DATABASE DESIGN

31.03.2014
(Muscat, Oman)

OUTLINE

- Database and Database Management System Definition
- Database Design Steps
- Entity-Relationhip (ER) Model
- Conceptional Data Design
- Logical Database Design
- Physical Database Design
- Normalization
- Denormalization
- DB Design Examples and Key Points

What is Database ?

A **database** is an organized collection of data

“It is hot today” => Not a data
“It is 5 C degrees” => Data

One can insert, update or delete the data in the database directly or via a web program, etc.

Today, databases are totally in our lifes: internet shopping, banking, administrative registers etc.

Database Management System (**DBMS**) is a software designed
To assist, managing, maintaning data.

Main Functions of the DBMS:

Make new databases
Define the concept of the database
Store data (Different types of data available)
Protect data
Query data
Encrypt data
Controlling access rights
Synchronize accesses
Organization of physical data structure

An **alternative** to DBMS may be using text files that are less complex

Advantages of DBMS

- ✓Can deploy huge size of data
- ✓Controlling redundancy
- ✓Data independence from applications
- ✓Reduced application development time
- ✓Efficient data access, techniques for efficient query
- ✓Data integrity with using referantial integrity techniques (data consistency)
- ✓Security, autherization for multiple users
- ✓Easy data administration
- ✓Backup and recovery mechanism

Disadvantages of DBMS

Size – very large

Complex

High cost

Increased hardware requirements

DATABASE DESIGN STEPS

✓ **Think before doing it !**

- Requirement Analysis
- Conceptional Design :
  - Relational Model (E-R Modelling)
  - Hierarchical Model
  - Network Model
  - Object Oriented Model
- Logical Design
- Physical Design

Requirement Analysis

During this phase, the below questions must be answered:

What will the system serve for?

Which requirements will this database meet?

Which data will this database store?

What will the tables of this database be?

Upon completing the answers for these questions on paper,

passing to the conceptional design can be advantegeous for your work.

ENTITY-RELATIONSHIP MODEL

ER Model is the concept of the Relational Databases.

ER Model describes data model using objects (Entity) and their relationships

Entity :

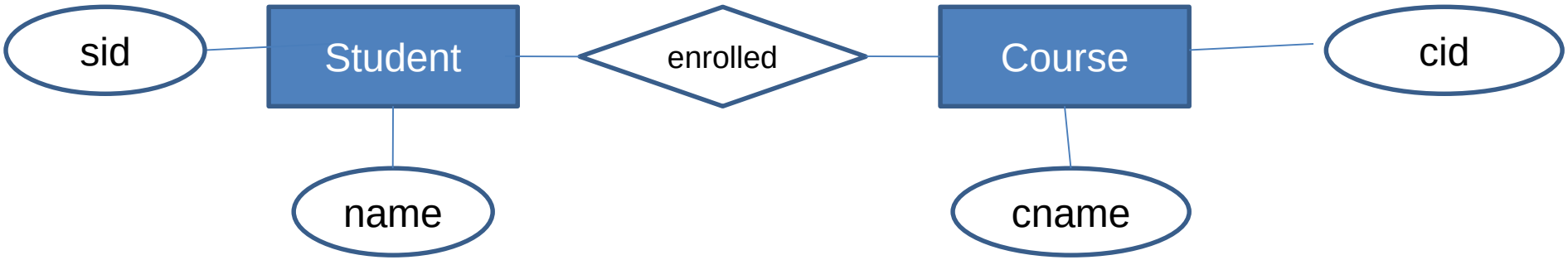
is an object that exists and can be distinguished from other objects
has attributes

Relationship:

Relate two or more entities
Relationship may have attributes

ER Model


sid
Student
name


sid
Student
enrolled
Course
cid
name
cname

Attributes

An entity is represented by a set of attributes, that is descriptive properties possessed by all members of an entity set.

Example:

instructor = (*ID, name, street, city, salary*)

course= (*course_id, title, credits*)

Domain – the set of permitted values for each attribute

Attributes Types

- **Simple** and **composite, component** attributes.

- **Single-valued** and **multivalued** attributes

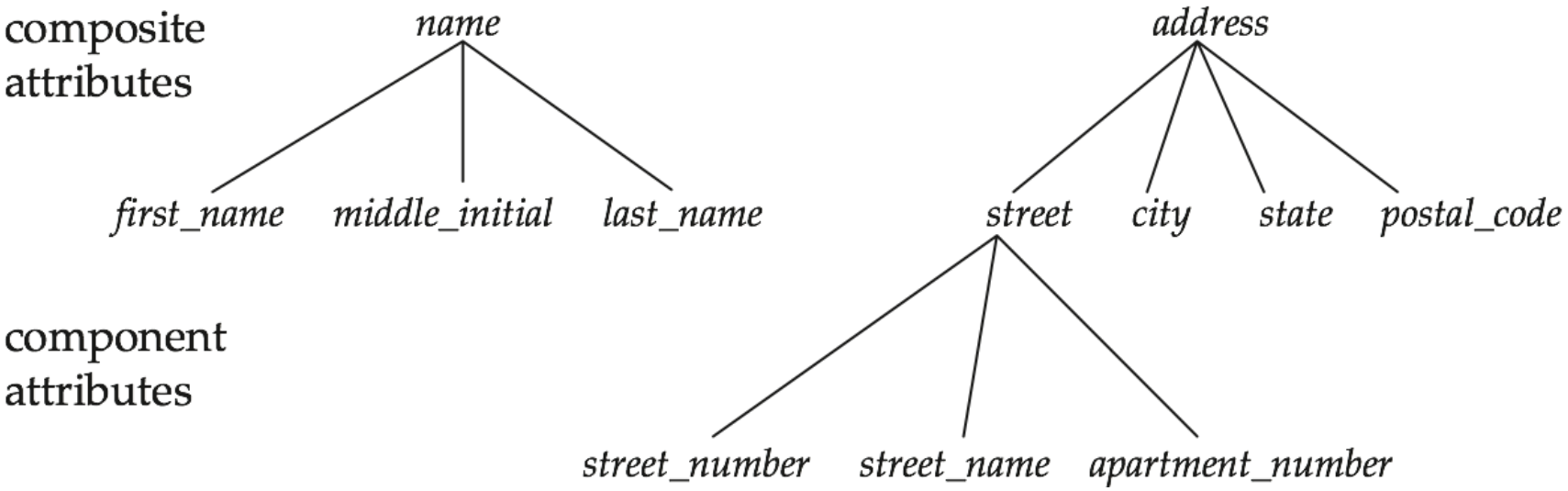
Example: multivalued attribute: *phone_numbers*

- **Derived** attributes

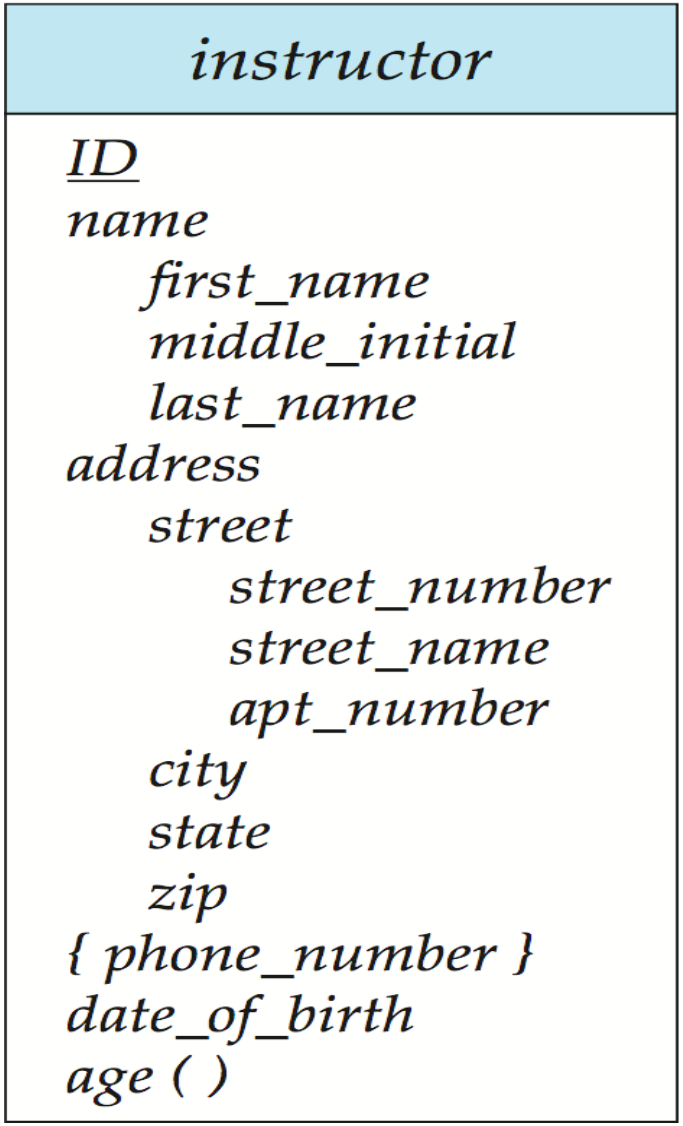
Can be computed from other attributes

Example: age, given date_of_birth

Composite Attributes



Entity With Composite, Multivalued, and Derived Attributes



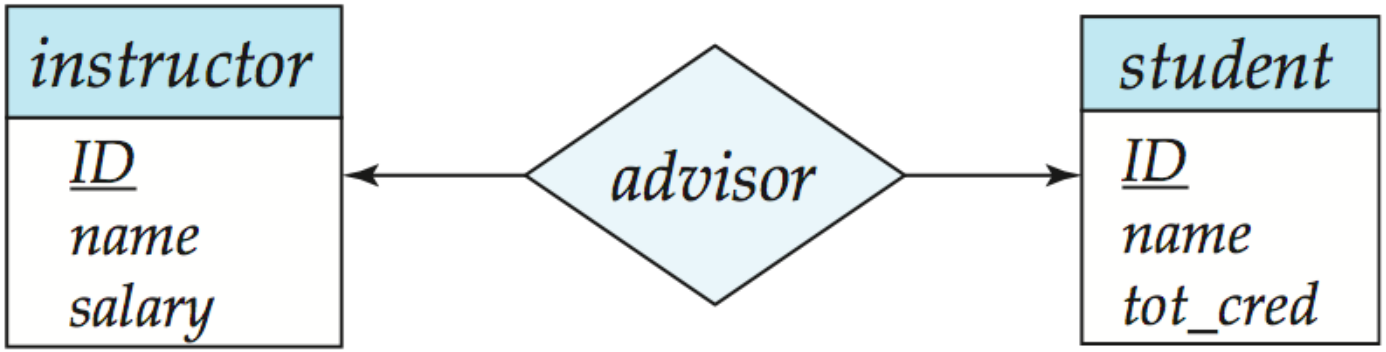
Mapping Cardinality Constraints

- **Relationship** can be one of the following types:

- One to one
- One to many
- Many to one
- Many to many

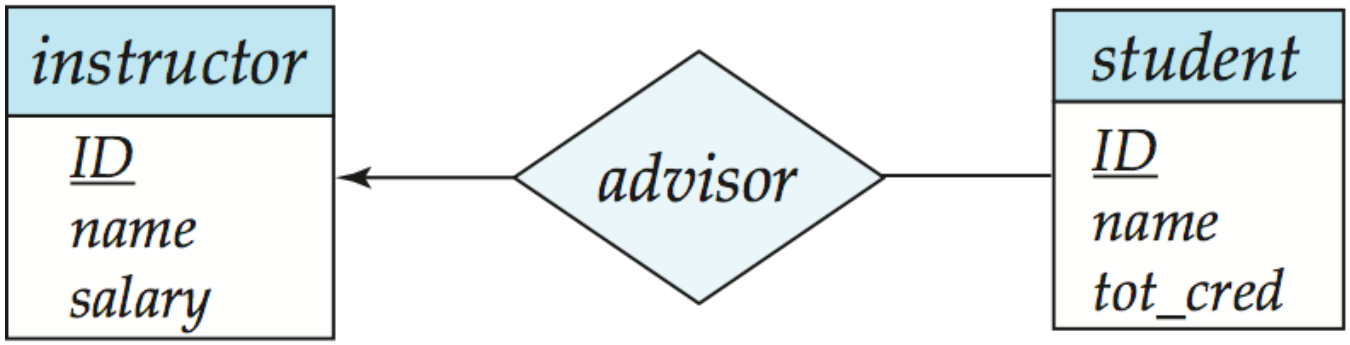
One-to-One Relationship

- one-to-one relationship between an *instructor* and a *student*
- an instructor is associated with at most one student via *advisor*
- and a student is associated with at most one instructor via *advisor*



One-to-Many Relationship

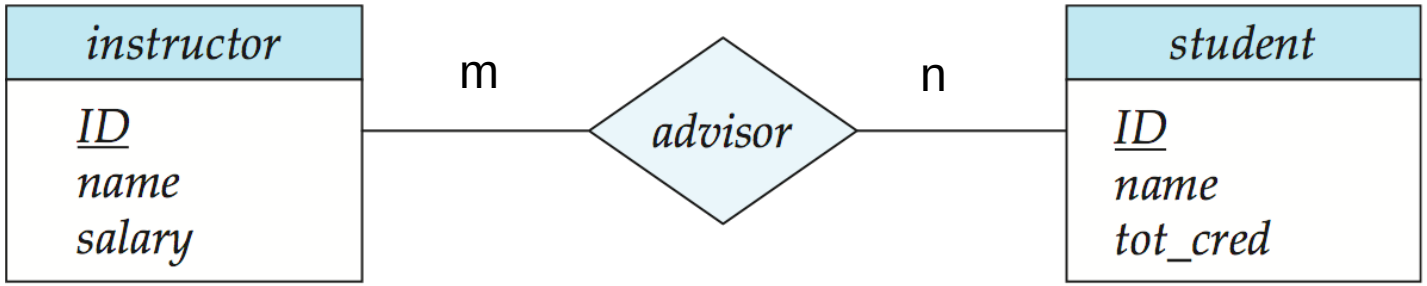
- one-to-many relationship between an *instructor* and a *student*
- an instructor is associated with several (including 0) students via *advisor*
- a student is associated with at most one instructor via advisor



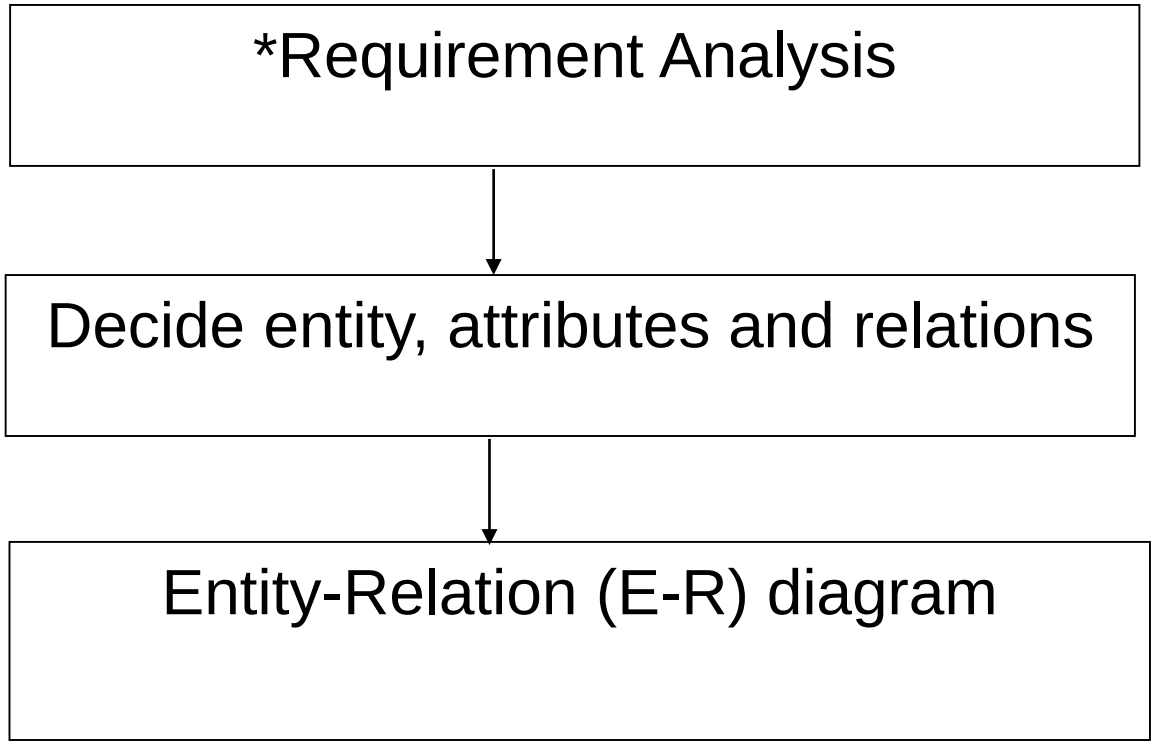
Many-to-Many Relationship

•An instructor is associated with several (possibly 0) students

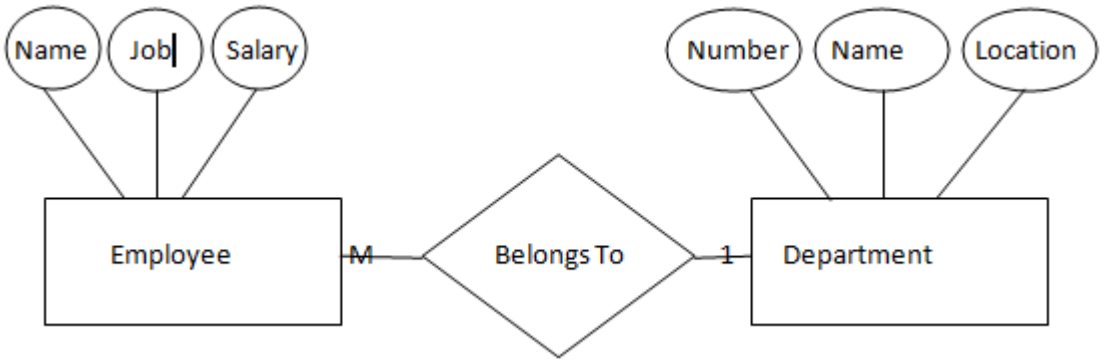
•A student is associated with several (possibly 0) instructors via advisor relationship



Conceptional Data Design with E-R Model



ER Diagram Example :



Logical Database Design

- Conceptional Design is used
- ER Model is converted to Relational Database Model.
- Entity at E-R or Classes in UML --> Table
- Many to many relations --> Table
- Attributes --> Columns
- Primary keys and foreign keys are defined

Physical Database Design

•It is **Physical** implementation of the Logical Model.

•Tables, columns, primary key constraints, foreign key constraints, check constraints, unique constraints, comments are created.

•**Normalization Rules** are checked , redundancy is minimized

•Disk capacity, partition strategy, security strategy are considered.

•Performance tuning is done.

Most Common Relational Databases

-MySQL
-PostgreSQL

-Access
-Oracle
-IBM Db2
-Interbase
-Microsoft SQL Server

Normalization

- process of organizing a database into tables correctly

- Finally unproblematic tables are designed
that providing Consistency, minimize redundancy

- We decide which atributes are used in a table

Normalization Forms

- (1NF) First Normal Form
- (2NF) Second Normal Form
- (3NF) Third Normal Form
- (BCNF) Boyce Codd Normal Form
- (4NF) Fourth Normal Form
- (5NF) Fifth Normal Form

First Normal Form

- Domain is **atomic** in First Normal Form
  if its elements are considered to be indivisible units

- UNF – Unnormalized Form Example:

staffNo	job	dept	dname	city	contact number
S01	Salesman	10	sales	London	12345,767642,4982423
S02	Manager	20	accounts	Barking	351632165
S03	Clerk	20	accounts	Barking	
S04	Clerk	30	operations	Barking	383131267

First Normal Form Example

Sales Table:

id	Name	Address	City	Product	Quantity	Price
1	Ahmet Seker	Address1	Ankara	CD-R	50	100
2	Ahmet Seker	Address1	Ankara	Mouse	2	2
3	Ahmet Seker	Address1	Ankara	CD-R	50	100
4	Ramazan Kaya	Address2	İstanbul	DVD-R	10	50
5	Gokhan Imam	Address3	Adana	CD-R	0	0

- Contains repeating data
- There are anomalies while inserting, updating and deleting
- **Let's think about avoiding repeating data ..**

Second Normal Form Example

- Repeating data is prevented

Customer table:

id	Name	Address	City
1	Ahmet Seker	Address1	Ankara
2	Ramazan Kaya	Address2	İstanbul
3	Gokhan Imam	Address3	Adana

Sales table:

Customer id	Product	Quantity	Price
1	CD-R	50	100
1	Mouse	2	2
1	CD-R	50	100
2	DVD-R	10	50
3	CD-R	0	0

Let's think about : What is the anomalies at these tables ? How to avoid?

Anomalies at the Second Normal Form

- If adding new city is wanted, a customer should be added
- When deleting a customer, the city will also be deleted
- So?

Third Normal Form

- Tables are divided into new tables
though there is no functionally dependencies

- In the example City table will be added

City:

City id	Name
c1	Ankara
c2	İstanbul
c3	Adana

Customer:

id	Name	City id
1	Ahmet Seker	C1
2	Ramazan Kaya	C2
3	Gokhan Imam	C3

Normalization

Advantages :

- ✓ More efficiently
- ✓ More accurate data
- ✓ Less hard drive
- ✓ Fewer data integrity problems

Disadvantages :

- ✓More slower
- ✓More complex queries
- ✓More work is needed
- ✓Unless normalizing, still do its business !

Denormalization for Performance

- May want to use non-normalized schema for performance
- More tables require more joining operations while querying

What is Denormalization ?

- It is a **strategy** that used to increase the performance of a database infrastructure
- involves adding redundant data
- İnvolves combining data from various tables into a single table.

Summary of Physical Database Design

Divide your information into tables regarding main subjects or entities.

Decide which columns will take place in each table
(eg. Surname and StartDate for EMPLOYEES table)

Decide the primary keys for all tables.
A primary key is used to define a record specifically
(eg. Province_code in PROVINCES tables)

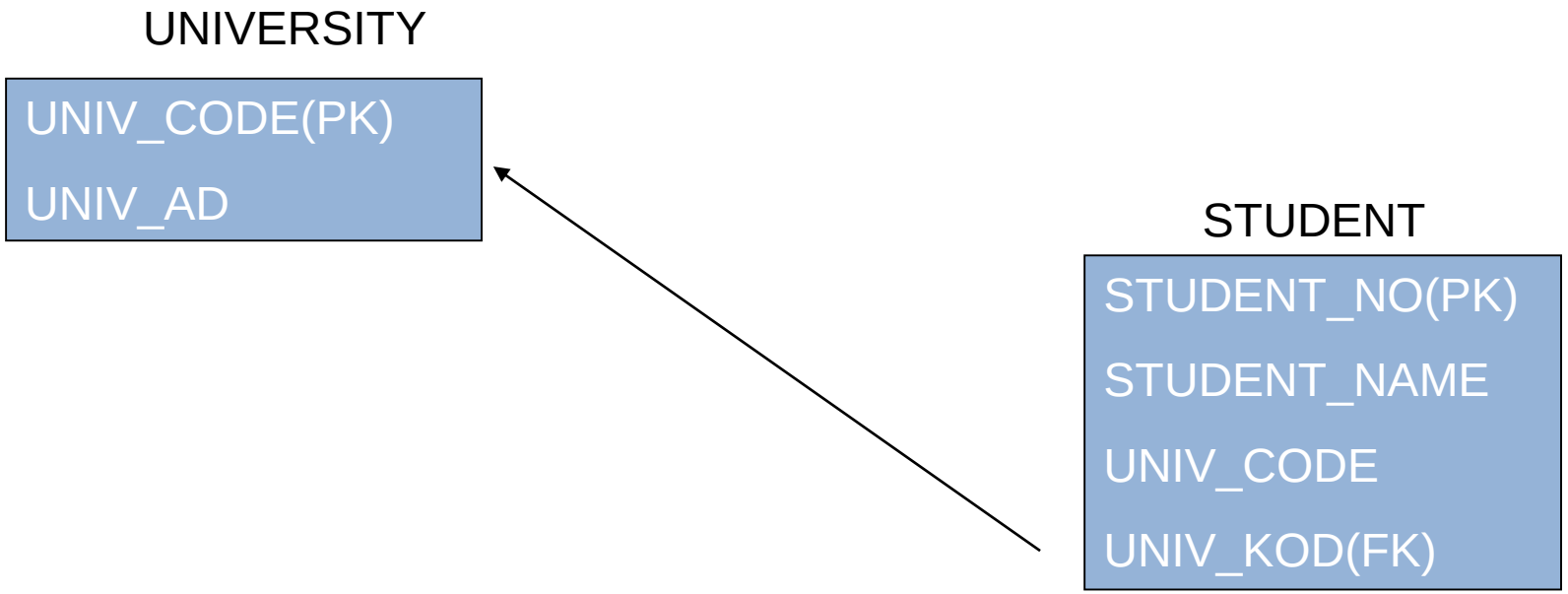
Establish the table relations
Analyze each table in order to decide which columns will take place in other tables.

Detail your design
Analyze your design for errors. Create the tables and insert test records to find whether there are anomalies in your design. Make arrangements on your design if necessary.

Summary of Database Design (Cont'd)

- Apply normalization rules
- Define constraints for integrity :
  - not null
  - primary key
  - unique constraint
  - check constraints
  - Foreign keys (References another tables)
- Use indexes for performance (DB already creates for PK)

PK – FK RELATIONS

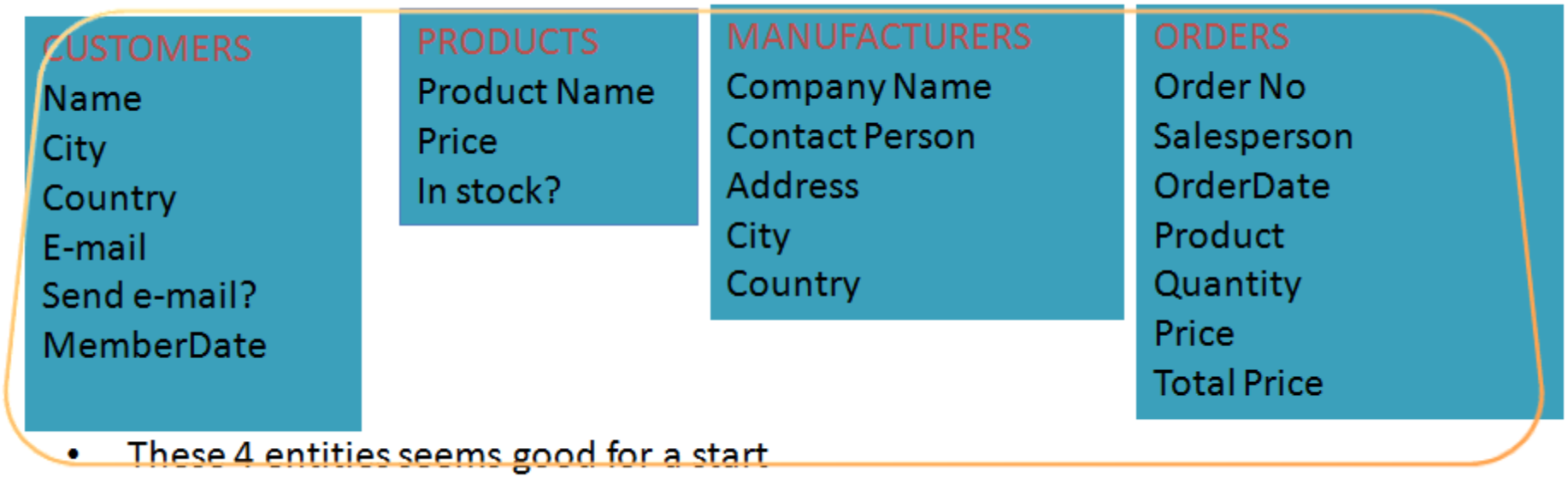


- There is **master-child** relation between university and student
- University codes of the Student should be contained in University table
- PK and Index are needed

DATABASE DESIGN EXAMPLES AND MAIN POINTS

Divide your information into tables

For example, the main entities or subjects for a Product Sales Database can be designed as below at first:



- These 4 entities seems good for a start

Divide your information into tables (Cont'd)

If you had designed a single table instead of 4 different tables:

Products and Manufacturers

Product	Manufacturer	Address
Çay	Yabancı İçkiler	Çiftlik Sok. 9/1
Bira	Yabancı İçkiler	Çiftlik Sok. 9/1
Vişne Likörü	Yabancı İçkiler	Çiftlik Sok. 9/1
Şefin Kajun Baharatları	New Orleans Kajun Tavu	PK 67343

Each record would contain data about both products and manufacturers
You may have many products coming from a manufacturer.
In this case, you need to enter the name and address **multiple times** causing unnecessary disk usage. Instead, a MANUFACTURERS table related with a PRODUCTS table would provide a single record for a manufacturer.

One other anomaly would be seen while **manipulating data**.
If the address of a company changes, you need to update all the records related with that company.

Divide your information into tables (Cont'd)

If you had designed a single table instead of 4 different tables:

Products and Manufacturers

Product	Manufacturer	Address
Çay	Yabancı İçkiler	Çiftlik Sok. 9/1
Bira	Yabancı İçkiler	Çiftlik Sok. 9/1
Vişne Likörü	Yabancı İçkiler	Çiftlik Sok. 9/1
Şefin Kajun Baharatları	New Orleans Kajun Tavu	PK 67343

Another anomaly; Assume a manufacturer has only one product. . If you want to delete this product, but want to keep the Manufacturer's data, you can not achieve this goal.

Note: create tables(entities) that represent a subject and include columns only related to that subject.

For example, Manufacturer address is a concept that belongs to manufacturer, not to product. Hence, this column should be in MANUFACTURERS table.

Decide which columns will take place in each table

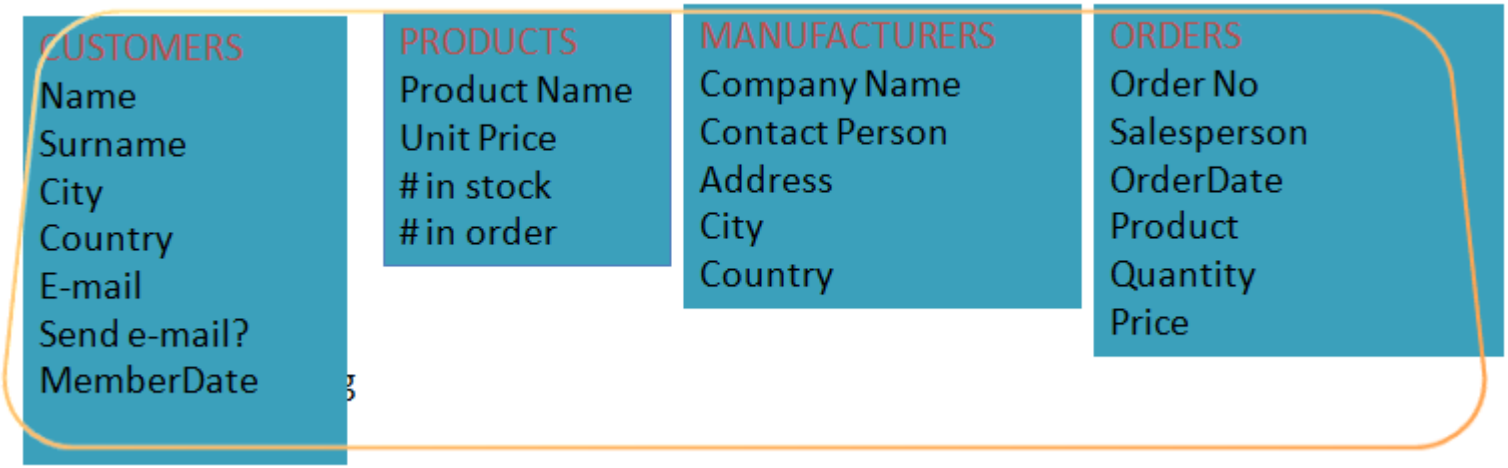
Assume you decided your address column to include country, province, and districts in a single column (Turkey, Ankara, Çankaya).

If you will produce reports based on province or order the reports by country, than you'd better divide this field into 3 seperate columns.
(Codes of these locations are preferable)

Seperate Name and Surname columns if surnames are important in your future reports.

No need to include derived columns in tables (eg. Age, Total Price)

For example, the main entities or subjects for a Product Sales Database can be designed as below at first:



every table, you are ready to choose the Primary Key of each.

Decide Primary Keys

Primary keys are the main factors that define a record uniquely.

For example, in a PERSONNEL table, SSN is an ideal candidate for being a PK.
PK s can not be null(empty) and can not have a repeating value in a table.

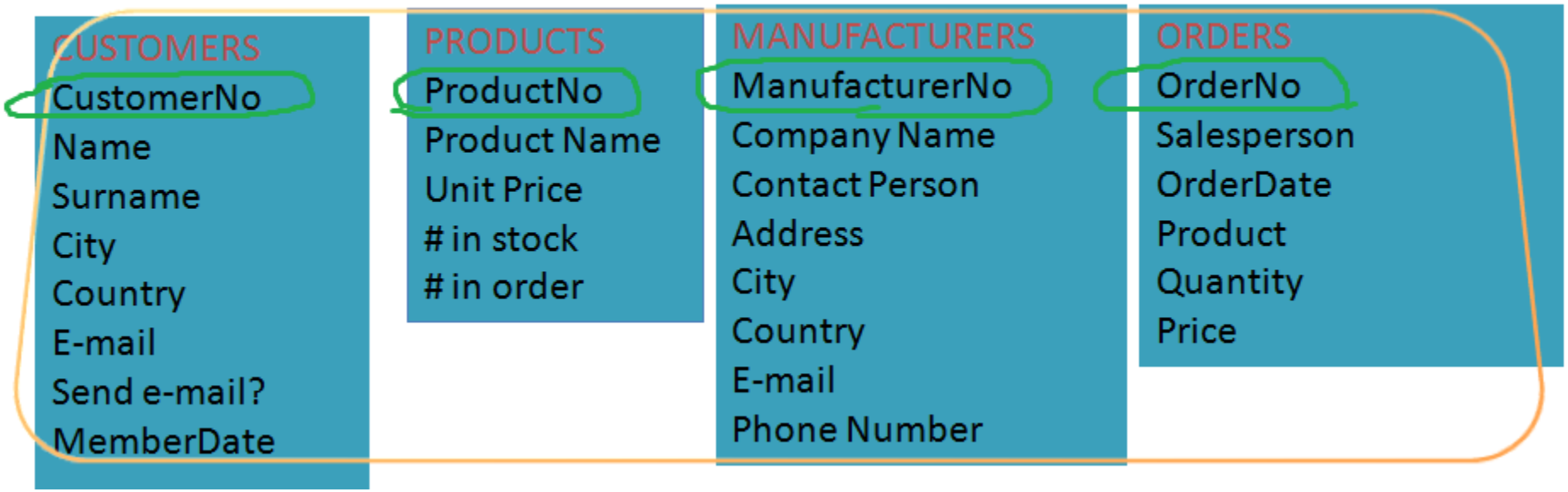
Name is a bad candidate for being a PK for a PERSONNEL,
because you will most probably have records that have the same name.

The PK for a table will probably be a reference (foreign key) for another table.
So, your PK's should be unchangeable, generally.

An auto-increment number (aka. surrogate key) can also be a PK for some tables
(eg. ORDERS table).

In some cases, your PK may consist of more than one column.

Our tables have Primary Keys:

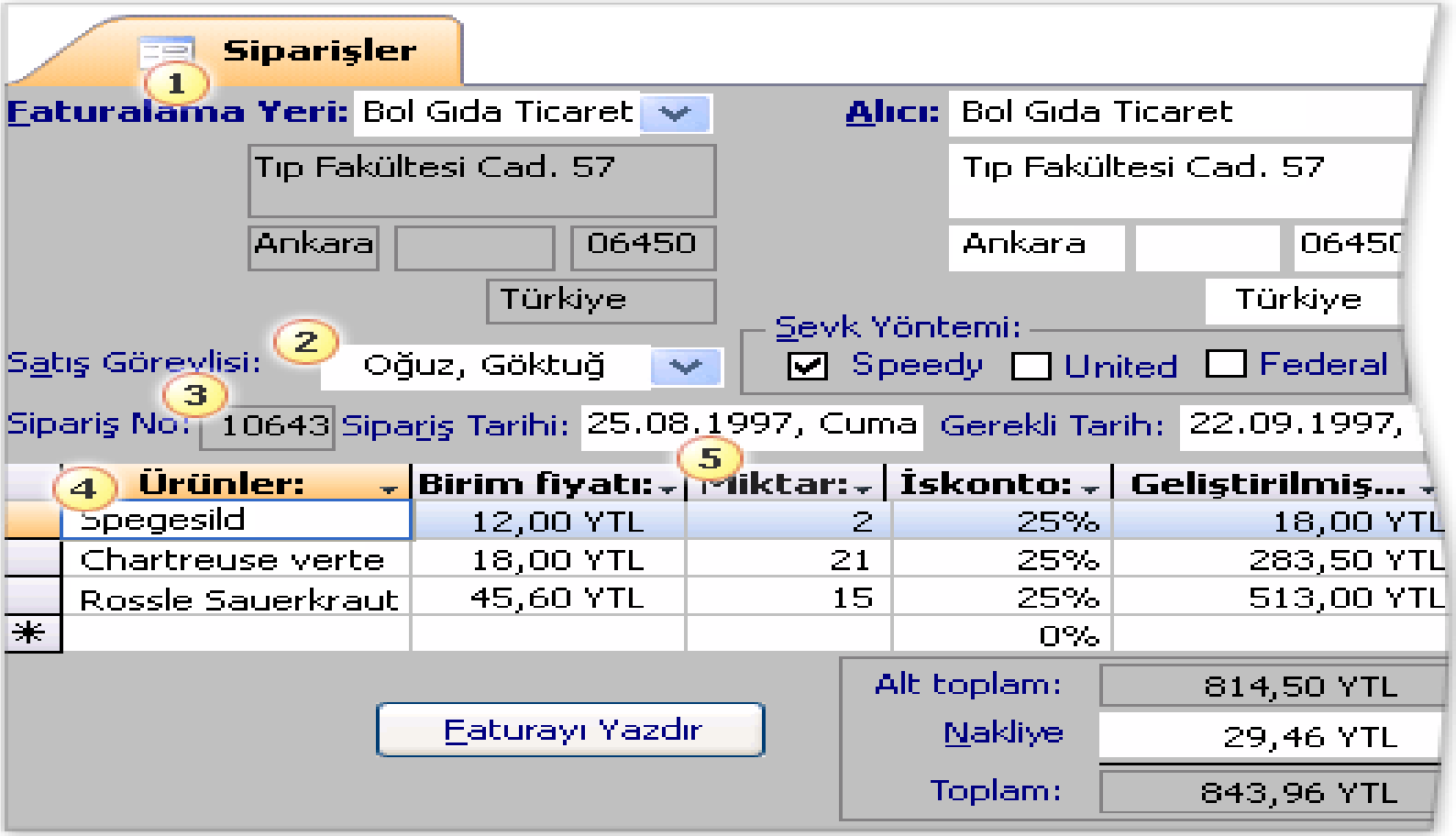


RELATE THE TABLES

On relational databases, you divide your data into subject based tables

Afterwards, you relate the tables in order to query more than one table at a time.

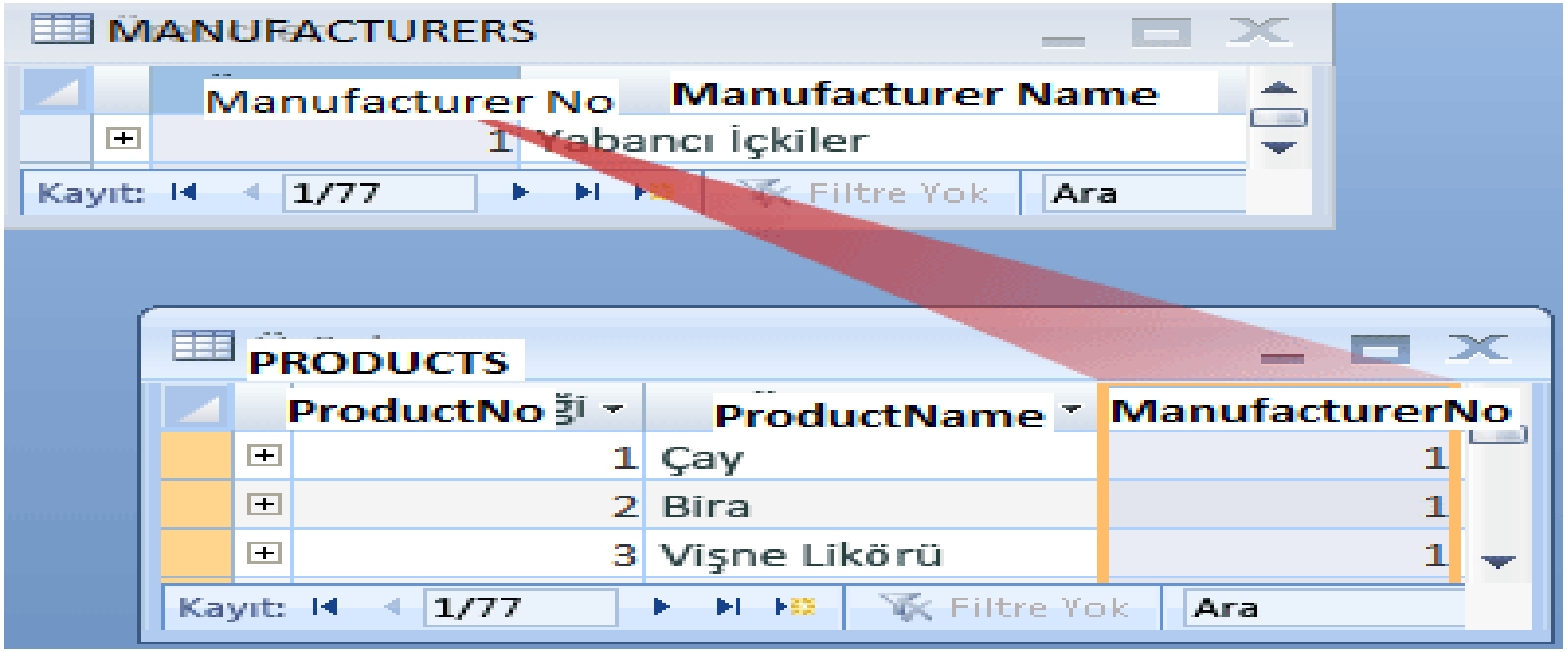
Different parts from different tables at Application



In our Product Orders database, there is MANUFACTURERS table and PRODUCTS table.

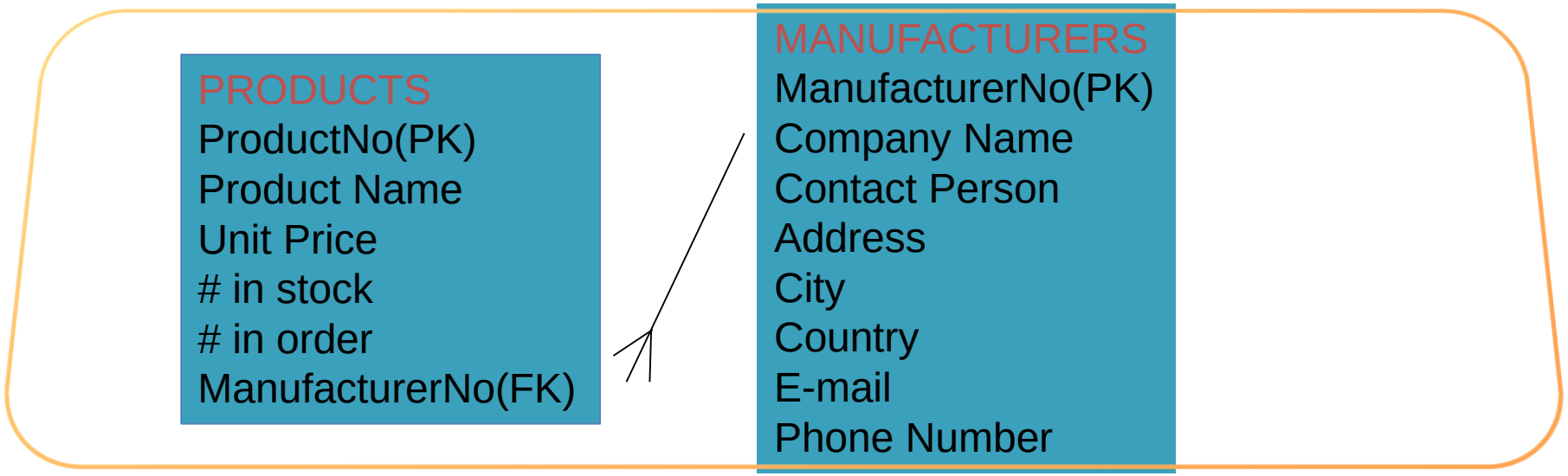
A manufacturer can produce more than one products.
As a result, many rows may exist in PRODUCTS table that belong to a manufacturer.

(1- M Relationship)



1-M Relationship

A Foreign Key must be the Primary key of another table.
It is the most common relationship used when creating relational databases.
A row in a table in a database can be associated with one or (likely) more rows in another table.



M-M Relationship

Let’s decide the relationship between PRODUCTS and ORDERS tables.
A single order may include more than one product.
For every record in PRODUCTS table, more than one record in ORDERS table may exist.

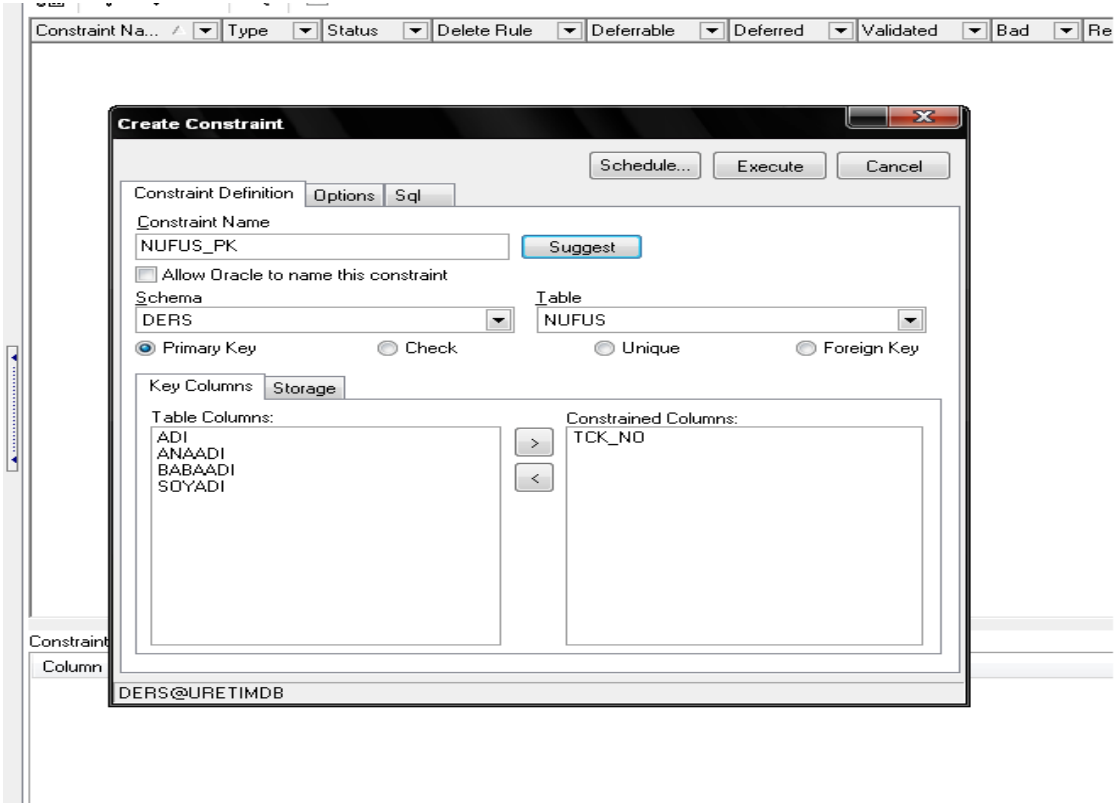
On the other hand, a product may be involved in more than one order.

For every record in ORDERS table, more than one record in PRODUCTS table may exist.

Let’s think what kind of problems may arise?

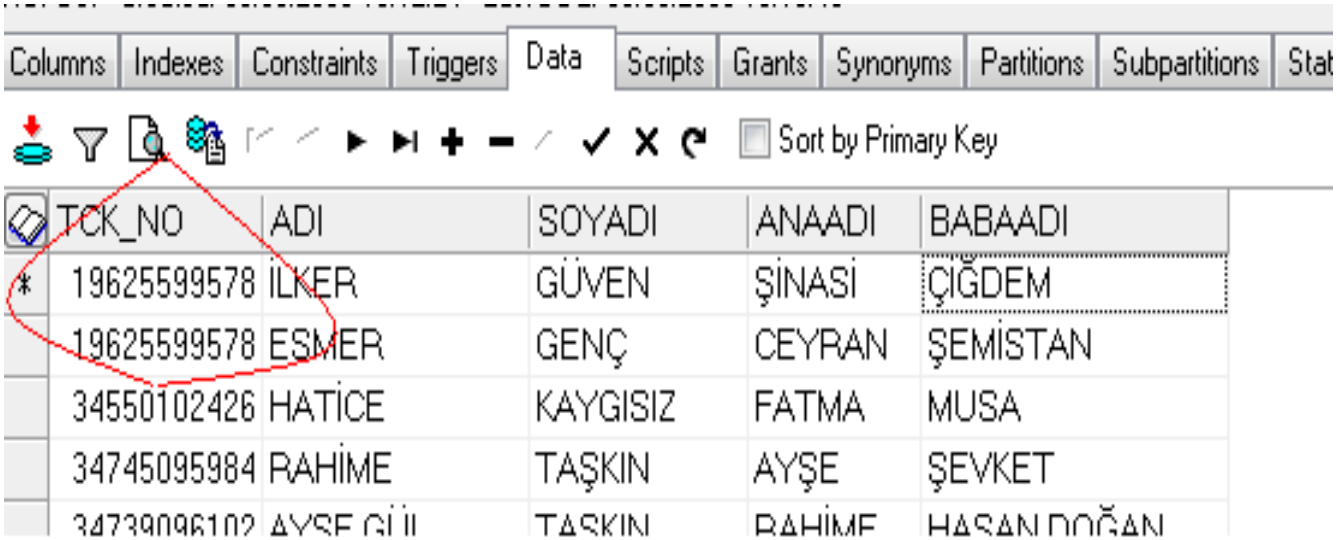
PREVENT DUPLICATION

For example, in the Citizen table, We chose tck_no as pk. :



PREVENT DUPLICATION (Cont'd)

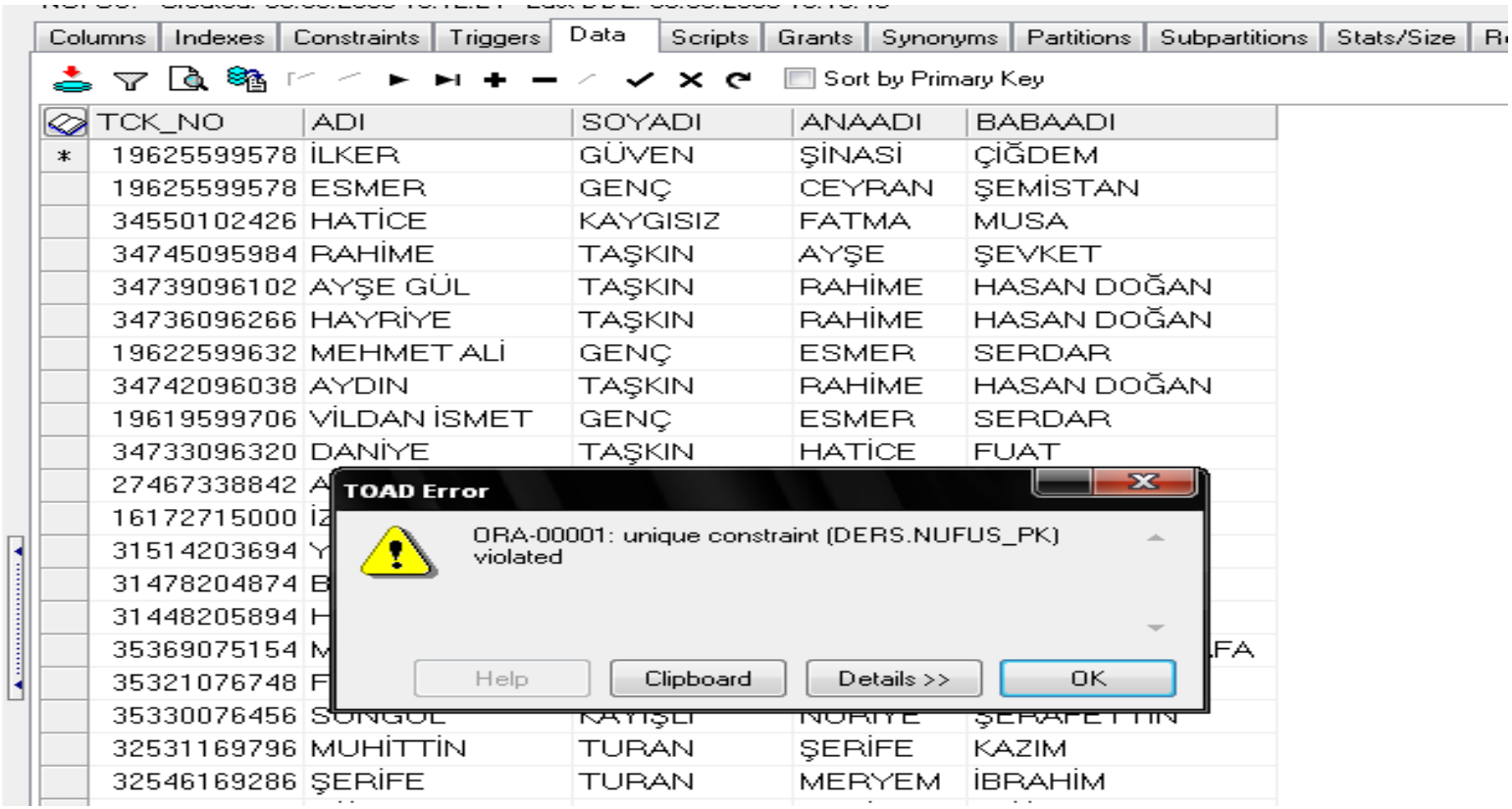
After making the TCK_NO column the Primary Key,
if we want to insert another record with the same TCK_NO :



TCK_NO	ADI	SOYADI	ANAADI	BABAADI
* 19625599578	İLKER	GÜVEN	ŞİNASİ	ÇİĞDEM
19625599578	ESMER	GENÇ	CEYRAN	ŞEMİSTAN
34550102426	HATİCE	KAYGISIZ	FATMA	MUSA
34745095984	RAHİME	TAŞKIN	AYŞE	ŞEVKET
34739096102	AYSE GÜL	TASKIN	RAHİME	HASAN DOĞAN

PREVENT DUPLICATION (Cont'd)

After making the TCK_NO column the Primary Key,
if we want to insert another record with the same TCK_NO :



FORCE TO ENTER A VALUE

- Another consistency check mechanism is to force the user to enter a value to columns.

- For example, no records with null values in NAME and SURNAME columns should exist.

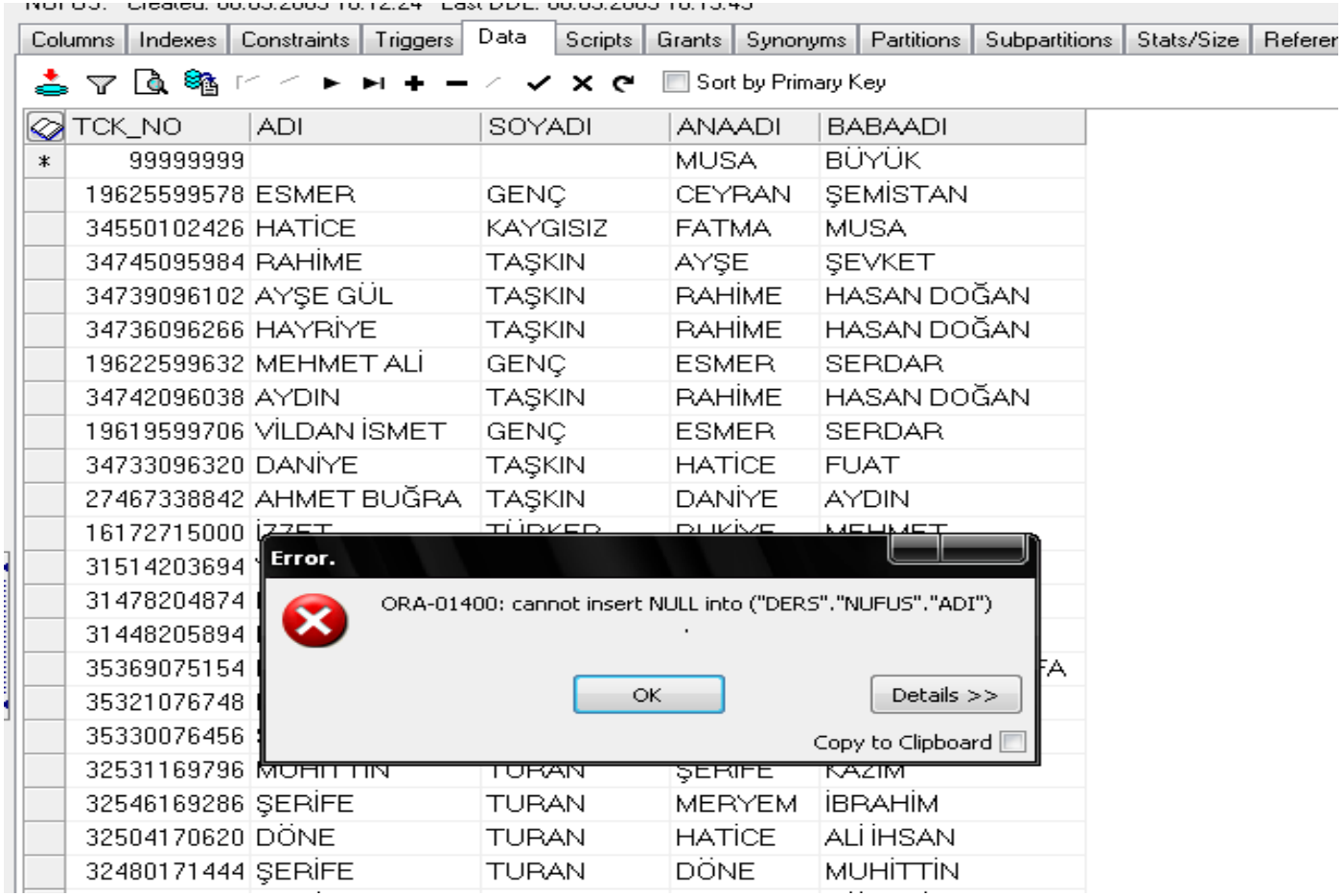
  To enforce this is also possible with a database constraint.

FORCE TO ENTER A VALUE (cont'd)

Columns | Physical Attributes | Additional Attributes | Constraints | Comments

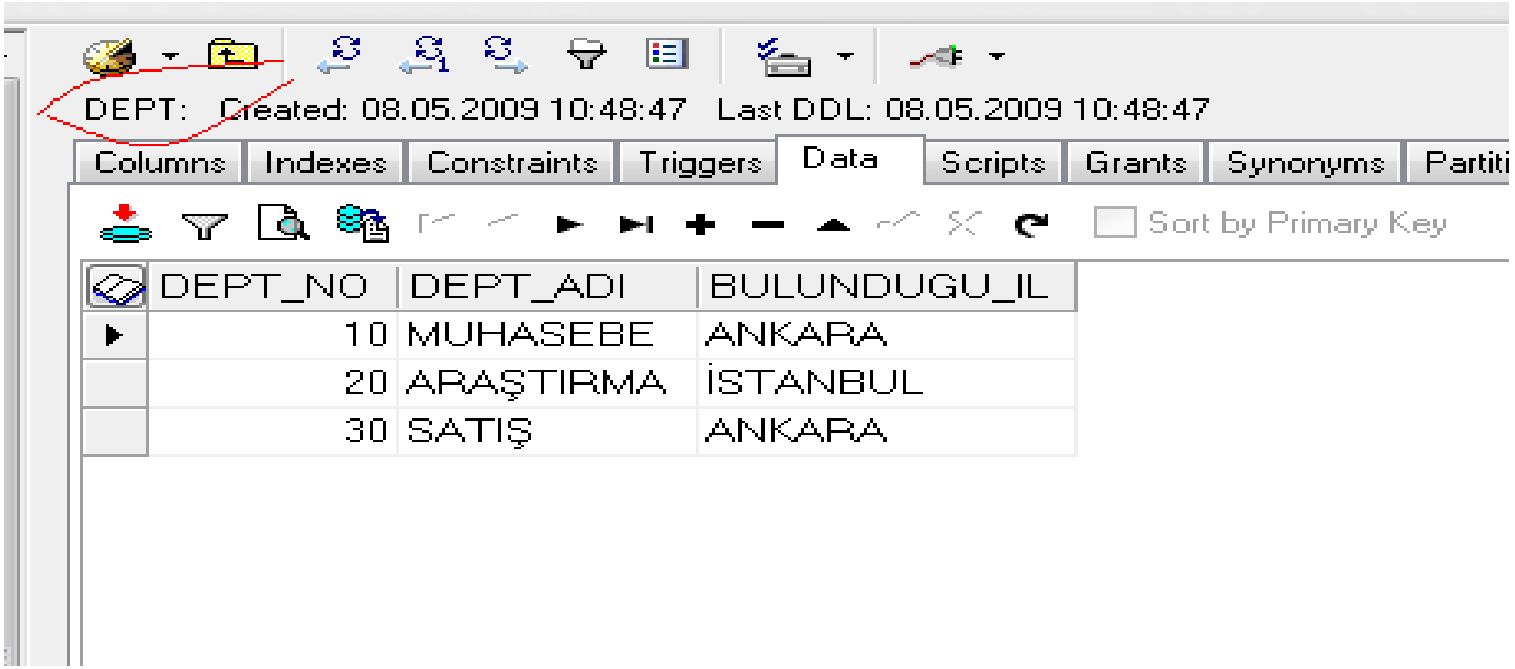
	ID	Column Name	DataType	Size	Byte/C...	Precision	Scale	Not Null	Default	Ref
	1	TCK_NO	NUMBER			11		☑		☐
▲	2	ADI	VARCHAR2	30	Byte			☑		☐
▲	3	SOYADI	VARCHAR2	30	Byte			☑		☐
	4	ANAADI	VARCHAR2	30	Byte			☐		☐
	5	BABAADI	VARCHAR2	30	Byte			☐		☐

FORCE TO ENTER A VALUE (cont'd)



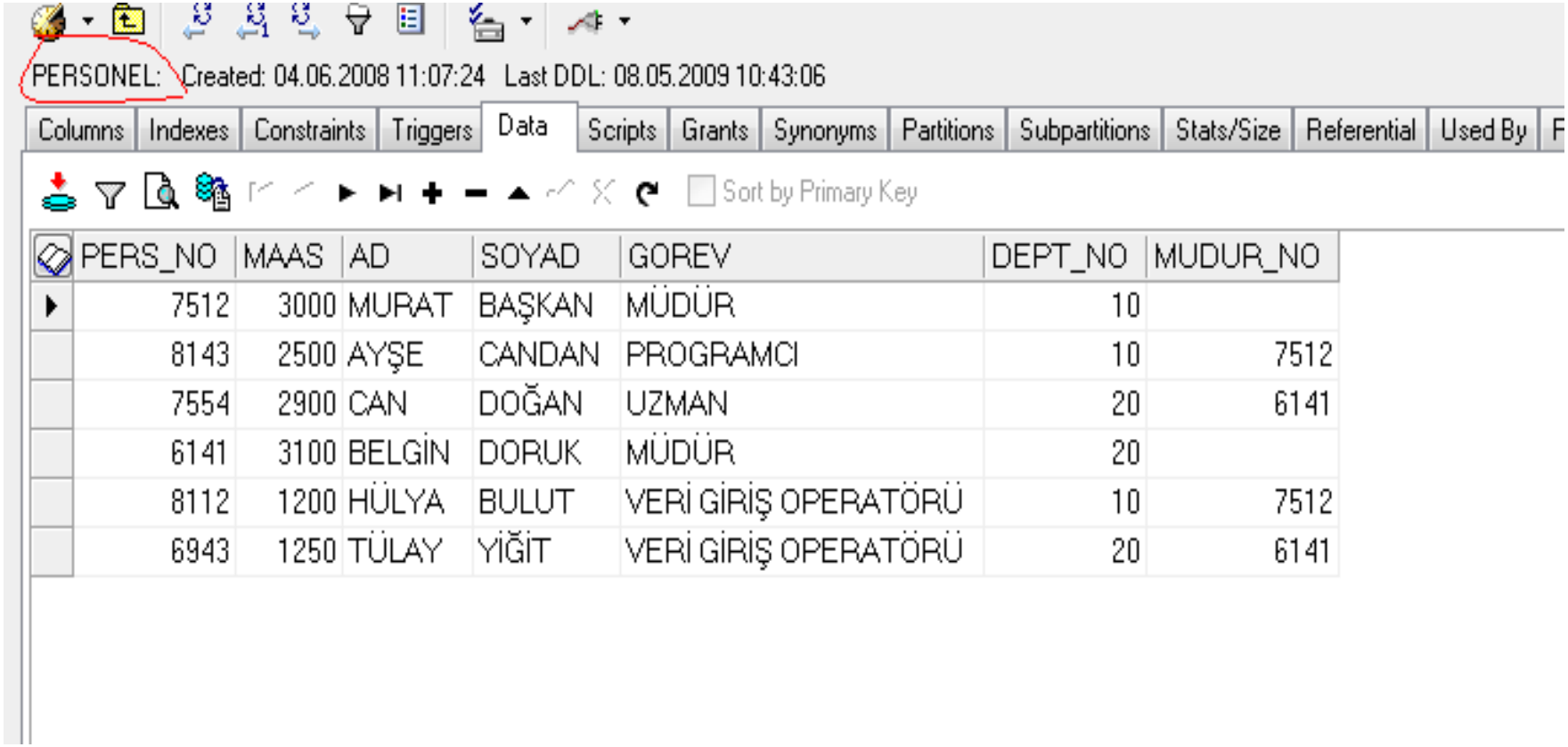
RELATIONAL DATABASES

Assume we have a PERSONEL table that stores personnel data and a DEPT table that stores the departments in the corporation. These 2 tables have such records:



DEPT_NO	DEPT_ADI	BULUNDUGU_IL
10	MUHASEBE	ANKARA
20	ARAŞTIRMA	İSTANBUL
30	SATIŞ	ANKARA

RELATIONAL DATABASES (Cont'd)



PERS_NO	MAAS	AD	SOYAD	GOREV	DEPT_NO	MUDUR_NO
7512	3000	MURAT	BAŞKAN	MÜDÜR	10	
8143	2500	AYŞE	CANDAN	PROGRAMCI	10	7512
7554	2900	CAN	DOĞAN	UZMAN	20	6141
6141	3100	BELGİN	DORUK	MÜDÜR	20	
8112	1200	HÜLYA	BULUT	VERİ GİRİŞ OPERATÖRÜ	10	7512
6943	1250	TÜLAY	YİĞİT	VERİ GİRİŞ OPERATÖRÜ	20	6141

RELATIONAL DATABASES (Cont'd)

- If you do not put a constraint, one day you may face with a personnel who has a department that does not exist in the DEPT table. Because, for example, data entry people may enter incorrectly while entering quickly

- A database constraint must be put for not to face with such a problem. The most important feature of the relational database is the ability to relate these tables and does not allow **inconsistent** data.

RELATIONAL DATABASES (Cont'd)

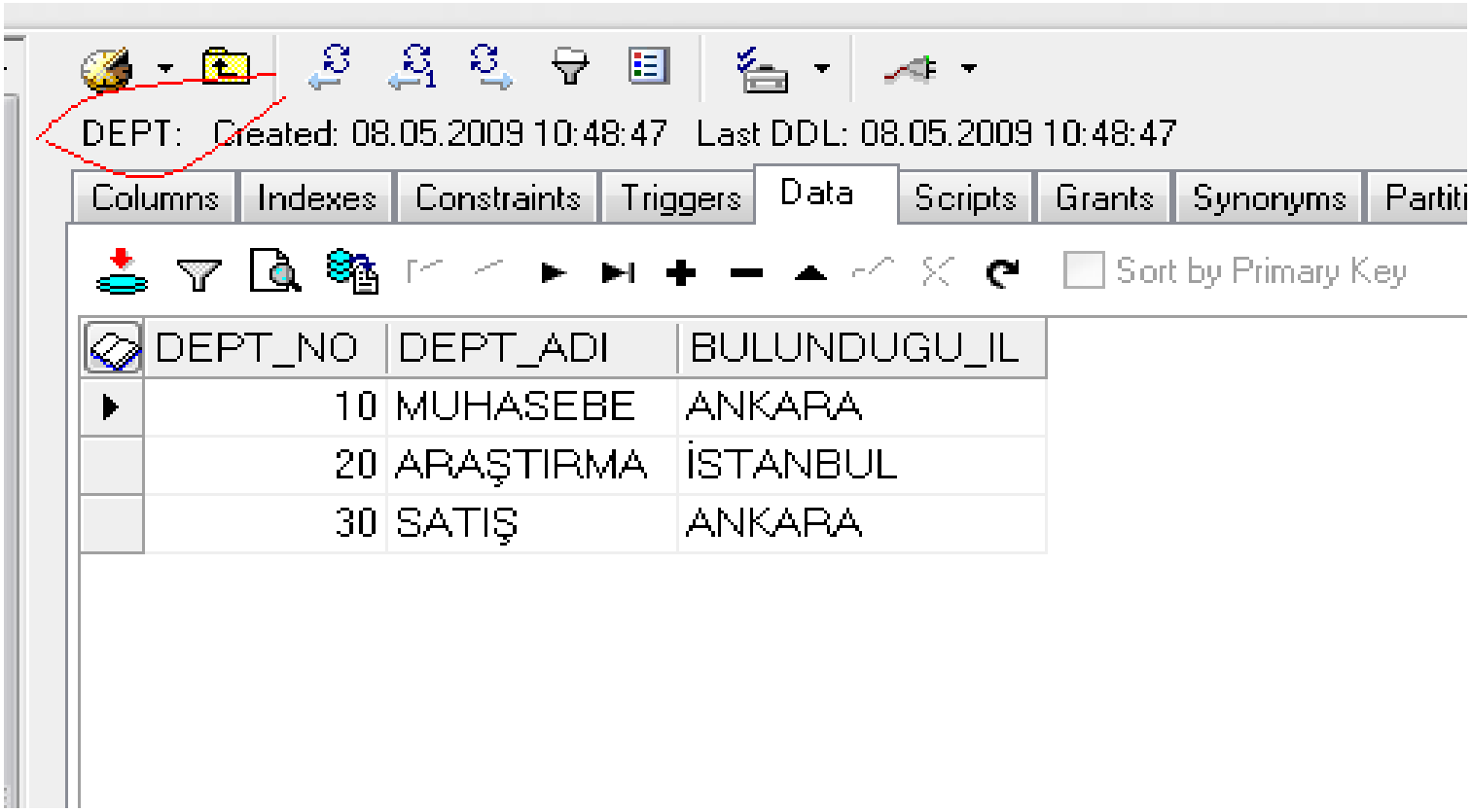
PERSONEL: Created: 04.06.2008 11:07:24 Last DDL: 08.05.2009 10:43:06

Columns | Indexes | Constraints | Triggers | Data | Scripts | Grants | Synonyms | Partitions | Subpartitions | Stats/Size | Referential | Use

Sort by Primary Key

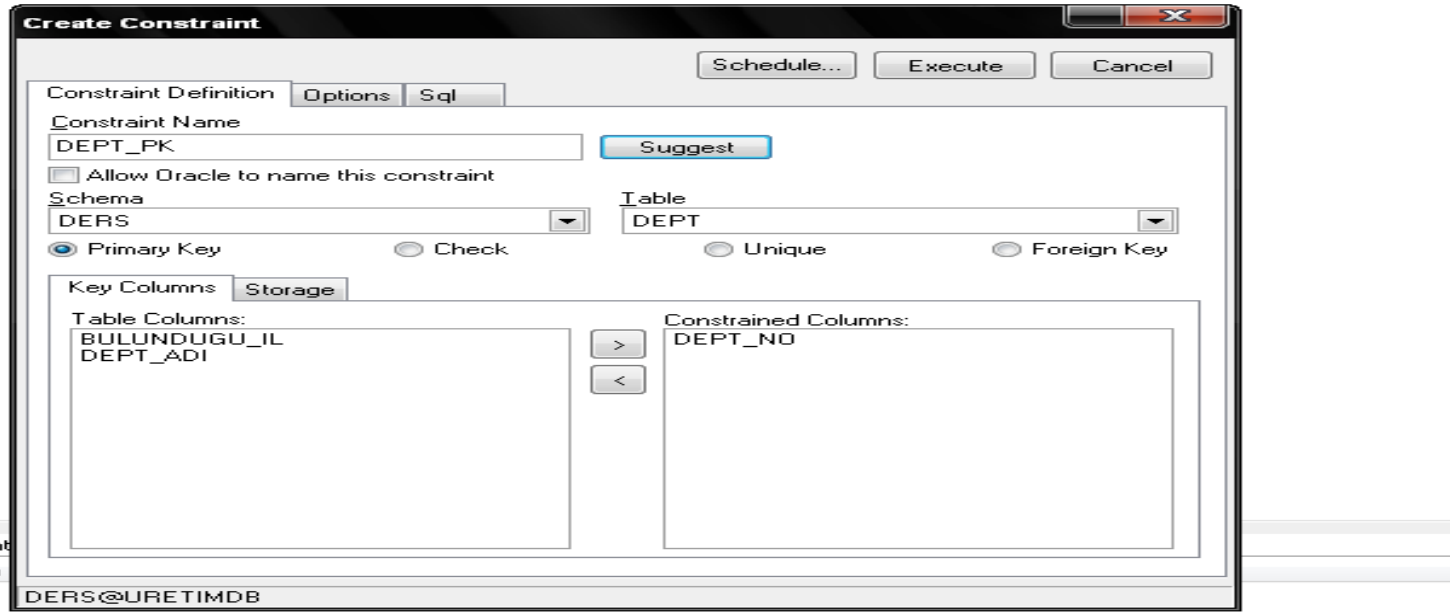
PERS_NO	MAAS	AD	SOYAD	GOREV	DEPT_NO	MUDUR_NO
8159	2000	CEYDA	MAYDA	UZMAN YARDIMCISI	40	
7512	3000	MURAT	BAŞKAN	MÜDÜR	10	
8143	2500	AYŞE	CANDAN	PROGRAMCI	10	7512
7554	2900	CAN	DOĞAN	UZMAN	20	6141
6141	3100	BELGİN	DORUK	MÜDÜR	20	
8112	1200	HÜLYA	BULUT	VERİ GİRİŞ OPERATÖRÜ	10	7512
6943	1250	TÜLAY	YİĞİT	VERİ GİRİŞ OPERATÖRÜ	20	6141

RELATIONAL DATABASES (Cont'd)

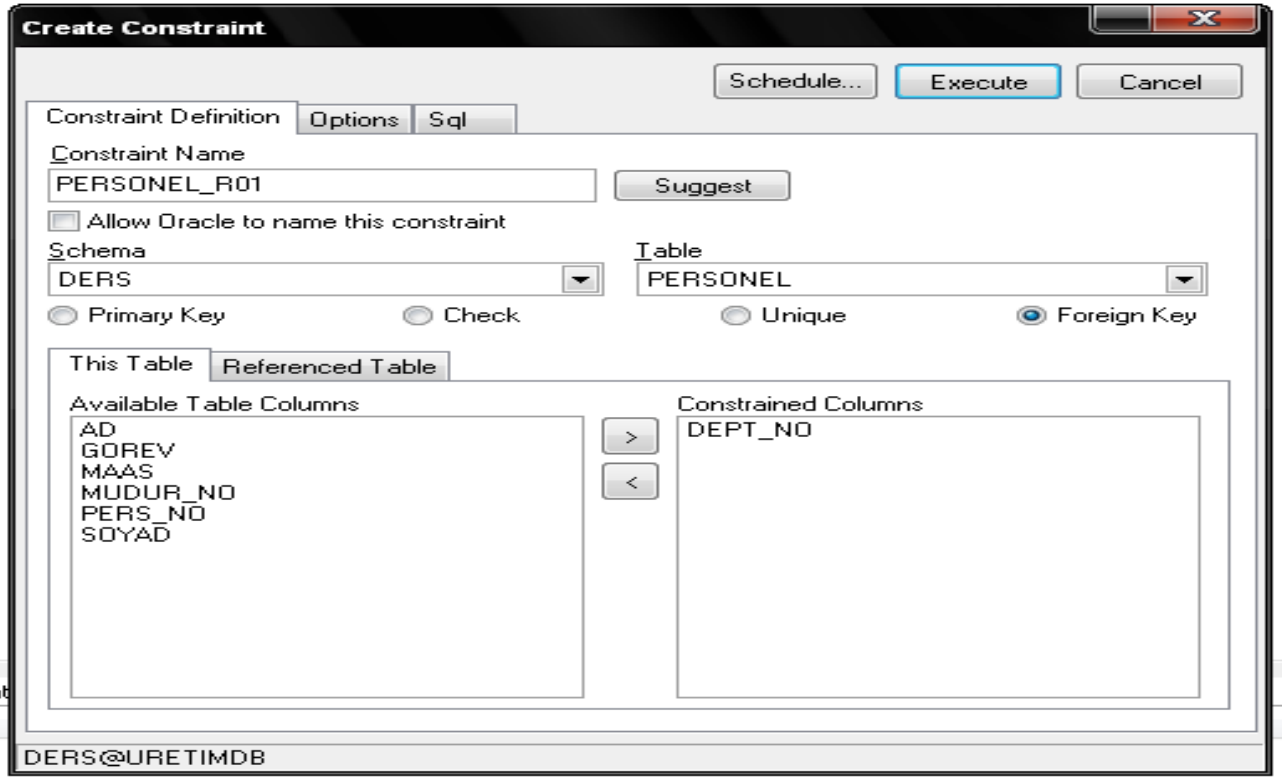


RELATIONAL DATABASES (Cont'd)

To prevent such an inconsistency,
DEPT_NO column of PERSONEL table should be related to
DEPT_NO column of DEPT table.



RELATIONAL DATABASES (Cont'd)



RELATIONAL DATABASES (Cont'd)

- After relating these 2 tables, the DEPT table is called as **master table** or **parent table** or **reference table.**
- PERSONEL table is called as **child table.**
- After this relation, no one can enter a value to PERSONEL.DEPT_NO column that does not exists in DEPT.DEP_NO column.

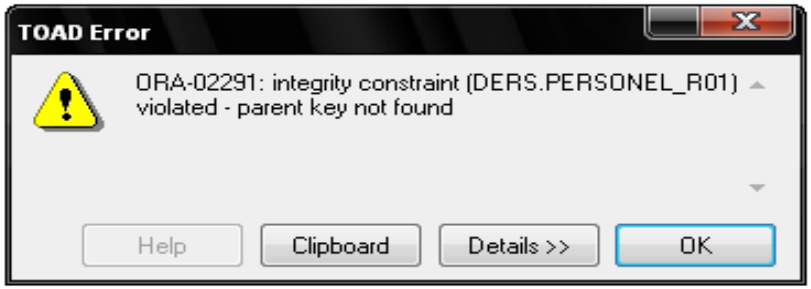
RELATIONAL DATABASES (Cont'd)

PERSONEL: Created: 04.06.2008 11:07:24 Last DDL: 08.05.2009 11:03:39

Columns | Indexes | Constraints | Triggers | Data | Scripts | Grants | Synonyms | Partitions | Subpartitions | Stats/Size | Referential | Used By

Sort by Primary Key

PERS_NO	MAAS	AD	SOYAD	GOREV	DEPT_NO	MUDUR_NO
8159	2000	CEYDA	MAYDA	UZMAN YARDIMCISI	40	
7512	3000	MURAT	BAŞKAN	MÜDÜR	10	
8143	2500	AYŞE	CANDAN	PROGRAMCI	10	7512
7554	2900	CAN	DOĞAN	UZMAN	20	6141
6141	3100	BELGİN	DORUK	MÜDÜR	20	
8112	1200	HÜLYA	BULUT	VERİ GİRİŞ OPERATÖRÜ	10	7512
6943	1250	TÜLAY	YİĞİT	VERİ GİRİŞ OPERATÖRÜ	20	6141



RELATIONAL DATABASES (Cont'd)

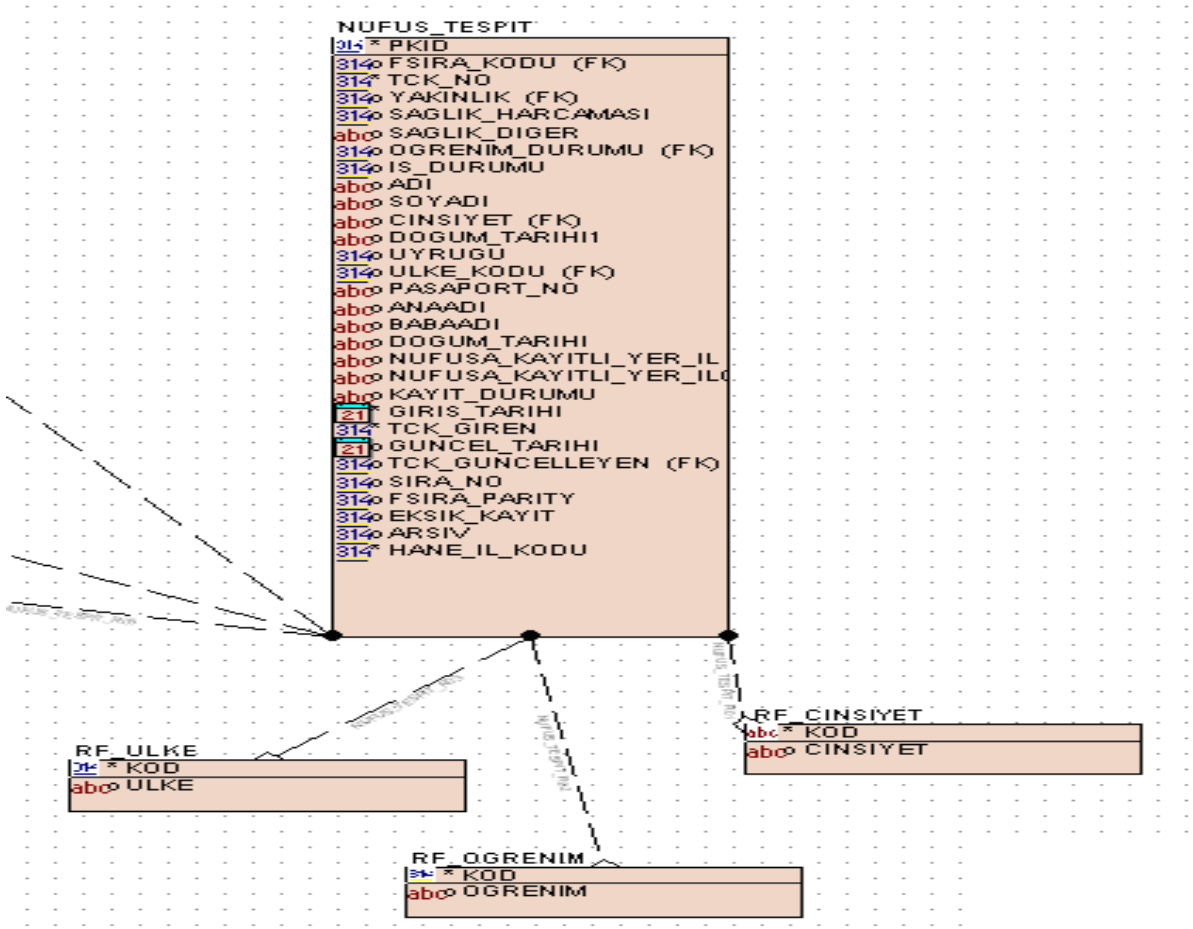
- But similarly, no one can also delete a record from DEPT table that has a child record in PERSONEL table.

RELATIONAL DATABASES (Cont'd)

- In a data entry application, the ideal solution for consistent data collection is to create **reference tables** and establish a PK-FK relation between the tables.

- To give an example, if you are making a population census project and the variables on the survey paper include EDUCATION LEVEL, COUNTRY CODE, GENDER the ideal design should be something like this:

RELATIONAL DATABASES (Cont'd)



RELATIONAL DATABASES (Cont'd)

- By this way, you guarantee that unknown values of education level, country code or gender can not be entered to POPULATION table.

- You can write your own constraints also in the application programming language like Java, .NET, etc.

- But it will be waste of time since any RDBMS guarantees these issues.

CONCLUSION

Database Design Steps are;

- Required Analysis
- Conceptional Data Design (ER)
- Logical Database Design (Relational)
- Physical Database Design

Before designing a database; requirements analysis should be done carefully !

Maintenance should be considered while designing

Normalization/Denormalization should be considered, evaluated.

Data integrity, consistency should be guaranteed using constraints

Performance of the system should be also evaluated

Case Study:

We are designing DB system for a shop. Sytem includes products and sales information.

1) Please draw the **ER** diagram for the system

2) After ER diagram; write table names, columns, undeline the primary keys and foreign keys columns

The requirements of the system are :

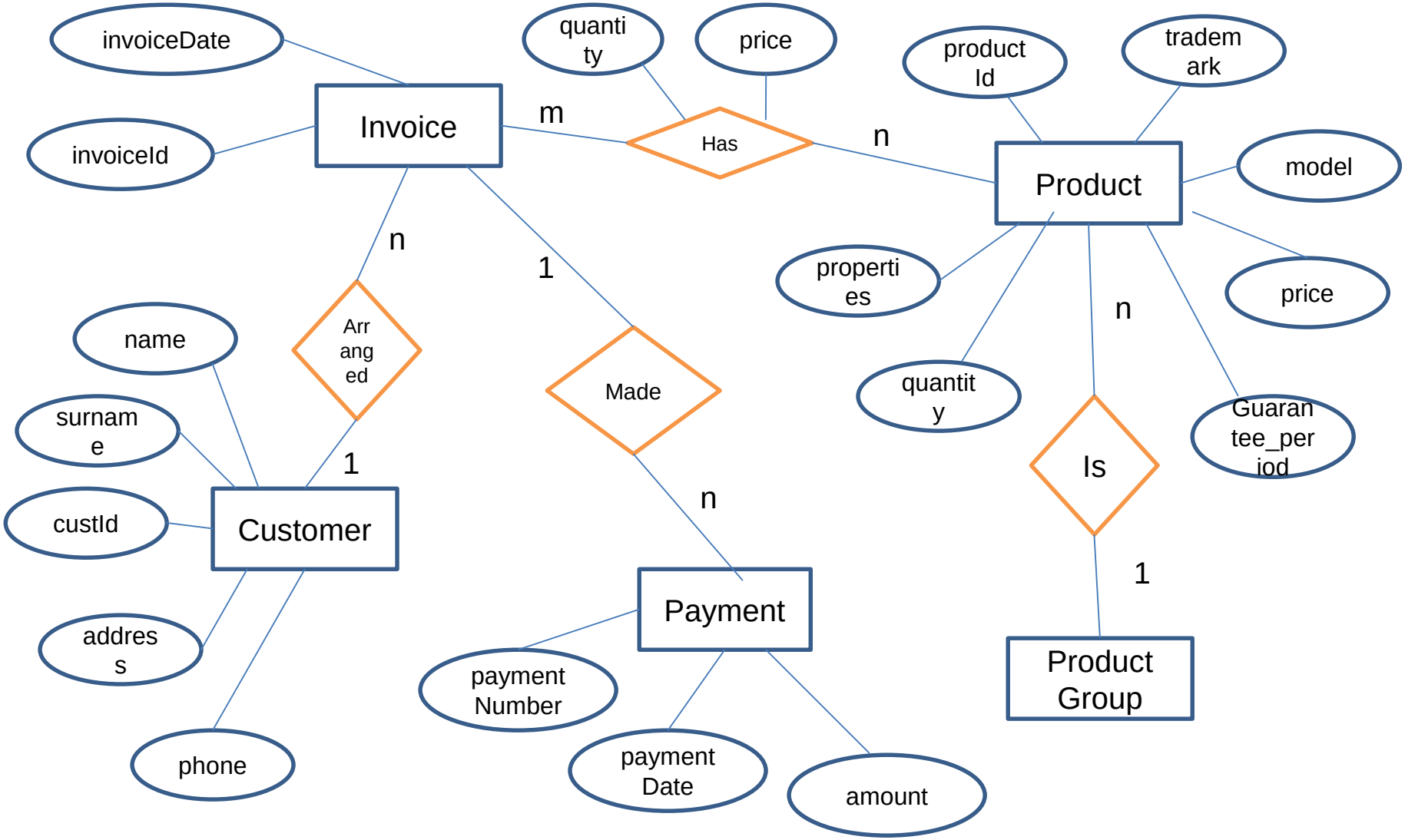
•Each product has product number, trademark, model, properties, unit price, guarantee period and stock quantity.

•Each product belongs to a product group, a product group may contain many products.

•The information of the customers -that buy products- are saved. Customers have id, name, surname, address, phone information. When a customer buy a product, invoice is arranged. More than one invoice may be arranged for a customer, but an invoice is arranged for one customer.

•There may be more products in an invoice. A product may exist more than one invoice. The price of the sale and the quantity are recorded.
•More than one payment is available for a sale. Each payment has date and paid price.

** Hint : You may Underline objects to find entities and their attributes


invoiceDate
quanti ty
price
product Id
tradem ark
Invoice
m
Has
n
Product
invoiceId
model
n
1
properti es
price
Arr ang ed
n
name
Made
quantit y
surnam e
Guaran tee_per iod
Is
1
1
custId
Customer
n
Payment
addres s
Product Group
payment Number
payment Date
amount
phone

Tables of the database:

- Customer (**customerId**, name, surname, address, phone)
- Invoice (**invoiceId,**invoiceDate, customerId)
- Product (**productId**, trademark,model,properties,unitPrice,guranteePeriod, quantity, productGroupId)
- ProductGroup (**productGroupId**, name)
- Payment (**invoiceId , paymentNumber,** date, amount)
- Sale (**saleId**, invoiceId, productId, quantity, salePrice)