EDEM
Escuela de Empresarios

Bases de Datos Relacionales

GFT

Esteban Chiner

Curso 2020/2021 - Edición 2

Fecha 23/10/2020

EDEM
Escuela de Empresarios

Esteban Chiner

- **Computer Engineer** from the Universitat de València & **Master on Software Engineering** from Universitat Politècnica de València
- Working in GFT as a **Senior Architect** specialized in Big Data and Blockchain
- Part of the technological **Tech Incubator** team
- Leading the **"Fast Data"** and **"DLT & Crypto"** domains
- **Lecturer**
  - Data Analytics for Enterprises (**EDEM**)
  - Big Data & Analytics Master (**UPV**) – "Big Data Architectures in Financial Services"
  - Seminaris d'Empresa (**UPC**) – "Developing decentralized apps with Ethereum"
- **Hadoop Certified** in developer, administration and HBase
- Paper published: "**A Big Data Financial Information Management Architecture for Global Banking**"

@echiner
echiner
esteban.chiner@gft.com

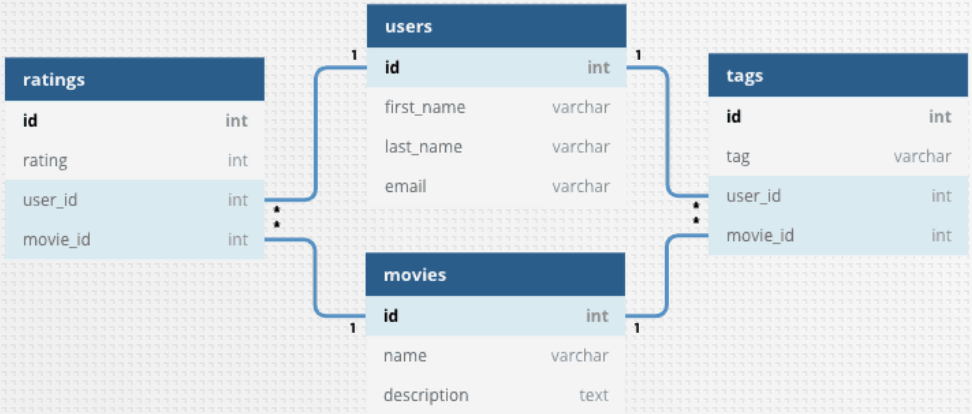
Agenda

1. Some History
2. Relational Databases & Data Model
3. Entity Relationship Model
4. Physical Data Model
5. Stored Procedures
6. Performance Tuning
7. Database Design

Introduction

"A **database** is an **organized collection of data**, generally stored and accessed electronically from a computer system. Where databases are more complex they are often **developed using formal design and modeling techniques**."

– Wikipedia



Relational vs NOSQL

Relational (or RDMS)

A relational database is a digital database based on the relational model of data, as proposed by E. F. Codd in 1970.

NOSQL

A NoSQL database provides a mechanism for storage and retrieval of data that is modeled in means other than the tabular relations used in relational databases.

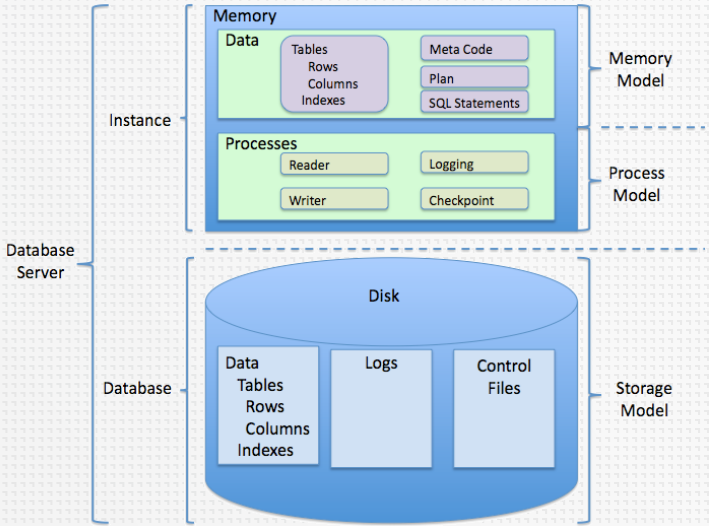
Most popular databases

Rank			DBMS	Database Model	Score		
Oct 2020	Sep 2020	Oct 2019			Oct 2020	Sep 2020	Oct 2019
1.	1.	1.	Oracle	Relational, Multi-model	1368.77	-0.59	+12.89
2.	2.	2.	MySQL	Relational, Multi-model	1256.38	-7.87	-26.69
3.	3.	3.	Microsoft SQL Server	Relational, Multi-model	1043.12	-19.64	-51.60
4.	4.	4.	PostgreSQL	Relational, Multi-model	542.40	+0.12	+58.49
5.	5.	5.	MongoDB	Document, Multi-model	448.02	+1.54	+35.93
6.	6.	6.	IBM Db2	Relational, Multi-model	161.90	+0.66	-8.87
7.	↑ 8.	7.	Elasticsearch	Search engine, Multi-model	153.84	+3.35	+3.67
8.	↓ 7.	8.	Redis	Key-value, Multi-model	153.28	+1.43	+10.37
9.	9.	↑ 11.	SQLite	Relational	125.43	-1.25	+2.80
10.	10.	10.	Cassandra	Wide column	119.10	-0.08	-4.12
11.	11.	↓ 9.	Microsoft Access	Relational	118.25	-0.20	-12.93
12.	12.	↑ 13.	MariaDB	Relational, Multi-model	91.77	+0.16	+5.00
13.	13.	↓ 12.	Splunk	Search engine	89.40	+1.51	+2.57
14.	14.	↑ 15.	Teradata	Relational, Multi-model	75.79	-0.61	-2.95
15.	15.	↓ 14.	Hive	Relational	69.55	-1.62	-15.19
16.	16.	16.	Amazon DynamoDB	Multi-model	68.41	+2.23	+8.24
17.	17.	↑ 25.	Microsoft Azure SQL Database	Relational, Multi-model	64.40	+3.95	+36.89
18.	18.	↑ 19.	SAP Adaptive Server	Relational	55.16	+1.15	-0.67
19.	19.	↑ 20.	SAP HANA	Relational, Multi-model	54.24	+1.38	-1.11
20.	20.	↓ 17.	Solr	Search engine	52.48	+0.86	-5.09

Source: https://db-engines.com/en/ranking

Relational Databases

- A relational database is a digital database based on the **relational model of data**, as proposed by E. F. Codd in 1970
- A software system used to maintain relational databases is a relational database management system (**RDBMS**)
- Many relational database systems have an option of using the **SQL (Structured Query Language)** for querying and maintaining the database



Data Model

- A Data Model is the result of an analysis to define from business perspective what **entities**, **attributes** and **relations** a company manages
- It does not define the business process
- Traditionally this data model has been implemented on **Relational Databases**, however nowadays that has changed using NOSQL technologies
- On traditional approaches this **data model is defined up-front** (schema on write)
  - That creates a rigid structure for current known processes
  - It is hard to perfrom data insights and data scientist work

Entity Relationship Model

Levels of Abstraction

Conceptual data model

- Least granular details (entities)
- Stablishes overall scope
- Usually used to define master reference data common to all organization
- High level structural metadata used by the Logical models

Logical data model

- The logical model contains more detail than the conceptual model
- In addition to master data entities, operational and transactional data entities are now defined
- The details of each data entity are developed and the relationships between these data entities are established
- It is independent of the specific database management system

Levels of Abstraction

Physical data model

- The physical ER model is normally developed to be instantiated as a database
- Contains enough detail to build it on a specific database technology
- The physical model usually defines the database schema such as database tables, attributes, keys, etc.

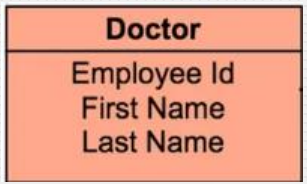
ERD features	Conceptual	Logical	Physical
Entity (Name)	Yes	Yes	Yes
Relationship	Yes	Yes	Yes
Columns		Yes	Yes
Column's Types		Optional	Yes
Primary Key			Yes
Foreign Key			Yes

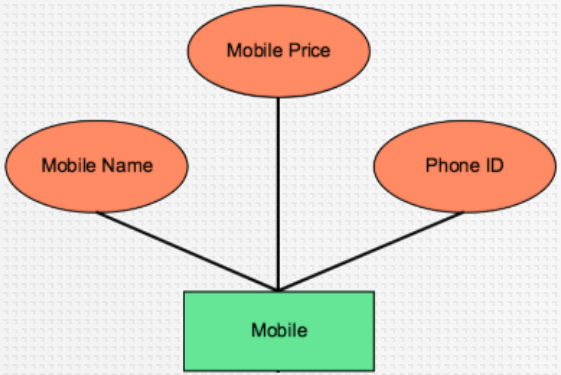
Entity Relationship Model

- Relational Databases relies on an **entity–relationship model**
- It is composed of entity types (which classify the things of interest) and specifies relationships that can exist between instances of them
- It does not define the business processes. Represents a business **data schema** in graphical form
- It is an **structured model**, that requires a systematic analysis to define and conceptualize how data should be stored to be used by the Business Processes

Entity

- It **represents and object or a concept** that exists independently, so it can be identified from others. For example:
  - Person
  - Telephone
  - Account
- Can be a physical thing or a conceptual thing
- It is described by its atrributes. For instance a Car has a Model, Id, Owner, Year, etc..



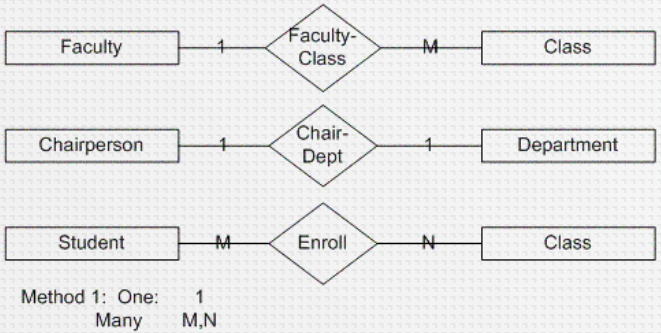


Attributes

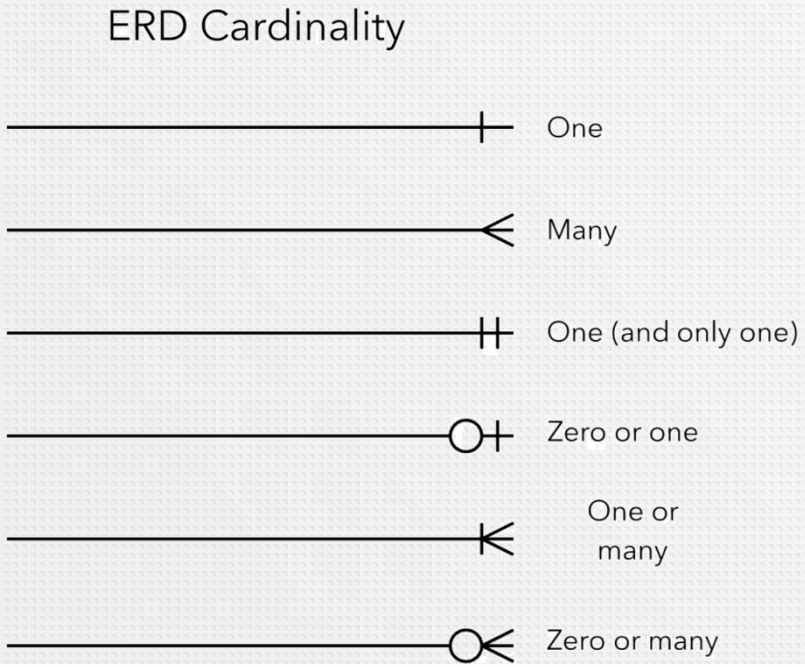
- Attributes describes entity **properties** and **features**
- **Each entity has it's own attributes values**, creating instance of an entity that can be identified unambiguously
  - Ex. A set of Doctors that has common attributes (id, name, surname):
    - (1, Alice, Williams)
    - (2, Bob, Smith)
    - (3, Salma, Spencer)
    - ...
- Entities pertaining to same set differentiate from others by the value of their attributes. **Two entities cannot have all values equal** (duplicate)
- An attribute can be a **key** that **identifies instances of entities**
- Also an attribute has a **type** (integer, string, etc.) and can have a **domain** of allowed values

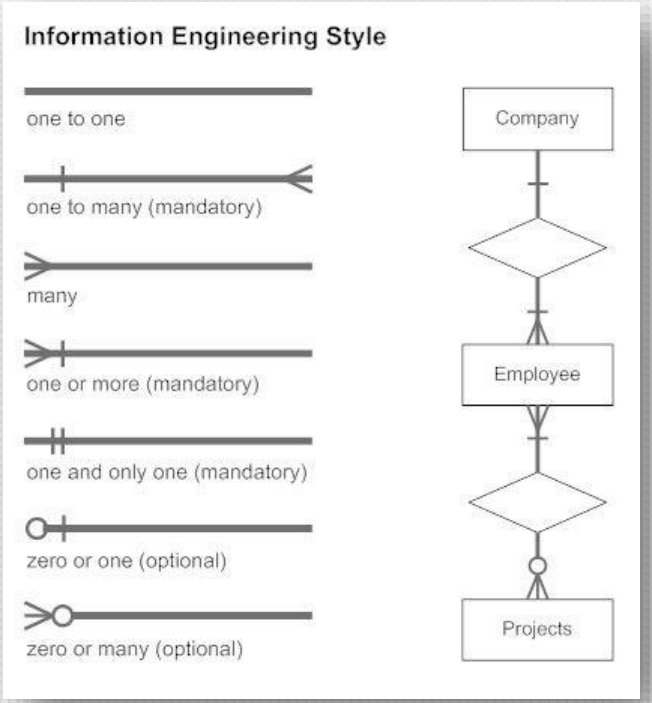
Relations

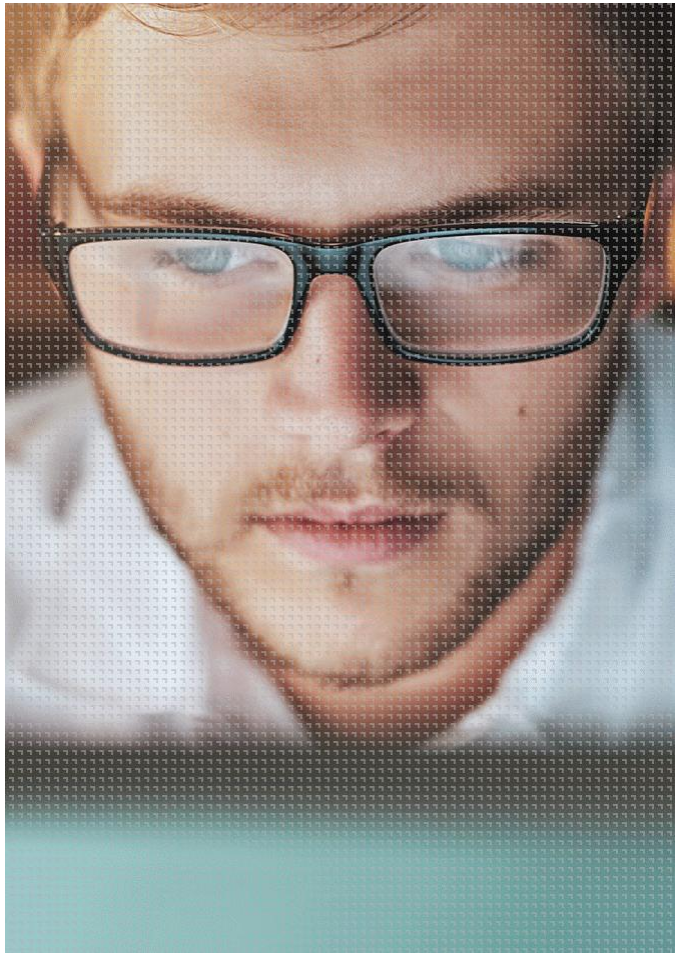
- A relation is defined by its **cardinality**:
  - Number of instances from one entity that has a relation with instances from other entity.
  - **"0"** if an instance of one entity is not mandatory to have a relation with instances from other entity.
    - Ex. A car can have one owner or not. So is a relation of 0:1
  - **"1"** if an instance of one entity is mandatory to have a relation with instances from other entity.
    - Ex. A Country has 1 and only one prime minister so is a relation of 1:1
  - **"N" , "M", or "*"** if an instance on one entity has a relation with more that one instances of other entity.
    - Ex. A person can have several cars. So the relation is 0:N



Cardinality (Crow's Foot Notation)







Exercise – Create a basic data model

Think (and draw) a simple data model which implements the following:

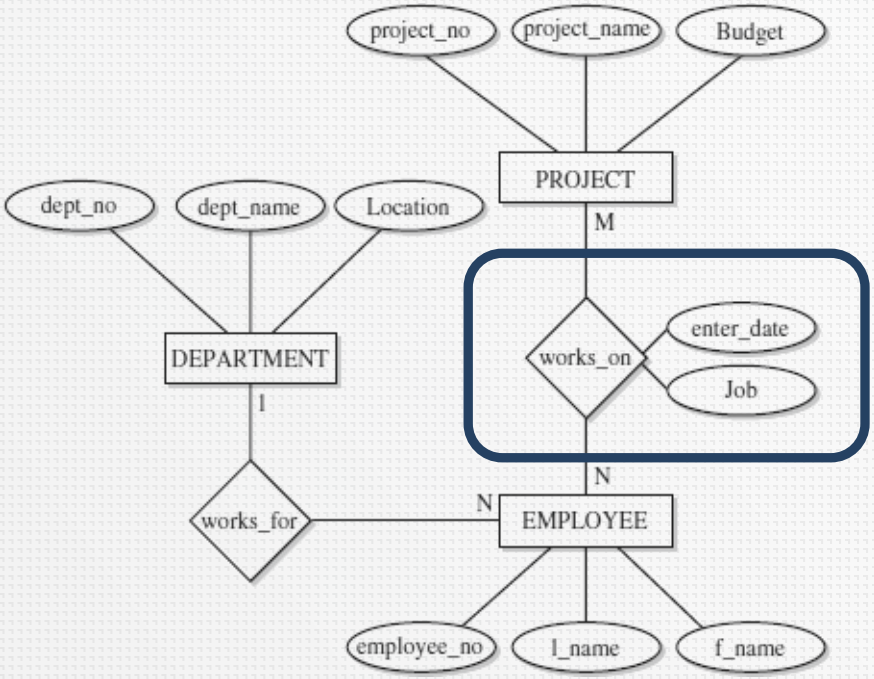
“A library catalog, that is, relationship between books, authors and borrowers.”

TIP1: You don’t need to add the attributes, just the **entities** and **relationships**.

TIP2: Feel free to use the “numeric” or “crow’s foot” notation.

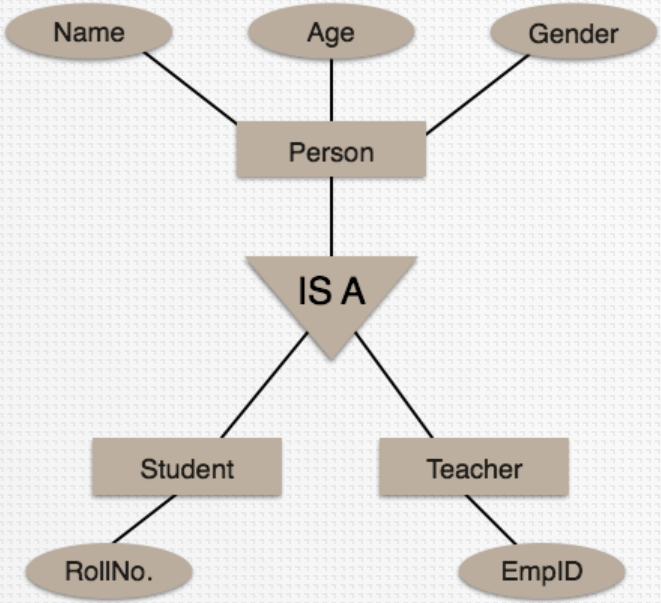
Relations Attributes

- **A relation can have also attributes**
- Usually those attributes are stored on a new entity that defines the relation or on one of two related entities



Inheritance

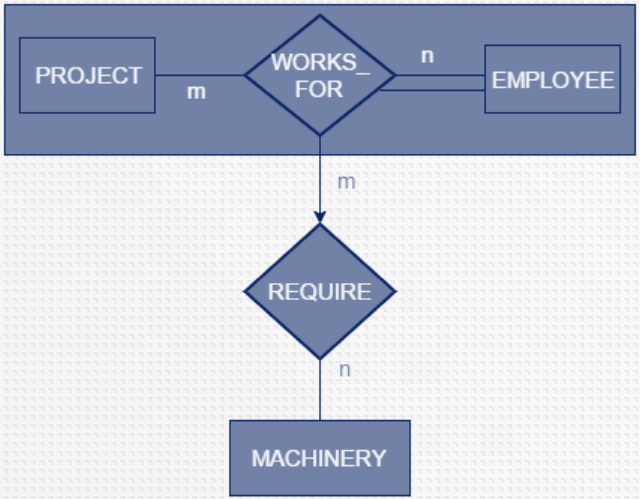
- Inheritance is something that tries to use **Object Oriented Programming** features to database models
- There is a **parent entity** and a **child entity**
- Child entity inherits all attributes and relations of parent one
- Only one parent is allowed (there is not multiple inheritance)



Aggregation

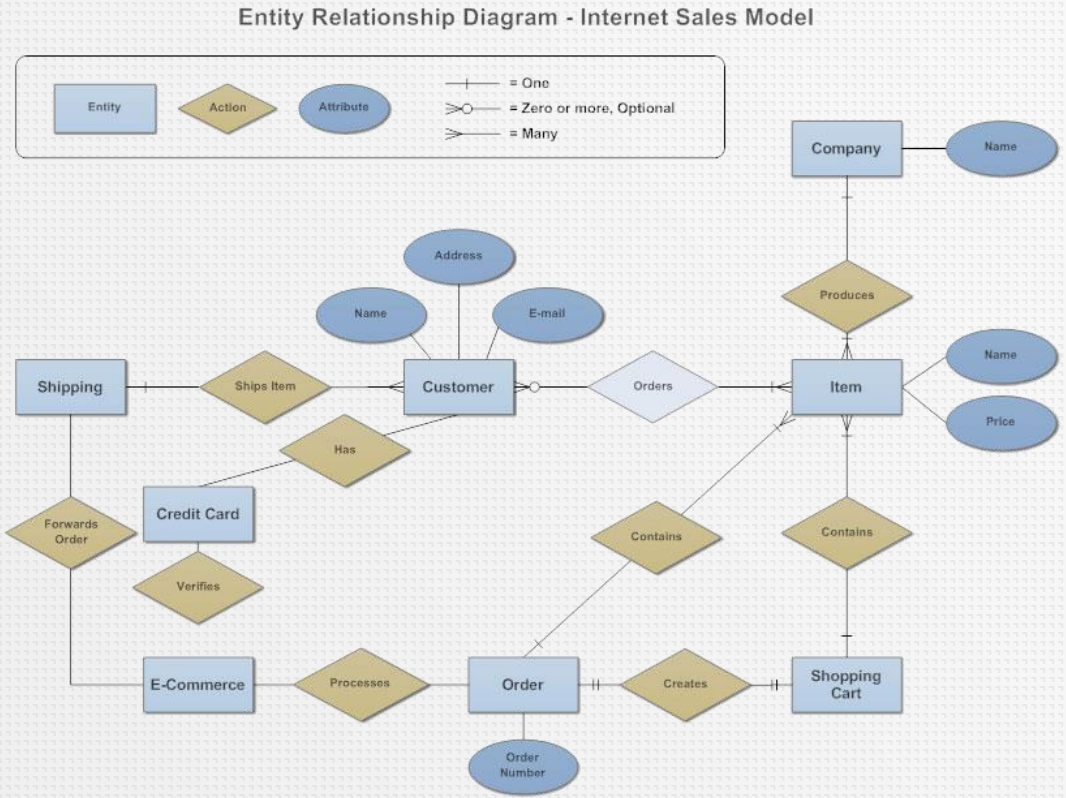
- An ER diagram is not capable of representing relationship between an entity and a relationship which may be required in some scenarios

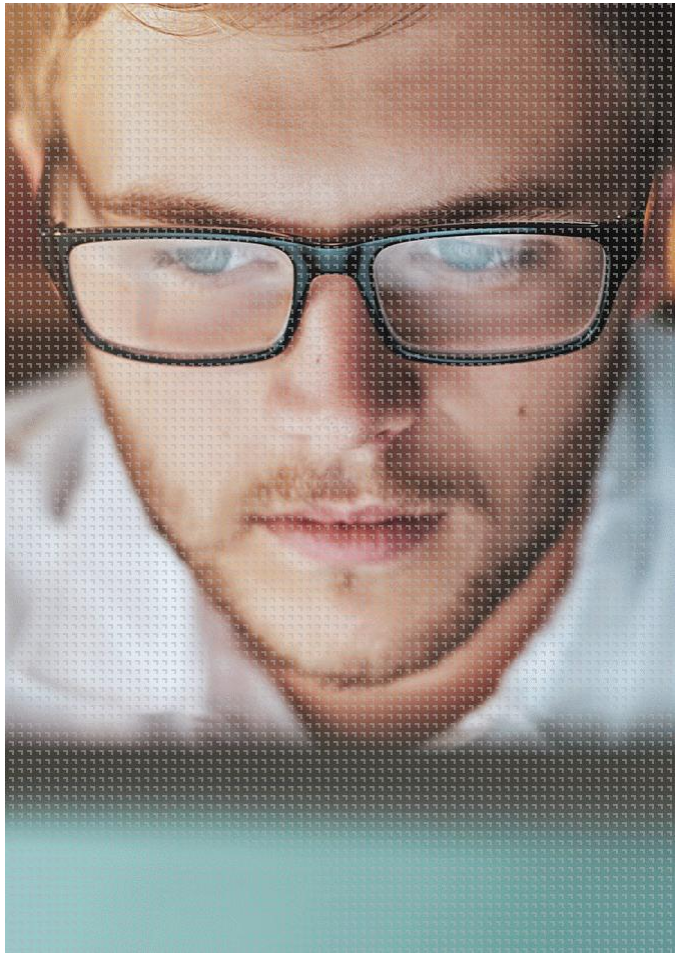
- A relationship with its corresponding entities is aggregated into a higher level entity.



Aggregation

Example of a diagram





Exercise 1 – Creating a Data Model

In this exercise you will design an **Entity Relationship Data Model** for a use case.

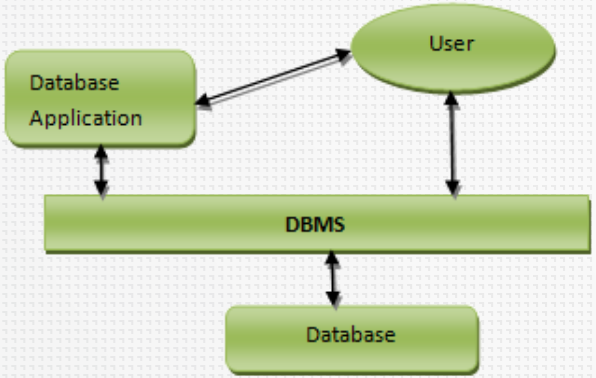
Follow the steps defined in **Exercise 1**:

https://github.com/echiner/edem-mda-relational-databases

Physical Data Model

Relational Database Architecture

- **User:** Users are the one who really uses the database. Users can be administrator, developer or the end users
- **Data or Database:** There are two types of data. It contains the data itself physically stored and structured depending on the database management system. Another data is Metadata, tables names, how many columns names, constraints, etc.
- **DBMS:** This is the software helps the user to interact with the data. It allows the users to insert, delete, update or retrieve the data, by means of database system (MySQL, Oracle, etc.)
- **Database Application:** It the application program which helps the users to interact with the database by means of query languages. It uses SQL language, so has no idea about underlying DBMS.



Tables

- Entities are translated to physical tables

PEN						
NAME	TYPE	PEN_COLOR	INK_CLOR	ORIENTATION	WEIGHT	EASE_OF_USE
Reynolds	BallPoint	Blue	Blue	Linear	0.5	Yes
Reynolds	BallPoint	Blue	Black	Linear	0.5	Yes
Reynolds	BallPoint	Red	Red	Linear	0.5	Yes
Reynolds	BallPoint	Black	Red	Linear	0.5	Yes

```
CREATE TABLE PEN (
            NAME VARCHAR2 (15),
            TYPE VARCHAR2 (10),
            PEN_COLOR VARCHAR2 (15),
            INK_COLOR VARCHAR2 (15),
            ORIENTATION VARCHAR2 (15),
            WEIGHT NUMBER (3, 2),
            EASE_OF_USE BOOLEAN
);
```

Views

- Views looks like a table but are generated by a **query** over other tables
- Also called named query or stored query
- Views are generated **on the fly** executing defined query
- They are **virtual tables** and hence doesn't use storage space
- Always have **updated data**

```
CREATE OR REPLACE VIEW vw_RedInkPen AS
          SELECT * FROM Pen
          WHERE INK_COLOR = 'Red';
```

Materialized Views

- Are like Views, but query results are **stored**
- That speedup accessing to the data
- If queried tables changes, that's not reflected on the materialized view
- Requires **refreshes**

```
CREATE MATERIALIZED VIEW mv_Design_Emp AS
 SELECT e.EMP_ID, e.EMP_FIRST_NAME, e.EMP_LAST_NAME, d.DEPT_ID, d.DEPT_NAME
 FROM EMPLOYEE e, DEPARTMENT d
 WHERE e.DEPT_ID = d.DEPT_ID AND d.DEPT_NAME = 'DESIGN';
```

Data Integrity

- Data Integrity ensures that the **data is consistent**, complete and and there is no mismatches

STUDENT_ID	STUDENT_NAME	ADDRESS	SUBJECT
100	Joseph	Alaiedon Township	Mathematics
101	Allen	Fraser Township	Chemistry
100	Joseph	Alaiedon Township	Physics
102	Chris	Clinton Township	Mathematics
103	Patty	Troy	Physics

Data Integrity

- Data integrity is achieved using **constraints**, like:
  - **NOT NULL** → Cannot insert a NULL value (that is, “empty”)
  - **UNIQUE** → Cannot have to rows with this same value
  - **PRIMARY KEY** → Identifies a row. It is usually NOT NULL, UNIQUE and indexed
  - **FOREIGN KEY** → References an ID in a different table (it must exist in the referenced table)
  - **CHECK** → Custom check for a value
  - **DEFAULT** → Sets a default value (if not entered by the user)
- Those constraints are computational expensive, for Big Data on NOSQL databases are relaxed

Domain Constraint

- Domain Constraint
  - Data types of the columns are correct
  - It pertains to an allowed list of values

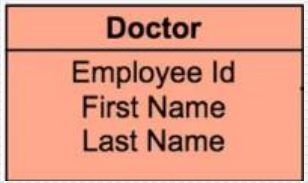
- On SQL it is defined using CHECK condition

```
CREATE TABLE ACCOUNT (
            ACCOUNT_ID NUMBER (10),
            ACCOUNT_TYPE CHAR(12) CHECK (VALUE IN (“Checking”,”Saving”)
);
```

Entity Constraint

- Each record of a table has following properties
  - It has a primary key
  - Primary key is not null

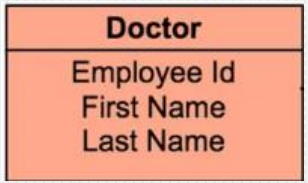
- **Primary Key**: atribute or list of atributes of an entity that identifies it from others, so their value is unique. For instance Person DNI.




```
CREATE TABLE DOCTOR (
            EMPLOYEE_ID NUMBER (10) PRIMARY KEY,
            FIRST_NAME VARCHAR2 (50),
            LAST_NAME VARCHAR2 (50)
);
```

Entity Constraint

- **NOT NULL**: This constraint forces the column to have non-null value




```
CREATE TABLE DOCTOR(
            EMPLOYEE_ID NUMBER (10) PRIMARY KEY,
            FIRST_NAME VARCHAR2 (50) NOT NULL,
            LAST_NAME VARCHAR2 (50)
);
```

Column Constraint

- This constraint ensures that the column values are meeting a defined business rules:
  - Age is not negative
  - Phone number has 9 digits
  - Value is Unique
  - It has a Default value

```
CREATE TABLE STUDENT (
          STUDENT_ID NUMBER (10),
          STUDENT_DNI VARCHAR2 (9) UNIQUE,
          AGE NUMBER CHECK (AGE >= 25 and AGE<= 32),
          CREATED_DATE DATE DEFAULT SYSDATE)
);
```

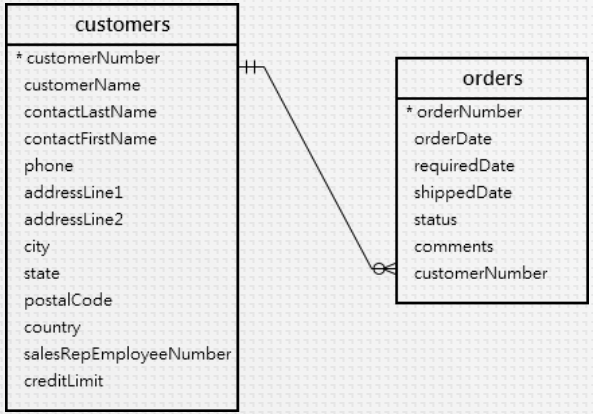
User Defined Column Constraint

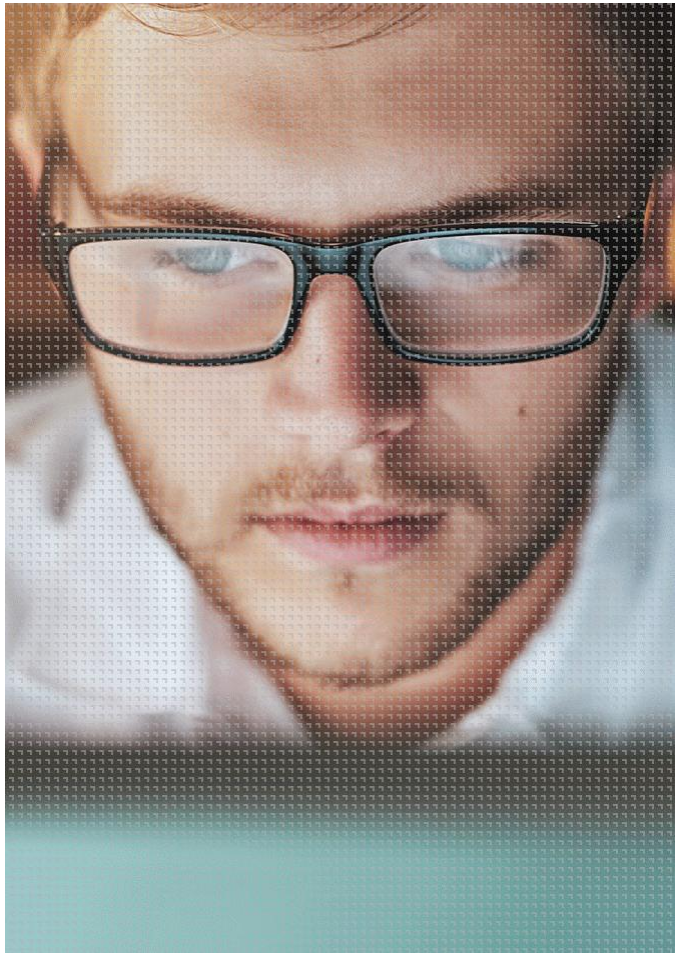
- Column value is checked against values of other column or records
  - That implies developing code
  - Ex. How to ensure that if you own a car you have a driver license

Referential Integrity

- Ensures that entities **relations are consistent**
  - Ex. If I'm a teacher, my department exists on Department table

- **Foreign Key**: It is used to describe relations, and atribute or list of atributes of an entity that links with other entity attributes





Exercise 2 – Implementing the Data Model

In this exercise you will create a database based on the ER created in Exercise 1.

Follow the steps defined in **Exercise 2**:

https://github.com/echiner/edem-mda-relational-databases

Adminer

MySQL®

Data Schema

OLTP vs. OLAP

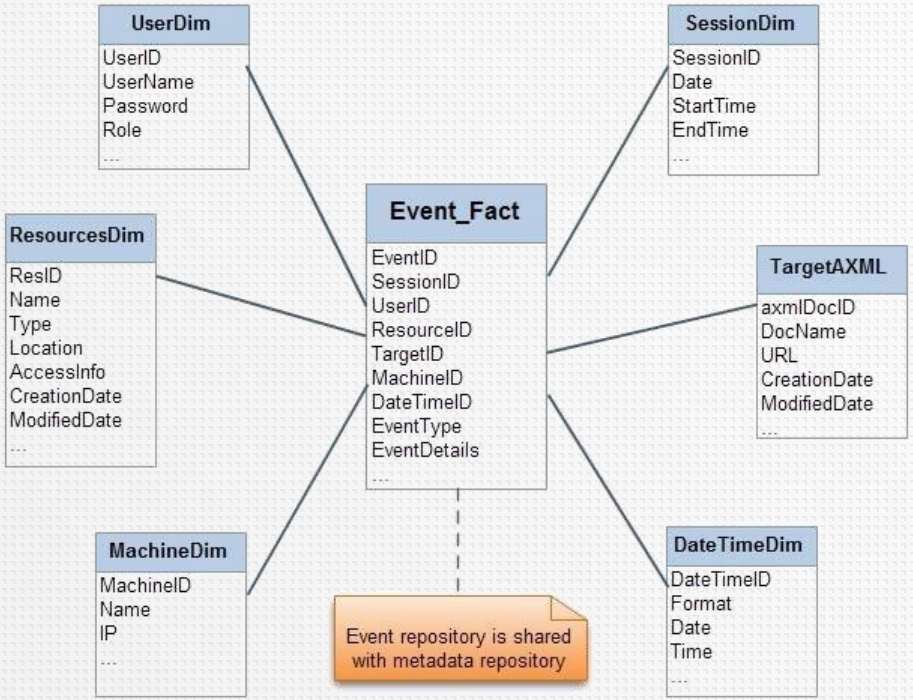
ONLINE TRANSACTION PROCESSING	ONLINE ANALYTICAL PROCESSING
Handles recent operational data	Handles all historical data
Size is smaller, typically ranging from 100 Mb to 10 Gb	Size is larger, typically ranging from 1 Tb to 100 Pb
Goal is to perform day-to-day operations	Goal is to make decisions from large data sources
Uses simple queries	Uses complex queries
Faster processing speeds	Slower processing speeds
Requires read/write operations	Requires only read operations

Data Schema

- How data is strucuted on tables, relationships, attributes is also called Data or Database Schema
- It is related to the use case you want to cover
  - OLTP
  - OLAP
- There are different levels of data normalization that you can apply
  - Star schema
  - Snowflake schema

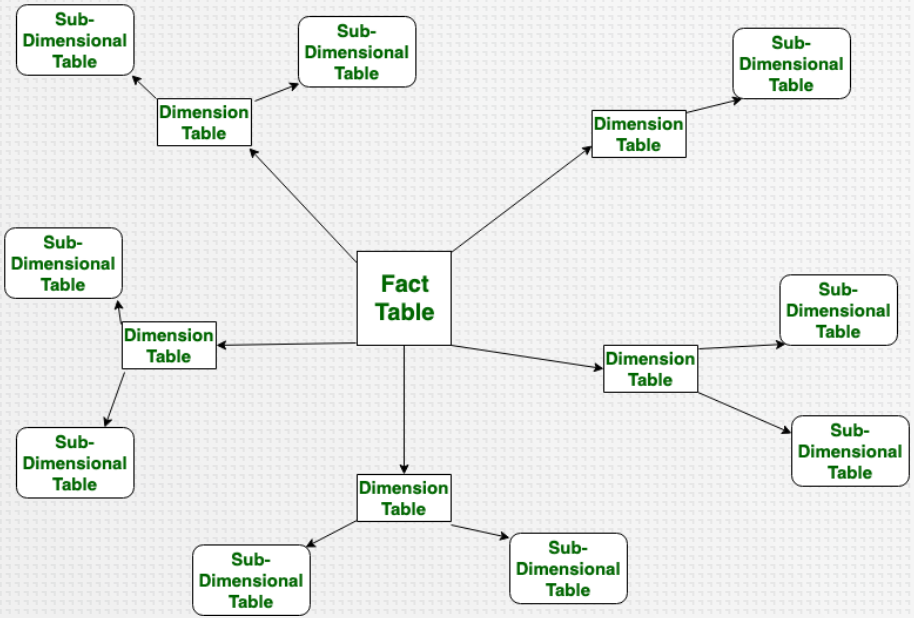
Star schema

- It is oriented to **Data Warehouses** (OLAP)
- The center of the star can have one or more fact tables (Ex. Transactions, operations,etc)
- Those tables has references to dimension tables (reference data)
- This schema is optimized for large datasets
- The dimension tables are **not normalized**



Snowflake schema

- It is also oriented to Data Warehouses (OLAP)
- Main differences is that **dimension tables are also normalized**



Stored procedures

PL/SQL

- **PL/SQL (Procedural Language for SQL)** is is an Oracle extension of SQL, but also used on other databases la IBM DB2. Usually used to build Stored Procedures.

  - **Function** → Piece of code that can be invoked several to compute and return a single value.
  - **Procedure** → Similar to a function but can return multiple values
  - **Packages** → Packages are groups of conceptually linked functions, procedures, variables,
  - **Triggers** → Is like a stored procedure that Oracle Database invokes automatically whenever a specified event occur (INSERT, UPDATE, DELETE, etc…)

Purpose of Triggers

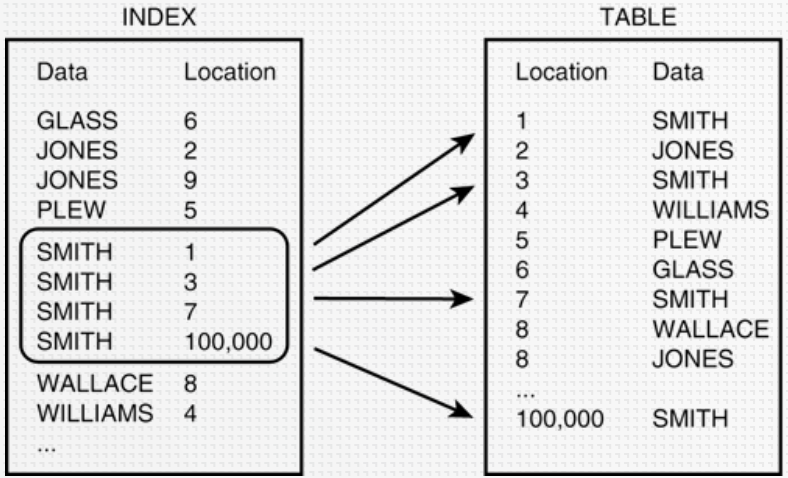
- Generating some derived column values automatically
- Event logging and storing information on table access
- Auditing
- Synchronous replication of tables
- Imposing security authorizations
- Preventing invalid transactions

Performance tuning

Indexes

- Data structure that **improves the speed of data retrieval**
- Allows to quickly locate data without having to search every row in a database on a query
- Can be created using one or more columns of a database table, providing the basis for both rapid random lookups and efficient access of ordered records.

```
CREATE INDEX idx_lastname
ON Persons (LastName);
```



Partitions

- A **partition** is a division of a logical database or its constituent elements into distinct independent parts
- Database partitioning is normally done for **manageability**, **performance**, **availability** reasons or for load balancing
- Partitioning criteria:
  - **Range partitioning** - selects a partition by determining if the partitioning key is inside a certain range.
  - **List partitioning** - a partition is assigned a list of values. If the partitioning key has one of these values, the partition is chosen.
  - **Round-robin partitioning** - the simplest strategy, it ensures uniform data distribution. With n partitions, the ith tuple in insertion order is assigned to partition (i mod n).
  - **Hash partitioning** - applies a hash function to some attribute that yields the partition number.
  - **Composite partitioning** - allows for certain combinations of the above partitioning schemes, by for example first applying a range partitioning and then a hash partitioning.

Database design

Factors to consider Designing a Database

- Data is structured and distributed among the right tables
- There is no duplication data
- There is no missing data
- The storage space
- Query patterns (indexes, partitioning)

GFT

Shaping the future of digital business

Esteban Chiner Sanz

GFT