Welcome to the Databases - SQL module!

Trainer: Diana Cavalcanti

Scope

- Relations
- Databases, Tables: Creating and Designing
- Data types, indexes, limitations
- SQL
- CRUD
- Complex queries with JOIN (INNER, OUTER, LEFT, RIGHT)
- having, group by, order by, limit
- (Optional)
- triggers, procedures
- Transactions
- ACID

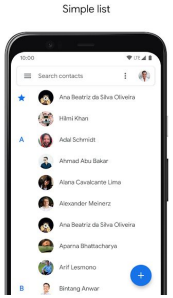
Software:
- MySQL 5.7.x+/8.x.y+
- MySQL Workbench 5.x.y+/8.x.y+

Important

Attendance list
Break time

Fundamentals

- Do you know what a database is?
  - A database is an organized collection of data
  - Would you know how to measure how much this area is present in your life?



Database system

A Database system is basically a computerized information storage system, that is, a computerized system whose main purpose is to maintain, store and make information available. ” (C.J. Date)

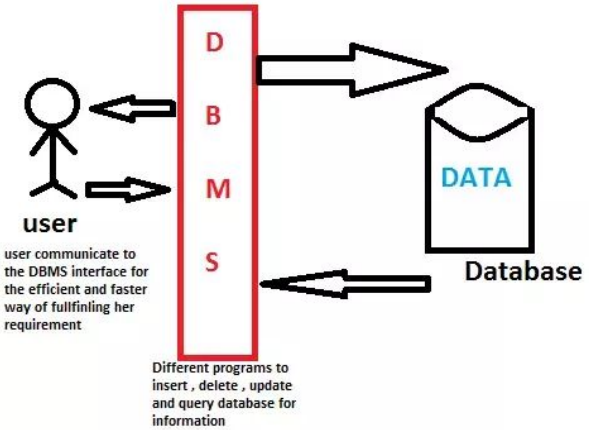
Main purpose:

- Organized storage aimed at:
  - System optimization
  - Facilitate insert, update, processing and consultation

https://en.wikibooks.org/wiki/Introduction_to_Database_Systems

A Database Management System (DBMS)

- DBMS is a system (software) that provides an interface to database for information storage and retrieval
  - capacity for large amount of data
  - an easy to use interface language (SQL-structured query language)
  - efficient retrieval mechanisms
  - multi-user support
  - security management
  - concurrency and transaction control
  - persistent storage with backup and recovery for reliability



https://en.wikibooks.org/wiki/Introduction_to_Database_Systems

A Database Management System (DBMS)

Examples of popular DBMS used these days:

- MySql
- Oracle
- SQL Server
- IBM DB2
- PostgreSQL

Relational databases

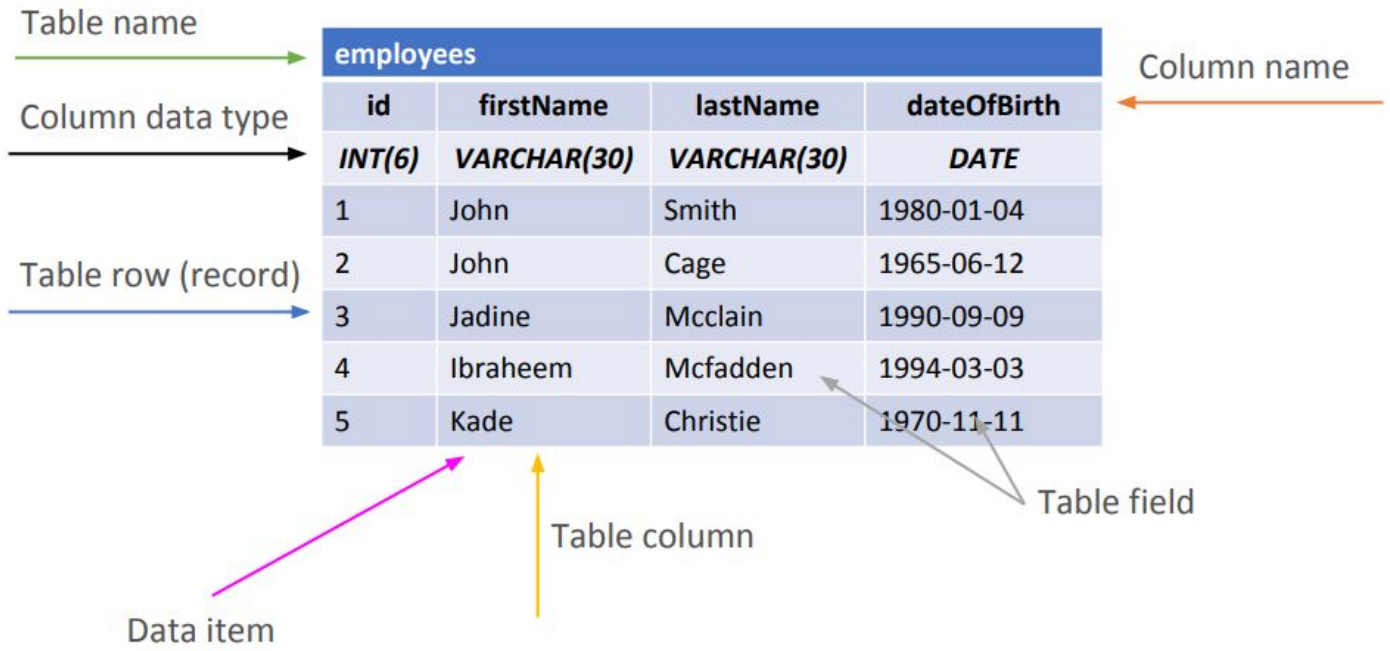
- This model organizes data into one or more tables (or "relations") of columns and rows, with a unique key identifying each row.

- A table is a collection of data held in a two dimensional structure.
- The two dimensions are rows and columns.
- A table is identified by a name.

https://www.oracle.com/database/what-is-database.html

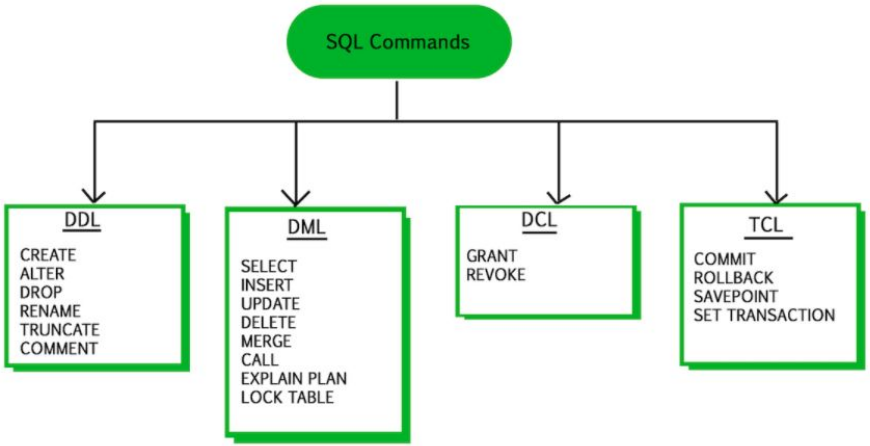
Relational databases

- Table



SQL - Structured Query Language

- **DDL** - **data definition language**. Helps users define what kind of data they are going to store and how they are going to model this data.
- **DML** - **data manipulation language**. Allows users to insert, update and delete data from the database.
- **DQL** - **data query language**. Helps users retrieve information from the database.
- **DCL** - **data control language**. Allows users to restrict and control access to the database.



SQL - Structured Query Language

- **DDL - Data Definition Language**
-
  - Create a database
    - **CREATE DATABASE** sda_course;
  - Select the database
    - use sda_course;
  - Delete a database
    - **DROP DATABASE** sda_course;

SQL - **DDL - Data Definition Language**

- Create a table

```
CREATE TABLE employees (
    id_employees INT,
    first_name VARCHAR(30),
    last_name VARCHAR(30),
    salary INT
);
```

- Column data types: The column data types define the type of information you can store in that particular column:
- **numeric:** int, tinyint, bigint, float, real, etc.,
- **date and time:** Date, Time, Datetime, etc.,
- **character and string:** char, varchar, text, etc.,
- **logical values:** TINYINT type value (0 or 1).

SQL - Structured Query Language

- **DDL - Data Definition Language**

  - describe employees;
  - Delete a table
    - **DROP TABLE** employees;

SQL - Structured Query Language

- **DDL - Data Definition Language**
-
    - Add a column

      ALTER TABLE employees
      ADD dateOfBirth **VARCHAR(10);**

    - Update a column

      ALTER TABLE employees
        MODIFY dateOfBirth **VARCHAR(50);**

SQL - Structured Query Language

- **DDL - Data Definition Language**
  - RENAME a column

    ALTER TABLE employees
    CHANGE COLUMN dateOfBirth date_of_birth **DATE**

  - DELETE a column

    ALTER TABLE employees
    DROP COLUMN date_of_birth **;**

SQL - Structured Query Language

- **DDL - Data Definition Language**

When defining a table the user can set certain properties on the columns:

- data type controls the type of values stored in the column,
- **NOT NULL** defines whether a column must be filled or not,
- **AUTOINCREMENT** states that the column value will be generated automatically (incrementation of the last inserted value) - this only works for numeric columns,
- **UNIQUE** states that there cannot be more than one row with the same value for that particular column.

SQL - Structured Query Language

-
- **NOT NULL**
  - **ALTER TABLE** employees **MODIFY** first_name **VARCHAR(30) NOT NULL;**
-
- **AUTOINCREMENT**
  - **ALTER TABLE** employees **CHANGE** id_employees id_employees **INT NOT NULL AUTO_INCREMENT PRIMARY KEY;**
  -
- **UNIQUE**
  - **ALTER TABLE employees ADD UNIQUE (last_name);**

Exercises

1. Create a new database: humanResources
2. Create a new table employees, with the following columns:
   a. employeeId - INTEGER ,
   b. firstName - VARCHAR,
   c. lastName - VARCHAR,
   d. dateOfBirth - DATE,
   e. postalAddress - VARCHAR.
3. Alter table employees and add the following columns:
   a. phoneNumber - VARCHAR,
   b. email - VARCHAR,
   c. salary - INTEGER.
4. Alter table employees and remove the postalAddress column.
5. Create a new table employeeAddresses,
   a. country_id - INTEGER
   b. country_name - VARCHAR.
6. Remove table employeeAddresses.

DML - Data Manipulation Language

- **Adding data**

INSERT INTO employees (id_employees, first_name, last_name, salary, date_of_birth) **VALUES**
(1, 'Michael', 'Harding', 20, '1937-07-25'),
(2, 'Ariana', 'Fox', 30, '1992-09-30'),
(3, 'Madelyn', 'Flynn', 35, '1953-03-05'),
(4, 'Fynley', 'Dodd', 40, '1973-03-27'),
(5, 'Aliza', 'Wyatt', 55, '1969-02-14'),
(6, 'Michael', 'Doss', 67, '1964-12-11')
(7, 'Michael', 'Watshon', 37, '1983-12-11');

*ALTER TABLE employees add date_of_birth DATE;

DML - Data Manipulation Language

- **Updating data**

```
UPDATE employees SET date_of_birth = '1988-12-11'
WHERE id_employees = 1 ;
```

```
SET SQL_SAFE_UPDATES=0;
```

```
SELECT * FROM employees
```

DML - Data Manipulation Language

- **Deleting data**

```
DELETE FROM employees WHERE id_employees = 7 ;
```

Exercises

Use the database: humanResources

1. Insert a new entry into employees table:
   a. employeeId - 1,
   b. firstName - John,
   c. lastName - Johnson,
   d. dateOfBirth - 1975-01-01,
   e. phoneNumber - 0-800-800-314,
   f. email - john@johnson.com,
   g. salary - 1000.
2. Update dateOfBirth of John Johnson to 1980-01-01.
3. Delete everything from employees table.
4. Add two more entries in employees:
   a. 1, 'John' , 'Johnson', '1975-01-01', '0-800-800-888' , 'john@johnson.com', 1000
   b. 2,'James' , 'Jameson', '1985-02-02', '0-800-800-999' , 'james@jameson.com', 2000

Exercises - Answer

Use the database: humanResources

1. Insert a new entry into employees table:
   a. employeeId - 1,
   b. firstName - John,
   c. lastName - Johnson,
   d. dateOfBirth - 1975-01-01,
   e. phoneNumber - 0-800-800-314,
   f. email - john@johnson.com,
   g. salary - 1000.

INSERT INTO employees (employeeId, firstName, lastName, dateOfBirth , phoneNumber, email, salary)
VALUES (1, 'John', 'Johnson', '1975-01-01', ' 0-800-800-314', 'john@johnson.com', 100)**;**

*If employeeId is auto-increment, remove it.

Exercises - Answer

1. Update dateOfBirth of John Johnson to 1980-01-01.

```
UPDATE employees SET dateOfBirth = '1980-01-01'
WHERE id_employees = 1;
```

also

```
UPDATE employees SET dateOfBirth = '1980-01-01'
WHERE first_name = '1980-01-01' AND  last_name = 'Johnson ';
```

1. Delete everything from employees table.

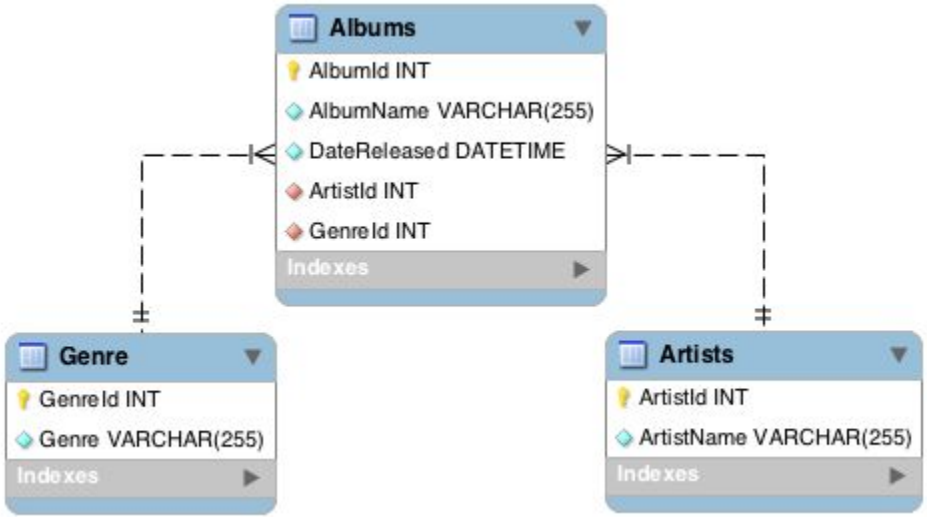
   ```
   DELETE FROM employees;
   ```

2. Add two more entries in employees:
   a. 1, 'John' , 'Johnson', '1975-01-01', '0-800-800-888' , 'john@johnson.com', 1000
   b. 2, 'James' , 'Jameson', '1985-02-02', '0-800-800-999' , 'james@jameson.com', 2000

Exercises

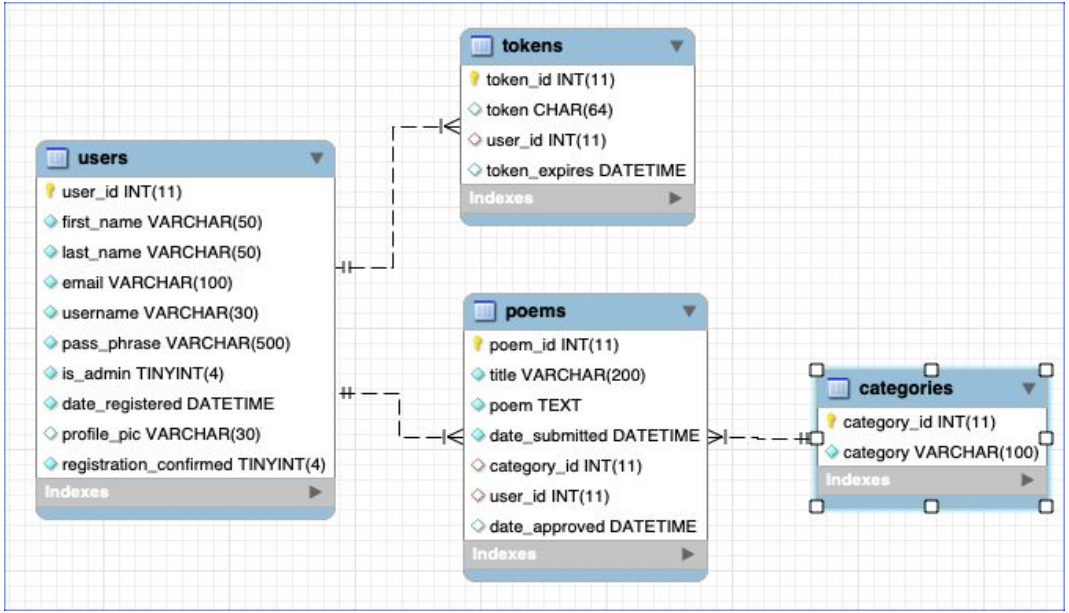
Using DDL

Create a new schema “music” and add the tables following the diagram below



Exercises

- Using DDL create a new schema “db_poems” and add the tables following the diagram below
- Use DML to insert data
- Ids are auto_increment
- Read and search about functions for Date
  - https://www.geeksforgeeks.org/sql-date-functions/
  - https://dataschool.com/learn-sql/dates/
  - https://www.tutorialspoint.com/sql/sql-date-functions.htm
- Insert data using a date function for the attribute ‘date_registered’



Read about string functions

https://www.w3schools.com/sql/sql_ref_sqlserver.asp

Day 2

DQL - Data Query Language

- **SELECT FROM**
  - The SELECT statement allows you to read data from one or more tables.

SELECT select_list FROM table_name [WHERE condition];

SELECT * FROM employees;

DQL - Data Query Language

- **SELECT FROM**
- **WHERE clause**
  - The WHERE clause allows you to specify a search condition for the rows returned by a query.
  - The search condition is a combination of one or more predicates using the logical operator AND, OR and NOT.

DQL - Data Query Language

- **SELECT FROM … WHERE clause**
  - **SELECT** * **FROM** employees **WHERE** last_name = 'Fox';
  - **SELECT DISTINCT** first_name **FROM** employees;
  - **SELECT** * **FROM** employees **WHERE** last_name **= 'Wyatt' AND** first_name **= 'Aliza';**
  - **SELECT** * **FROM** employees **WHERE** salary **>** 40;
  - **SELECT** * **FROM** employees **WHERE** salary **IN (10, 20, 30)**;
  - **SELECT** * **FROM** employees **WHERE** salary **IS NULL**;
  - **SELECT** * **FROM** employees **WHERE** salary **IS NOT NULL** ;
  - **SELECT** * **FROM** employees **WHERE** salary **!= 20**;
  - **SELECT** * **FROM** employees **WHERE** salary **BETWEEN 30 AND 50** ;
  - **SELECT** * **FROM** employees **WHERE** first_name **LIKE 'A%';**
  - **SELECT** * **FROM** employees **WHERE** first_name **LIKE '%n';**
  - **SELECT** * **FROM** employees **WHERE** first_name **LIKE '%e%';**

AGGREGATE functions

An aggregate function performs a calculation on multiple values and returns a single value

- **AVG** - takes multiple numbers and returns the average value of the numbers
  - **SELECT AVG**(salary) **FROM** employees;
- **SUM** - returns the summation of all values
  - **SELECT SUM**(salary) **FROM** employees;
- **MAX** - returns the highest value
  - **SELECT MAX**(salary) **FROM** employees;
- **MIN** - returns the lowest value
  - **SELECT MIN**(salary) **FROM** employees;
- **COUNT** - returns the number of rows
  - **SELECT COUNT**(*) **FROM** employees;

SQL EXTRAS

- **ORDER BY**
  - Used to sort the result-set in ascending or descending order:
    SELECT column1, column2, … FROM table_name ORDER BY column1 [ASC|DESC];

```
SELECT first_name
FROM employees
ORDER BY first_name ASC;
```

```
SELECT first_name
FROM employees
ORDER BY first_name DESC;
```

SQL EXTRA

- **AS**
  - Aliases are used to give a table, or a column in a table, a temporary name:
  - **SELECT column1 as newName, column2, … FROM table_name;**
    - **SELECT** first_name **as FIRST_NAME FROM** employees;

- **LIMIT**
  - Used to restrict the number of results retrieved from the database
  - **SELECT * FROM** employees **LIMIT 3;**

SQL EXTRAS

- **GROUP BY**
  - statement groups rows that have the same values into summary rows, like “find the number of customers in each country”:
  - SELECT column1, column2, … FROM table_name GROUP BY column1;
  - **SELECT COUNT(CustomerID), Country FROM Customers GROUP BY Country;**

```
SELECT first_name, COUNT(*) AS 'occurences count'
FROM employees
GROUP BY first_name;
```

SQL EXTRA

- **HAVING**
  - clause was added to SQL because the WHERE keyword could not be used with aggregate functions:
  - **SELECT column1, column2, … FROM table_name GROUP BY column1 HAVING condition;**

```
SELECT first_name AS 'NAME'
FROM employees
GROUP BY first_name
HAVING COUNT(*) > 1;
```

SubQueries

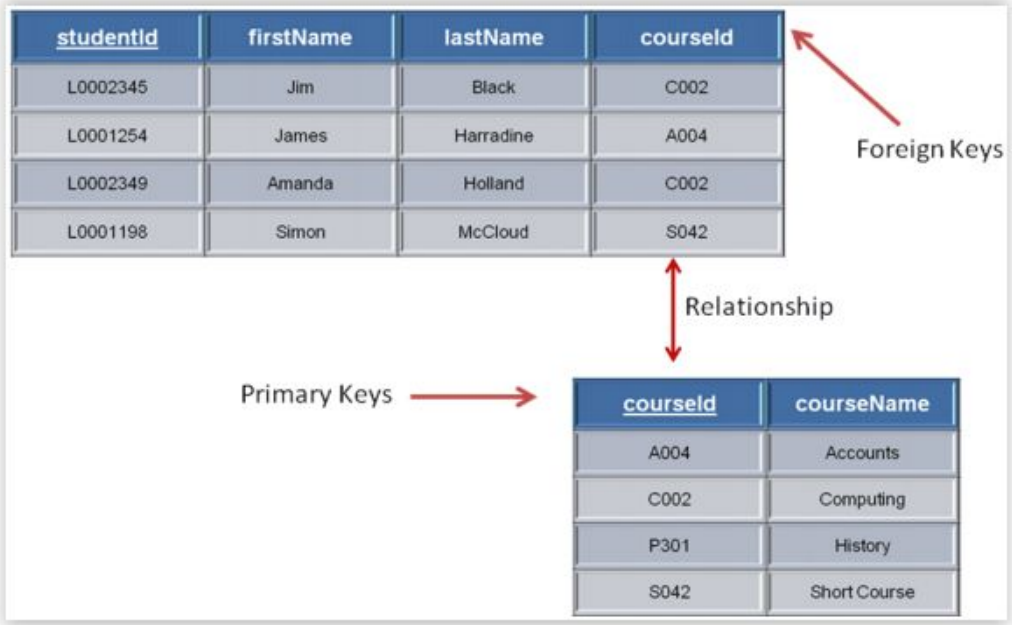
https://www.mysqltutorial.org/mysql-subquery/

https://www.essentialsql.com/get-ready-to-learn-sql-server-20-using-subqueries-in-the-select-statement/

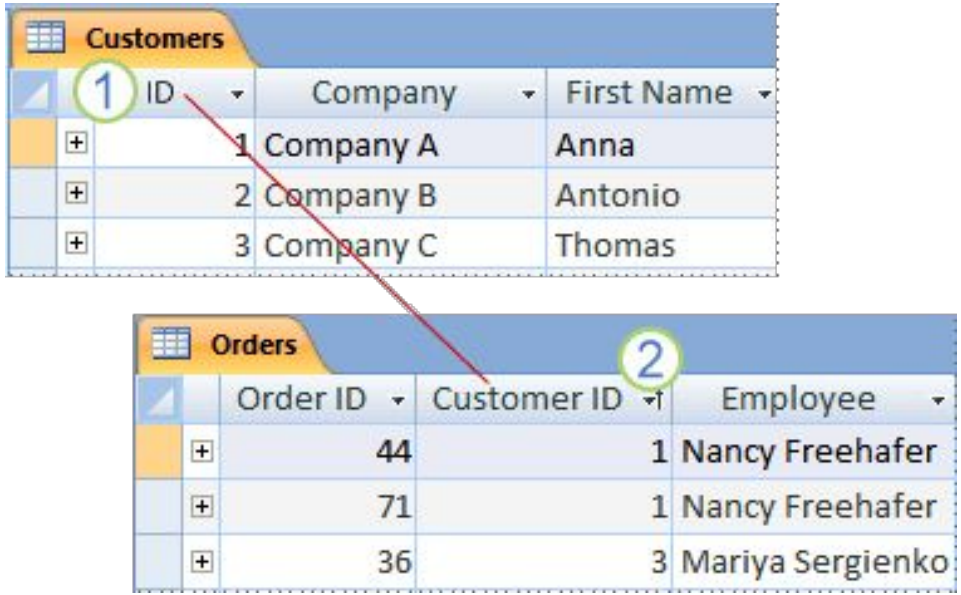
https://levelup.gitconnected.com/how-and-when-to-write-mysql-subqueries-8d5d580b1729

```
SELECT first_name, salary
FROM   employees
WHERE  salary = (SELECT MIN(salary) FROM   employees);
```

PRIMARY and FOREIGN Keys



PRIMARY and FOREIGN Keys



PRIMARY Keys

- A primary key is a column or a set of columns that uniquely identifies each row in the table.
- A primary key must contain unique values. If the primary key consists of multiple columns, the combination of values in these columns must be unique.
- A primary key column cannot have NULL values.
- A table can have one an only one primary key.
- A primary key column often has the AUTO_INCREMENT attribute that automatically generates a sequential integer whenever you insert a new row into the table.

FOREIGN Keys

```
CREATE TABLE employees (
    id_employees INT AUTO_INCREMENT PRIMARY KEY NOT NULL,
    first_name VARCHAR(30),
    last_name VARCHAR(30),
    salary INT,
    date_of_birth DATE
);
```

OR

```
ALTER TABLE employees ADD PRIMARY KEY NOT NULL (id_employees);
```

FOREIGN Keys

- A foreign key is a column or group of columns in a table that links to a column or group of columns in another table.
- The foreign key places constraints in the related tables, so MySQL can maintain referential integrity. The table containing the foreign key is called the child table, and the referenced table is the parent table.
- Typically, the foreign key columns of the child table often refer to the primary key columns of the parent table.
- A table can have more than one foreign key where each foreign key references to a primary key of the different parent tables.
- Once a foreign key constraint is in place, the foreign key columns from the child table must have the corresponding row in the parent key columns of the parent table or values in these foreign key columns must be NULL.

FOREIGN Keys

```
ALTER TABLE employees ADD id_departments INT(6);

CREATE TABLE departments (
    id_departments INT(6) AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(30) NOT NULL
);


ALTER TABLE employees ADD FOREIGN KEY(id_departments)
(REFERENCES departments (id_departments);
```

Exercises - HomeWork

1. Alter table employees:
   a. make employeeId column PRIMARY KEY, NOT NULL and AUTO INCREMENT.
2. See what happens when you add two more entries in employees, this time without setting the employeeId manually:
   a. 'Julie', 'Juliette', '1990-01-01', '0-800-900-111', 'julie@juliette.com', 5000
   b. 'Sofie', 'Sophia', '1987-02-03', '0-800-900-222', 'sofie@sophia.com', 1700 3.
3. Create a new table departments, with columns:
   a. departmentId - Integer, PRIMARY KEY, NOT NULL, AUTO INCREMENT
   b. name - Varchar, NOT NULL
4. Add two entries in table departments:
   a. HR,
   b. Finance.
5. Connect the two tables together - employees should have a reference to departments:
   a. add departmentId - Integer column to employees table,
   b. assign John to HR and Julie to Finance.

Exercises - HomeWork

6. Delete entry HR from departments table:
    a. Does this work?
    b. Should we be able to delete it? If John is assigned to HR and we delete, is the data still correct?
7. Create a foreign key in employees table to departments table:
    a. departmentId column in employees should reference departmentId column in departments,
    b. remember the naming convention: fk_employees_departments.
8. Now try to delete entry HR from departments table:
    a. Does this still work?
9. Now try to add a new employee and set its departmentId to 10:
    a. Does this work? Should it?
    b. Try to add this new employee and set its departmentId to 1. Does this work?
10. Try deleting the newly added employee:
    a. Does it work? Should it?

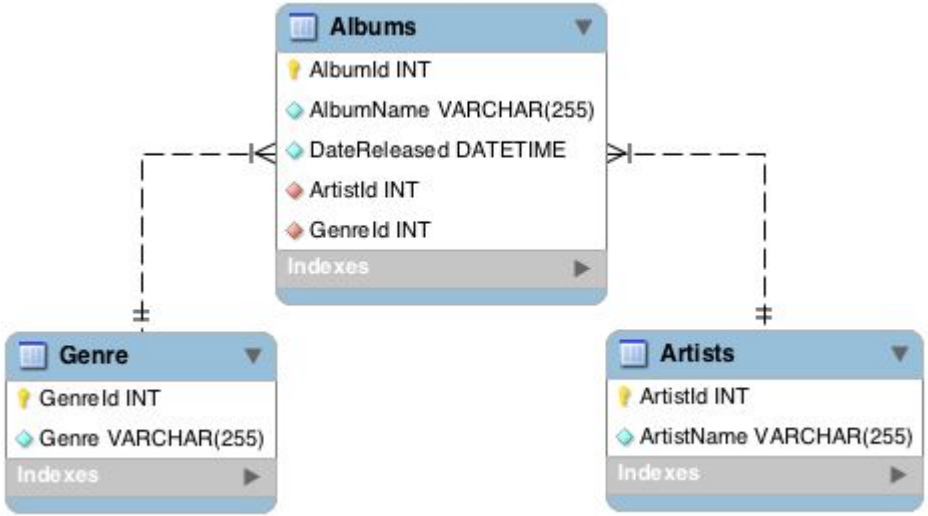
EXERCISES

Use the database: humanResources

1. Select everything from table employees.
2. Select only firstName and lastName from table employees.
3. Select all employees with lastName Johnson.
4. Select all employees whose lastName starts with J.
5. Select all employees whose lastName contains so.
6. Select all employees born after 1980.
7. Select all employees born after 1980 and whose firstName is John.
8. Select all employees born after 1980 or whose firstName is John.
9. Select all employees whose lastName is not Jameson.
10. Select maximum salary.
11. Select minimum salary.
12. Select average salary.

Exercises

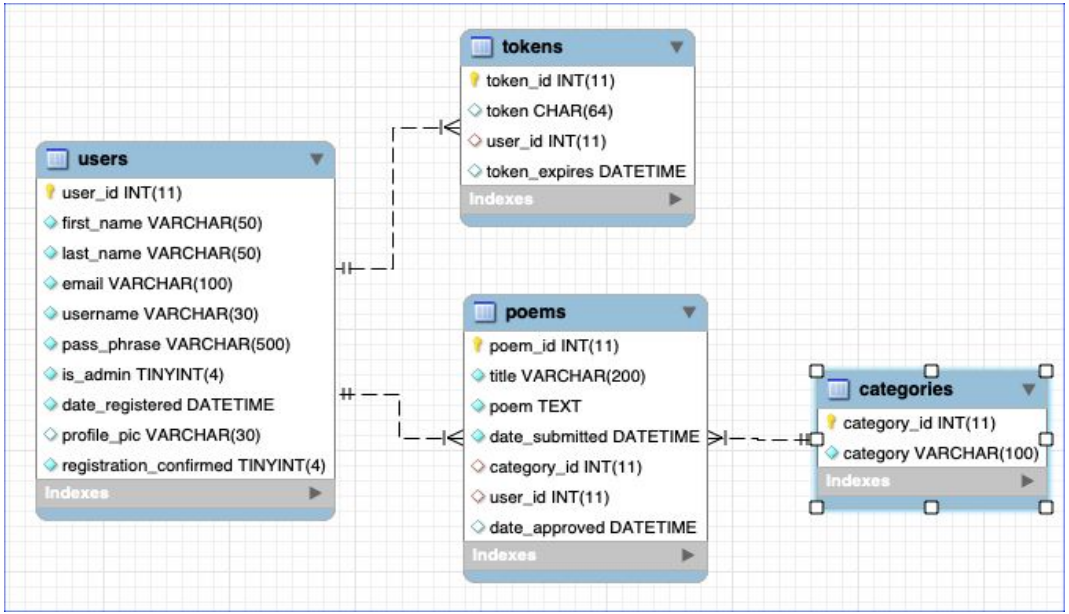
- Using SQL add the relationship between the tables described on diagram below, use the reverse engineer and compare your diagram
- Add data, create a query to answer how many albums exist by ‘genre’
  - SELECT genrerId, count(genrerId) FROM albums group by genrerId;
- Create a query to answer what is the lasted album released?



```
SELECT albumName, dateReleased FROM albums

ORDER BY dateReleased DESC LIMIT 1;


SELECT albumName, max(dateReleased) FROM albums

GROUP BY albumName;


SELECT albumName, dateReleased FROM albums

WHERE dateReleased IN (SELECT MAX(dateReleased) FROM albums);
```

Exercises

- Using SQL add the relationship between the tables described on diagram below, use the reverse engineer and compare your diagram
- How many users was registered by date?
- List the ‘token_id’ that has expired



SELECT date_registered, count(date_registered) FROM USERS GROUP BY date_registered;

SELECT token_id FROM tokens WHERE token_expires < now();

select curdate();

select now();

token 10/10/2022 10:22:13 > 30/01/2021 09:43:21

30/01/2021 08:43:21 < 30/01/2021 09:43:21

Day 3

Relationships

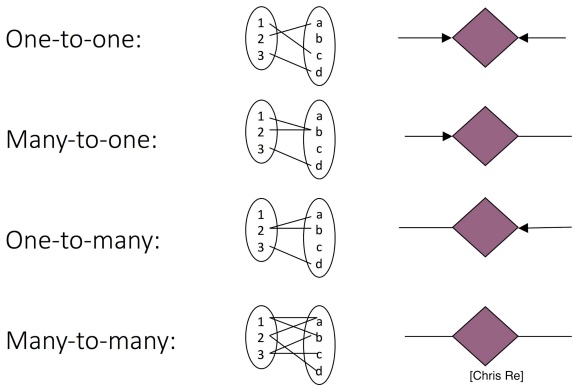
When designing a database, common sense dictates that we use different tables for different types of entities.

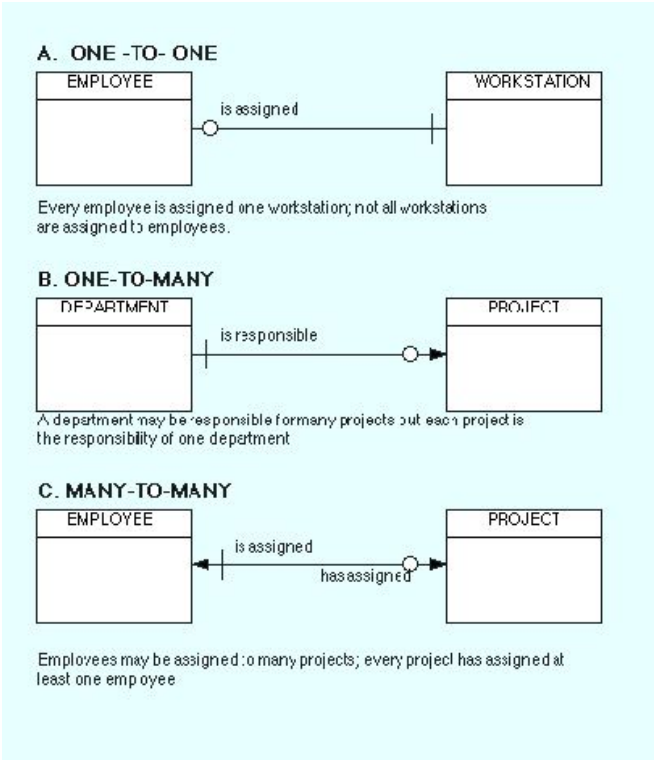
Take for example an online store which has information about: customers, orders, items, etc. We would have separate tables for all of these. But we also need to have relationships between these tables. For instance a customer can place several orders, an order belongs to only one customer, an order can contain multiple items. These relationships need to be represented in the database.

Relationships

There are several types of database relationships:

- **One to One** relationships,
- **One to Many** and Many to One relationships,
- **Many to Many** relationships



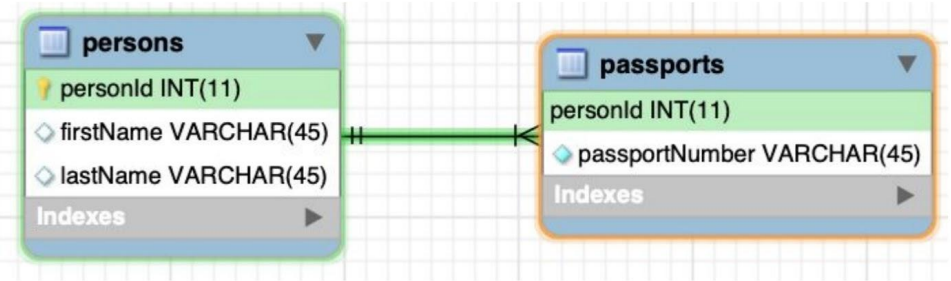


One-to-One

One-to-One relationships occur when there is only one record in the first table that corresponds to only one record in the related table.

This is achieved by adding a foreign key from the primary key in the first table to the primary key in the second table.

Keep in mind that this kind of relationship is not very common. It is usually simpler to combine the two tables into a single larger one.

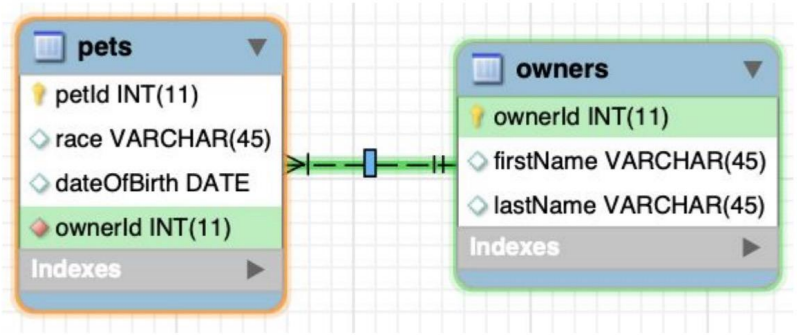


Example: a person has a single passport and a passport belongs to a single person.

One-to-Many

One-to-Many relationship is defined as a relationship between two tables where a row from one table can have multiple matching rows in another table.

In order to model this you need to identify the table representing the many side of the relationship and add an additional column with a foreign key referencing the primary key of the table representing the one side of the relationship.

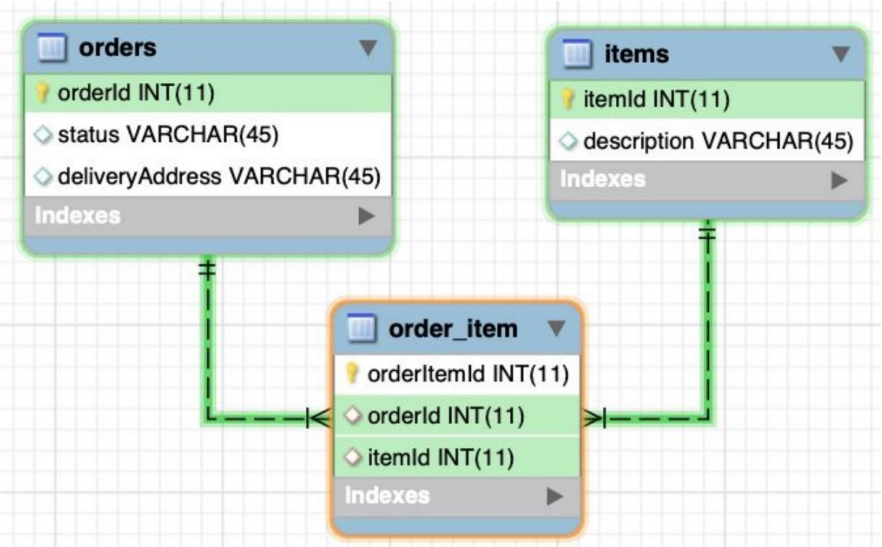


Example: a pet belongs to a single owner while the owner can have multiple pets.

Many-to-Many

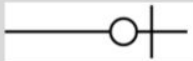
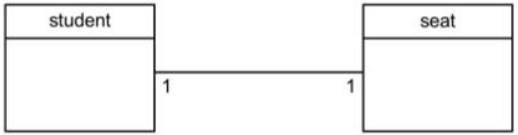
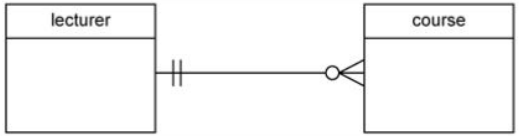
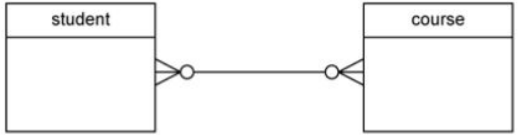
Many-to-Many relationship occurs when multiple records in one table are associated with multiple records in another table.

In order to model this you can break the many-to-many relationship into two one-to-many relationships by using a third table, called a join table. Each record in a join table includes a match field that contains the value of the primary keys of the two tables it joins (in the join table, these match fields are foreign keys). These foreign key fields are populated with data as records in the join table are created from either table it joins.

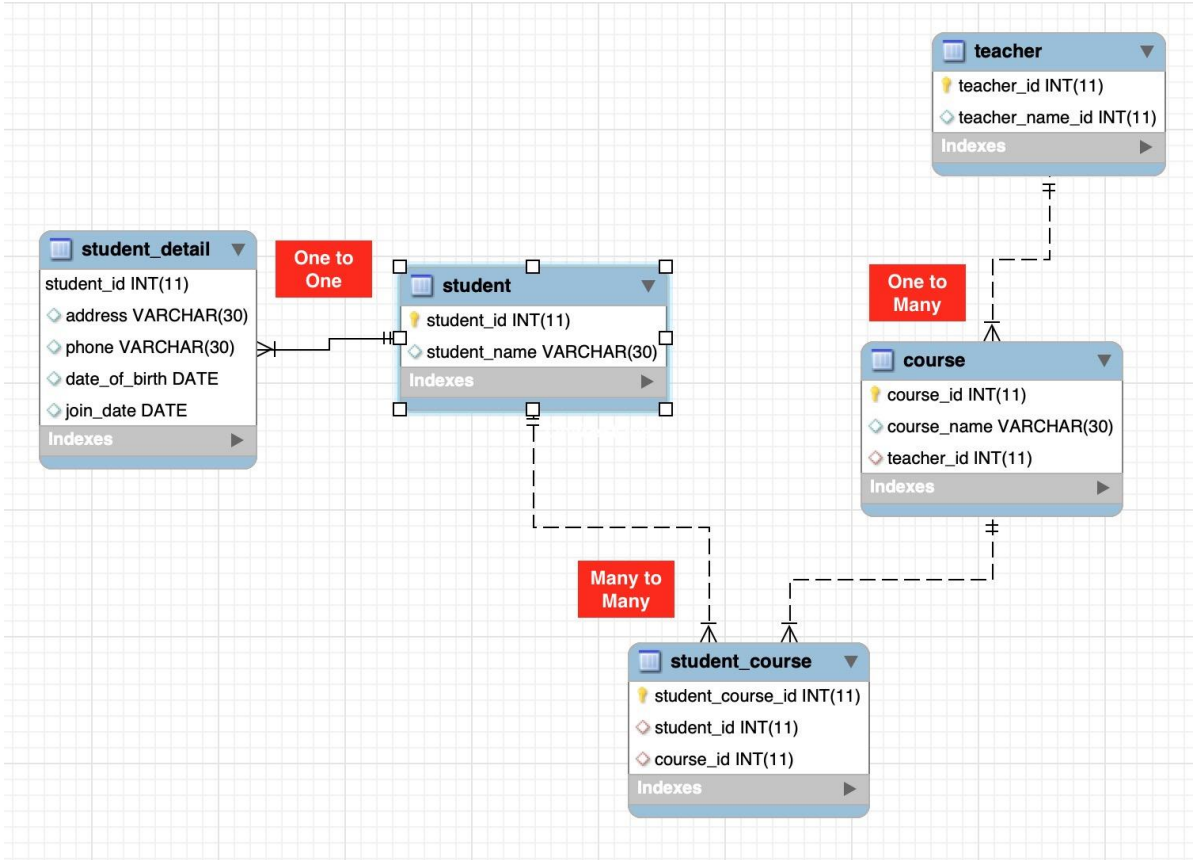


Example: an order can have multiple items and an item can belong to multiple orders.

Relationships

Relationship	UML (Unified Modeling Language)	Crow's Foot Notation
One and Only One	[1]	
Zero or One	[0 .. 1]	
One or Many	[1 .. *]	
Zero or Many	[0 .. *]	
One-to-One *One student has one seat*		
One-to-Many *One lecturer can have many courses*		
Many-to-Many *Many students can have many courses*		

Relationships



Exercise

```
CREATE DATABASE school;
USE school;

CREATE TABLE student_detail (
    student_id INT PRIMARY KEY NOT NULL,
    address VARCHAR(30),
    phone VARCHAR(30),
    date_of_birth DATE,
    join_date DATE
);

CREATE TABLE student (
    student_id INT AUTO_INCREMENT PRIMARY KEY NOT NULL,
    student_name VARCHAR(30)
);

CREATE TABLE teacher (
    teacher_id INT AUTO_INCREMENT PRIMARY KEY NOT NULL,
    teacher_name VARCHAR(30)
);
```

Exercise

```
CREATE TABLE course (
    course_id INT AUTO_INCREMENT PRIMARY KEY NOT NULL,
    course_name VARCHAR(90),
    teacher_id  INT
);


CREATE TABLE student_course (
    student_course_id INT AUTO_INCREMENT PRIMARY KEY NOT NULL,
    student_id INT,
    course_id INT
);
```

Exercise

ALTER TABLE student_detail **ADD FOREIGN KEY(**student_id**) REFERENCES** student **(**student_id**);**

ALTER TABLE course **ADD FOREIGN KEY**(teacher_id**) REFERENCES** teacher (teacher_id**);**

ALTER TABLE student_course **ADD FOREIGN KEY**(student_id**) REFERENCES** student (student_id**);**

ALTER TABLE student_course **ADD FOREIGN KEY**(course_id**) REFERENCES** course (course_id**);**

Add student

```
INSERT INTO `student` (`student_name`) VALUES ('Neeme ');
INSERT INTO `student` (`student_name`) VALUES ('Christopher ');
INSERT INTO `student` (`student_name`) VALUES ('Kristina ');
INSERT INTO `student` (`student_name`) VALUES ('Susanna');
INSERT INTO `student` (`student_name`) VALUES ('Aleksander ');
INSERT INTO `student` (`student_name`) VALUES ('Karl');
INSERT INTO `student` (`student_name`) VALUES ('Joel ');
INSERT INTO `student` (`student_name`) VALUES ('Silver');
INSERT INTO `student` (`student_name`) VALUES ('Miquel');
INSERT INTO `student` (`student_name`) VALUES ('Kitaek ');
INSERT INTO `student` (`student_name`) VALUES ('Leonardo');
INSERT INTO `student` (`student_name`) VALUES ('Pedro Iglesias');
INSERT INTO `student` (`student_name`) VALUES ('Mikael ');
INSERT INTO `student` (`student_name`) VALUES ('Valentin ');
INSERT INTO `student` (`student_name`) VALUES ('Airika ');
INSERT INTO`student` (`student_name`) VALUES ('Märt');
INSERT INTO `student` (`student_name`) VALUES ('Anastasia');
```

Add teacher

```
INSERT INTO `teacher` (`teacher_name`) VALUES ('Peter');
INSERT INTO `teacher` (`teacher_name`) VALUES ('Allan');
INSERT INTO `teacher` (`teacher_name`) VALUES ('Julia');
INSERT INTO `teacher` (`teacher_name`) VALUES ('Daniel');
INSERT INTO `teacher` (`teacher_name`) VALUES ('Catarine');
INSERT INTO `teacher` (`teacher_name`) VALUES ('Sophia');
INSERT INTO `teacher` (`teacher_name`) VALUES ('Lily');
INSERT INTO `teacher` (`teacher_name`) VALUES ('Natalie');
INSERT INTO `teacher` (`teacher_name`) VALUES ('Audrey');
INSERT INTO `teacher` (`teacher_name`) VALUES ('Sarah');
INSERT INTO `teacher` (`teacher_name`) VALUES ('Jacob');
INSERT INTO `teacher` (`teacher_name`) VALUES ('Jack');
INSERT INTO `teacher` (`teacher_name`) VALUES ('Luke');
INSERT INTO `teacher` (`teacher_name`) VALUES ('Anthony');
INSERT INTO `teacher` (`teacher_name`) VALUES ('Andrew');
INSERT INTO `teacher` (`teacher_name`) VALUES ('Adrian');
INSERT INTO `teacher` (`teacher_name`) VALUES ('George');
INSERT INTO `teacher` (`teacher_name`) VALUES ('Edward');
```

Add course

```
INSERT INTO `course` (`course_name`, `teacher_id`) VALUES ('Java - Fundamentals', '1');
INSERT INTO `course` (`course_name`, `teacher_id`) VALUES ('Java - Fundamentals: Coding', '2');
INSERT INTO `course` (`course_name`, `teacher_id`) VALUES ('Software Testing - Fundamentals', '3');
INSERT INTO `course` (`course_name`, `teacher_id`) VALUES ('Java - Advanced Features', '4');
INSERT INTO `course` (`course_name`, `teacher_id`) VALUES ('Design Patterns - Good Practices', '5');
INSERT INTO `course` (`course_name`, `teacher_id`) VALUES ('Java - Advanced Features: Coding', '6');
INSERT INTO `course` (`course_name`, `teacher_id`) VALUES ('Databases - SQL', '7');
INSERT INTO `course` (`course_name`, `teacher_id`) VALUES ('JDBC - Hibernate', '8');
INSERT INTO `course` (`course_name`, `teacher_id`) VALUES ('Practical Project', '9');
INSERT INTO `course` (`course_name`, `teacher_id`) VALUES ('Introduction to HTTP', '10');
INSERT INTO `course` (`course_name`, `teacher_id`) VALUES ('HTML. CSS. JavaScript', '11');
INSERT INTO `course` (`course_name`, `teacher_id`) VALUES ('Front-end Technologies', '12');
INSERT INTO `course` (`course_name`, `teacher_id`) VALUES ('Front-end Technologies & HTML. CSS. JavaScript', '13');
INSERT INTO `course` (`course_name`, `teacher_id`) VALUES ('Spring', '14');
INSERT INTO `course` (`course_name`, `teacher_id`) VALUES ('Software Testing - Advanced Features', '15');
INSERT INTO `course` (`course_name`, `teacher_id`) VALUES ('Agile. Scrum', '16');
INSERT INTO `course` (`course_name`, `teacher_id`) VALUES ('Cybersecurity and Blockchain', '1');
INSERT INTO `course` (`course_name`, `teacher_id`) VALUES ('IOT', '4');
INSERT INTO `course` (`course_name`, `teacher_id`) VALUES ('AI and Big Data', '6');
INSERT INTO `course` (`course_name`, `teacher_id`) VALUES ('Data Science and Analytics', '1');
INSERT INTO `course` (`course_name`, `teacher_id`) VALUES ('Robotics', '4');
INSERT INTO `course` (`course_name`, `teacher_id`) VALUES (' Pi Course - Raspberry Pi Foundation', '1');
INSERT INTO `course` (`course_name`, `teacher_id`) VALUES ('Python-Data-Structures', '4');
INSERT INTO `course` (`course_name`, `teacher_id`) VALUES ('Machine-Learning');
INSERT INTO `course` (`course_name`, `teacher_id`) VALUES ('Network and IT Security', '6');
INSERT INTO `course` (`course_name`, `teacher_id`) VALUES ('Artificial Intelligence', '4');
INSERT INTO `course` (`course_name`, `teacher_id`) VALUES ('Database', '13');
INSERT INTO `course` (`course_name`, `teacher_id`) VALUES ('HTML & CSS Course', '11');
INSERT INTO `course` (`course_name`, `teacher_id`) VALUES ('esponsive Web Design', '4');
INSERT INTO `course` (`course_name`, `teacher_id`) VALUES ('Word 2016', '9');
INSERT INTO `course` (`course_name`, `teacher_id`) VALUES ('Perl Specialist ', '1');
```

```
INSERT INTO `student_course` (`student_id`, `course_id`) VALUES ('2', '8');
INSERT INTO `student_course` (`student_id`, `course_id`) VALUES ('2', '2');
INSERT INTO `student_course` (`student_id`, `course_id`) VALUES ('4', '9');
INSERT INTO `student_course` (`student_id`, `course_id`) VALUES ('4', '6');
INSERT INTO `student_course` (`student_id`, `course_id`) VALUES ('4', '1');
INSERT INTO `student_course` (`student_id`, `course_id`) VALUES ('5', '9');
INSERT INTO `student_course` (`student_id`, `course_id`) VALUES ('5', '7');
INSERT INTO `student_course` (`student_id`, `course_id`) VALUES ('6', '8');
```

Joins

A SQL Join statement is used to combine data or rows from two or more tables based on a common field between them:

- CROSS JOIN
- INNER JOIN
- LEFT JOIN
- RIGHT JOIN

https://www.guru99.com/joins.html

CROSS JOIN

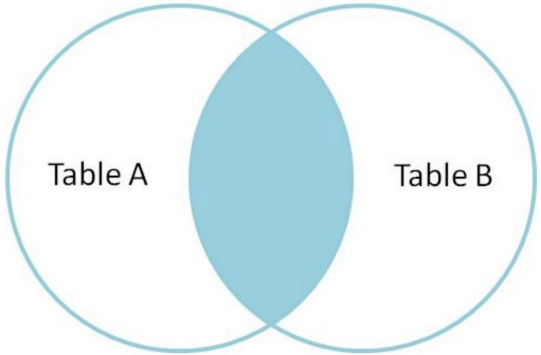
- Matches each row from the first table with each row of the second table.
- If each table had 4 rows, we should be getting a result of 16 rows.

- SELECT * FROM table1 JOIN table2
- SELECT * FROM owners JOIN pets

INNER JOIN

INNER JOIN selects all rows from both tables as long as the condition is met.

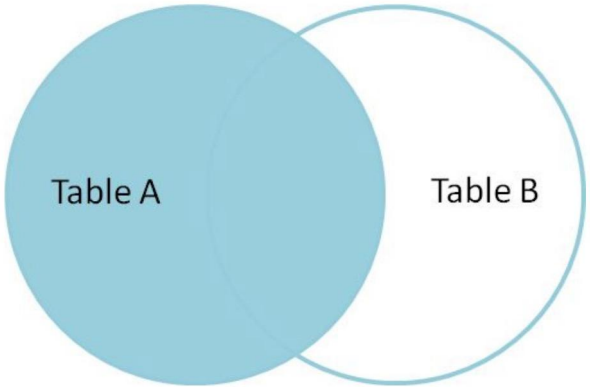
intersection




```
SELECT * FROM pets
INNER JOIN owners
ON pets.ownerId = owners.ownerId
```

LEFT JOIN

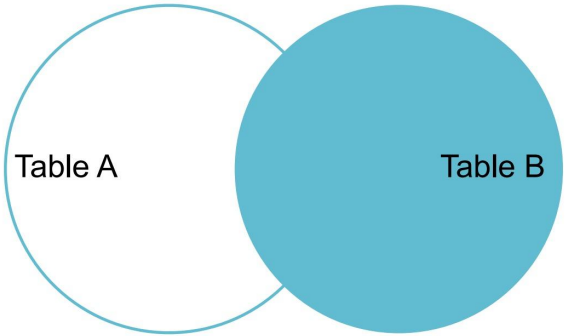
Returns all the rows of the table on the left side of the join and matching rows of the table on the right side of the join. For the rows for which there is no matching row on right side, the result-set will contain null.




```
SELECT * FROM pets
LEFT JOIN owners
ON pets.ownerId = owners.ownerId
```

RIGHT JOIN

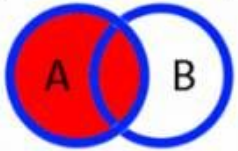
Returns all the rows of the table on the right side of the join and matching rows of the table on the left side of join. For the rows for which there is no matching row on left side, the result-set will contain null.



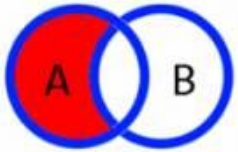

```
SELECT * FROM pets
RIGHT JOIN owners
ON pets.ownerId = owners.ownerId
```

SQL JOINS

LEFT OUTER JOIN

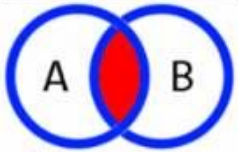



```
SELECT *
FROM TableA a
LEFT JOIN TableB b
    ON a.KEY = b.KEY
```



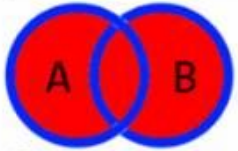

```
SELECT *
FROM TableA a
LEFT JOIN TableB b
    ON a.KEY = b.KEY
WHERE b.KEY IS NULL
```

INNER JOIN

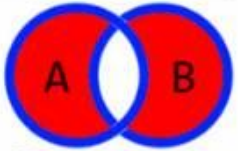



```
SELECT *
FROM TableA a
INNER JOIN TableB b
    ON a.KEY = b.KEY
```

FULL OUTER JOIN

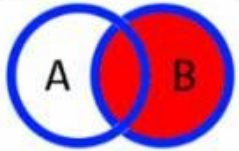



```
SELECT *
FROM TableA a
FULL OUTER JOIN TableB b
    ON a.KEY = b.KEY
```

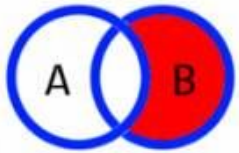



```
SELECT *
FROM TableA a
FULL OUTER JOIN TableB b
    ON a.KEY = b.KEY
WHERE a.KEY IS NULL
    OR b.KEY IS NULL
```

RIGHT OUTER JOIN

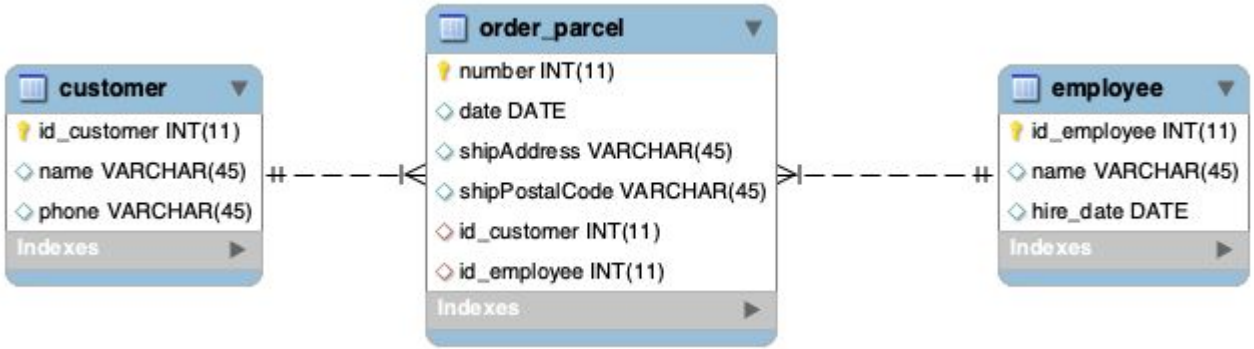



```
SELECT *
FROM TableA a
RIGHT JOIN TableB b
    ON a.KEY = b.KEY
```




```
SELECT *
FROM TableA a
RIGHT JOIN TableB b
    ON a.KEY = b.KEY
WHERE a.KEY IS NULL
```

Exercises



1. **What type of Relationships can you identify at the diagram above?**
2. **Add a new database: dql_exercise**
3. **Check the diagram above and add the tables and relationships**
4. **Insert data for each table**

   Using DQL:
5. **Display all orders number ordered by date, with employee name, customer name**
6. **Display the total of orders by shipPostalCode**
7. **Display the total of orders by shipPostalCode and customer name**
8. **Display the total of orders by shipPostalCode, date and employee name**

Joins - Resume

Here are the different types of the JOINs in SQL:

- **(INNER) JOIN**: Returns records that have matching values in both tables
  - "SELECT * FROM TableA a INNER JOIN TableB b ON a.KEY = b.KEY
- **LEFT (OUTER) JOIN**: Returns all records from the left table, and the matched records from the right table
  - "SELECT * FROM TableA a LEFT JOIN TableB b ON a.KEY = b.KEY
  - "SELECT * FROM TableA a RIGHT JOIN TableB b ON a.KEY = b.KEY WHERE b.KEY IS NULL
- **RIGHT (OUTER) JOIN**: Returns all records from the right table, and the matched records from the left table
  - "SELECT * FROM TableA a RIGHT JOIN TableB b ON a.KEY = b.KEY
  - "SELECT * FROM TableA a RIGHT JOIN TableB b ON a.KEY = b.KEY WHERE a.KEY IS NULL
- **FULL (OUTER) JOIN**: Returns all records when there is a match in either left or right table
  - "SELECT * FROM TableA a FULL OUTER JOIN TableB b ON a.KEY = b.KEY
  - "SELECT * FROM TableA a FULL OUTER JOIN TableB b ON a.KEY = b.KEY WHERE a.KEY IS NULL OR b.KEY IS NULL

Example

```
SELECT * FROM student_course;
```

```
SELECT stc.student_id, s.student_name, c.course_name

FROM student_course as stc

INNER JOIN course as c ON stc.course_id = c.course_id

INNER JOIN student as s ON s.student_id = stc.student_id
```

Example

```
SELECT * FROM course;
```

```
SELECT * FROM course AS c
INNER JOIN teacher AS t ON t.teacher_id = c.teacher_id;
```

```
SELECT * FROM course AS c
LEFT JOIN teacher AS t ON t.teacher_id = c.teacher_id;
```

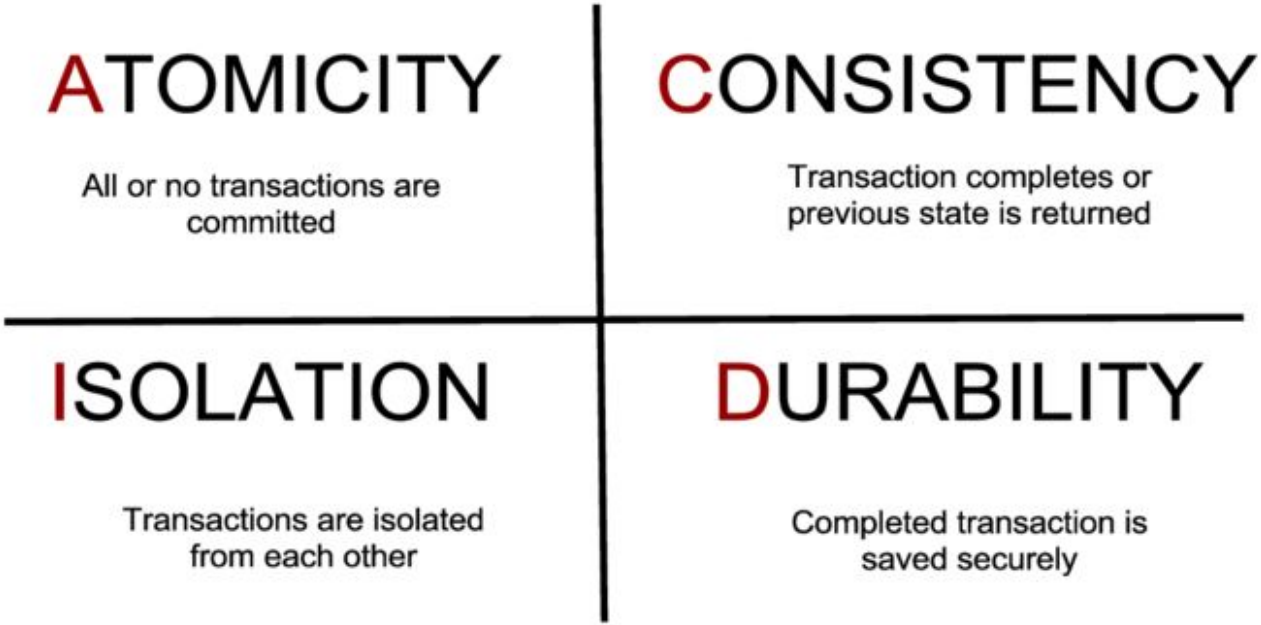
Example

```
SELECT stc.student_id, s.student_name, c.course_name, t.teacher_name

FROM student_course as stc

INNER JOIN course as c ON stc.course_id = c.course_id

INNER JOIN student as s ON s.student_id = stc.student_id

INNER JOIN teacher AS t ON t.teacher_id = c.teacher_id;
```

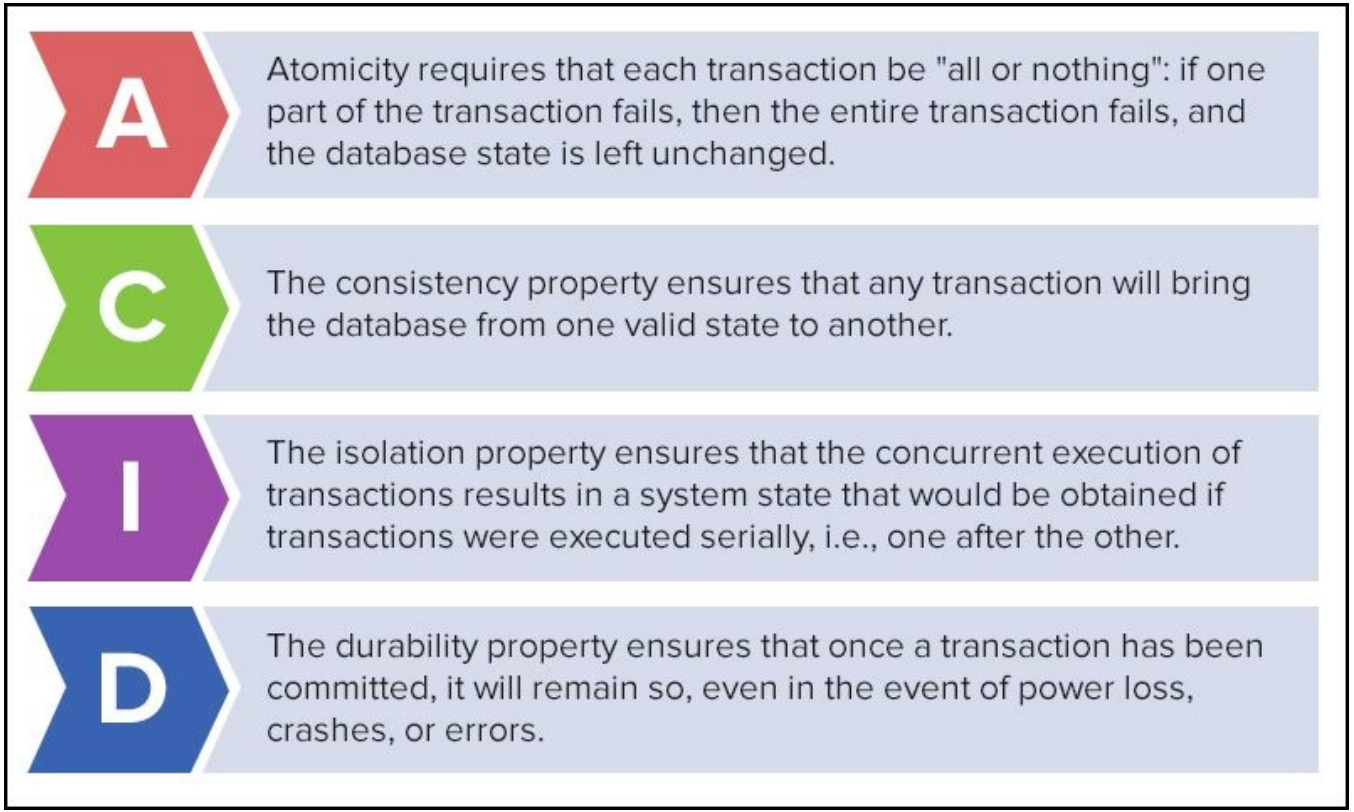
Read more

- Read more about JOIN:
  - https://www.guru99.com/joins.html
  - https://www.edureka.co/blog/sql-joins-types
  - https://www.geeksforgeeks.org/sql-join-set-1-inner-left-right-and-full-joins/

ACID


ATOMICITY
All or no transactions are committed
CONSISTENCY
Transaction completes or previous state is returned
ISOLATION
Transactions are isolated from each other
DURABILITY
Completed transaction is saved securely

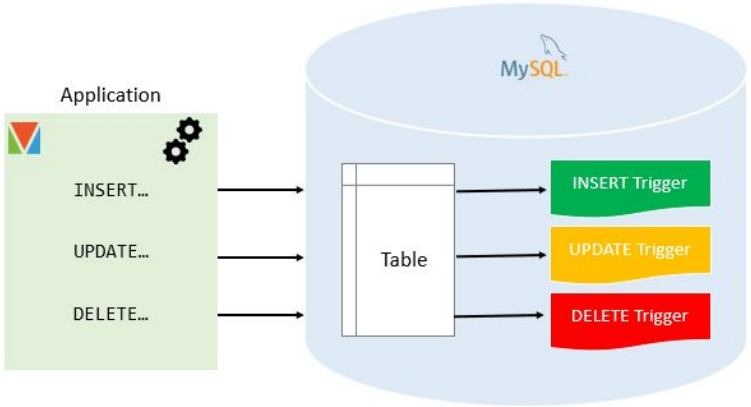
ACID


A
Atomicity requires that each transaction be "all or nothing": if one part of the transaction fails, then the entire transaction fails, and the database state is left unchanged.
C
The consistency property ensures that any transaction will bring the database from one valid state to another.
I
The isolation property ensures that the concurrent execution of transactions results in a system state that would be obtained if transactions were executed serially, i.e., one after the other.
D
The durability property ensures that once a transaction has been committed, it will remain so, even in the event of power loss, crashes, or errors.

Atomicity	• Ensuring that the transaction is completed either fully or not at all, and not left partially complete after a failure.
Consistency	• Ensuring that data are in a valid state at all times.
Isolation	• Ensuring independence of transactions one from another.
Durability	• Ensuring the persistence of a transaction in the event of any failure.

Trigger

- A trigger is a stored program invoked automatically in response to an event such as insert, update, or delete that occurs in the associated table
- Read more:
  - https://www.mysqltutorial.org/mysql-triggers.aspx/
  - https://www.mysqltutorial.org/create-the-first-trigger-in-mysql.aspx/



Trigger

```
CREATE TRIGGER trigger_name

{BEFORE | AFTER} {INSERT | UPDATE| DELETE }

ON table_name FOR EACH ROW

trigger_body;
```

Read: https://www.mysqltutorial.org/create-the-first-trigger-in-mysql.aspx/

Trigger - Example

- Add new table

```
CREATE TABLE school_students (
    id INT AUTO_INCREMENT PRIMARY KEY,
    total_students INT NOT NULL,
    up_to_date DATE
);
```

- Add the trigger

```
CREATE TRIGGER after_insert_students
    AFTER INSERT ON student
    FOR EACH ROW
 INSERT INTO school_students (total_students, up_to_date)
 VALUES((select count(*) from student), NOW());
```

- Check the new table

```
select * from school_students;
```

- Insert value to student table

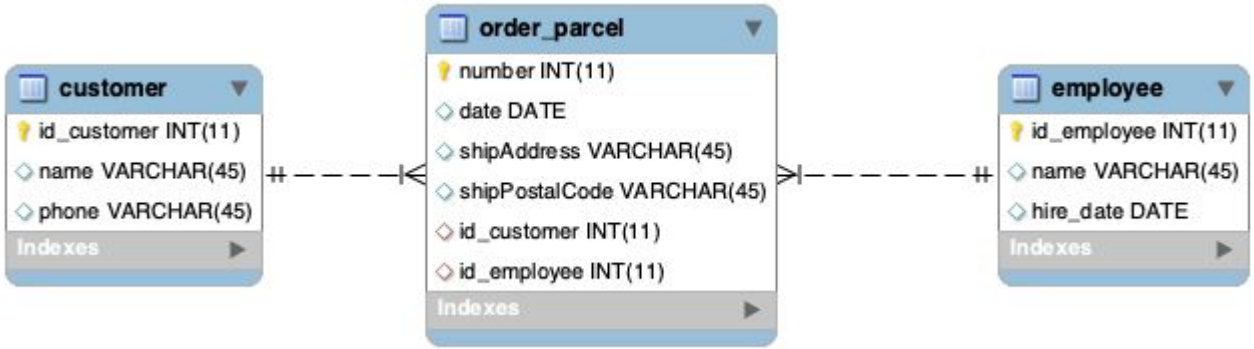
```
Insert into student (student_name) values ('Alonso');

select * from school_students;
```

Trigger

https://www.mysqltutorial.org/mysql-triggers.aspx/

https://www.mysqltutorial.org/create-the-first-trigger-in-mysql.aspx/

Exercises



1. **What type of Relationships can you identify at the diagram above?**
2. **Add a new database: dql_exercise**
3. **Check the diagram above and add the tables and relationships**
4. **Insert data for each table**

   Using DQL:
5. **Display all orders number ordered by date, with employee name, customer name**
6. **Display the total of orders by shipPostalCode**
7. **Display the total of orders by shipPostalCode and customer name**
8. **Display the total of orders by shipPostalCode, date and employee name**

Exercises

1. Write a query to display the firstName, lastName and department name for all employees.
2. Write a query to display the firstName and lastName of the employees assigned to a department.
3. Write a query to display the firstName and lastName of the employees assigned to the HR department.
4. Write a query to display all employees that haven’t been assigned to a department.
5. Write a query to display all employees that work on the Java project.
6. Write a query to display the firstName, lastName, department name and project name for all of the employees that work in the HR department.
7. Write a query to display the firstName, lastName, department name and project name for all of the employees that work on the Java project.
8. Write a query to display the firstName, lastName, department name and project name for all of the employees that work on the Java project and their last name starts with J.
9. Write a query to display only the projects that have employees assigned to them.
10. Write a query to display the departments that don’t have any employees.