

Amazon.com	Shopping list
• Keeps track of products, reviews, purchase orders, credit cards, users, media, etc. • Needs to store trillions of pieces of information, and they need to be readily available. • Information is extremely valuable and critical to Amazon.com's functioning. • Security is essential, Amazon stores peoples' personal information: ◦ Credit card #, SSN, Address phone. • Information is stored on a computer.	• Keeps track of customer products that need to be purchased. • Stores 10-20 pieces of information, this also needs to be readily available. • Information is for convenience sake only and not necessary for shopping. • Security is not important. • Information is stored on a piece of paper, or even just in someone's memory.

1.4. Database Management System (DBMS)

- A database can be as simple as a txt file, or excel file, but generally if you need to store large amounts of information a better solution is to use special software designed to create and maintain a database, this is called a Data Management System.
- A special software program that helps users create and maintain a database.
  - Makes it easy to manage large amounts of information.
  - Handles security.
  - Backup your data.
  - Importing and exporting data.
  - Concurrency.
  - Interacts with software applications:
    - Programming software.
- The database management system is not the database it is the software application that is creating, managing, updating, etc the database.



1.5. C.R.U.D

- Create, Read (Retrieve), Update, Delete.
- CRUD represents the 4 main operations that can be done in a database.
- Any good database management systems are able to perform these operations.

1.6. Two types of databases

- Relational Database (SQL): (The most popular kind of database.)
  - Organize data into one or more tables.
  - Each table has columns and rows.
  - A unique key identifies each row.
  - It is a lot like an Excel spreadsheet.
- Non-relational (noSQL / no just SQL):
  - Organize data is anything but a traditional table.
  - Key-value stores.
  - Documents (JSON, XML, etc).
  - Graphs.
  - Flexible tables.
  - Any type of database that is not a non-relational database. Organize data in anything but a table.

1.7. Relational Database (SQL)

Student Table

*ID #	Name	Major
1	Jack	Biology
2	Kate	Sociology
3	Claire	English
4	John	Chemistry

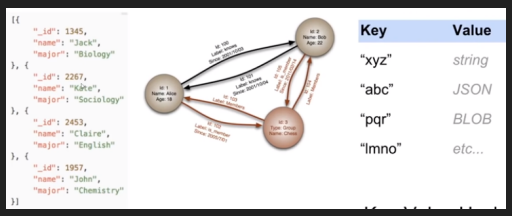
Users Table

*Username	Password	Email
jsmith22	wordpass	...
catlover45	apple223	...
gamerkid	...	...
giraffe	...	...

- Relational Database Management Systems (RDBMS):
  - Help users create and maintain a relational database.
    - mySQL, Oracle, postgreSQL, mariaDB, etc.
- Structured Query Language (SQL):
  - Relational Database Systems use SQL to interact with relational DB.
  - Used to perform CRUD operations, as well as other administrative tasks (user management, security, backups, etc.)
  - Used to define tables and structures.
  - SQL code used on one RDBMS is not always portable to another without modification. Not all SQL code used on one RDBMS will be able to be used on others.

1.8. Non-relational databases

- Anything that is not relational.
- For example:



- Store data on graphs, nodes, key value hash, documents (JSON, BLOB, XML).

- Non-relational Database management systems (NRDBMS):
  - Help users create and maintain a non-relational database.
    - mongoDB, dynamoDB, apache cassandra, firebase, etc.
  - Implementation specific:
    - Unlike RDMBS where there is a standard (SQL), this is implementation specific, there is no standard language for interacting with the non-relational database.
    - Each implementation will include the implementation for managing the database and performing the CRUD operations.
    - Most NRDBMS will implement their own language for performing CRUD operations and administrative operations on the database.

1.9. Database Queries

- Queries are request made to the database management system for specific information:
  - Query is asking the DBMS for information.
- As the database's structure becomes more and more complex it becomes more difficult to get the specific pieces of information we want.
- A Google search is a query.
  - With a relational database management system we cannot search for information in the same way google searches for it, we must adhere to a specific language in this case SQL.

1.10. Wrap up

- Database is any collection of related information.
- Computers are great for storing databases.
- Database Management Systems (DBMS) make it easy to create, maintain and secure a database.
- DBMS allow you to perform the CRUD operations and other administrative tasks.
- Two types of databases, relational and non-relational.
- Relational databases use SQL and store data in tables with rows and columns.
- Non-relational databases store data using other data structures.
- Queries are request made to the database management system for specific information.

Capítulo 2

Tables & Keys

2.1. Primary keys, Surrogate and Natural keys

- In every table of a relational database you have rows and columns, columns denote specific attributes, and rows denote each entry or object.
- Each table needs to have a *primary key* attribute. In the above example the student id would be the primary key because it uniquely identifies each student. This allows us to have repeated names, notice that student number 2 and 4 have the same name and major, the primary key differentiates them and therefore there is no problem.
- A primary key can be anything such as a string, a number or character, the only requisite is that they are unique in the table.
- Example:

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

2.1.1. Another example

- Example:

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

- Notice the emp_id, this is a primary key. In this particular example it is a Surrogate Key or a key that has no mapping in the real world:

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

- Notice here the primary key is the social security number. This is called a natural key, this is a key that has a purpose or a mapping in the real world.

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

2.1.2. Difference between Surrogate and natural keys

- Both are types of primary keys.
- A surrogate key serves only to identify a particular object inside a database such as an id or a hash, this key has no significance nor mapping outside the database.
- A natural key is a registration of an individual object using natural attributes, such as a social security number, this type of key has significance outside and inside the database.

2.2. Foreign Keys and composite keys

- An attribute that you can store that allow the object to be linked to another database table.
- For example in the below figure suppose that we have employees and in the company we have different branches, and we want to store the information, in this case we can store that using a foreign key:

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

  - In the above, we can see that for example Josh Porter belongs to the third branch, we can look up what is the third branch, in this case in another table, we quickly see that the third branch matches up to the Stamford branch.
  - Notice how the foreign key "branch_id"is the primary key of the Branch dable.
- A foreign key is a primary key inside another table.
- It is basically a way to define relationships with other tables.
- You can have multiple foreign keys:

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

- In the above figure, take for example, Angela Martin has the supervisor id=101, this means that whoever has the id 101 is Angela's supervisor, in this case id 101 is Michael Scott. Michael Scott's supervisor is Jan (id=100), and Jan has no supervisor.

- Taking the same example, lets add another foreign key that denotes who the suppliers are for each branch:

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

  - In this case we use a composite key, composite keys are keys that only makes sense in groups. For example the above only makes sense if we say Hammer Mill supplies branch 2, even though 2 is repeated three times and Hammer Mill is repeated twice.
  - We can say "Hammer Mill supplies Paper to branch 2".
  - Composite keys are when you define the combination of two columns to be a primary key, which is to say the columns by themselves have repeated values but when they are in combination they do not repeat.
  - This is why we've marked in orange (primary key) two columns.

- Suppose we have another database of clients and another database of sales:

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

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
97	404	33,000

  - Notice in this case emp_id and client_id are composite primary keys and at the same time they are foreign keys.
  - We can see that we can calculate how much an employee has sold for example.

- Notice as your databases get more and more complex you can split the database up in to multiple databases and establish a relationship between them using composite keys or foreign keys.

Capítulo 3

SQL Basics

3.1. Structured Query Language (SQL)

- SQL is a language used for interacting with relational Database Management Systems (RDBMS)
- It is kind of like a programming language, it is not strictly a programming language.
- You can use SQL to get the RDBMS to do things for you.
  - Create, Retrieve, Update and Delete data.
  - Create and Manage database.
  - Design and create database tables.
  - Perform administration tasks (security, user management, import/export, etc)
- RDBMS do not speak English, they speak SQL.
- SQL implementations vary between systems:
  - Not all RDBMS' follow the SQL standard to a 'T'.
  - The concepts are the same but the implementation may vary.
  - Keep in mind that certain instructions might work on certain RDBMS and not work on others.
- SQL is actually a hybrid language, it's basically 4 types of languages in one:
  - It is a Data Query Language (DQL):
    - Used to query the database for information.
    - Get information that is already stored there.
  - Data Definition Language (DDL):
    - Used for defining database schema (A schema is the overall layout of the database such as what columns and rows it will have, what data types are they going to be able to store.)
  - Data Control Language (DCL):
    - Used for controlling access to the data in the database.
    - Users and permissions management.
  - Data Manipulation Language (DML):
    - Used for inserting, updating and deleting data from the database.

3.2. Queries

- A query is a set of instructions given to the RDBMS (written in SQL) that tell the RDBMS what information you want it to retrieve for you.
  - There are TONS of data in a database.
  - Often hidden in a complex schema.
  - The goal is to only get the data you need.
- Example: if we want to query the ages and the names of employees in a company that make more than 30000.

```sql
SELECT employee.first_name, employee.last_name
FROM employee
WHERE employee.salary >30000;
```

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

Capítulo 4

Creating Tables

4.1. Required software

- MySQL: is a relational database management system.
  - We can set a mySQL database server, which is a server were when it is running you can search for things in the database and perform the usual CRUD operations.
- popsql: is an IDE, we will use this to have ease of use with the mySQL server.

4.2. Data types

- There are lots of data types you can use on mySQL.
- The most common are:

`INT`	Whole number.
`DECIMAL(M,N)`	Decimal Numbers - Exact value. The (whole number part, how many decimals it will have).
`VARCHAR(1)`	String of text of length 1. Inside the parenthesis you put the length of the string you want, for example (100) means 100 characters.
`BLOB`	Binary Large Object, Stores large data, such as images or files.
`DATE`	'YYYY-MM-DD'
`TIMESTAMP`	'YYYY-MM-DD HH:MM:SS' Used for recording when something happened.

4.3. Creating a table

- To create a table we must instruct the program which data types the database will store.
- Writing things in all caps is optional but as convention it is preferred because it makes it easy to distinguish SQL from other elements.
- Every command in SQL needs to be terminated with a semicolon.
- Create a table:

```
CREATE TABLE student (
    -- columns:
    student_id INT PRIMARY KEY,
    name VARCHAR(20),
    major VARCHAR(20)
);
```

- You can define the primary key after table creation:

```
CREATE TABLE student (
    -- columns:
    student_id INT,
    name VARCHAR(20),
    major VARCHAR(20)
    PRIMARY KEY(student_id)
);
```

- You can use the DESCRIBE to list the table elements:

```
DESCRIBE student;
```

- To delete the table you can use DROP TABLE:

```
DROP TABLE student;
```

- To add a column:

```
ALTER TABLE student ADD gpa DECIMAL(3,2); -- store student's GPA
```

- To delete a column:

```
ALTER TABLE student DROP COLUMN gpa;
```

Capítulo 5

Inserting Data

- Let's create the following table:

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

- Use the following code:

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

DESCRIBE TABLE student;

-- eliminate the table.
DROP TABLE student;

-- add another column called gpa.
ALTER TABLE student ADD gpa DECIMAL;

-- eliminate the column gpa.
ALTER TABLE student DROP COLUMN gpa;
```

5.1. Insert data into the database

- Using the same table as the previous section.
- In order to insert a piece of information to a database in SQL type the command:

```sql
INSERT INTO student VALUES(1,'Jack','Biology');
INSERT INTO student VALUES(2,'Kate','Sociology');
```

- The syntax is:

```
INSERT INTO <table> VALUES(<values list in order>);
```

- Use the `SELECT *` command to retrieve anything from the table:

```sql
SELECT * FROM student;
```

student_id	name	major
1	Jack	Biology
2	Kate	Sociology
NULL	NULL	NULL

- Let's say we have a student of which we do not know the major. In this case we want to modify specific attributes we must use the following syntax:

```
INSERT INTO student(<student_id, name>) VALUES (<new student_is, new name>);
```

  - In this case since we want register a student into the database but don't know what major she is studying then we will leave that attribute as null.

```sql
INSERT INTO student(student_id, name) VALUES (3, 'Claire');
```

student_id	name	major
1	Jack	Biology
2	Kate	Sociology
3	Claire	NULL
NULL	NULL	NULL

- SQL will not allow you to enter duplicate keys if the key entered is a primary key, remember the primary key needs to be unique to each object.

Capítulo 6

Constraints

6.1. NOT NULL, UNIQUE

- `NOT NULL` keyword is used to describe an attribute in a table which cannot be null, it has to have a value.
- `UNIQUE` keyword is used to say that that attribute needs to be unique in that column or attribute:

```
USE giraffe;
CREATE TABLE student (
    student_id INT,
    name VARCHAR(20) NOT NULL, -- This attribute cannot be null.
    major VARCHAR(20) UNIQUE, -- Whatever values this attribute hlds must be
    ↪  unique among all others.
    PRIMARY KEY(student_id)
);

INSERT INTO student VALUES(1, 'Jack', 'Biology'); -- OK
INSERT INTO student VALUES(2, 'Kate', 'Sociology'); -- OK
INSERT INTO student VALUES(3, NULL, 'Chemistry'); -- Error Code: 1048. Column
↪  'name' cannot be null.
INSERT INTO student VALUES(4, 'Jack', 'Biology'); -- Error Code: 1062. Duplicate
↪  entry 'Biology' for key 'student.major'
```

- A primary key can be thought of as an attribute that is `NOT NULL` and `UNIQUE`.
- You can use the `DEFAULT` to define default values to attributes in the situation no in which no value is provided:

```
CREATE TABLE student (
    student_id INT,
    name VARCHAR(20) ,
    major VARCHAR(20) DEFAULT 'undecided',
    PRIMARY KEY(student_id)
);

INSERT INTO student(student_id, name) VALUES(1, 'Jack'); -- 1, Jack, undecided
```

- You can use the `AUTOINCREMENT` keyword to make a primary automatically increment when an object is added:

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

INSERT INTO student(name, major) VALUES('Jack', 'Biology'); -- 1,  Jack, Biology
INSERT INTO student(name, major) VALUES('Kate', 'Sociology'); -- 2, Kate, Sociology
```

- Notice that we obtained the primary keys of 1 and 2 despite not telling SQL explicitly.

Capítulo 7

Update & Delete

7.1. Updating and deleting rows in sql databases

- Starting off from the following table:

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

INSERT INTO student(name, major) VALUES('Jack', 'Biology'); -- 1,  Jack, Biology
INSERT INTO student(name, major) VALUES('Kate', 'Sociology'); -- 2, Kate, Sociology
INSERT INTO student(name, major) VALUES('Claire', 'Chemistry'); -- 3, Claire,
↪  Chemistry
INSERT INTO student(name, major) VALUES('Jack', 'Biology'); -- 4, Jack, Biology
INSERT INTO student(name, major) VALUES('Mike', 'Computer Science'); -- 5, Mike,
↪  Computer Science
```

- Lets say that the major in biology officially changed in name to Bio, and we want our database to update and call them “bio” instead of “Biology”.

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

- You can use boolean logic to update:

```sql
UPDATE student
SET major = 'Comp Sci'
WHERE major = 'Bio' OR major = 'Chemistry';
```

- The `WHERE` statement is optional, if you omit it what ever you are doing in the `SET` statement is going to be applied to everything in the table.

```sql
UPDATE student
SET major = 'undecided';
-- sets the major of every student in the table to 'undecided'.
```

- `DELETE FROM` statements allows you to eliminate a specific row, if you don't specify a condition to decide which rows get eliminated, the whole table will be cleared.

```
DELETE FROM student WHERE major = 'Biology'; -- eliminates all the biology majors
↪  in the table.
```

Capítulo 8

Basic Queries

8.1. Basic Queries

- SELECT keyword allows us to ask the database management system for a particular piece of information, it is like a google search.
- When you want to select everything you just put an *.
- Example:

```
SELECT major FROM student; -- gets all the majors that exist in the table.
SELECT name FROM student; -- gets all the names that exist in the table.
SELECT name, major FROM student; -- gets all the names and majors that exist in
↪  the table
SELECT student.name, student.major FROM student; -- gets the names and the majors.
```

- The following orders the query results by name, in alphabetical order.

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

- Order by query results in descending order, in this case reverse alphabetical order.

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

- The following orders by major and if they have the same major then sql will order them according to student_id:

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

- You can also limit the amount of results, in this case 2:

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

- To select the name and major if the major is chemistry or biology:

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

- To select all people with major not equal to chemistry:

```
SELECT name, major FROM student WHERE major <> 'Chemistry'; -- <> means not equals.
```

- Select all people with student id less than 3:

```
SELECT * FROM student WHERE student_id < 3;
```

- Select everything from the table where the name is Clair, Kate or Mike: Then select all where the majors are biology or chemistry:

```
SELECT * FROM student WHERE name IN ('Clair', 'Kate', 'Mike');
SELECT * FROM student WHERE major IN ('Biology', 'Chemistry');
```

- Writing queries gets more complex as the database schema gets more complex.

Capítulo 9

Creating Company Database

9.1. Complicated Company Database Schema

Company Database

Employee

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

Branch

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

Client

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

Works_With

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

Branch Supplier

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

Labels

	Primary Key
	Foreign Key
	Attribute

9.2. Coding the schema of the company

```
use giraffe;
create database giraffe;
drop database giraffe;
CREATE TABLE employee (
    emp_id INT PRIMARY KEY,
    first_name VARCHAR(40),
    last_name VARCHAR(40),
    birth_date DATE,
    sex VARCHAR(1),
    salary INT,
    super_id INT,
    branch_id INT
);
CREATE TABLE branch (
    branch_id INT PRIMARY KEY,
    branch_name VARCHAR(20),
    mgr_id INT,
    mgr_start_date DATE,
    FOREIGN KEY(mgr_id) REFERENCES employee(emp_id)
    ON DELETE SET NULL
);

-- Make employee.super_id and employee.branch_id foreign keys,
    -- we cannot do that just yet
ALTER TABLE employee
ADD FOREIGN KEY(branch_id)
REFERENCES branch(branch_id)
ON DELETE SET NULL;

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

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

```sql
    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
);

-- inserting the information
-- Corporate branch.
INSERT INTO employee VALUES(100,'David','Wallace','1967-11-17','M',350000,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 branch.
INSERT INTO employee VALUES(102, 'Michael', 'Scott', '1954-03-15','M',75000,100,NULL);
INSERT INTO branch VALUES(2, 'Scranton',102,'1992-04-06');
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 branch.
INSERT INTO employee VALUES(106, 'Josh', 'Porter', '1959-09-05','M',78000,100,NULL);
INSERT INTO branch VALUES(3, 'Scramford',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 Utensil');
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 Utensil');
INSERT INTO branch_supplier VALUES(3,'Hammer Mill', 'Paper');
INSERT INTO branch_supplier VALUES(3,'Stamford Lables', 'Custom Forms');

-- WORKS WITH TABLE
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);
```

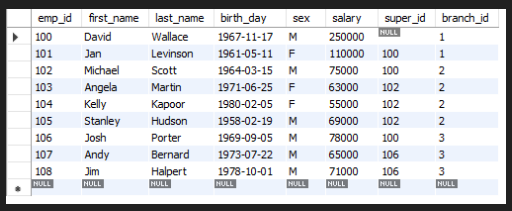
Capítulo 10

More Basic Queries

10.1. SELECT statements and prompts

- Find all employees:

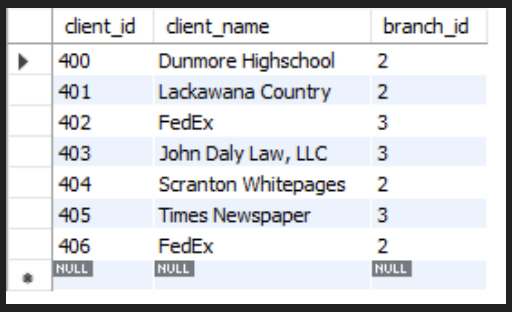
```
SELECT * FROM employee;
```



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
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

- Find all the clients:

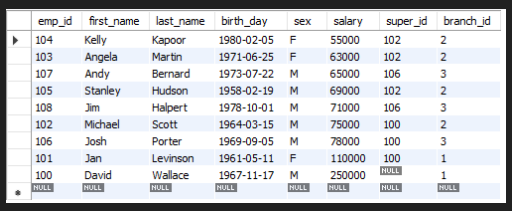
```
SELECT * FROM client;
```



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
NULL	NULL	NULL

- Find all employees ordered by salary:

```
SELECT * FROM employee ORDER BY salary;
```



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
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

- Find all employees ordered by salary from largest to smallest and then smallest to largest:

```
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
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

```
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
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

- Order all employees ordered by sex and 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
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

- 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
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

- Find the first and last names of all employees:

```
SELECT first_name, 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 of all employees:

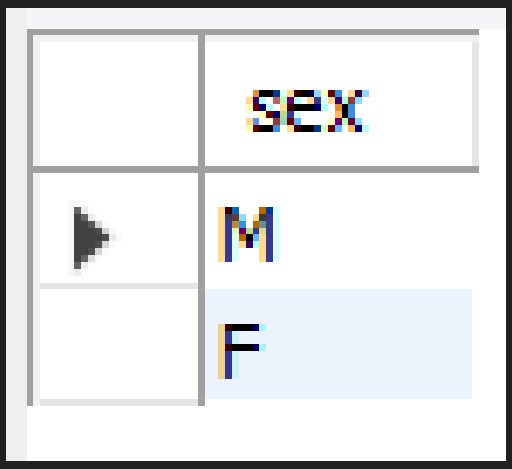
```
1 SELECT first_name AS forename, last_name AS surname FROM employee;
```

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

  - `AS` Allows you to select the columns differently to their names.

- Find out all the different genders:

```
1 SELECT  DISTINCT sex FROM employee;
```



- `DISTINCT` allows you to know all the values stored in a column.

- Find all male employees:

```
SELECT * FROM employee WHERE sex = 'M';
```

emp_id	first_name	last_name	birth_day	sex	salary	super_id	branch_id
100	David	Wallace	1967-11-17	M	250000	NULL	1
102	Michael	Scott	1964-03-15	M	75000	100	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 employees at branch 2:

```
SELECT * FROM employee WHERE branch_id = 2;
```

emp_id	first_name	last_name	birth_day	sex	salary	super_id	branch_id
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

- Find all employee's id's and names who were born after 1969:

```
SELECT emp_id, first_name, last_name FROM employee WHERE birth_day >= 1970-01-01;
```

	emp_id	first_name	last_name
▶	100	David	Wallace
	101	Jan	Levinson
	102	Michael	Scott
	103	Angela	Martin
	104	Kelly	Kapoor
	105	Stanley	Hudson
	106	Josh	Porter
	107	Andy	Bernard
	108	Jim	Halpert
*	NULL	NULL	NULL

- Find all female employees at branch 2:

```
SELECT * FROM employee WHERE branch_id = 2 AND sex = 'F';
```

	emp_id	first_name	last_name	birth_day	sex	salary	super_id	branch_id
▶	103	Angela	Martin	1971-06-25	F	63000	102	2
	104	Kelly	Kapoor	1980-02-05	F	55000	102	2
*	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

- Find all employees who are female & born after 1969 or who make over 80000:

```
SELECT * FROM employee WHERE (birth_day >= '1970-01-01' AND sex = 'F') OR salary >
↪  80000;
```

	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	102	2
	104	Kelly	Kapoor	1980-02-05	F	55000	102	2
*	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

- Find all employees born between 1970 and 1975:

```
SELECT * FROM employee WHERE birth_day BETWEEN '1970-01-01' AND '1975-01-01';
```

	emp_id	first_name	last_name	birth_day	sex	salary	super_id	branch_id
▶	103	Angela	Martin	1971-06-25	F	63000	102	2
	107	Andy	Bernard	1973-07-22	M	65000	106	3
*	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

- Find all employees named Jim, Michael, Johnny or David:

```
SELECT * FROM employee WHERE first_name IN ('Jim', 'Michael', 'Johnny', 'David');
```

	emp_id	first_name	last_name	birth_day	sex	salary	super_id	branch_id
▶	100	David	Wallace	1967-11-17	M	250000	NULL	1
	102	Michael	Scott	1964-03-15	M	75000	100	2
	108	Jim	Halpert	1978-10-01	M	71000	106	3
*	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

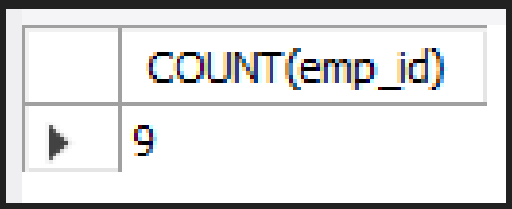
Capítulo 11

Functions

11.1. Prompts to functions

- Find the number of employees are inside the employee table:

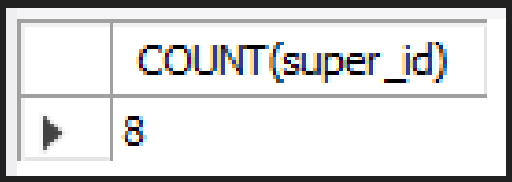
```
1 SELECT COUNT(emp_id) FROM employee;
```



	COUNT(emp_id)
▶	9

- Find how many employees have supervisors:

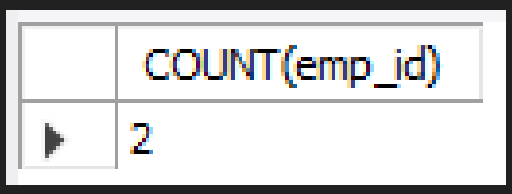
```
1 SELECT COUNT(super_id) FROM employee;
```



	COUNT(super_id)
▶	8

- Find the number of female employees born after 1970:

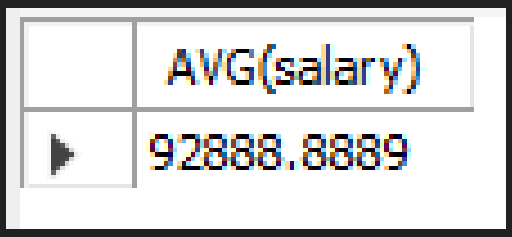
```
1 SELECT COUNT(emp_id) FROM employee WHERE sex = 'F' AND birth_day > '1971-01-01';
```



	COUNT(emp_id)
▶	2

- Find the average of all employee's salaries:

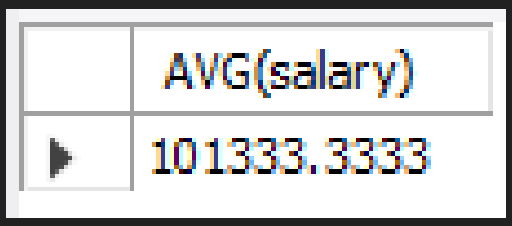
```sql
SELECT AVG(salary) FROM employee;
```



	AVG(salary)
▶	92888.8889

- Find the average of all employee's salaries who are male:

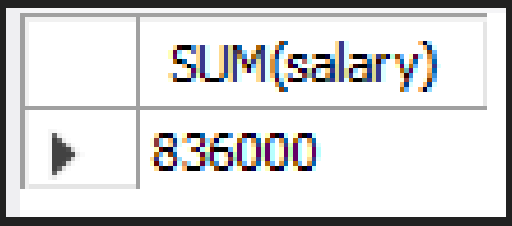
```sql
SELECT AVG(salary) FROM employee WHERE sex = 'M';
```



	AVG(salary)
▶	101333.3333

- Find the sum of all employee's salaries:

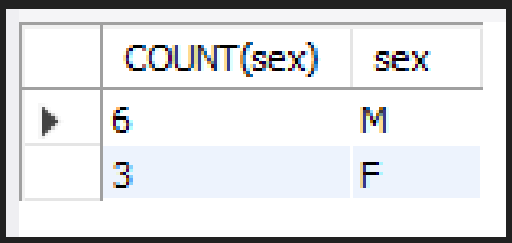
```sql
SELECT SUM(salary) FROM employee;
```



	SUM(salary)
▶	836000

- Find out how many males and how many females there are:

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



	COUNT(sex)	sex
▶	6	M
	3	F

  - This is called aggregation.
  - The `GROUP BY` allows us to display the count(sex) and sex in columns so that we can see what number belongs to what.

- Find the total sales of each salesman:

```
SELECT SUM(total_sales), emp_id FROM works_with GROUP BY emp_id;
```

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

- Find the total purchases made from clients:

```
SELECT SUM(total_sales), emp_id FROM works_with GROUP BY client_id;
```

	SUM(total_sales)	emp_id
▶	55000	105
	267000	102
	22500	108
	17000	107
	33000	105
	26000	107
	145000	102

Capítulo 12

Wildcards

12.1. Wildcards

- Ways of grabbing data that matches a specific pattern.
- Find any client's who are an LLC:

```
SELECT * FROM client WHERE client_name LIKE '%LLC';
```

client_id	client_name	branch_id
403	John Daly Law, LLC	3
NULL	NULL	NULL

  - The condition will be true if the last three characters of the client name are LLC, the % wildcard allows you to not specify from which character index you must start.

- Use characters to define a 'template' for pattern matching.
  - % means any number of characters.
  - _ means one character.
- 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
NULL	NULL	NULL

- Find any employee born in october:

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

emp_id	first_name	last_name	birth_day	sex	salary	super_id	branch_id
108	Jim	Halpert	1978-10-01	M	71000	106	3
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

- Find any clients who are schools:

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

client_id	client_name	branch_id
400	Dunmore Highschool	2
NULL	NULL	NULL

Capítulo 13

Union

13.1. Unions

- Unions are used to combine the results of several select statements into one.

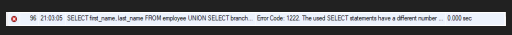
- Return the first name of the employees and the branches:

```
SELECT first_name FROM employee UNION
SELECT branch_name FROM branch;
```

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

- When using unions the number of columns selected in each select statement needs to be the same in terms of quantity:

```
SELECT first_name, last_name FROM employee UNION
SELECT branch_name FROM branch;-- error
```



- They need to be of similar data types.
- The union can be performed with lots of select statements:

```
SELECT first_name FROM employee UNION
SELECT branch_name FROM branch UNION
SELECT client_name FROM client;
```

	first_name
▶	David
	Jan
	Michael
	Angela
	Kelly
	Stanley
	Josh
	Andy
	Jim
	Corporate
	Scranton
	Stamford
	Dunmore ...
	Lackawan...
	FedEx
	John Daly...
	Scranton ...
	Times Ne...

- Find a list of all clients and 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 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
	5000
	26000
	22500
	12000

Capítulo 14

Joins

- Joins are used to combine rows from two or more tables based on a related column between them.
- Used to combine tables to consolidate a result.
- We want to know the names and employee id of each of the branches.

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

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

- Left joins:
  - On a left join all of the table is included but only the one that matches.

```sql
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

- Right join:

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

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

- There is an outer join but this is not available in mysql.

Capítulo 15

Nested Queries

- Nested query is when you use multiple select statements within select statements to get specific information.

- Find names of all employees who have sold over 30,000 to a single client:

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

	first_name	last_name	emp_id
▶	Michael	Scott	102
	Stanley	Hudson	105
*	NULL	NULL	NULL

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

```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

Capítulo 16

On Delete

- How to delete entries when they have foreign keys linked to them.
- ON DELETE SET NULL is basically when we delete an employee all foreign associated to that employee is going to be set to null.
- ON CASCADE is basically when we delete an employee the rows in other tables containing employee ids pointing to the deleted employee are also going to get eliminated.
- The code below signifies that if the employee id in the employee table gets deleted the value of that foreign key is set to 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
);
```

- You might want to consider using set null when the attribute being deleted is no essential, if it is essential then use cascade.

Capítulo 17

Triggers

- A trigger is basically is a block of SQL code that will define what needs to happen when a certain operation gets performed on a database.
- Lets create a table called `trigger_test` to store the trigger calls.

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

- Making a trigger needs to be done from the command line.

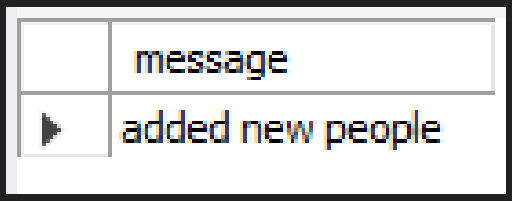
```
DELIMITER $$
CREATE
    TRIGGER my_trigger BEFORE INSERT
    ON employee
    FOR EACH ROW BEGIN
        INSERT INTO trigger_test VALUES('added new people');
    END$$
DELIMITER ;
```

```
Enter password: ********
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 10
Server version: 8.0.22 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> DELIMITER $$
mysql> CREATE
    -> TRIGGER my_trigger BEFORE INSERT
    -> ON employee
    -> FOR EACH ROW BEGIN
    ->   INSERT INTO trigger_test VALUES('added new people');
```

```
    -> END$$
Query OK, 0 rows affected (0.03 sec)
mysql> DELIMITER ;
```

- Now we can see a message whenever we make the specified action, in this case inserting Oscar Martinez and then using select statements see the registered actions in the table "trigger_test".

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



	message
▶	added new people

- We now want to store the first name in the trigger test table.

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

```
Enter password: ********
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 10
Server version: 8.0.22 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> DELIMITER $$
mysql> CREATE
    -> TRIGGER my_trigger1 BEFORE INSERT
    -> ON employee
    -> FOR EACH ROW BEGIN
    ->     INSERT INTO trigger_test VALUES(NEW.first_name);
    -> END$$
Query OK, 0 rows affected (0.03 sec)
mysql> DELIMITER ;
```

- Using select statements we can see the registry the trigger has left when inserting Kevin Malone.

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

	message
▶	added new people
	added new people
	Kevin

- Example:

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

```
Enter password: ********
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 10
Server version: 8.0.22 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> 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$$
Query OK, 0 rows affected (0.03 sec)
mysql> DELIMITER ;
```

- Using select statements we can see the registry the trigger has left when inserting Pam Beesly.

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

	message
▶	added new people
	added new people
	Kevin
	added new people
	Pam
	added female

- Use `DROP TRIGGER` command to eliminate the trigger.

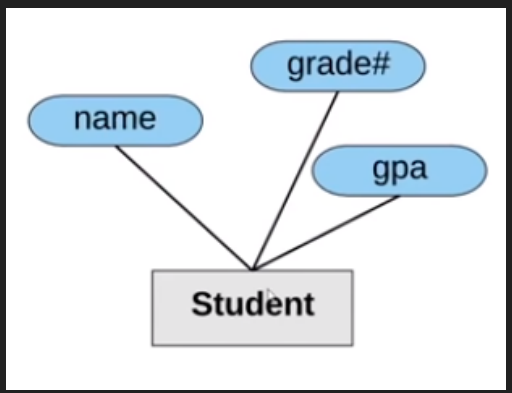
Capítulo 18

ER Diagrams Intro

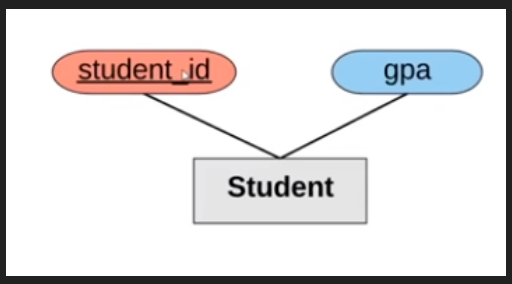
To design a database schema we must first understand it, for understanding we design ER programs, they stand for Entity Relationship Diagrams. An ER diagram would be absolutely helpful in defining relationships so that we can understand the relations, object and attributes associated with each table in our database. ER diagrams act as a middle man between the requirements and the actual schema that gets implemented.

- ER diagrams consists of different shapes and symbols that establish relationships.
- An entity is an object we want to model and store information about. In ER diagrams they are modeled using a square or rectangle.
- ER Diagrams: used to design the database, it is used to model an entity.
- An entity is depicted as a square, in this example the entities we want to model are students.

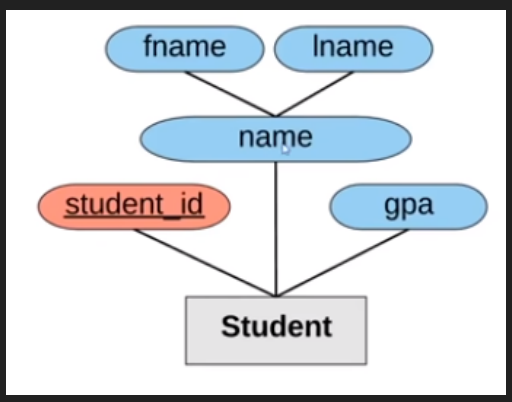
- An attribute is a specific piece of information about an entity. In ER diagrams they are modeled using ovals connected to entity.



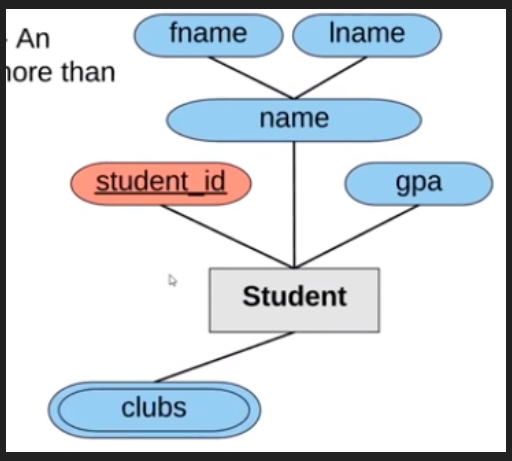
- Primary key: an attribute that uniquely identifies an entry in the database table. Modeled inside an oval, the primary key name is underlined.



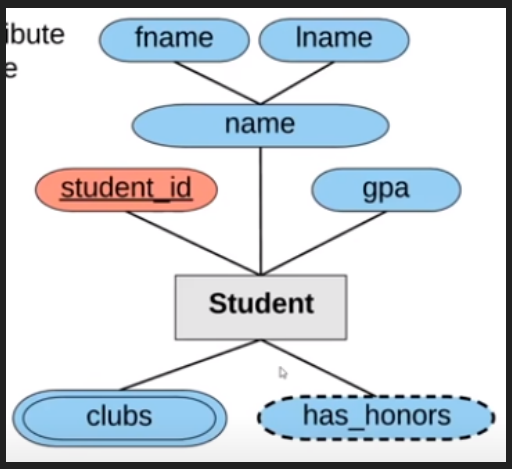
- Composite attributes: Attributes that can be broken up into sub-attributes. In this example the name attribute can be broken up further into first name and last name, this is a composite attribute.



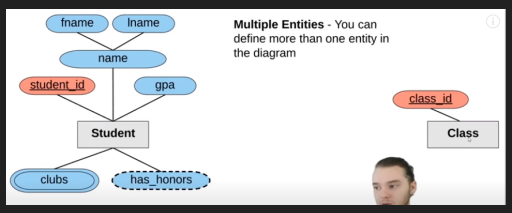
- Multi-valued attribute: an attribute that can have more than one value. Modeled with an oval with an inner oval. In this case "clubs" is a multi-values attribute because a student could be involved in more than one club or zero, it can have more than one value, they are not going to have more than one gpa or more than one name, but they can be involved in more than one club.



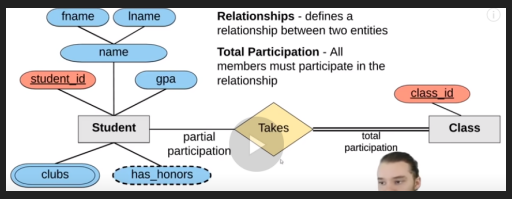
- Derived attribute: an attribute that can be derived from the other attributes, modeled using oval with dashed lines. A derived attribute is not stored in the database, they can be figured out whenever they are needed but for the sake of efficient memory usage they are not stored because they can be derived whenever they are needed. Has honors can be derived from the gpa.



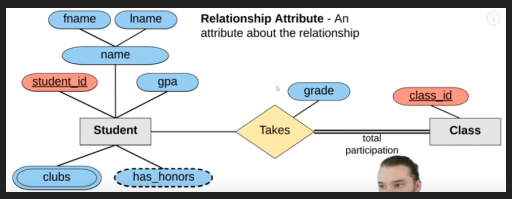
- Multiple entities: you can define more than one entity in the diagram. We can have multiple entities and establish relations between them, in this case lets define an entity called class. Notice the class entity has a primary key called class id.



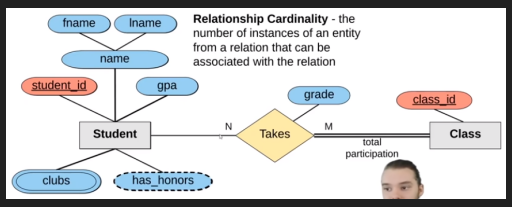
- Relationship: defines a relationship between two entities. Modeled using lines and a diamond shape. Using a double line means that task has total and one line means partial. When we have two or more entities we can define relationships between those two entities, here lets establish a relationship between student and class:
  - A relationship is denoted by the diamond, in this case "takes", the lines connecting the entities to the relationship also hold particular information.
  - The relationship here is that a student can take a class and a class can be taken by a student.
  - In many ways relationships are denoted as verbs, and they can go both ways, in this case you can say the student takes a class and the class takes a student.
  - One line means partial participation, meaning that not all students need to take a class.
  - Two lines means total participation, meaning that all the classes need to be taken by at least a minimum of students, it doesn't make sense to have a class that has zero students taking it.



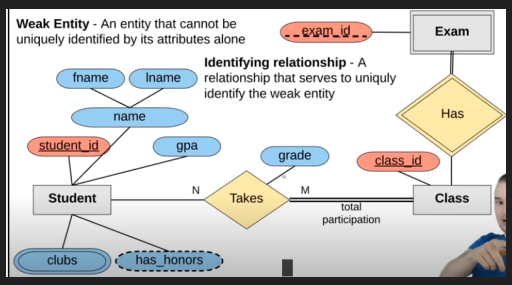
- Relationship attribute: an attribute about the relationship. In this case there is an attribute on the relationship, the entity student takes a class, while he takes the class the entity student will receive a grade, this only happens when the student takes the class. It doesn't make sense to store in student grades on classes he never took or never will.



- Relationship cardinality: the number of instances of an entity from a relation that can associated with the relation. This means for our example that a student can take multiple classes, and we can define the same thing for the class, we can say the class is taken by any number of students. However, we can define cardinality relationships as the following:
  - 1:1, meaning that a student can take one class and a class can be taken one student. This means that the class can only have one student and the student can only take one class. Doesn't make very much sense in this example.
  - 1:N, N meaning a constant number of students, 1:N means that a student can take one class and a class can be taken by N number of students.
  - N:M, means that a student can take any number of classes and a class can be taken by any number of students.
  - This is relevant because it is useful in data modeling requirements, for example if you require your database to check that a student can only take one class at a time, then that is something you want to consider in your ER diagrams.

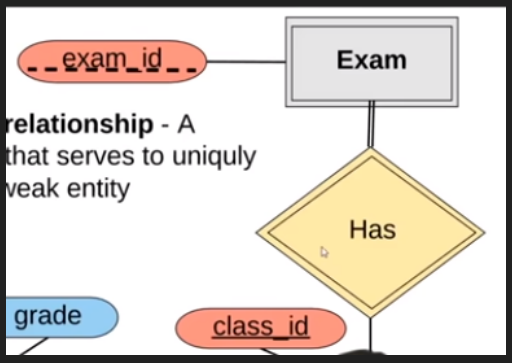


- Weak entity: an entity that cannot be uniquely identified by its attributes alone, an entity that will depend on another.
  - In this example a weak entity is an exam, notice that a class can have an exam, the exam is an entity, the exam has the primary key of exam id, but in this case an exam can't exist without a class, in order for an exam to exist it has to be associated with a class.



- Identifying relationship: a relationship that serves to uniquely identify the weak entity.
  - In this example the identifying relationship is the "has", an exam can be uniquely *identified* only if there is a class associated with it.
  - The exam does not exist on its own, it relies on a class for an identity.

- Whenever we have a weak entity and identifying relationship the weak entity always has to have total participation in the identifying relationship, all exams must have a class but not all classes need to have an exam.



Knowing this, we can take this ER diagram and convert it to a actual database schema.

Capítulo 19

Designing an ER Diagram

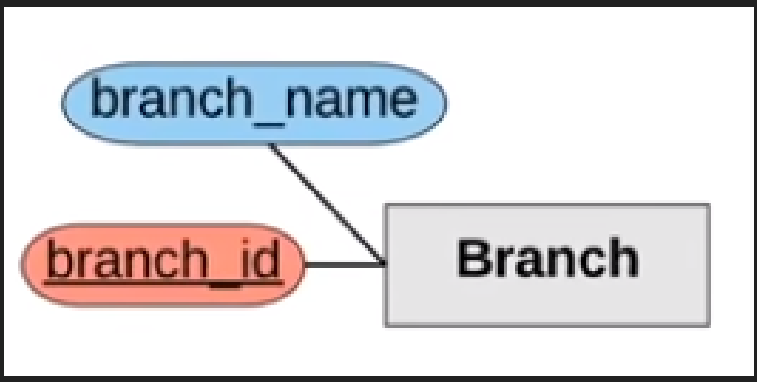
Using a Data Requirements document we can construct an ER diagram, take for example the following:

```
Company Data Requirements:
The company is organized into branches. Each branch has a unique number, a name, and a particular e

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

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

An employee can work for one branch at a time, and each branch will be managed by one of the employ

An employee can act as a supervisor for other employees at the branch, an employee may also act as

A branch may handle a number of clients, with each client having a name and a unique number to iden

Employees can work with clients controlled by their branch to sell them stuff. If necesary, multipl

Many branches will need to work with suppliers to buy inventory. For each supplier we'll keep track
```

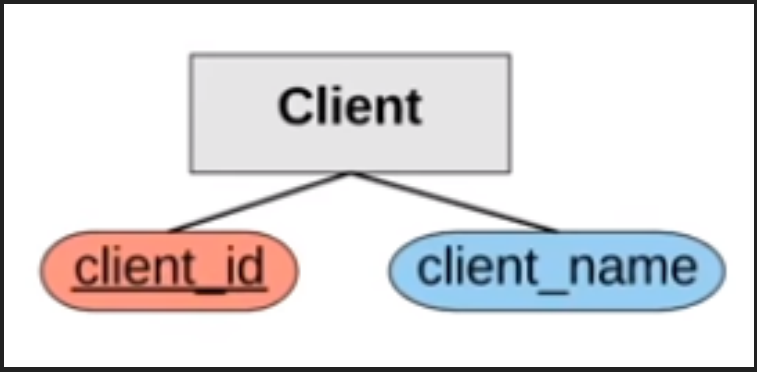
19.1. Conversion of the Data Requirements Document into an ER diagram

Each sentence conveys critical information, lets look at it one at a time and see how the ER diagram turns out:

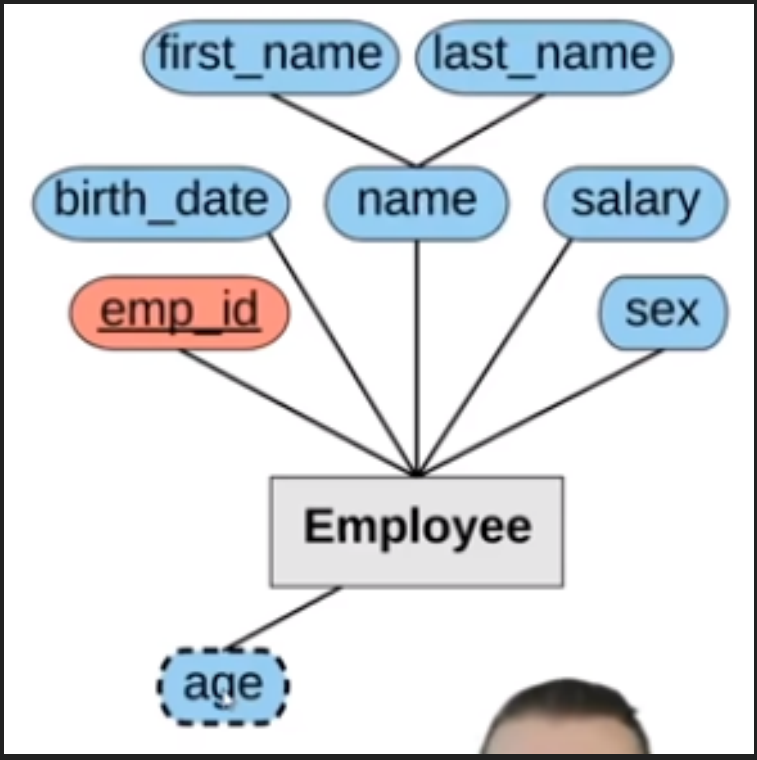
- **The company is organized into branches. Each branch has a unique number, a name, and a particular employee who manages it.**



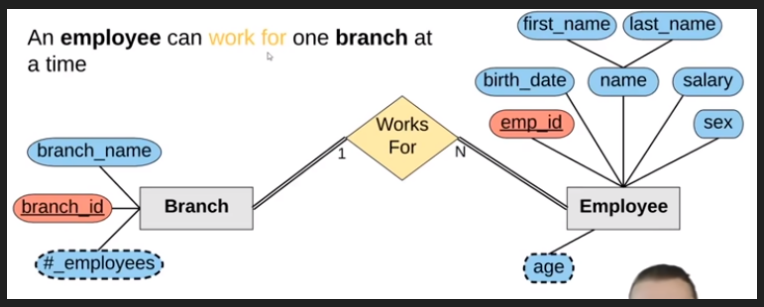
- **The company makes it's money by selling to clients. Each client has a name and unique number to identify it.**



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



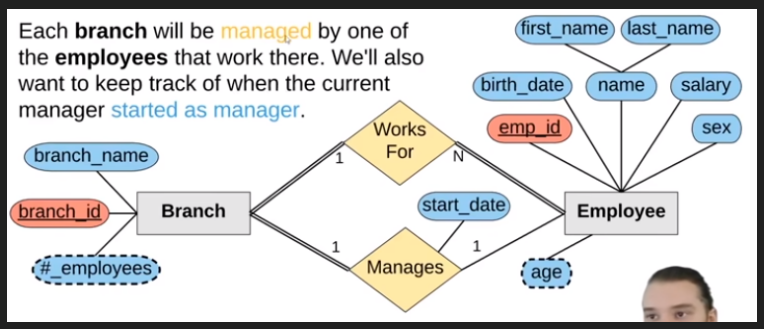
- **An employee can act as a supervisor for other employees at the branch, an employee may also act as the supervisor for employees at other branches. An employee can have at most one supervisor.**
  - Notice here the data is defining a relationship.
  - The relationship is the "work for".
  - All branches must have employees working for them, and all employees must work for a branch. This means a both way total participation.
  - An employee can work only for ONE branch, and a branch can have multiple employees, hence this is a 1:N cardinality relationship.



- **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.**

  - Notice the cardinality relationship here, a branch can be managed by one employee and an employee can manage one branch, its 1:1.

  - On the "manages" relationship we must store the attribute start date, this records when the manager started as manager.

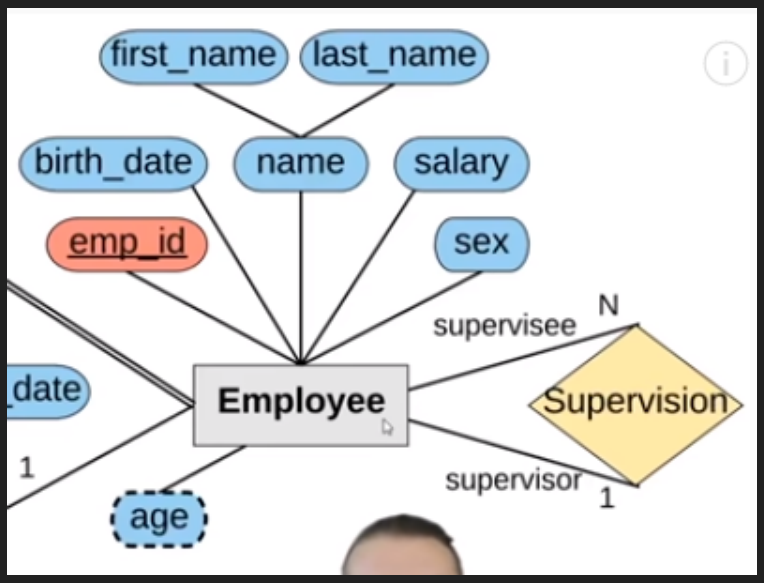


- **An employee can act as a supervisor for other employees at the branch, an employee may also act as the supervisor for employees at other branches. An employee can have at most one supervisor.**

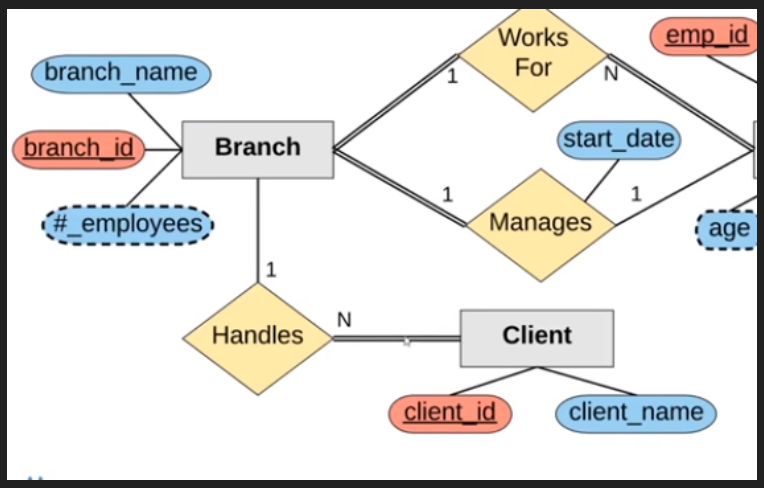
  - Here notice that we have a relationship that is with one's self, meaning that an employee can act as a supervisor to other employees, but the supervisor is himself an employee as well.

  - Lets call the supervisor relationship the "supervision" relationship, notice that it is comprised with a supervisor and the supervisee, an employee can have at most one supervisor, and a supervisor can have multiple people to supervise, all the while being an employee also.

  - The cardinality relationship is thus a 1:N relationship. Because an employee can have at most one supervisor but a supervisor can supervise several employees.

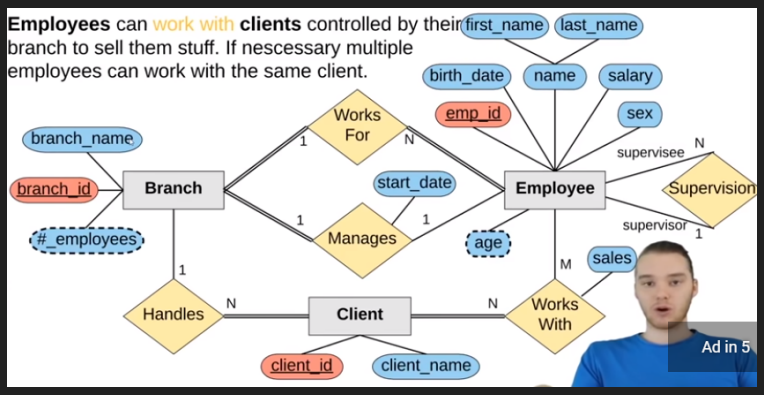


- **A branch may handle a number of clients, with each client having a name and a unique number to identify it. A single client may only be handled by one branch at a time.**
  - Now notice the relationship happening between the branch and the client, the client can be handled any number of times by a branch and a single client can only be handled by one branch at a time.
  - Notice the cardinality relationship, the branch can handle any number of clients but the client can only be handled by only one branch at a time.
  - Notice that the client has a total participation, because every client needs to be handled by a branch, but the branch has a partial participation because not all branches need to handle every client, in this case it needs to be one branch at a time.



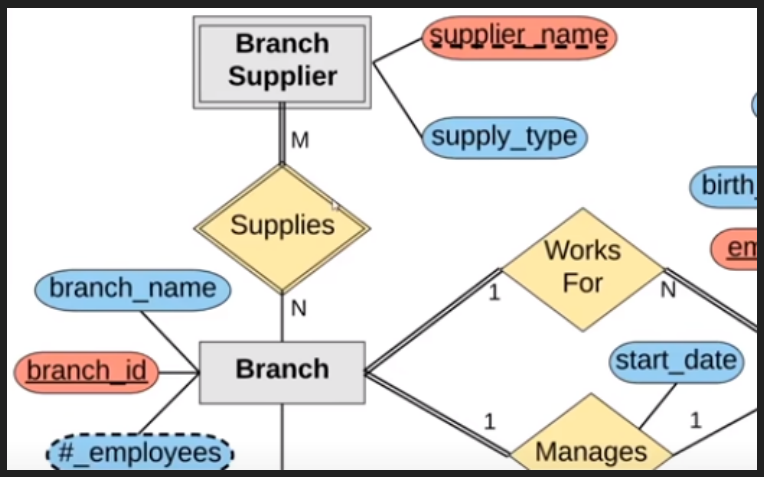
- **Employees can work with clients controlled by their branch to sell them stuff. If necesary, multiple employees can work with the same client. We'll want oto keep track of how many dollars worth of stuff each employee sells to each client they work with.**
  - Notice the cardinality, employees can work with clients, any number of clients and clients can have any number of employees working with them, It's an N:M cardinality relationship.
  - Notice here the relationship is the client and the employee "working" together, or "works with". When this relationship takes place a sale may take place, this is a relationship attribute "sales".

- Notice the participation, all clients must work with and interact with employees but not all employees must work with clients.

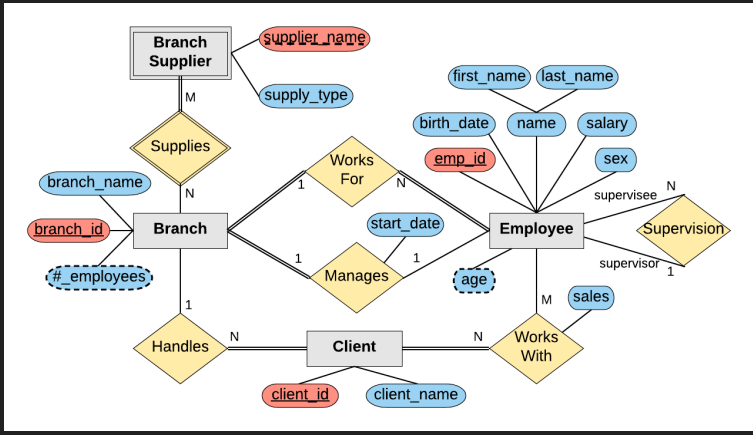


- **Many branches will need to work with suppliers to buy inventory. For each supplier we'll keep track of their name and the type of product they're selling the branch. A single suppler may supply products to multiple branches.**

  - This is a case where we would need to define a weak entity and weak relationship.
  - Notice the branch supplier supplies a specific branch or branches, we also want to keep track of which supplier is supplying with branch. Do this using the weak relationship "supplies".
  - Notice that there can be any number of suppliers per branch and any branches per supplier. The cardinality relationship is N:M.
  - Notice the participation, as always weak entities have total participation in the relationship and the string entity haves in this case partial participation, because not all branches must be supplied.



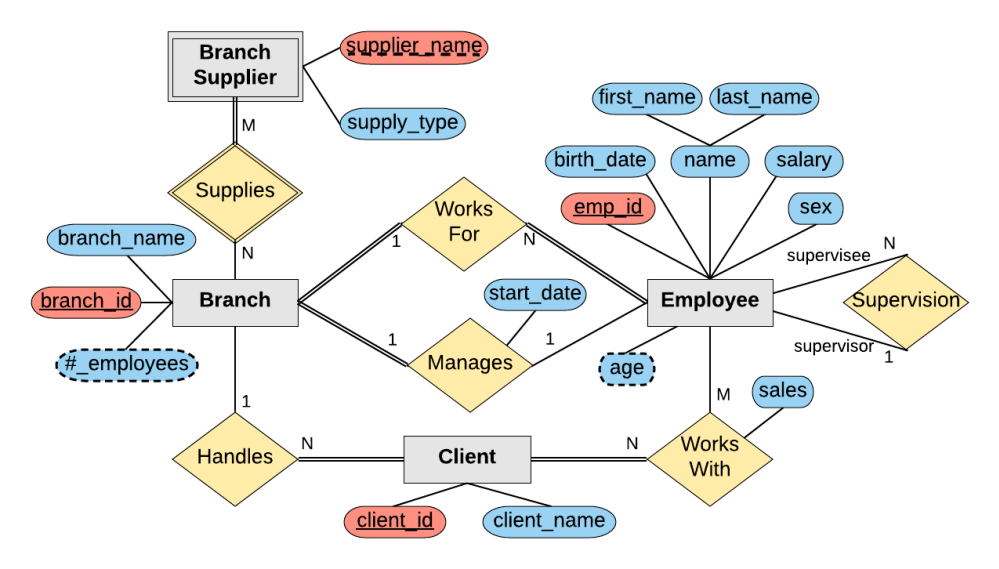
- The whole diagram looks like this:


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

Capítulo 20

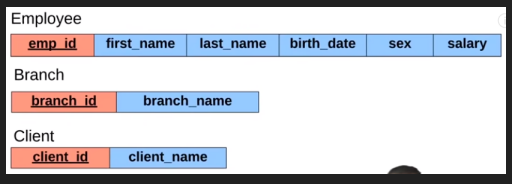
Converting ER Diagrams to Schemas

Converting the following into a schema:



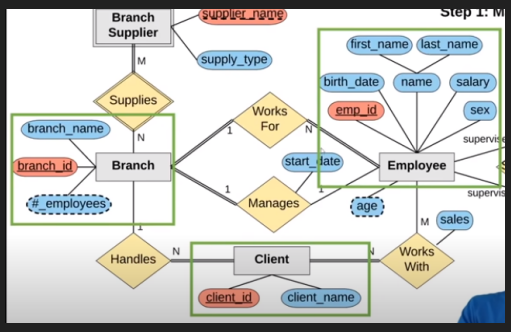
20.1. Steps

1. Mapping of regular entity types:



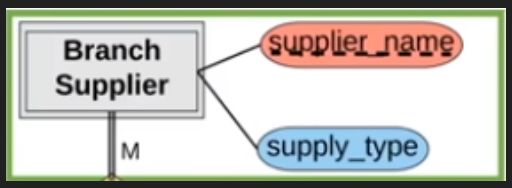
- For each regular entity type create a relation or table that includes all the simple attributes of that entity.
- In this case the regular entities are Branch, Employee, and Client. The columns of the table of the entities will be the simple attributes of the entities.
- Notice we don't mark the relationships between the entities nor the branch supplier as an entity because in this case the branch supplier is a weak entity.

- In composite attribute it is common to store the attributes that compose the composite attribute rather than storing it as a composite attribute. This means that for example name is composed of first name and last name, but we will not store it as name but rather as first and last name directly. Composite attributes are stored as subattributes.

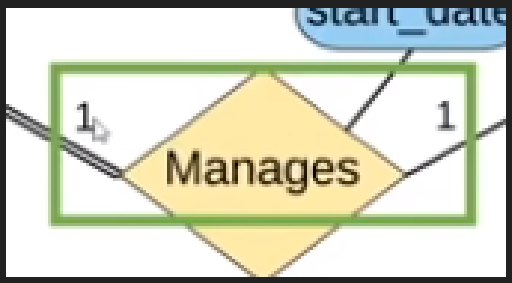


2. Mapping of weak entity types:

- For each weak entity type create a relation (table) that includes all simple attributes of the weak entity.
- The primary key of the weak entity of the new table should be the partial key of the weak entity plus the primary key of its owner. In this case the primary key will be branch id.



3. Mapping of Binary 1:1 relationship types:



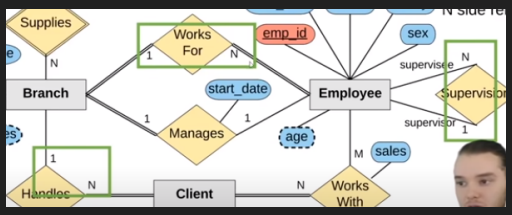
- For each one to one binary relationship we want to include one side of the relationship as a foreign key in the other favor total participation.
- Basically include the primary key of one of these entities as a foreign key in the other's entity relation. And if a particular entity has total participation in the relationship then you want to add the foreign key on to the one that has total participation, in this case to branch, add the employee primary key as a foreign key to branch.

- If both of them are total participation or both are partial participation then it is left to your discretion to decide which entity has the foreign key.
- To the table Branch, add the foreign key of employee, in this case called mgr_id will be foreign key to link the relationship.

Branch Supplier

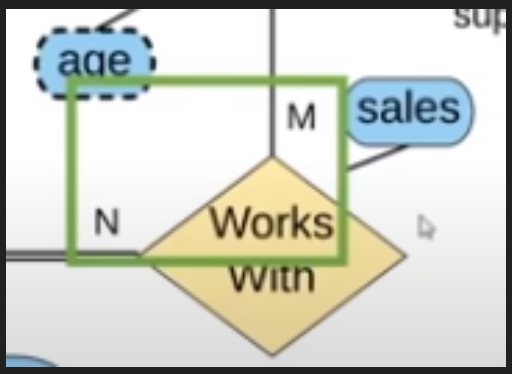
branch_id	supplier_name	supply_type

4. Mapping of binary 1:N relationship types:



- Include the 1 side's primary key as a foreign key in the N side relation table.
- In this case we have three.
- Basically, in the case where we establish the relationship between the branch works for employee, I want to include the 1's side primary key, which means the branch side, as a foreign key in the employee table, this means that employee will have a column as a foreign key specifying the branch.
- In the employee supervision relation, we grab the 1's side, which is employee and store in the N's side, which is employee, the foreign key to the supervisor.
- In the branch handles client relationship we store as a foreign key a branch in client.

5. Mapping of binary M:N relationship types:

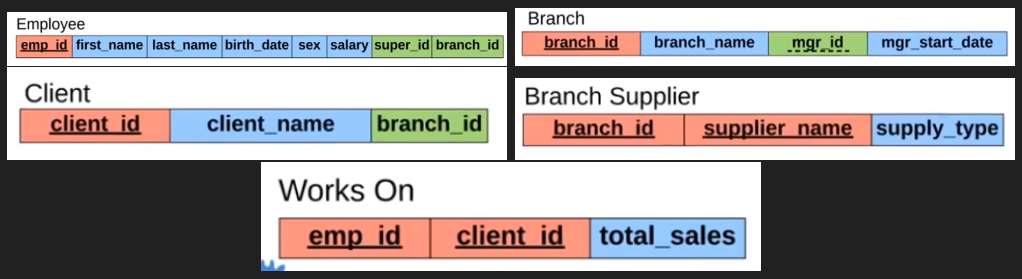


- Create a new table who's primary key is a combination of both entities' primary key's. Also include any relationship attributes.
- In this case store sales.

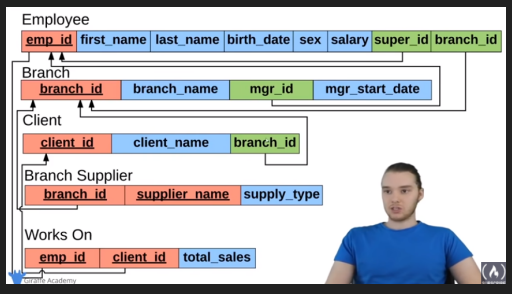
Works On

emp_id	client_id	total_sales

You should end up with the tables looking like this:



You can see the relationships like this:



Now we can populate the database in the way weve already learned.

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

Labels

	Primary Key
	Foreign Key
	Attribute