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.

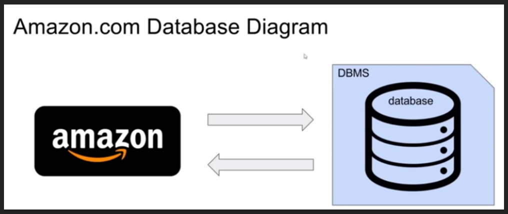
1.3. Computers with databases

- 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.



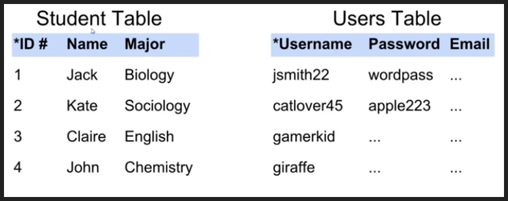
1.5. C.R.U.D

- 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

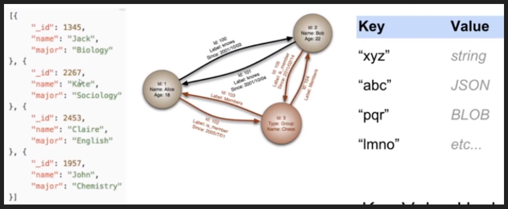
Users Table

*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:



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

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

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

- 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

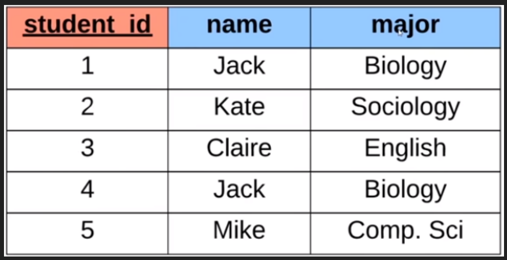
2.1. Primary keys, Surrogate and Natural keys

- 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.

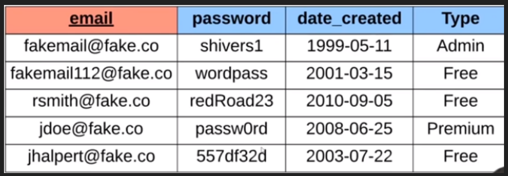
- Example:



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

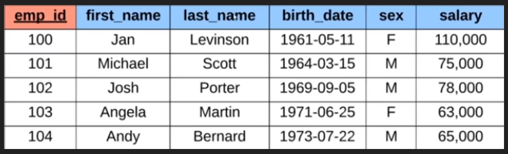
2.1.1. Another example

- Example:



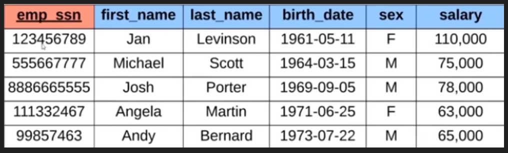
email	password	date_created	Type
fakemail@fake.co	shivers1	1999-05-11	Admin
fakemail112@fake.co	wordpass	2001-03-15	Free
rsmith@fake.co	redRoad23	2010-09-05	Free
jdoe@fake.co	passw0rd	2008-06-25	Premium
jhalpert@fake.co	557df32d	2003-07-22	Free

- 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.



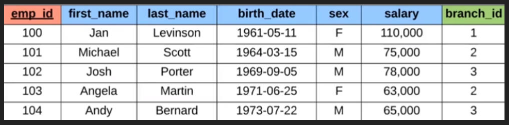
emp_ssn	first_name	last_name	birth_date	sex	salary
123456789	Jan	Levinson	1961-05-11	F	110,000
555667777	Michael	Scott	1964-03-15	M	75,000
8886665555	Josh	Porter	1969-09-05	M	78,000
111332467	Angela	Martin	1971-06-25	F	63,000
99857463	Andy	Bernard	1973-07-22	M	65,000

2.1.2. Difference between Surrogate and natural keys

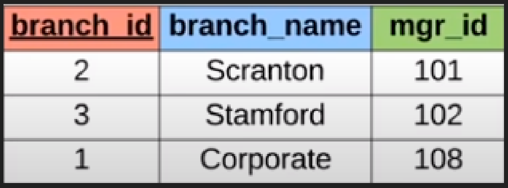
- Both are types of primary keys.
- A surrogate key serves only to identify a particular object inside a database such as an id or a hash, this key has no significance nor mapping outside the database.
- A natural key is a registration of an individual object using natural attributes, such as a social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



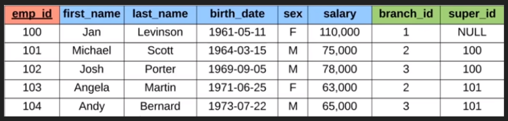
emp_id	first_name	last_name	birth_date	sex	salary	branch_id
100	Jan	Levinson	1961-05-11	F	110,000	1
101	Michael	Scott	1964-03-15	M	75,000	2
102	Josh	Porter	1969-09-05	M	78,000	3
103	Angela	Martin	1971-06-25	F	63,000	2
104	Andy	Bernard	1973-07-22	M	65,000	3



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.

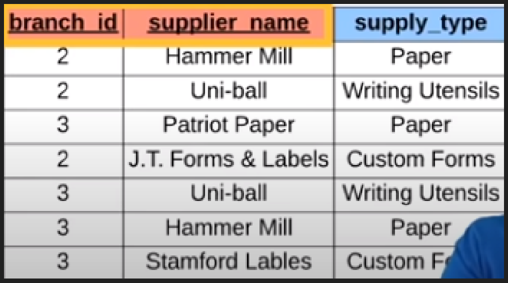
- A foreign key is a primary key inside another table.
- It is basically a way to define relationships with other tables.
- You can have multiple foreign keys:



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.

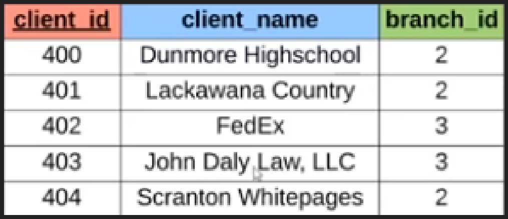
- Taking the same example, lets add another foreign key that denotes who the suppliers are for each branch:



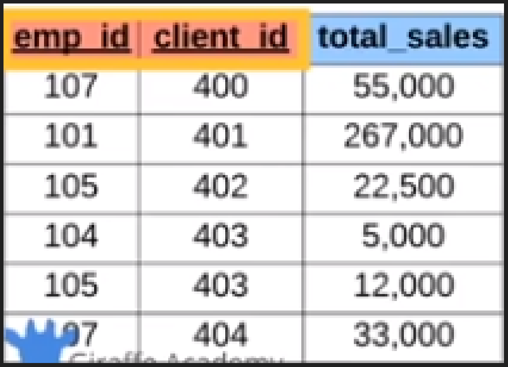
branch_id	supplier_name	supply_type
2	Hammer Mill	Paper
2	Uni-ball	Writing Utensils
3	Patriot Paper	Paper
2	J.T. Forms & Labels	Custom Forms
3	Uni-ball	Writing Utensils
3	Hammer Mill	Paper
3	Stamford Lables	Custom F

  - 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
07	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.

```
SELECT employee.name, employee.age
FROM employee
WHERE employee.salary > 30000;
```

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
`VARCHAR(1)`	String of text of length 1. Inside the parenthesis you put the length of the string you want, for exan
`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 happe

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:

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

- You can define the primary key after table creation:

```sql
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:

```sql
DESCRIBE student;
```

- To delete the table you can use DROP TABLE:

```sql
DROP TABLE student;
```

- To add a column:

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

- To delete a column:

```sql
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:

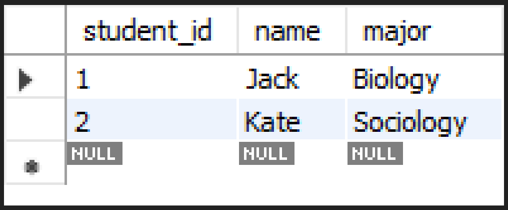
```
INSERT INTO student VALUES(1,'Jack','Biology');
INSERT INTO student VALUES(2,'Kate','Sociology');
```

- The syntax is:

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

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

```
SELECT * FROM student;
```



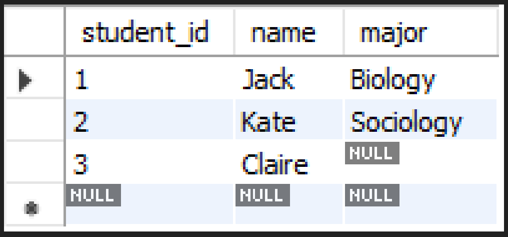
	student_id	name	major
▶	1	Jack	Biology
	2	Kate	Sociology
✱	NULL	NULL	NULL

- 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.

```
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 other
    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' fo
```

- 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:

```
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:

```
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”.

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

- You can use boolean logic to update:

```
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.

```
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:

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

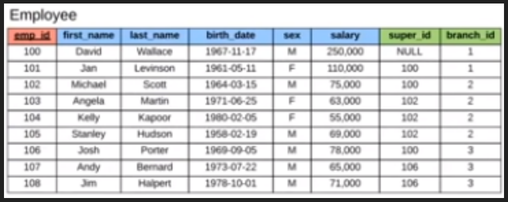
- Writing queries gets more complex as the database schema gets more complex.

Capítulo 9

Creating Company Database

9.1. Complicated Company Database Schema

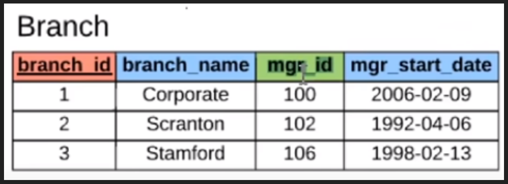
- Employee table:



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

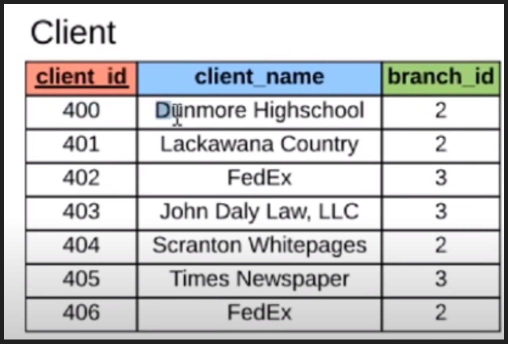
- Branch table:



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

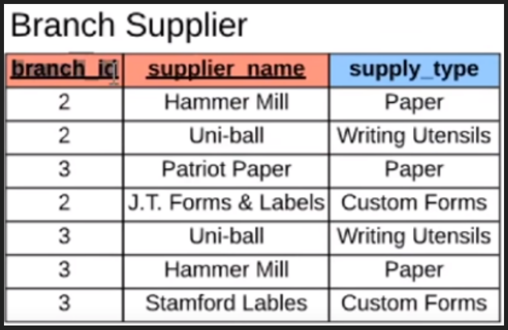
- Client table:



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

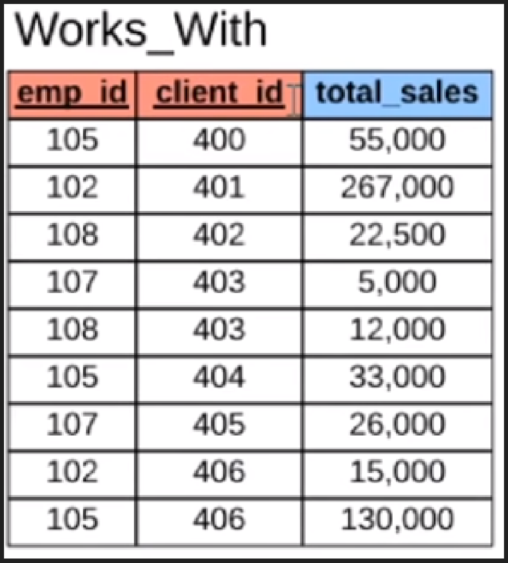
- Branch supplier:



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

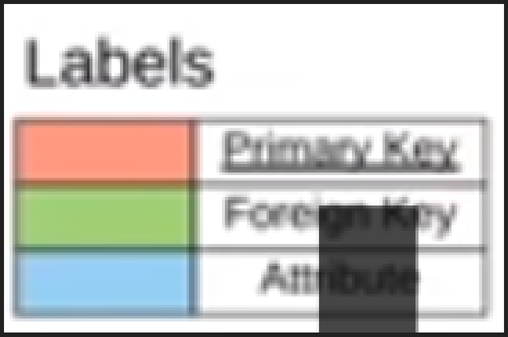
- Works with table:



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

- Labels:



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),
```

```sql
    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 (
    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;`

- Find all the clients:

  `SELECT * FROM client;`

- Find all employees ordered by salary:

  `SELECT * FROM employee ORDER BY salary;`

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

  `SELECT * FROM employee ORDER BY salary DESC;`

- Order all employees ordered by sex and then name:

  `SELECT * FROM employee ORDER BY sex, first_name, last_name;`

- Find the first 5 employees in the table:

  `SELECT * FROM employee LIMIT 5;`

- Find the first and last names of all employees:

  `SELECT first_name, last_name FROM employee;`

- Find the forename and surnames of all employees:

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

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

- Find out all the different genders:

  `SELECT  DISTINCT sex FROM employee;`

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

Capítulo 11

Functions

11.1. Prompts to functions

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

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

- Find how many employees have supervisors:

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

- Find the number of female employees born after 1970:

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

- Find the average of all employee's salaries:

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

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

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

- Find the sum of all employee's salaries:

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

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

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

  - 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;
  ```

- Find the total purchases made from clients:

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

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';
  ```

  - 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%';
  ```
- Find any employee born in october:

  ```
  SELECT * FROM employee WHERE birth_day LIKE '____-10%';
  ```
- Find any clients who are schools:

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

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;
```

- 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;
```

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

```
SELECT client_name FROM client UNION
SELECT supplier_name FROM branch_supplier;
```

- Find a list of all money spent or earned by the company:

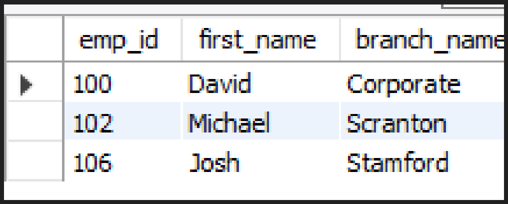
```
SELECT salary FROM employee UNION
SELECT total_sales FROM works_with;
```

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.

```
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.

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

- Right join:

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

- 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:

```
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
);
```

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

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

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.
- 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 ;
```

- Example:

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

- 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. Given the following database schema an ER diagram would be absolutely helpful in defining relationships so that we can understand the relations, object and attributes associated with each table.

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

- ER Diagrams: used to design the database, it is used to model an entity.

- An entity is an object we want to model and store information about. In ER diagrams they are modeled using a square or rectangle.

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

- 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.

- Multi-valued attribute: an attribute that can have more than one value. Modeled with an oval with an inner oval.

- Derived attribute: an attribute that can be derived from the other attributes, modeled using oval with dashed lines.

- Multiple entities: you can define more than one entity in the diagram.

- 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.

- Relationship attribute: an attribute about the relationship.

- Relationship cardinality: the number of instances of an entity from a relation that can associated with the relation.

- Weak entity: an entity that cannot be uniquely identified by its attributes alone, an entity that will depend on another.

- Identifying relationship: a relationship that serves to uniquely identify the weak entity.

Capítulo 19

Designing an ER Diagram

-

Capítulo 20

Converting ER Diagrams to Schemas