01 - Designs and develops database systems to manage data efficiently and effectively

Designs and develops database systems to manage data efficiently and effectively

Structured Vs. unstructured data

Structured Data

- Structured data refers to any data that resides in a fixed field within a record or file.
- This includes defining what fields of data will be stored and how that data will be stored
- Ability to enter easily
- Managed using Structured Query Language (SQL)
- (numeric, currency, alphabetic, name, date, address

Unstructured Data

- Unstructured data is all those things that can't be so readily classified and fit into a neat box
- (photos and graphic images, videos, streaming instrument data, webpages, PDF files)

Structured Data	Unstructured Data
Fit into a pre-defined model	Doesn't fit into a pre-defined model
Easy to search	Difficult to seach
Usually text based	Could text based or binary based
Often stored in Relational databases	Often stored in Non-relational databases (NoSQL)
Examples: Dates, Addresses, NIC, Names	Examples: PDF files, Images, Videos, Audio files

	Structured Data	Unstructured Data
Characteristics	• Pre-defined data models • Usually text only • Easy to search	• No pre-defined data model • May be text, images, sound, video or other formats • Difficult to search
Resides in	• Relational databases • Data warehouses	• Applications • NoSQL databases • Data warehouses • Data lakes
Generated by	Humans or machines	Humans or machines
Typical applications	• Airline reservation systems • Inventory control • CRM systems • ERP systems	• Word processing • Presentation software • Email clients • Tools for viewing or editing media
Examples	• Dates • Phone numbers • Social security numbers • Credit card numbers • Customer names • Addresses • Product names and numbers • Transaction information	• Text files • Reports • Email messages • Audio files • Video files • Images • Surveillance imagery

What is a Database?

```
Database is a specialized structure that allow computer-based systems to store, manage and retrieve data very quickly.
```

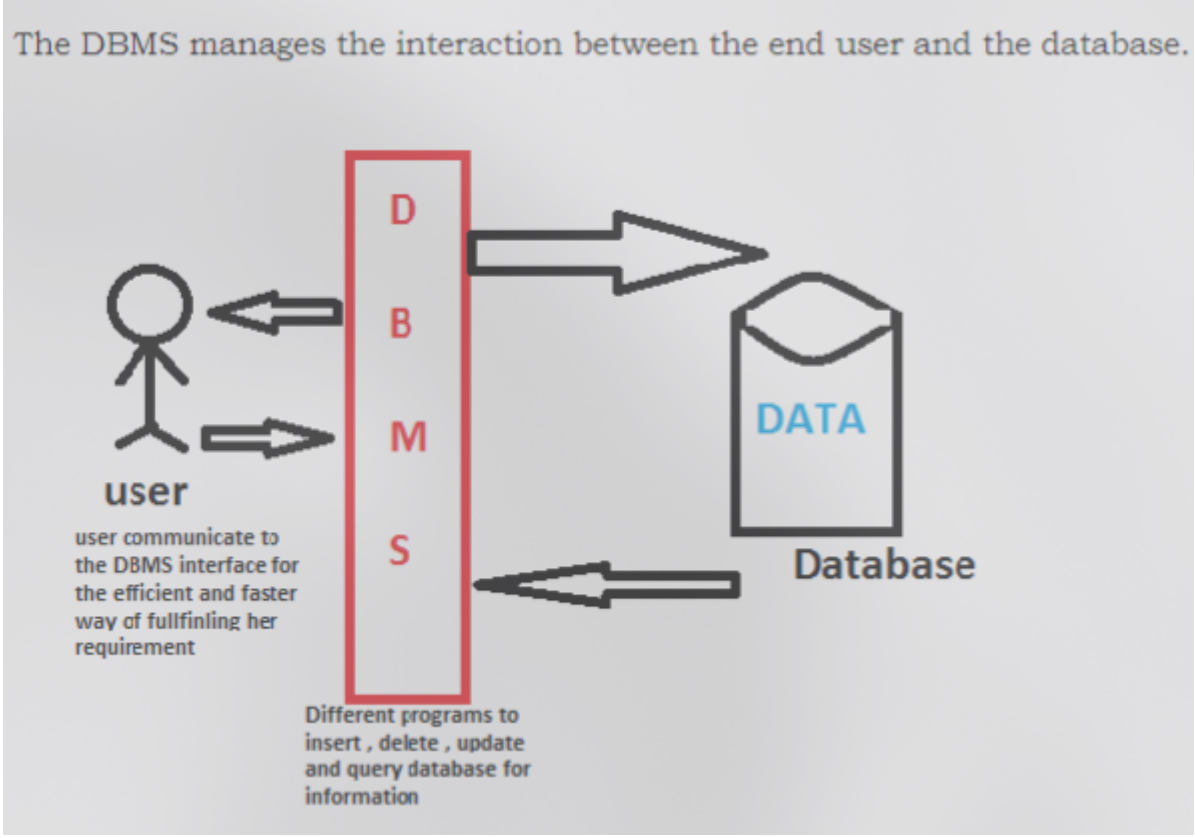
Problems with not using the database.

- Security
- Accuracy
- Redundancy
- Incomplete Data

File Based Systems Vs Databases	
File	**Database**
• Data scattered every where	• Data is integrated
• Sometimes different formats are used	• Data is stored in a common format for all applications to use
• Same data may be stored more than once (duplicated)	• Data duplication is reduced
	• Data can be used simultaneously by multiple users

What is a DBMS?

```
DBMS is a collection of programs trhat manage database sturcture, control access to data, facilitate the sharing of data, anables efficient and effective data management.
```



Benefits of a DBMS

- Provides consistent view of data and operations
- enables ad-hoc queries
- Reduces data inconsistencies and errors.
- Helps with data security

Database models

- Flat file system
- Hierarchical model
- Network model
- Relational model
- Object Relational model

Flat File system

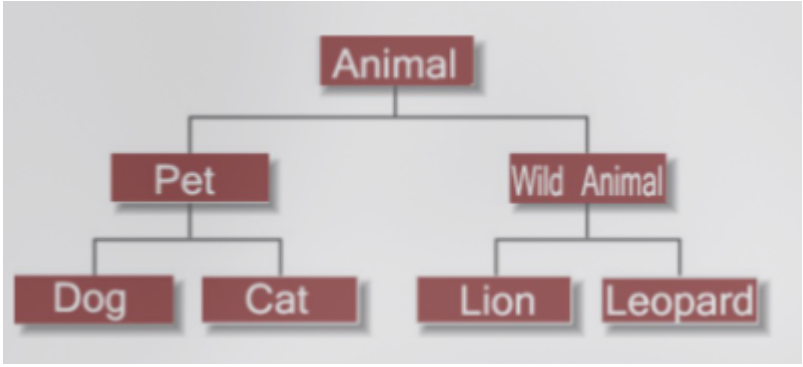
- Firstly introduced model
- All the data used is kept in the same plane.
- This model is not so scientific.
- (Excel, Calc)

ID	Title	First name	Surname	Address	City	Postcode	Telephone
1	Mr	Tom	Smith	42 Mill Street	London	WE13GW	0103440044
2	Mrs	Sandra	Jones	10 Low Lane	Hull	HU237HJ	022344033
2	Mr	John	Jones	10 Low Lane	Hull	HU237HJ	022344033

Hierarchical Model

- Hierarchical model has one parent entity with several children entity

- But at the top we should have only one entity called **root**
- The root can have several children entities.

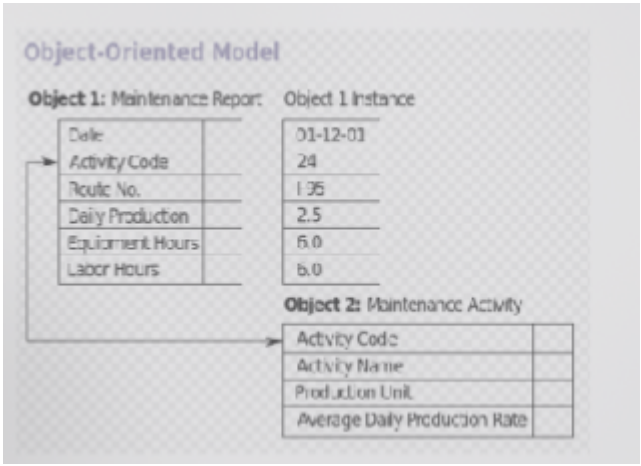


Network Model

- Entities are stored in a graphical representation
- Some entities are accessible through several paths.

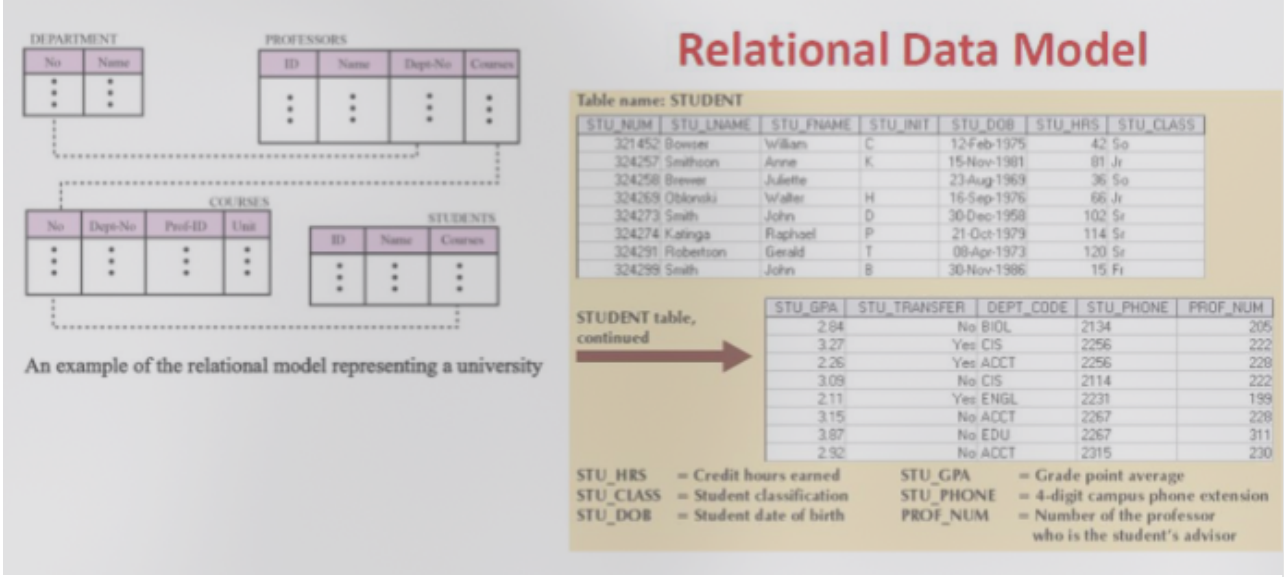
Object Relational Model

- widely used in computer applications to **hold audio, video and graphic files**
- consist of data piece and the methods which are the DBMS instructions.



Relational Model

- **most popular** model and the **most extensively** used model
- data can be stored in the tables, these storing is called relations



Comparison of database models

Criteria	Flat file system	Hierarchical model	Network model	Relational model	Object relational model
Duration	1950 - 1960	1960 -1970 -	1960 - 1970	1970 - to date	1980 – to date
Short Name	DBF	DBMS	DBMS	RDBMS	ODBMS
Used computer programming languages	Assembler, Fortran, COBOL	COBOL, PL1, Fortran	COBOL, PL1, Fortran	SQL, ODBC	Java, C++, Pascal, Python

02 - Describes the main components of the relational database model

Describes the main components of the relational database model

Characteristics of Tables

- A table contains data about a group of related entities
- Table: entity set, a two-dimensional structure composed of rows and columns
- Each table row (tuple) represents a single entity occurrence with the entity set - a Student
- Each table Column represents an attribute and each column has a distinct name - Student ID
- All values in a column must conform to the same data type/format
- Each table must have a key, one or more attributes that uniquely identifies each row

DBMS Keys

A DBMS key is an attribute or set of an attribute which helps you to identify a row

Primary Key

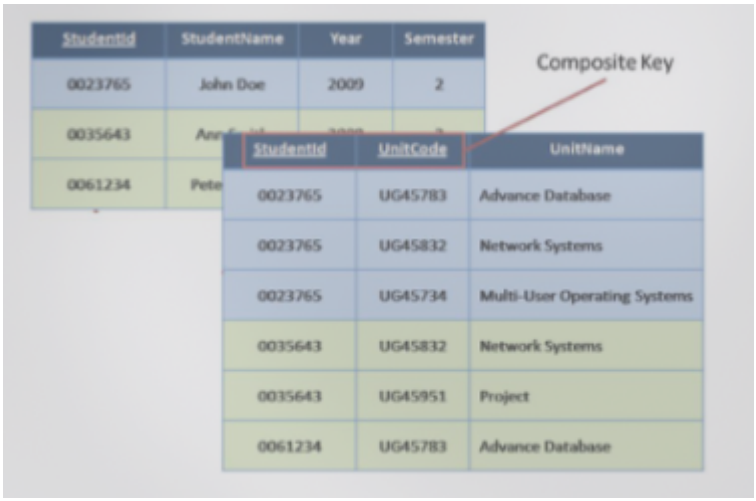
A key with a singel attribute that is used to identify a certain record uniquely is called a primary key.

Rules for defining Primary key

- Two rows can't have the same primary key value
- Its a must for every row to have a primary key value.
- The primary key field cannot be null.
- The value in a primary key column can never be modified or updated **if any foreign key refers to that primary key**
- Each table must have a Primary Key

Composite key

A key which has multiple attributes to uniquely identify rows in a table is called a composite key



Candidate key

A super key with no repeated attribute is called a candidate key.

> A candidate key is a possible primary key, as it doesn't repeat itself

- The Primary key should be selected from the candidate keys. Every table must have at least a single candidate key.

Properties of a candidate key

- It must contain unique values

- Candidate key may have multiple attributes
- Must not contain null values
- It should contain minimum fields to ensure uniqueness
- Uniquely identify each record in a table.

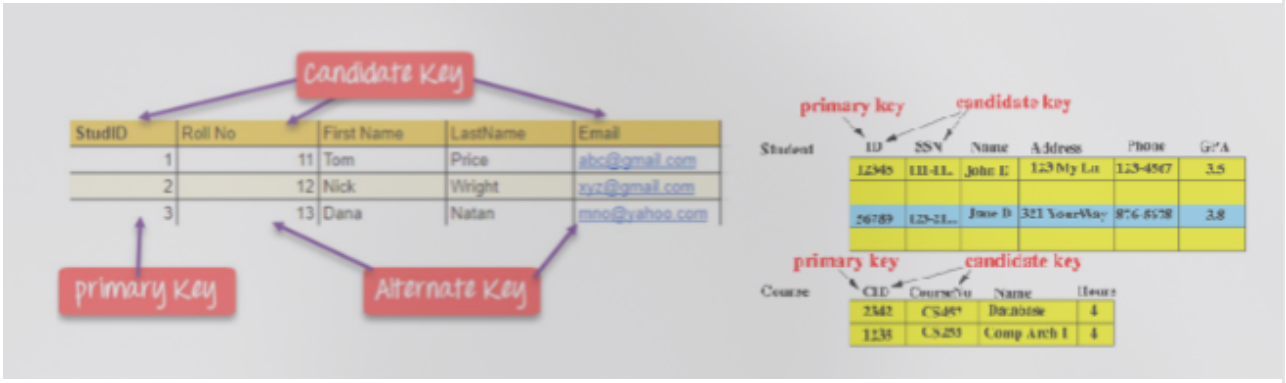
Example: In the given table Stud ID, Roll No, and email are candidate keys which help us to uniquely identify the student record in the table.

StudID	Roll No	First Name	LastName	Email
1	11	Tom	Price	abc@gmail.com
2	12	Nick	Wright	xyz@gmail.com
3	13	Dana	Natan	mno@yahoo.com

Alternate Key

Keys that are left of candidates keys after chosing the primary key.

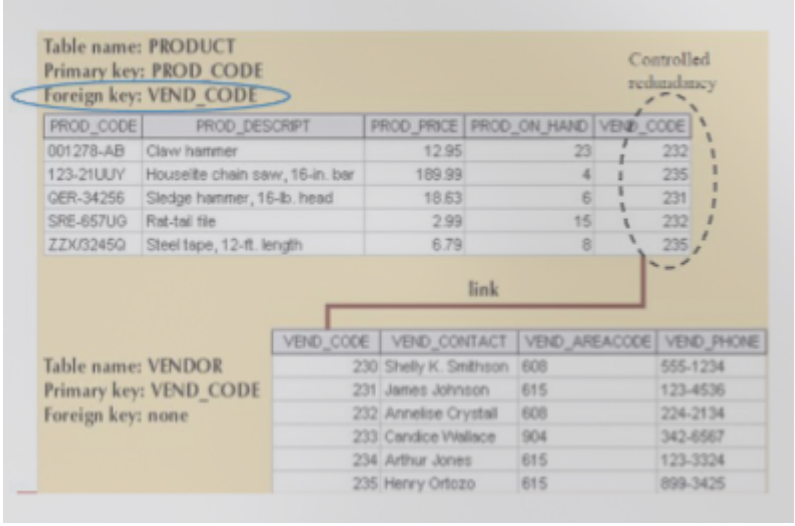
- Technically, it is a candidate key which is currently not the primary key.
- A table may have single or multiple choices for the primary key



Foreign key (FK)

A foreign key is a column which is added to create a relationship with another table

- helps to maintain integrity of data
- allow navigation easy
- if using a foreign key, every model needs to be supported by a foreign key.
- Helps reduce redundancy. (Redundancy is unnecessary duplication of data)
- Foreign key is nullable.



Difference between Primary key & foreign key

Primary Key	Foreign Key
Helps you to uniquely identify a record in the table.	It is a field in the table that is the primary key of another table.
Primary Key never accept null values.	A foreign key may accept multiple null values.
Primary key is a clustered index and data in the DBMS table are physically organized in the sequence of the clustered index.	A foreign key cannot automatically create an index, clustered or non-clustered. However, you can manually create an index on the foreign key.
You can have the single Primary key in a table.	You can have multiple foreign keys in a table.

Key Type	Description
Super key	An attribute (or combination of attributes) that uniquely identifies each row in a table.
Candidate Key	A minimal (irreducible) super key. A super key that does not contain a subset of attributes that is itself a super key.
Primary key	A candidate key selected to uniquely identify all other attribute values in any given row. Cannot contain null entries.
Secondary key	An attribute (or combination of attributes) used strictly for data retrieval purposes.
Foreign key	An attribute (or combination of attributes) in one table whose values must either match the primary key in another table or be null.

Integrity Rules

Primary key

- All PK entries are unique.
- No part of a PK may be null.

Foreign key may have:

- An entry matching a PK value in the related table.
- A null entry.

- How a foreign key is set for a table

```
-- Create the Customers table
CREATE TABLE Customers (
    CustomerID INT PRIMARY KEY,
    CustomerName VARCHAR(50),
    CustomerAddress VARCHAR(100)
);

-- Create the Orders table
CREATE TABLE Orders (
    OrderID INT PRIMARY KEY,
    OrderDate DATE,
    CustomerID INT,
    FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)
);
```

Domain

```
A domain is defined as the set of all unique values permitted for an attribute
```

- For example, a domain of date is the set of all possible valid dates, a domain of integer is all possible whole numbers

Examples

Attribute	Domain	Valid Data Values
No	Int(5)	0 to 9 numbers
Name	Varchar(20)	A to Z letters

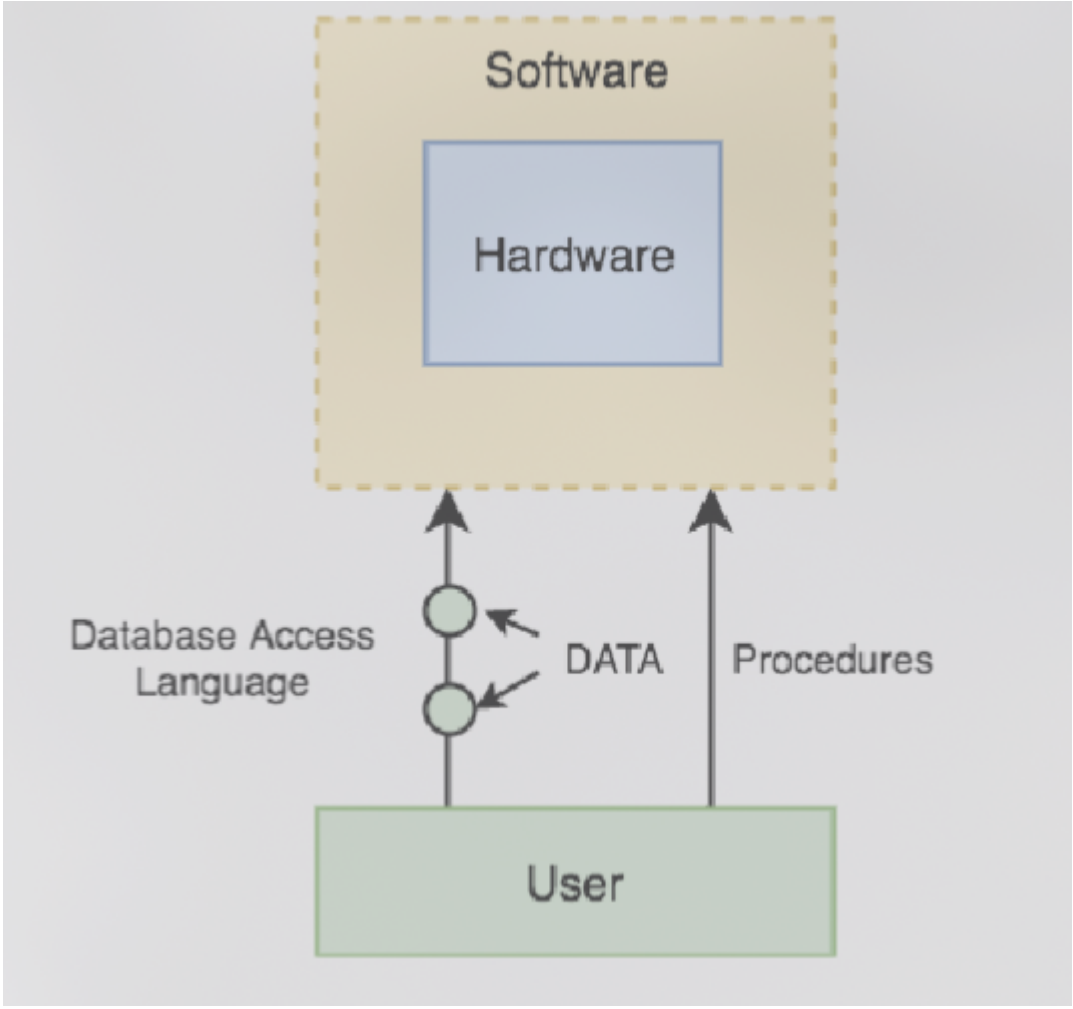
03 - Analyses the main components of a database system

Analyses the main components of a database system

Components of DBMS

- Hardware

- Software
- Data
- Procedures
- DAL (Database Access Language)



SQL

- SQL stands for Structured Query Language
- SQL lets you access and manipulate databases
- SQL became a standard of the American National Standards Institute (ANSI) in 1986
- SQL commands falls into 2 basic groups
  - **Data Definition Language (DDL)**
  - **Data Manipulation Language (DML)**

Data Definition Language (DDL)

- This is the SQL commands **used for defining and modifying** the data and its structure
- It is used to build and modify the structure of your tables

COMMAND OR OPTION	DESCRIPTION
CREATE SCHEMA AUTHORIZATION	Creates a database schema
CREATE TABLE	Creates a new table in the user's database schema
NOT NULL	Constraint that ensures that a column will not have null values
UNIQUE	Constraint that ensures that a column will not have duplicate values
PRIMARY KEY	Defines a primary key for a table
FOREIGN KEY	Defines a foreign key for a table
DEFAULT	Defines a default value for a column (when no value is given)
CHECK	Constraint used to validate data in a column
CREATE INDEX	Creates an index for a table
CREATE VIEW	Creates a dynamic subset of rows/columns from one or more tables
ALTER TABLE	Modifies a table's definition (adds, modifies, or deletes attributes or constraints)
CREATE TABLE AS	Creates a new table based on a query in the user's database schema
DROP TABLE	Permanently deletes a table (and thus its data)
DROP INDEX	Permanently deletes an index
DROP VIEW	Permanently deletes a view

Data Manipulation Language (DML)

- These are the commands used for **selecting, inserting, deleting and updating** data in a database.
- It is used to retrieve and manipulate data in a relational database
- DML **performs read-only queries** of data.

COMMAND OR OPTION	DESCRIPTION
INSERT	Inserts row(s) into a table
SELECT	Selects attributes from rows in one or more tables or views
WHERE	Restricts the selection of rows based on a conditional expression
GROUP BY	Groups the selected rows based on one or more attributes
HAVING	Restricts the selection of grouped rows based on a condition
ORDER BY	Orders the selected rows

Writing SQL queries

Comparison Operators for Conditional Restrictions

SYMBOL	MEANING
=	Equal to
<	Less than
<=	Less than or equal to
>	Greater than
>=	Greater than or equal to
<> or !=	Not equal to

Arithmetic Operators

ARITHMETIC OPERATOR	DESCRIPTION
+	Add
-	Subtract
*	Multiply
/	Divide

Precedence Rule

- Perform **parentheses/brackets**
- Perform **power**
- Perform **multiplications and divisions**
- Perform **additions and subtractions**

Special Operators

- BETWEEN - Used to check whether an attribute value is within a range

- **BETWEEN**
  - List all products whose prices are between $50 and $100, inclusive.

```
SELECT * FROM Product
WHERE P_Price BETWEEN 50.00 AND 100.00;
```

or

```
SELECT * FROM Product
WHERE P_Price >= 50.00 AND P_Price <= 100.00;
```

- IS NULL - Used to check whether an attribute value is null

- **IS NULL**
  - List all products that do not have a vendor assigned

```
SELECT * FROM Product
WHERE V_CODE IS NULL;
```

- LIKE - Used to check whether an attribute value matches a string pattern
  - `%` is a special character used to mean any characters (plural)
  - `_` is a special character used to mean any single character (singular)

- **LIKE** (to find patterns within string attributes)
  - List all vendors whose contact names begin with Smith.

```
SELECT V_Name, V_Contact, V_Phone FROM Vendor
WHERE V_Contact LIKE 'Smith%';
```

% means any characters

  - List all vendors whose contact names do not begin with Smith.

```
SELECT V_Name, V_Contact, V_Phone FROM Vendor
WHERE V_Contact NOT LIKE 'Smith%';
```

- **LIKE** (to find patterns within string attributes)
  - List all vendors whose contact names may be Johnson or Johnsen.

    SELECT V_Name, V_Contact, V_Phone **FROM** Vendor
    WHERE V_Contact **LIKE** 'Johns_n';

    _ means any one character

- IN - Used to check whether an attribute value matches any value within a value list

- **IN**
  - List those products whose V_Code is either 21344 or 24288.

    SELECT * **FROM** Product
    WHERE V_Code **IN** (21344, 24288);

  or If V_Code is defined as CHAR(5)

    SELECT * **FROM** Product
    WHERE V_Code **IN** ('21344', '24288');

- EXISTS - Used to check if a sub query returns any rows

- **EXISTS**
  - List all vendors only if there are products to order (i.e. P_QoH <= P_Min).
  - **P_QoH:** Product Quantity on Hand
    P_Min: Product Minimum Quantity

    SELECT * **FROM** Vendor
    WHERE EXISTS (SELECT * FROM Product
    WHERE P_QoH <= P_Min);

```
1. SELECT * FROM STAFF;
2. SELECT Sno, FName, LName, salary from STAFF;
```

```sql
SELECT DISTINCT Pno FROM PROPERTY_FOR_RENT;
SELECT Sno, FName, LName, (Salary * 30) AS monthly_salary FROM STAFF;
SELECT * FROM STAFF WHERE Salary > 10000;
SELECT Street FROM BRANCH WHERE City = 'London' OR City = 'Glasgow';
SELECT * FROM STAFF WHERE DOB >= '1975-01-01' AND DOB <= '1995-12-31';
SELECT * FROM STAFF WHERE Position = 'Manager' OR Position = 'Deputy';
SELECT * FROM STAFF WHERE Address LIKE 'Glasgow';
SELECT * FROM VIEWING WHERE Pno = 'PG4' AND Comment IS NULL;
```

SQL aggregate functions

FUNCTION	OUTPUT
COUNT	The number of rows containing non-null values
MIN	The minimum attribute value encountered in a given column
MAX	The maximum attribute value encountered in a given column
SUM	The sum of all values for a given column
AVG	The arithmetic mean (average) for a specified column

Examples

OrderId	CustomerId	Total	OrderDate
1	321	10	1/2/2014
2	455	40	3/2/2014
3	456	20	3/10/2014

- SUM

```
SELECT SUM(Total) FROM Orders WHERE OrderDate BETWEEN '3/1/2014' AND '3/31/2014'
```

- AVG -

```
SELECT AVG(Total) FROM Orders WHERE OrderDate BETWEEN '3/1/2014' AND '3/31/2014'
```

- COUNT

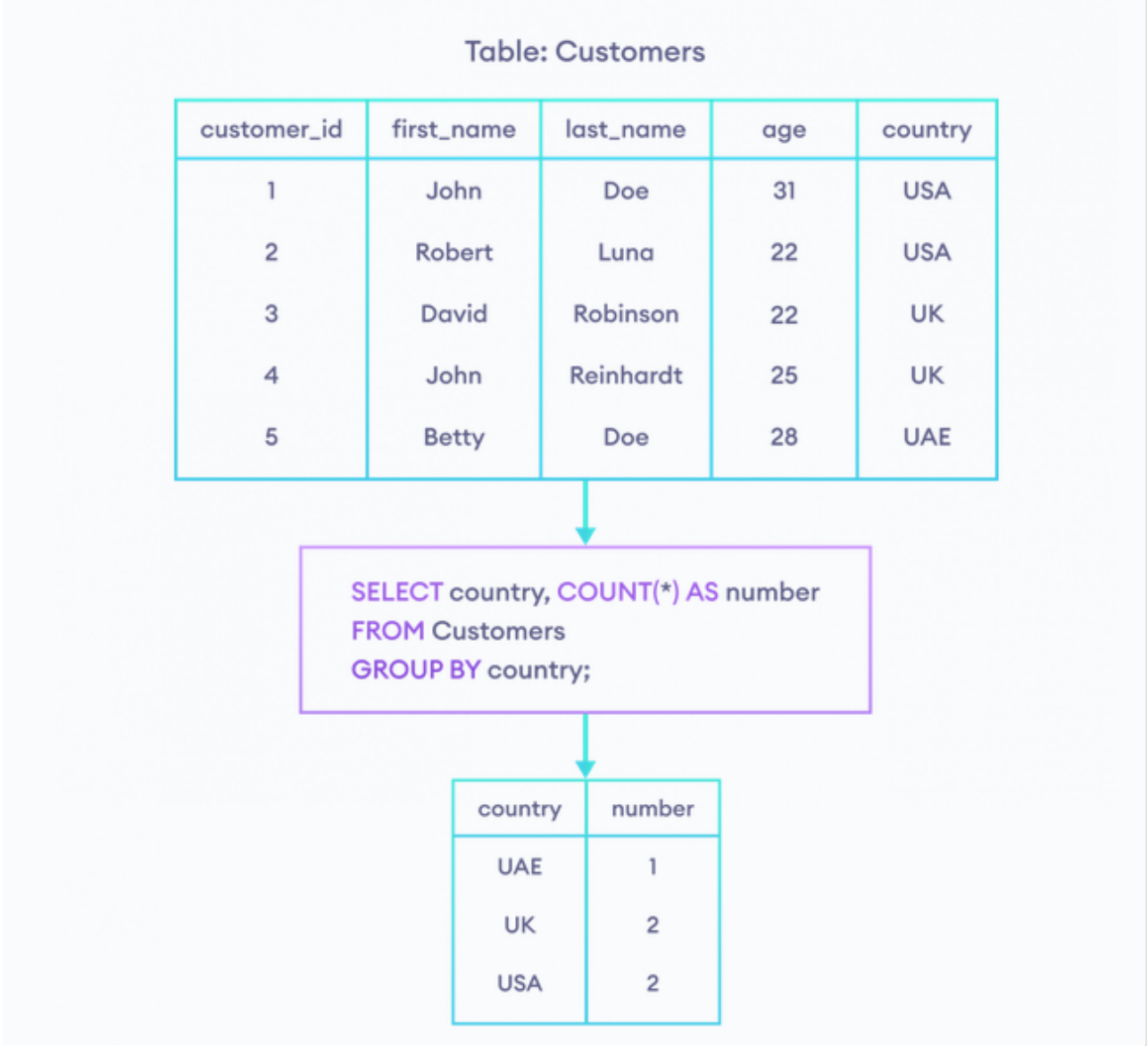
```
SELECT COUNT(*) FROM Orders
SELECT COUNT(OrderId) FROM Orders
```

- GROUP BY
  - Used to group and kind of sort out the results we get.
  - I.e say you have a list of customers from different countries. You can use GROUP BY to identify how many customers are from a certain country, like this

```
SELECT country, COUNT(customer_id) FROM customers GROUP BY country;
```

  - We can even use 2 columns if needed, like say you have the customer's country and the city, so you can further group it like this

```
SELECT country, COUNT(customer_id) FROM CUSTOMERS GROUP BY country, city;
```



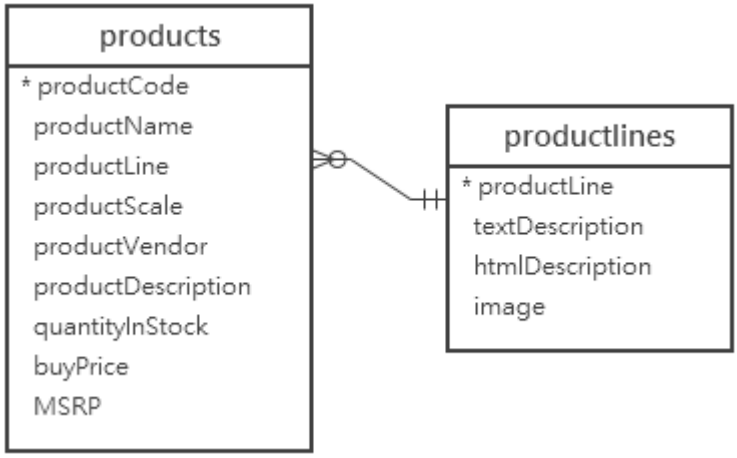
- JOIN - to perform queries relating more than 1 table
  - common attributes is an important distinction in relational database
  - This general idea is

```
get rows from Table A
and rows from Table B
where the PK(Table A) = FK(Table B)
```

  - refer - https://www.mysqltutorial.org/mysql-inner-join.aspx
  - this is the syntax

```
SELECT select_list FROM t1 INNER JOIN t2 ON join_condition1
```

  - Example



- the table `products` has the column `productLine` that references the column `productline` of the table `productlines` . The column `productLine` in the table `products` is called the foreign key column.
- Suppose you want to get:
  - The `productCode` and `productName` from the `products` table.
  - The `textDescription` of product lines from the `productlines` table.
- To do this, you need to select data from both tables by matching rows based on values in the `productline` column using the `INNER JOIN` clause as follows:

```
SELECT
    productCode,
    productName,
    textDescription
FROM
    products t1
INNER JOIN productlines t2
    ON t1.productline = t2.productline;
```

04 - Designs the conceptual schema of a database

Designs the conceptual schema of a database

- What is a Data model?

```
Simple representation (usually graphical) of more complex real world data
structures
```

Building a House	Building a DB
Blueprints	Data Models

- A data model organizes data for different users

- Data modeling is the first step in designing a DB

Building Blocks of Data Models

- Entities
  - Things about which data is to be collected
  - e.g., employee, students, courses
- Attributes
  - Characteristics of the entity
  - e.g., ID, name, DOB, manufacturer number
- Relationships (cardinalities)
  - 1:1 (one to one)
  - 1:M (one to many)
  - M:N (many to many)

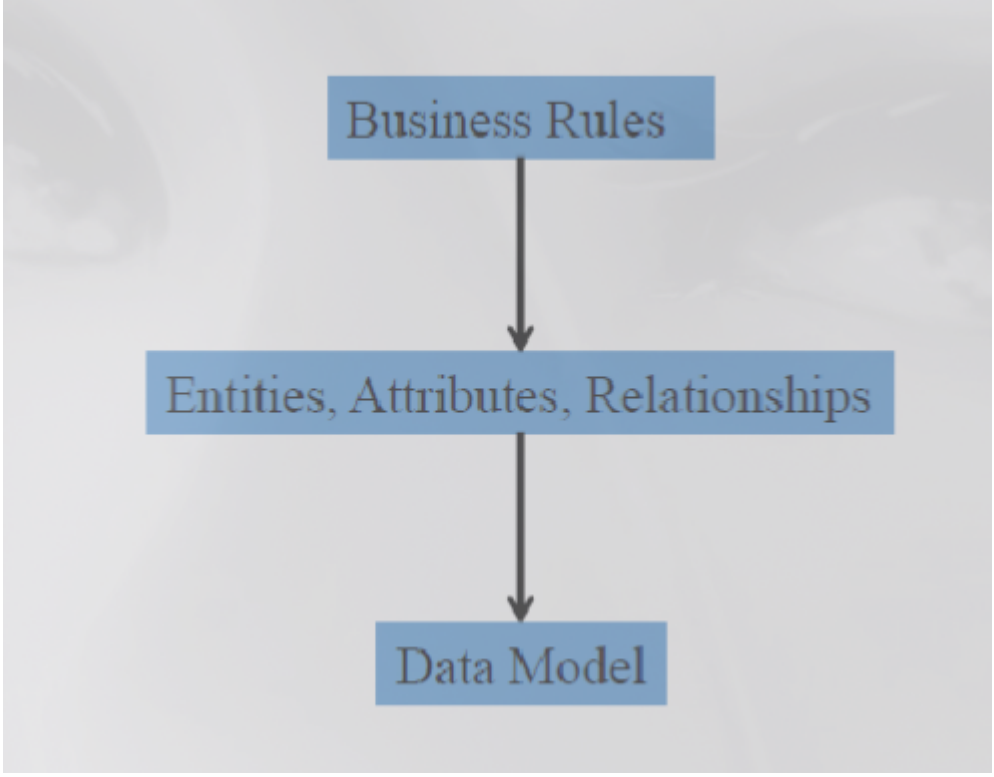
Business Rules

```
Business rules determine the building blocks
```

- Above-mentioned building blocks are determined from the business rules.
- Any organization that stores data has business rules
  - Policies, procedures, or principles
  - For example, each full-time student must select at least 3 units each semester

Importance of Business Rules

- Help verify the data model
  - Without knowing the business rules, we can't identify the entities, relationships and attributes
- Communication tool
  - helps to the relationship between the user and the designer to understand the nature of the database model.



Evolution of Data Models

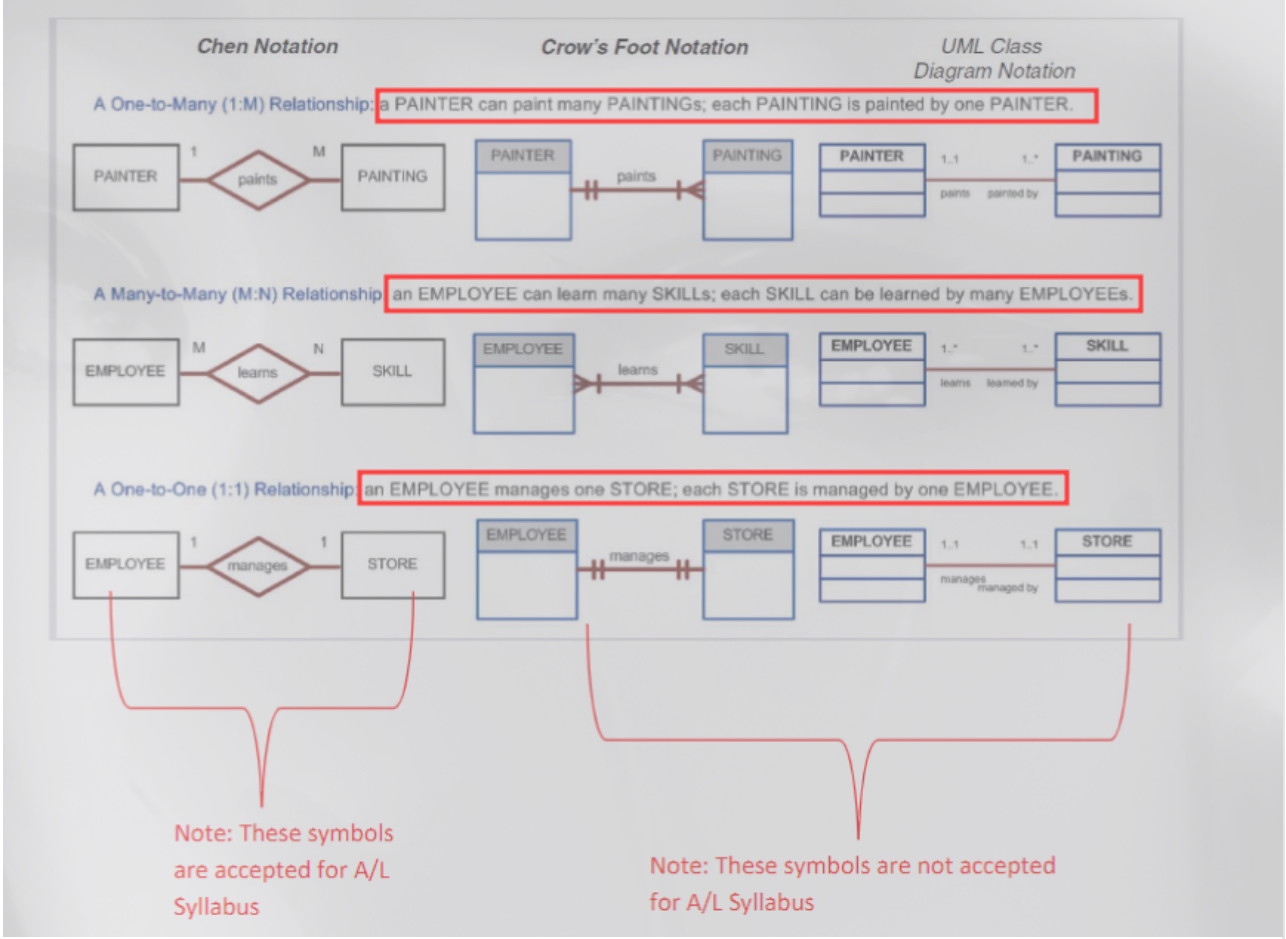
- Early age
  - Hierarchical and Network models
- Modern age
  - Relational model
  - Entity Relationships (ER) model
- New age
  - Object-Oriented, Object/Relational, XML, Big Data and NoSQL

Relational Model

- Developed by E.F. Codd
- Conceptually simple
- Powerful modelling capabilities
- But considered impractical in 1970due to lack of computing power
- Modern computers can handle relational databases with no problem

Merits	Limitations
Structural Independence	Substantial hardware
Conceptual Simplicity	System Overhead
Implementation Simplicity	Can facilitate poor design

Entity Relationship (ER) Model



ER Model

- A graphical data model
- Introduced by Chen in 1976
- Defines database structures in terms of the **entities** and their **relationships**

ER-Relational

- An ER model maps to relational tables
- Each **entity set** maps to a **relational table** `Entity set --> table`
- Each **entity instance** maps to a **table entry (row)** `Entity occurence --> row`

Merits	Limits
Exceptional conceptual simplicity	Representation of constraints and relationships
Effective communication tool	No data manipulation language
Complements the relational data model	

Four Tests for an Entity Type

You have to check if these 4 rules are fulfilled when you are choosing whether if it's an entity or not.

- **It should be of importance to the system being studied**
- **Should be able to have of a number of attributes that belong to the entity type.**

- **There should be several occurrences of the entity type**
- **Each occurrence should be uniquely identifiable** (should be possible to define a primary key)

Entity

> A uniquely identifiable object or concept that is of significance to the system and about which information is to be held.

- Entity types, Entity sets → Tables
- Entity instances, Entity Occurrences → Rows

Organization	Entity Types
School	Student, Subject, Teacher
Hospital	Patient, Ward, Disease
Grocery	Customer, Order

Entity type

- Two classes of entity types
  - Weak entity
  - an entity whose existence depends on other entities
  - Can be identified uniquely only by considering the primary key of another strong entity
  - Strong/Regular Entity
  - an entity that is not weak (independent)

Attributes

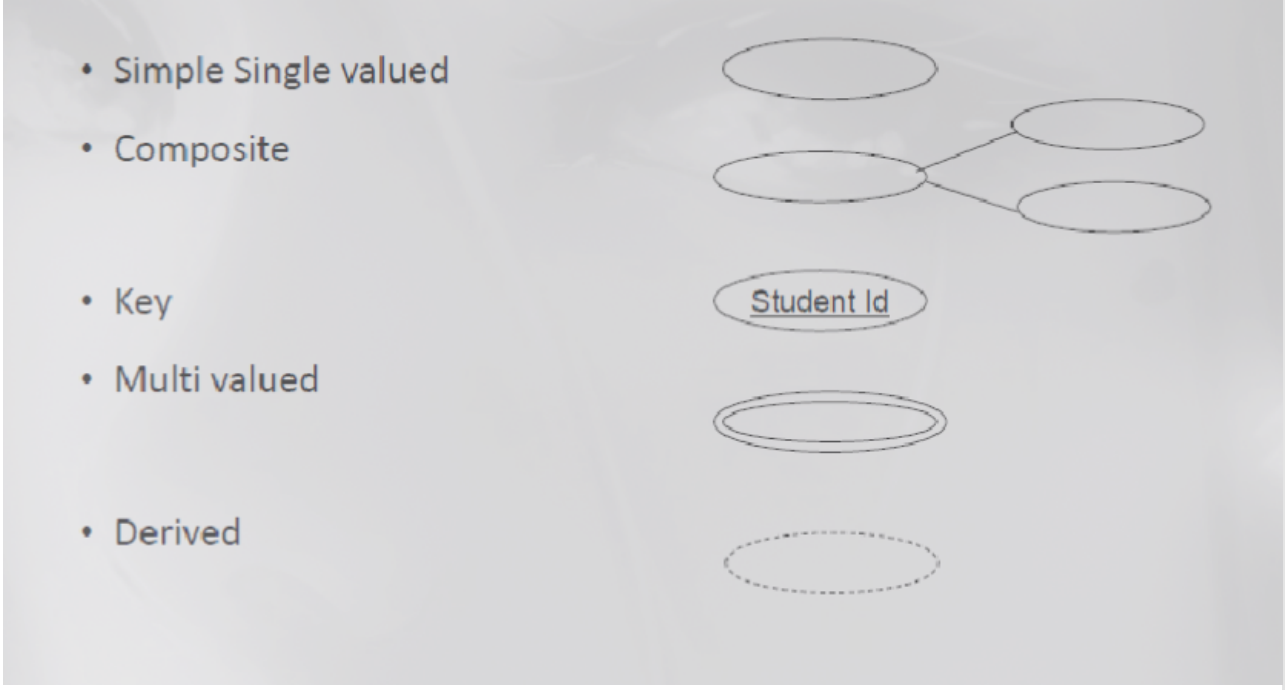
> A property for an entity

- e.g holds all data elements associated with entities

Entity	Attribute
Student	Student-No, Name, Address

Attribute types

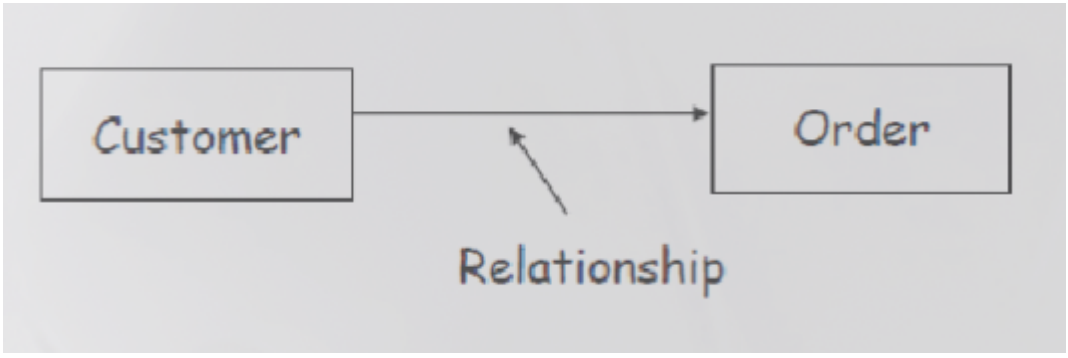
- Single / Composite
  - single
    - Can have a single value from the associated domain
    - like the student id
  - composite
    - Can have more than one value from the associated domain
    - like the name, `First name` and `last name`
- Key
- Base / Derived
  - base - raw value we input to the database, cannot be deduced (date of birth)
  - derived - value we derive from process (age --> `current date - date of birth`)



Relationships

> A meaningful association among entity types

- Relationships of the same type are grouped or typed into a relationship type
- Entities are linked together by relationships
- Exists if one or more attributes are **common** to two entities

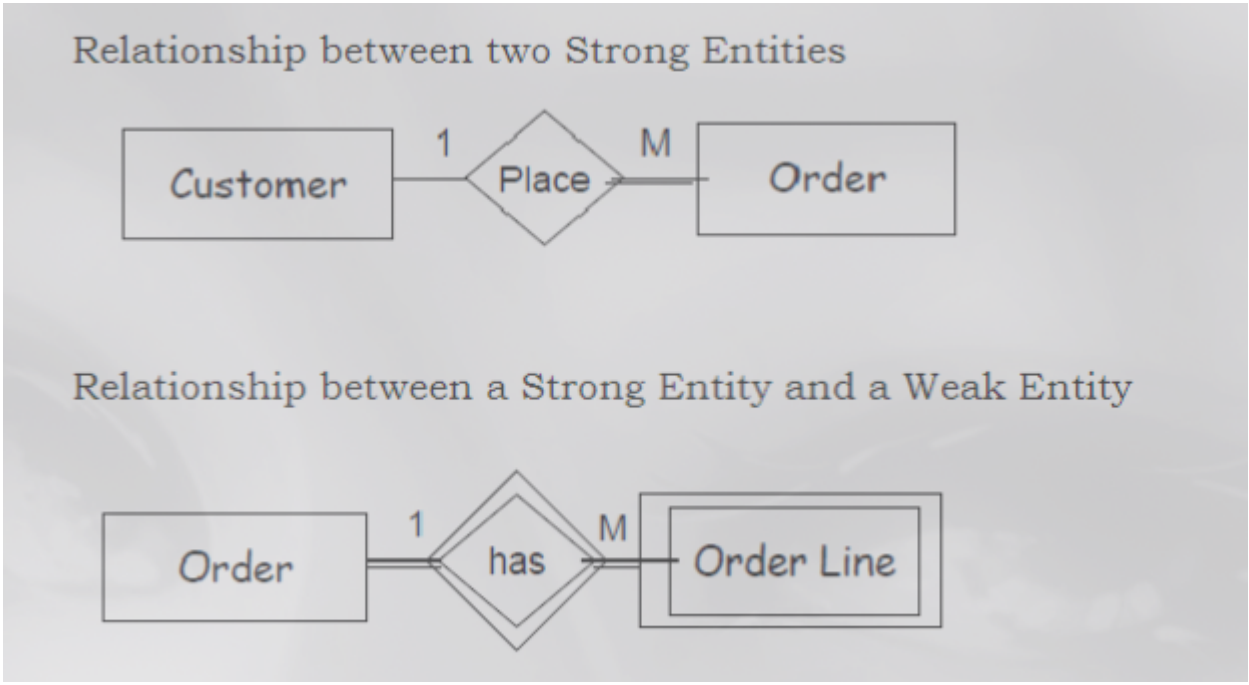


Degree

> The number of entities associated with the relationship

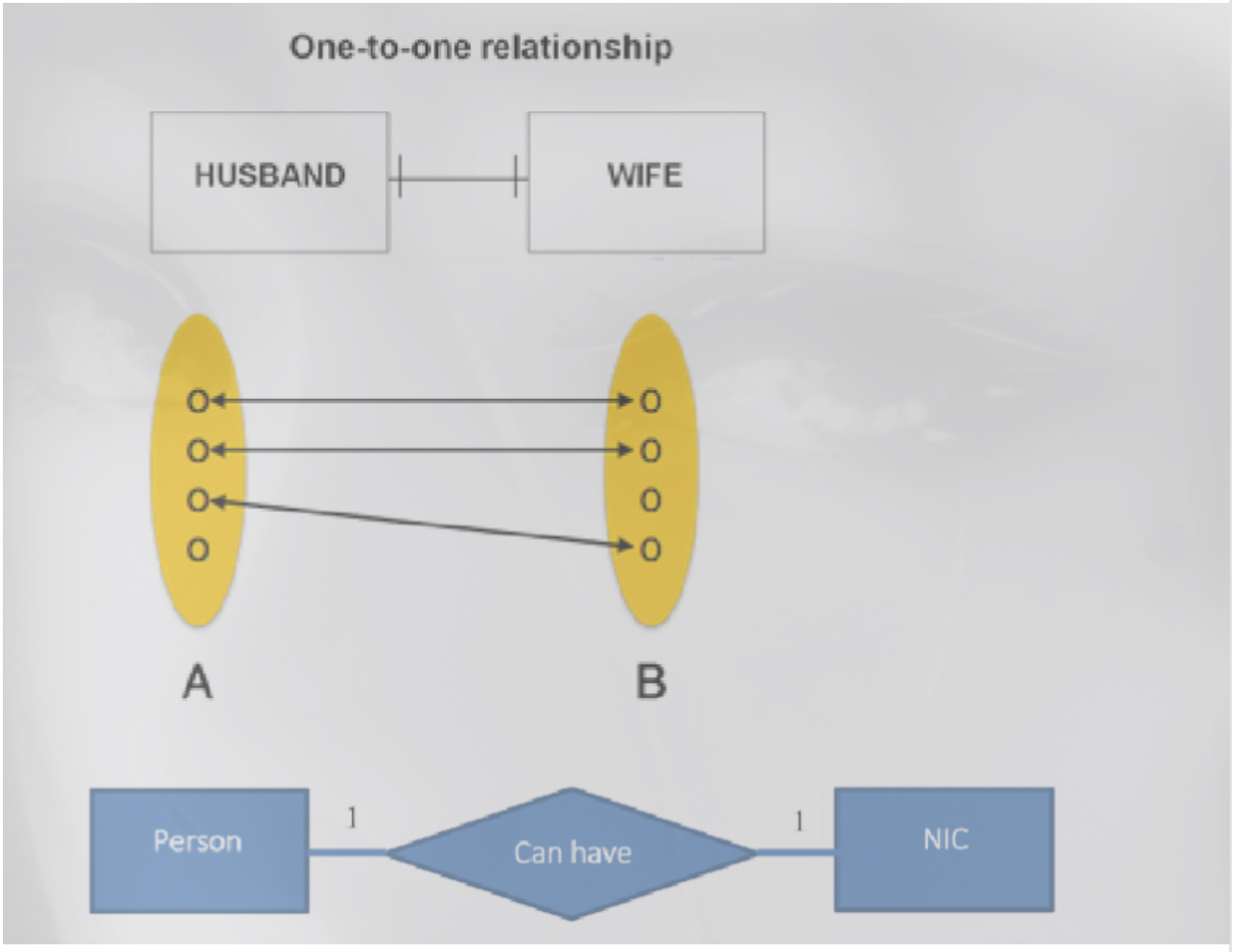
- Unary relationships
  - Only one entity is involved
    - Employee of a company is married to another employee of the same company
- Binary relationships
  - 2 entities are involved
  - the most common type in the real world
- Ternary relationship
  - when n number of entities are involved

ER Notation Relationships

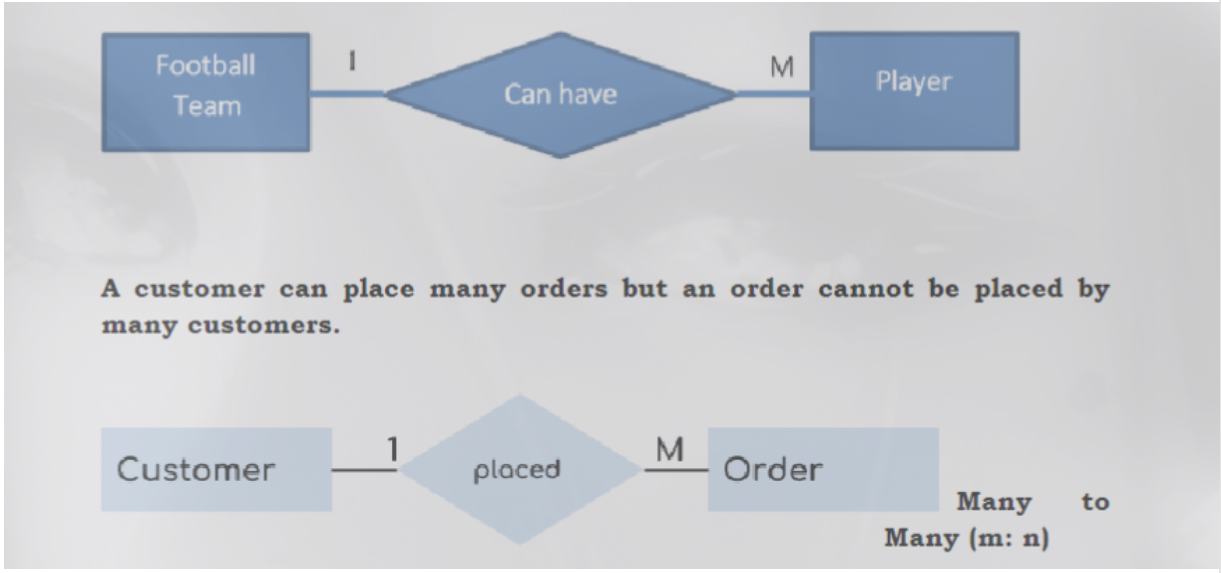


Structural Constraints - Cardinality

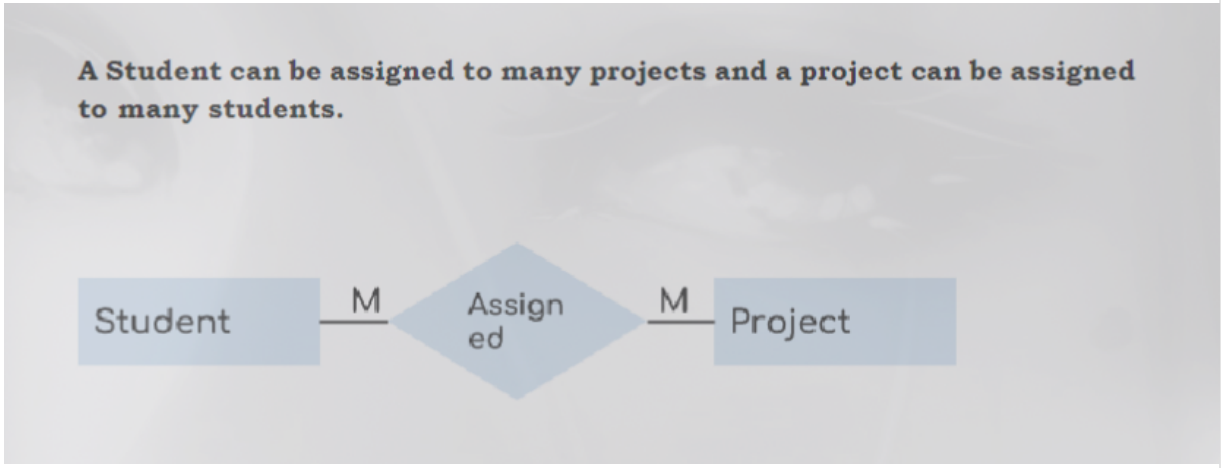
- A relationship may allow multiple occurrences of subject / object.
- Possible Cardinalities
  - One to One 1:1
  - One to Many 1: m
  - Many to Many m: n
- one to one



- one to many



- many to many



> Note - relational databases are unable to handle m: n relationships. Therefore, if they occur they must be resolved by two 1: n relationships

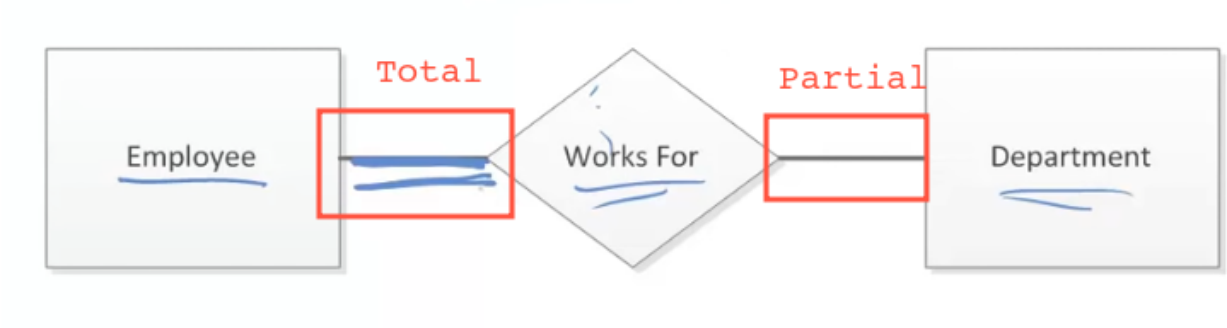
Structural Constraints - Participation/Optionality Constraints

```
Participation constraints determine whether the existence of an entity depends upon its being related to another entity through the relationship
```

There are 2 types of participation constraints

- **Total** - A mandatory relationship is where one occurrence of an entity type must be associated with an occurrence of the other entity type (mentioned by a double line) (all occurrences of table A has to be connected to a row of table B)
- **Partial** - A relationship is said to be optional between 2 entities if an occurrence of one may exist without associated occurrences of the other (mentioned by a single line)

REF: https://www.youtube.com/watch?v=W7SVMKIwOzs

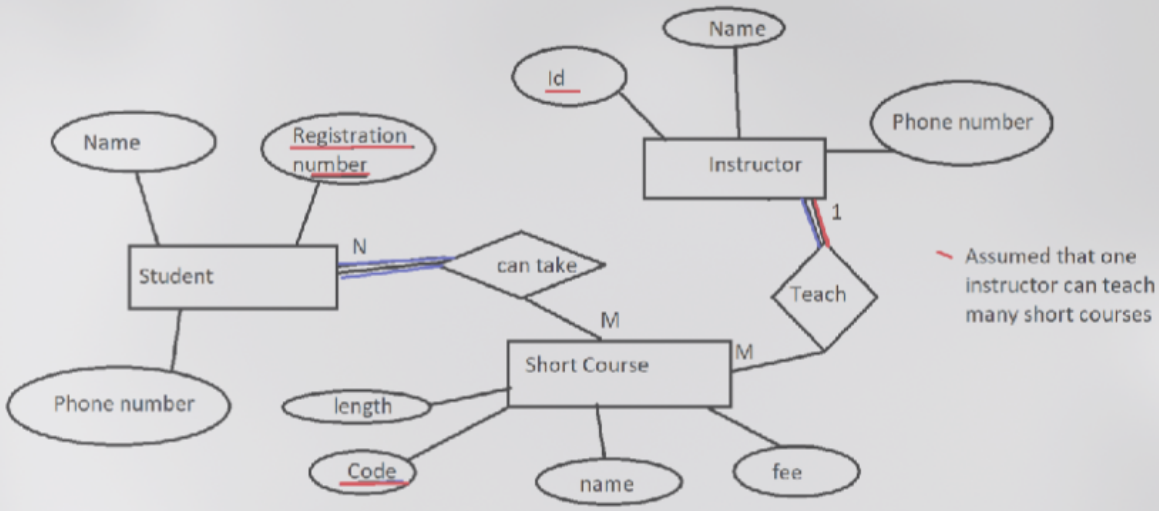


> According to this example, an employee must at least work for 1 Department. However, there can be departments where no employees are in work. (This could be the case for newly formed departments)

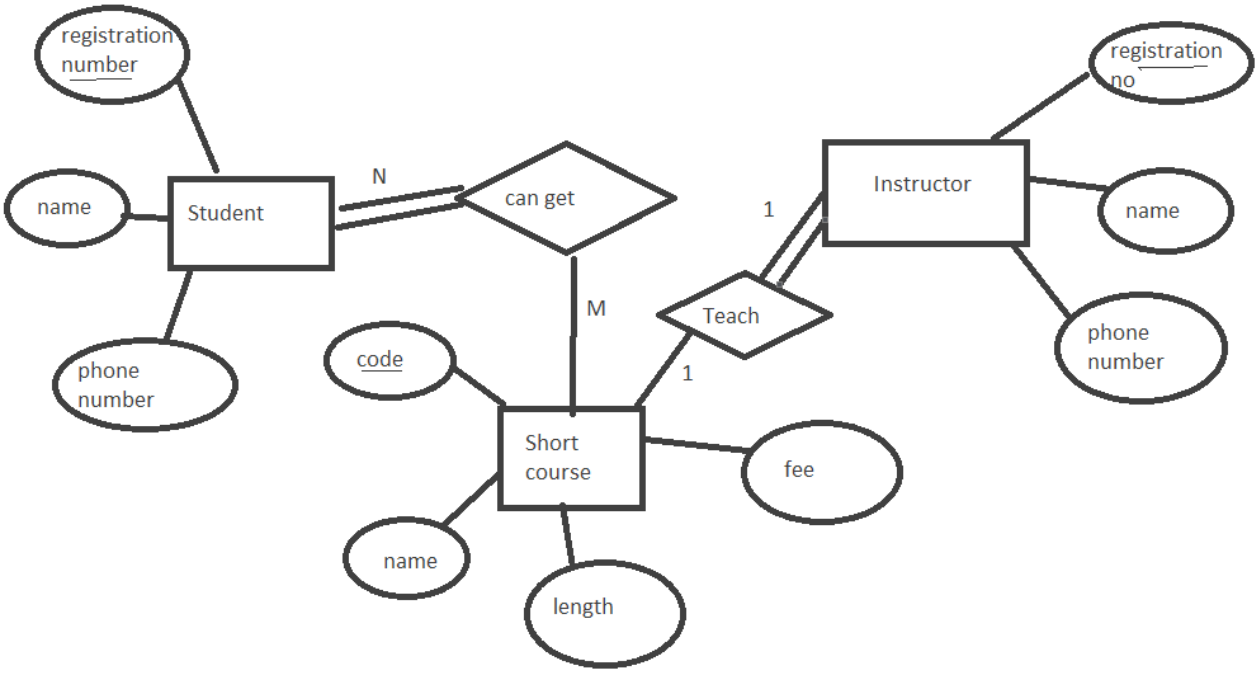
Example

"I am the director of the university's Short Course Programme Each of the short courses we teach has a code, a name, and a fee Beginners Systems Analysis and Web Design are two of our more popular courses.

Courses vary in length from three to ten days Reza and Jas are two of our best instructors We need each instructor's id, name and phone number The students can take several courses over time, and many do this We like to have each student's registration no, name and phone number"



- Entities should be inside rectangles (Students, Instructor)
- Attributes should be in ovals (name, code)
- Key attribute (primary key) should be underlined
- Relations should be marked according to the cardinality (1:1, 1:m, n:m)



Assumption is made as: One instructor should teach one course
Using this assumptoin, we get the relation b/w instructor and couses

Extended ER Diagram -EERD

EER is a high-level data model that incorporates the extensions to the original ER model. There are a couple of ways of doing this but for our syllabus, we only talk about **Specialization** and **Generalization**

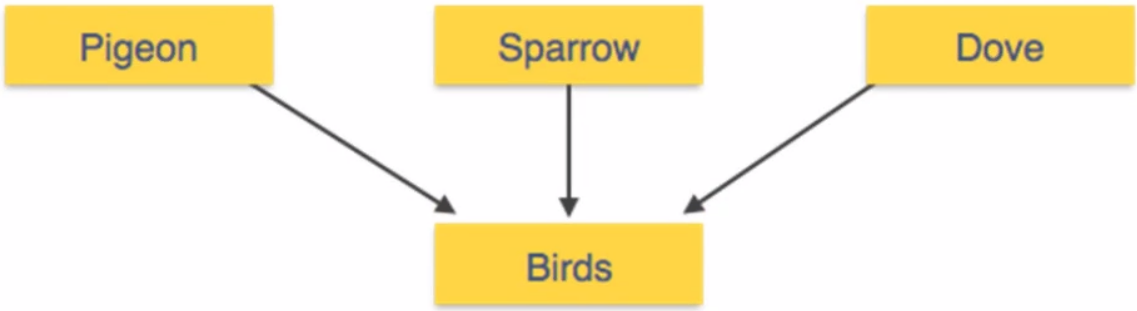
Specialization

```
Specialization is a process of identifying subsets of an entity that share some different characteristic
```

For example, we have an entity called vehicles. This vehicle entity could have values like car, bike which have special attributes.
I.e car could have a special characteristic called engine type and the bike could have a special characteristic called tyres. To represent these in an ER diagram, this technique is used.

Generalization

This is kinda the opposite of the specialization. Here we take 2 or more sub entities, like Pigeon, Sparrow and Dove which can be generalized into **birds**



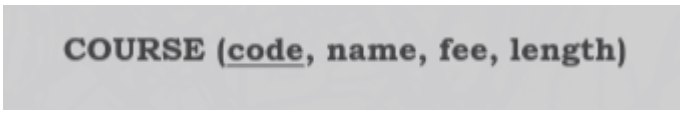
Designs the logical schema of a database

There are different kinds of database sachems

1. Conceptual schema - The plane and the structure we have in our head (ER diagrams)
2. Logical(Relational) schema - The tables we make using the structure
3. Physical schema - Final tables filled with data

Transform ER Diagram into Relational schema

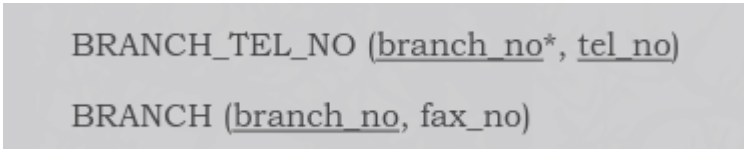
The given ER diagram should be written in the following form




```
ENTITY_NAME(PRIMARY_KEY (underlined), OTHER_ATRIBUTES)
```

- **Multivalued attributes**

I.e if we have 2 telephone numbers for the same person, we have to **go for another additional table** to represent them, and use the primary key of the initial table as the foreign key for the additional table.



The `BANCH` table is the main table and the `BRANCH_TEL_NO` is the additional table we added to add the telephone numbers. Here to present the foreign key, an `*` is used.

- **Composite attributes**
  These attributes are stored in the same table as 2 columns.
  For example the name -> (First name, Last name). These both are saved in the same table.

```
STAFF(Staff_ID,f_name,l_name)
```

> Only multi valued attributes are made into a seperate table.

- **Derived attributes.**

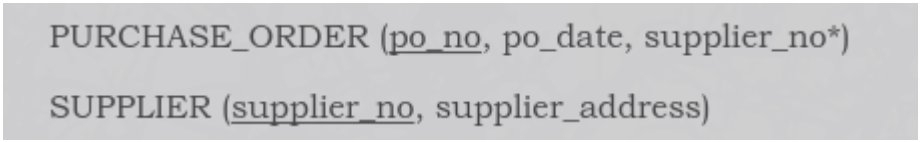
These values **aren't represented in the table**. When needed, they are derived at the time to retrieval. I.e if in the ER diagram we have name, **age** and date of birth as fields, when making the table we only include the name and date of birth as we can derive the age based on that.

```
STUDENT(name, date_of_birth)
```

Relationships

- **One to many relationship**

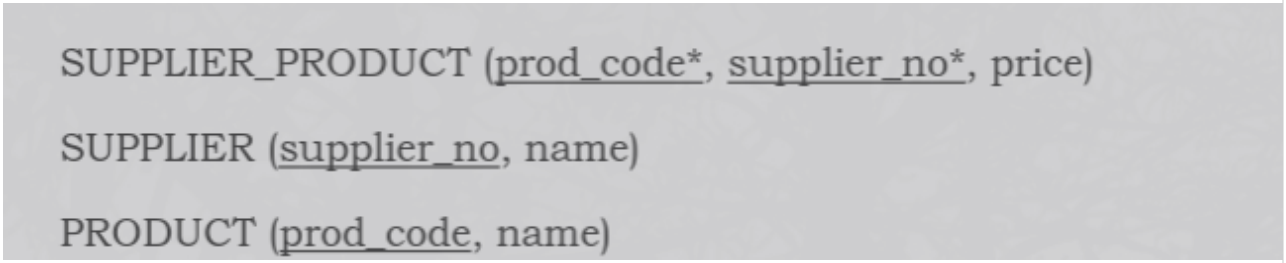
Here, the primary key of the 1 side is set as the foreign key of the many side.



`SUPPLIER` is the 1 side and the `PURCHASE_ORDER` is the many side. `supplier_no` which is the primary key from the 1 side is related to the foreign key of the many sides

- Many to many

Here we have to use 3 tables to depict this relation.

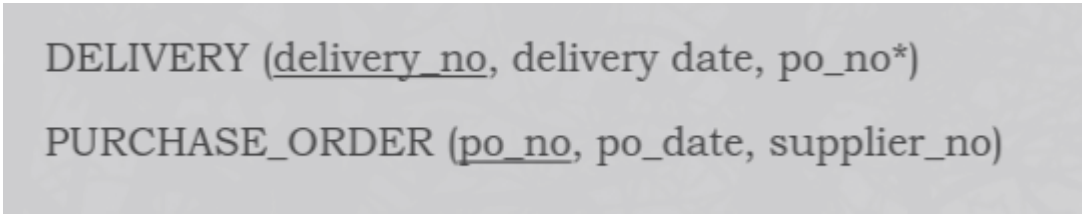


In the example shown , `prod_code` and `supplier_no` are foreign keys in the new SUPPLIER_PRODUCT relation. The primary key of the new relation must also include both these foreign keys, unless a surrogate/artificial key is produced

Here, the primary keys of the SUPPLIER table and the PRODUCT key are used as the **composite key** for the new SUPPLIER_PRODUCT table.

- **One to one**

Here the primary key of each table can be the foreign key of the other table according to your preference. No order here.



07 - Normalizes database schema to improve performance

Normalizes database schema to improve performance

```
Normalization is a process to evaluate and correct the table structure
```

> This is another database design tool except ER diagrams

> Here what happens is that we are given a document with a mixed up records, and here with normalization we just clean it up according to a couple of stages and insert those data into a database

Drawing an ER diagram is a **top-bottom** approach

- Here we start from 0 and build everything from there
  But normalization is a **bottom-top** approach
- Here we take a finalized document and then build a database off of that

The method we use from ER or normalization depends on the story line

- What is the main purpose of using this method (including ER diagram)

```
To minimize data redundency
```

Normalization Objectives

- To ensure tables have the following characteristics
  - Each table represents a single subject (One to student, one for teaching etc)
  - No data duplication
  - All the attributes in a table are uniquely identified by a primary key
  - To avoid anomalies (happens due to data redundancy) for data insertion, updating and deletion.

Normalization Stages

Normalization goes through number of stages

- First normal form (1NF)
- Seconds normal form (2NF)
- Third normal form (3NF)

In each of these stages, there are certain things that happen to the document that's not normalized yet.

Normal form	Actions
1NF	Format the tables
	Make sure there are no repeating groups
	Define a primary key
2NF	No partial dependencies
3NF	No transitive dependencies

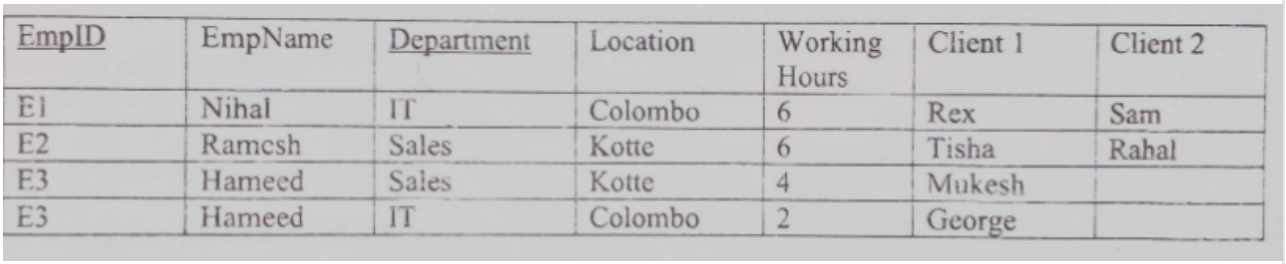
First Normal Form (1NF)

1. A table must not contain repeating groups
    - **Eliminate the repeating groups** by making sure each attribute contains an appropriate data value
2. Choose a primary key
    - A “key” should identify **one record** not a group
3. Identify all dependencies

> Physically, they way we identify the document is in 1NF is by seeing whether if it has any repeating groups. If not, it is in 1NF, else not.

Eliminating the repeating groups

Think of an example like this.



EmpID	EmpName	Department	Location	Working Hours	Client 1	Client 2
E1	Nihal	IT	Colombo	6	Rex	Sam
E2	Ramesh	Sales	Kotte	6	Tisha	Rahal
E3	Hameed	Sales	Kotte	4	Mukesh	
E3	Hameed	IT	Colombo	2	George	

This table is in the 0NF form. Why? Even though the primary keys are defined here (as `EmpID` and `Department`) there are repeating groups as `Client1` and `Client2`. So we need to remove those first. For that we need to create `client` table

CLIENT(ClinetID, ClinetName,EmpID*)
EMP_DEP(EmpID,EmpName,Department,Location,Working_Hours)

> So like this now we have 2 tables, which are both in 1NF form, as there are **no repeating groups** and both tables h**ave a primary key defined**.

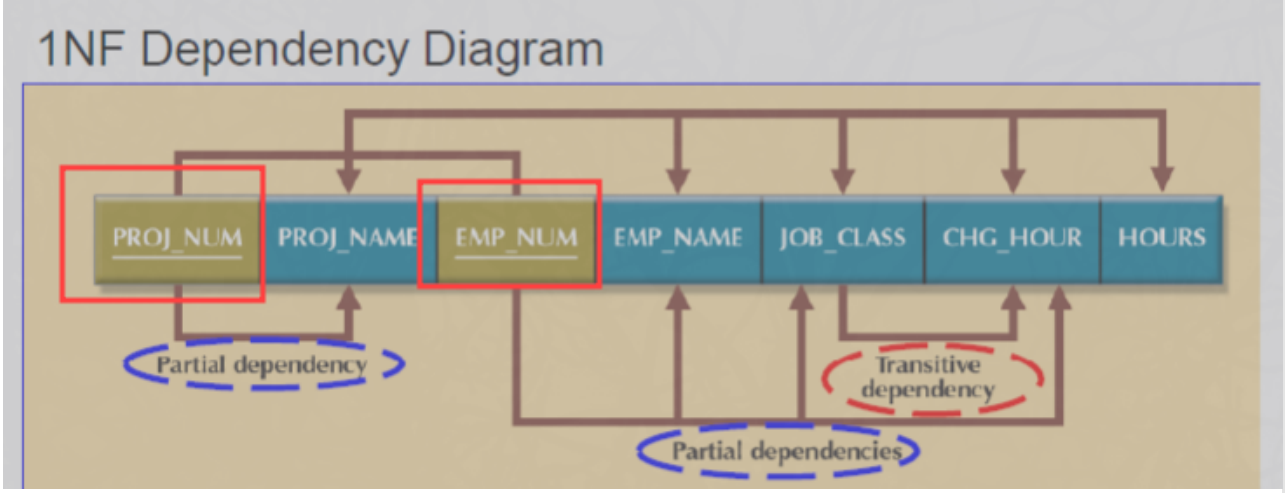
Functional Dependency

There are 2 main types of dependencies

1. Partial dependency
2. Transitive dependency

Partial dependency

- This happens only when there is a composite key is present
- In a composite key, there could be 2 or more key attributes. Partial dependency is **when other non-key attributes depend on any of those key attributes**.

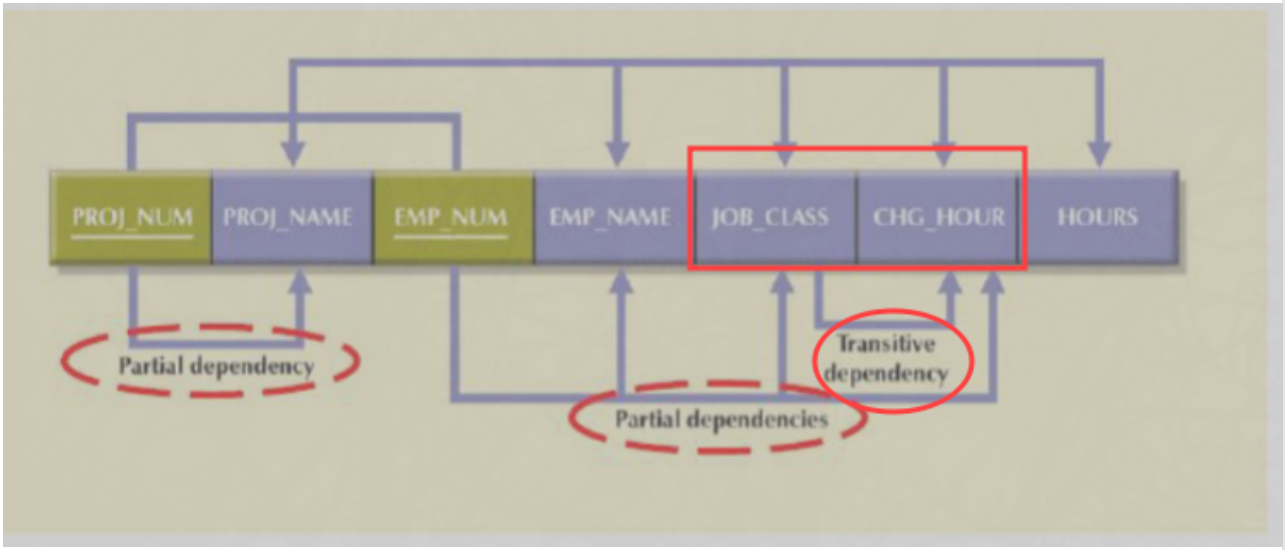


Here, in this diagram, `PROJ_NUM` and `EMP_NUM` are the 2 key attributes used to define the composite key. When take individually,

- `PROJ_NAME` is dependent on `PROJ_NUM`
- `EMP_NAME`, `JOB_CLASS`, `CHG_HOUR` are dependent on `EMP_NUM`
  This is partial dependency.

Transitive dependency

- This happens when **2 non-key attributes are related to each other.**
- Gets identified in the 2NF stage (even though, we can identify this in the 1NF as well)



Here the `JOB_CLASS` and the `CHG_HOUR` attributes are related to the `EMP_NUM` key attribute. Those 2 are also related to each other, as the hourly charging rate depends on the job class. This is when transitive dependency happens.

If transitive dependency exists, we need to go from 2NF to 3NF stage

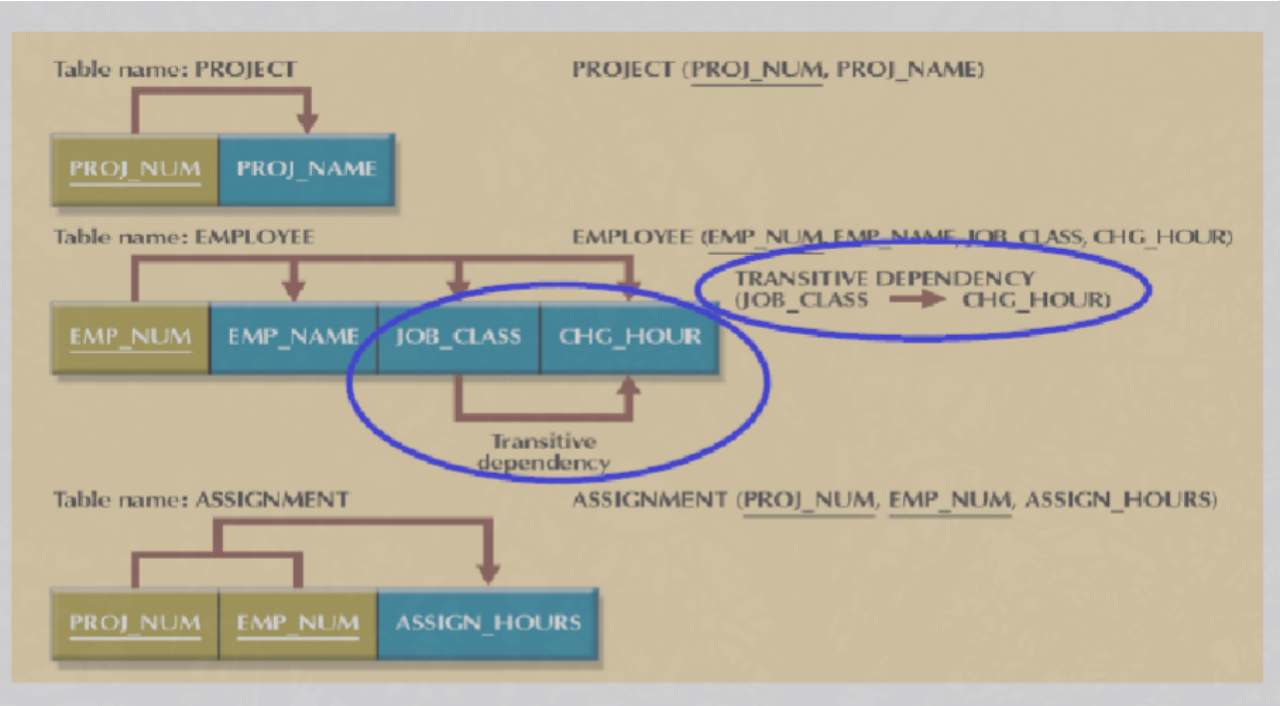
Second Normal Form (2NF)

> Goal of 2NF is to remove partial dependencies

- What is the problem of 1NF?

```
There should be NO partial dependencies in a table
```

- In the 2NF, **partial dependencies are removed and separate tables are made for the key attributes** of the partial dependencies.
- And then, transitive dependencies are identified.

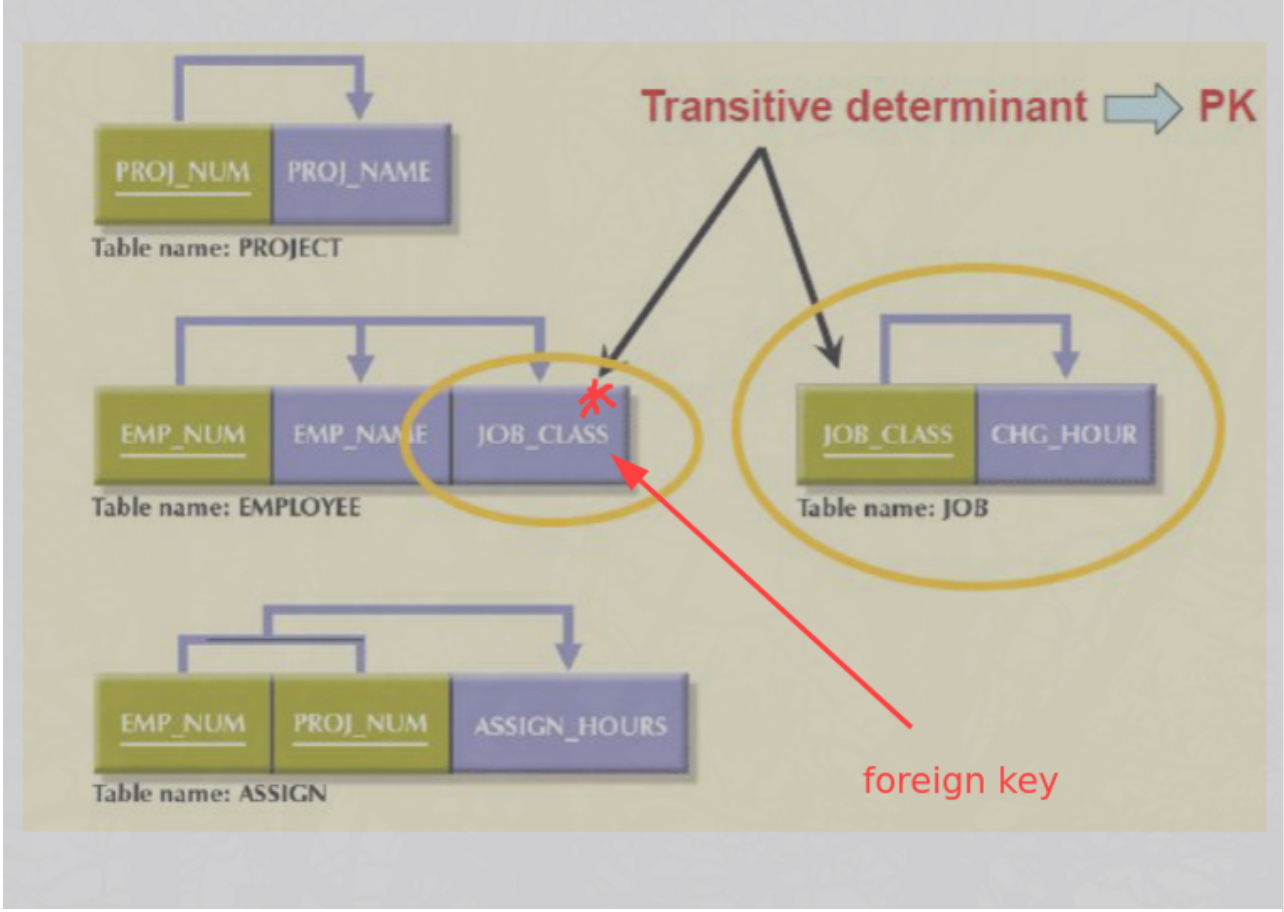


> Note that here to relate the `EMPLOYEE` and `PROJECT` tables, **an additional table called `ASSIGNMENT` has been added.**

Here, once the tables are made according to the 2NF, we can see the transitive dependency in the `EMPLOYEE` table.

Third Normal Form (3NF)

Now that have identified the transitive dependencies, we need to make an **additional table for the transitive dependency** to remove them so that it becomes a 3NF



> Since `JOB_CLASS` **depends on the employee**, and we need to **make a connection between the additional table we make and the** `EMPLOYEE` **table**, we are taking the `JOB_CLASS` as the primary key of the new table and make it the **foreign key** of the `EMPLOYEE` table

After all is done, the initial document we had should include the table which connects the 2 main tables we made according to the partial dependencies. Plus

And also we should have a separate table for the transitive dependency

> So in summary, for every partial, transitive dependency we find **represents a relation** between table. Which means for every dependency, separate tables must be made