Database Design

Some facts, rules and information

34014 - DATA BASE DESIGN (2013-14)
Degree in Computer Engineering
University of Alicante
General information

- Course facts
- Context
- Main goals
- Theoretical syllabus
- Basic bibliography
- Assessment
Course facts

- **Code:** 34014
- **Degree:** Computer Engineering
- **Type:** Compulsory
- **Credits:** 6 ECTS
- **Prerequisites:** None
- **Recommendation:** Fundamentals of Databases
  - *Fundamentos de las Bases de Datos*
- **More details:** [http://www.ua.es](http://www.ua.es)
Main goals

• Specific competences (CE)

  • **CE12**: understand and apply the characteristics, functionalities and structure of *databases* to allow a correct use and *design, analyze and implement* applications based on the same

  • **CE13**: understand and apply the *tools* needed for the storage, process and access Information Systems, including web-based systems
Main goals

• Learning goals

• Acquire **basic knowledge and techniques** to be able to learn and develop new methods and technologies, as well as skills to face up to novel situations

• **Ability for solving problems** with own initiative, informed decision making, autonomy and creativity

• **Communicate** knowledge, capabilities and skills related to Computer Engineering profession
Main goals

- **Specific goals**
  - **Design** a database
    - Conceptual design by using the EER model
    - Logical design by using the Relational model
    - Physical design by using the Oracle DBMS
  - Know issues related to **management** and **security** of databases
Theoretical syllabus

- Database design method: conceptual, logical and physical design
- EER and Relational models

  - Conceptual design with the EER model
    - Logical design with the relational model
    - Physical design according to a specific DBMS
      - Including design of ETL processes
    - SQL sentences to create, query, and manage a database

- If enough time
  - Secure database design
  - Database audit and laws
Planning

- 10/09 → Presentation and overview of database design method
- 17/09 → Conceptual design (I)
- 24/09 → Conceptual design (II)
- 01/10 → Conceptual design (III)
- 08/10 → First presentation of project
- 15/10 → Logical design (I)
- 22/10 → Logical design (II)
- 29/10 → Logical design (III)
- 05/11 → Second presentation of project
- 12/11 → Physical design
- 19/11 → Design of ETL processes
- 26/11 → Exercises
- 03/12 → Final presentation of project
- 10/12 → Exercises
- 17/12 → Exercises
Basic bibliography

- Slides and other resources to be available on CV

- Diseño y administración de bases de datos
  Autor(es): HANSEN, Gary W.; HANSEN, James V.
  ISBN: 84-8322-002-4

- Fundamentos de sistemas de bases de datos
  Autor(es): ELMASRI, Ramez; NAVATHE, Shamkant B.
  Edición: Madrid : Pearson Addison-Wesley
  Introducción a los sistemas de bases de datos
  Autor(es): DATE, C. J.
  ISBN: 968-444-4192

- Sistemas de Bases de Datos : un enfoque práctico para diseño, implementación y gestión
  Autor(es): CONNOLLY, Thomas M.; BEGG, Carolyn E.
  ISBN: 84-7829-075-3

- Up-to-date editions are in English!
Assessment

First assessment period - January

- Continuous assessment
  - 60% theory + 40% practices
  - Marks range from 0 to 10
- Final mark
  - Addition of weighted marks of each project and examination provided both theory and practices are passed, otherwise the course is failed

<table>
<thead>
<tr>
<th></th>
<th>% mark</th>
</tr>
</thead>
<tbody>
<tr>
<td>THEORY</td>
<td></td>
</tr>
<tr>
<td>Intermediate project assignments</td>
<td>20%</td>
</tr>
<tr>
<td>THEORY</td>
<td></td>
</tr>
<tr>
<td>Final project</td>
<td>40%</td>
</tr>
<tr>
<td>PRACTICES</td>
<td></td>
</tr>
<tr>
<td>First practical examination</td>
<td>15%</td>
</tr>
<tr>
<td>PRACTICES</td>
<td></td>
</tr>
<tr>
<td>Final practical examination</td>
<td>25%</td>
</tr>
</tbody>
</table>
Assessment
Second assessment period - July

- Marks of theory or practices are kept from first assessment period (January) provided you passed

- One or two examinations can be done
  - Exam A about theoretical concepts (60%)
  - Exam B about practical concepts (40%)
Overview on using a methodology for database design
Goal of the session

- Recall concepts and terminology from previous related subjects, such as Fundamentals of Databases
- Introducing students to the convenience of using a methodology for database designing
- Locate database development within a whole software engineering project
- Defining the main actors involved in the development of a database within a software engineering project
Database & software design

- Software application consists of several interrelated and mutually required artifacts
  - Interfaces, architecture, database, etc.

- The design of these artifacts are approached from different perspectives and different subjects

- Regarding data
  - Most applications require data persistence
    - Data storage repository of databases
  - Fundamentals of databases (1st course) was mainly dedicated to the study of the relational model
  - However, a method for “engineering” databases is required
Database & software design

SOFTWARE

Database & software design

Interfaces, Architecture, etc.

SOFTWARE

DATABASE

Database Management System (DBMS)
Database design methodology

- Conceptual (C)
- Logical (L)
- Physical (P)

Schema
- Class or EER (C)
- Relational or graph (L)
- Tool specific (P)

Based on
- Conceptual (C)
- Logical (L)
- Physical (P)

Model
- UML or EER (C)
- Relational (L)
- Tool specific (P)

Design

Produces
Database design methodology

CONCEPTUAL

EER model

LOGICAL

Relational model

PHYSICAL

Oracle
Actors involved in database designing

- **Database manager**
  - Manages the database by monitoring all aspects of the DBMS
    - implementation security policy, management of space allocation, etc.

- **System analyst**
  - Specifies the database requirements of the end users from the documentation of the software application and interviews with end users
Actors involved in database designing

• Database designer
  • Defines all needed schemas to allow an optimal final implementation of the static part of the database (data repository) and dynamic (functionality that complements the data repository) from the requirements

• Programmer
  • Implements the necessary interfaces and applications that interact with the database implemented by the designer.

• End users
  • Different types depending on the software application
Normalization

- From conceptual schema to logical schema
Next session

- Conceptual design as a cornerstone in database design

- Fundamentals of conceptual design by using EER (Extended Entity-Relationship) model
  - Entities
  - Attributes
  - Binary relationships

- Exercises
Extended Entity-Relationship model

Database conceptual design

34014 - DATA BASE DESIGN (2013-14)
Degree in Computer Engineering
University of Alicante
Goal of the session

- Introduce student to the database conceptual database by means of the Extended Entity-Relationship (EER) model
- Defining some EER concepts
- Using EER concepts in practice to conceptually design some sample databases
What is the EER?

- **Graphical semantic data model**
  - Includes meaning of concepts in order to represent information systems
    - Conceptual schema
    - First step in database designing
  - High level of abstraction
    - Without excessive details of the future implementation

- **Focus on static properties of the information system**
  - Only data structure, not processes nor transactions

- **No DBMS based on EER**
  - It is easily translated to the relational model and directly translated to third normal form
EER originator

- **Dr. Peter Chen**
  - Louisiana State University (USA)

- **Seminal paper**
  - Download from (free by using UA network)
    - [http://dl.acm.org/citation.cfm?doid=320434.320440](http://dl.acm.org/citation.cfm?doid=320434.320440)
EER notation used in DBD

- **WARNING!**
  - Our notation is not totally standard
  - Mixture between UML and classical notation
  - The important things are the structures of the model and the concepts to be represented
Representation: entity

- Concept, class of objects
  - An entity is described by its attributes
    - Key attribute(s)
    - Composite attribute(s)
    - Multivalued attribute(s)
    - NULL value(s)
    - ...and more
      - derived
      - alternative key attribute(s)
      - ...

<table>
<thead>
<tr>
<th>EMPLOYEE</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
</tr>
<tr>
<td>location</td>
</tr>
<tr>
<td>name</td>
</tr>
<tr>
<td>first name</td>
</tr>
<tr>
<td>last name</td>
</tr>
<tr>
<td>telephone</td>
</tr>
<tr>
<td>account</td>
</tr>
<tr>
<td>(0..N)</td>
</tr>
<tr>
<td>(1..1)</td>
</tr>
</tbody>
</table>
Representation: generalizations

- **Types within an entity**
  - Common and specific data
- **Coverage properties**
  - Total or Partial
    - If every entity has a type, then the generalization is TOTAL
    - Otherwise, it is PARTIAL
  - Disjoint or Overlapping
    - If every entity only has one of the types, then it is DISJOINT
    - Otherwise it is OVERLAPPING

- There are “good” and “bad” clients
  - The “good” ones can have discounts and points; the “bad” ones don’t
  - Both “good” and “bad” clients have nif, name, birthdate and address
- All the clients have a type (TOTAL) but it is not possible to be a “good” and “bad” client at the same time (DISJOINT)
Representation: relationship

- Indicates an action (or relation) between 2 entities
- Cardinality or multiplicity
  - Number of elements of each entity that can be related to each other
Representation: relationship

- Employees **belong** to companies
- Each employee belongs to none, or no more than 1 company
  - employees **CAN** belong to 1 company
- Each company has none or many employees (N)
  - companies **CAN** have many employees
  - There is not a maximum limit
Representation: relationship

- Employees **belong** to companies
- Each employee belongs to none or many companies
  - employees **CAN** belong to many companies
- Each company has none or just 1 employee
  - companies **CAN** have 1 employee
Representation: relationship

- Employees **belong** to companies
- Each employee belongs to none or just 1 company
  - employees CAN belong to 1 company
- Each company has none or just 1 employee
  - companies CAN have 1 employee
Representation: relationship

- Employees **belong** to companies
- Each employee belongs to none or many companies
  - employees **CAN** belong to many companies
- Each company has none or many employees (N)
  - companies **CAN** have many employees
Representation: relationship

- Terminology for cardinalities or multiplicity

One-to-one relation: 1:1

One-to-many relation: 1:N

Many-to-many relation: N:M
Representation: constraints

- Existence or mandatory participation
  - All entity occurrences participate in a relationship, at least, once
  - Employees **must** belong, at least, to 1 company and **just** one
  - Companies **can** have many employees
    - Or none also (optional participation)

```
EMPLOYEE
- ID
- location
- name
- first name
- last name
- telephone (0..N)
- account (1..1)

COMPANY
- cif
- name
- head_office_country
```

Work_for

(1,1)

(0,N)
Representation: constraints

- **Existence or mandatory participation**
  - All entity occurrences participate in a relationship, at least, once
  - Employees **can** belong to no more than one company
  - Companies **must** have employees
    - **At least one**
    - And also many
      - Not very useful in cardinalities ... to many

---

**COMPANY**
- cif
- name
- head_office_country

**EMPLOYEE**
- ID
- location
- name
  - first name
  - last name
- telephone: 0..N
- account: 1..1

Work_for: (0,1) -> (1,N)
Representation: constraints

- **Existence or mandatory participation**
  - All entity occurrences participate in a relationship, at least, once
  - Employees must belong to 1 and just one company
  - Companies must have employees
    - It is very debatable if it is useful or not...

- Employees must belong to 1 and just one company
  - Companies must have employees

- It is very debatable if it is useful or not...
representation: constraints

- No limitations
  - Companies **must** have 8 employees
  - But we will have problems to translate it to the relational model

```sql
EMPLOYEE
- ID
- location
- name
  - first name
  - last name
- telephone (0..N)
- account (1..1)

COMPANY
- cif
- name
- head_office_country
```

- Work_for
  - (1,1)
  - (8,8)
**Representation: constraints**

- **Existence or mandatory participation**
  - The constraint affects only one relationship and one entity
  - Employees **must** belong to 1 and just one company
  - Employees **can** work on one project
  - Companies **can** have employees
  - Projects **can** have employees working for them
Representation: constraints

- Existence or mandatory participation
  - The constraint affects only to one relationship and one entity
    - Employees must belong to 1 and just 1 company
    - Employees must work on 1 and just 1 project
  - Companies can have employees
  - Projects can have employees working for them
Representation: relationships and attributes

- They can belong to relationships
- They will have values only when a client rents a vehicle
Reflexive relationships

(*) A player can only be another player’s mate.
Ternary relationships

- Indicates an action or relation between 3 entities

**Representation from an entity towards every one of the others in an independent way**

<table>
<thead>
<tr>
<th>One professor can</th>
<th>many courses</th>
</tr>
</thead>
<tbody>
<tr>
<td>or cannot teach</td>
<td>many groups</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>One course</th>
<th>many professors</th>
</tr>
</thead>
<tbody>
<tr>
<td>cannot be or can be taught by</td>
<td>many groups</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>In one group, nothing can be taught or it can be related to</th>
<th>many courses</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>many professors</td>
</tr>
</tbody>
</table>

**Diagram:**

- **PROFESSOR**
  - dni
  - city
  - Name
  - o..N

- **GROUP**
  - code
  - numStudents
  - o..N
  - Teaching
  - o..N

- **Course**
  - code
  - name
  - credits
  - o..N
Ternary relationships

- Indicates an action or relation between 3 entities

**Representation from every TWO (pair) entities**

<table>
<thead>
<tr>
<th>One professor</th>
<th>in</th>
<th>1 course</th>
</tr>
</thead>
<tbody>
<tr>
<td>One group</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>One professor</th>
<th>with</th>
<th>many groups</th>
</tr>
</thead>
<tbody>
<tr>
<td>one course</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>One course</th>
<th>in</th>
<th>1 professor</th>
</tr>
</thead>
<tbody>
<tr>
<td>one group</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Entities**

- **PROFESSOR**
  - dni
  - city
  - Name
  - o..N

- **GROUP**
  - code
  - numStudents
  - Teaching
  - o..N

- **Course**
  - code
  - name
  - credits
  - o..N
Ternary relationships

- Terminology

- 1:1:1 relationship
- 1:1:M relationship
- 1:M:M relationship
- M:M:M relationship
Ternary relationships

- Existence constraint
  - The constraint affects only one relationship and one entity
    - Groups must have assigned at least 1 course taught by one professor
    - Courses can be taught by professors in groups
    - Professors can teach courses in groups
### Identification dependency

- The case of “invoice”

<table>
<thead>
<tr>
<th>num</th>
<th>client</th>
<th>date</th>
</tr>
</thead>
<tbody>
<tr>
<td>F001</td>
<td>ANTONIO</td>
<td>02/10/2009</td>
</tr>
<tr>
<td>F002</td>
<td>PEPE</td>
<td>05/10/2009</td>
</tr>
<tr>
<td>F003</td>
<td>ANTONIO</td>
<td>06/10/2009</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>línea</th>
<th>artículo</th>
<th>unidades</th>
<th>precio</th>
</tr>
</thead>
<tbody>
<tr>
<td>L001</td>
<td>pencil</td>
<td>10</td>
<td>0.35</td>
</tr>
<tr>
<td>L002</td>
<td>rubber</td>
<td>3</td>
<td>0.10</td>
</tr>
<tr>
<td>L003</td>
<td>dinA4</td>
<td>100</td>
<td>0.05</td>
</tr>
<tr>
<td>L004</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>line</th>
<th>article</th>
<th>Units</th>
<th>price</th>
</tr>
</thead>
<tbody>
<tr>
<td>L001</td>
<td>sharpener</td>
<td>2</td>
<td>0.15</td>
</tr>
<tr>
<td>L002</td>
<td>pen</td>
<td>300</td>
<td>1.10</td>
</tr>
<tr>
<td>L003</td>
<td>pencil</td>
<td>100</td>
<td>0.25</td>
</tr>
<tr>
<td>L004</td>
<td>Folder</td>
<td>50</td>
<td>0.10</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>num</th>
<th>client</th>
<th>date</th>
</tr>
</thead>
<tbody>
<tr>
<td>F003</td>
<td>ANTONIO</td>
<td>06/10/2009</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>line</th>
<th>article</th>
<th>units</th>
<th>price</th>
</tr>
</thead>
<tbody>
<tr>
<td>L001</td>
<td>pencil</td>
<td>100</td>
<td>0.10</td>
</tr>
</tbody>
</table>
Representation: constraints

- Identification dependency
  - The case of “invoice”

<table>
<thead>
<tr>
<th>num</th>
<th>Client</th>
<th>date</th>
</tr>
</thead>
<tbody>
<tr>
<td>F001</td>
<td>ANTONIO</td>
<td>02/10/2009</td>
</tr>
<tr>
<td>F002</td>
<td>PEPE</td>
<td>05/10/2009</td>
</tr>
<tr>
<td>F003</td>
<td>ANTONIO</td>
<td>06/10/2009</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>line</th>
<th>Article</th>
<th>Units</th>
<th>price</th>
</tr>
</thead>
<tbody>
<tr>
<td>L001</td>
<td>sharpener</td>
<td>2</td>
<td>0.15</td>
</tr>
<tr>
<td></td>
<td>pen</td>
<td>300</td>
<td>1.10</td>
</tr>
<tr>
<td>L002</td>
<td>pencil</td>
<td>100</td>
<td>0.25</td>
</tr>
<tr>
<td>L003</td>
<td>Folder</td>
<td>50</td>
<td>0.10</td>
</tr>
<tr>
<td>L001</td>
<td>Pencil</td>
<td>100</td>
<td>0.10</td>
</tr>
<tr>
<td>L001</td>
<td>pencil</td>
<td>100</td>
<td>0.05</td>
</tr>
<tr>
<td>L002</td>
<td>rubber</td>
<td></td>
<td></td>
</tr>
<tr>
<td>L003</td>
<td>dinA4</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Not enough information in the detail lines
Identification dependency
- An entity needs to be identified also with the key attribute of the other
- Only in 1:N

<table>
<thead>
<tr>
<th>num</th>
<th>cliente</th>
<th>fecha</th>
</tr>
</thead>
<tbody>
<tr>
<td>F001</td>
<td>ANTONIO</td>
<td>02/10/2009</td>
</tr>
<tr>
<td>F002</td>
<td>PEPE</td>
<td>05/10/2009</td>
</tr>
<tr>
<td>F003</td>
<td>ANTONIO</td>
<td>06/10/2009</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>factura</th>
<th>linea</th>
<th>artículo</th>
<th>unidades</th>
<th>precio</th>
</tr>
</thead>
<tbody>
<tr>
<td>F002</td>
<td>L001</td>
<td>sacaps</td>
<td>2</td>
<td>0.15</td>
</tr>
<tr>
<td>F002</td>
<td>L002</td>
<td>pilot</td>
<td>300</td>
<td>1.10</td>
</tr>
<tr>
<td>F002</td>
<td>L003</td>
<td>lápiz nº2</td>
<td>100</td>
<td>0.25</td>
</tr>
<tr>
<td>F002</td>
<td>L004</td>
<td>carpeta</td>
<td>50</td>
<td>0.10</td>
</tr>
<tr>
<td>F003</td>
<td>L001</td>
<td>lápiz nº2</td>
<td>100</td>
<td>0.10</td>
</tr>
<tr>
<td>F001</td>
<td>L001</td>
<td>lápiz nº2</td>
<td>10</td>
<td>0.35</td>
</tr>
<tr>
<td>F001</td>
<td>L002</td>
<td>borrador</td>
<td>3</td>
<td>0.10</td>
</tr>
<tr>
<td>F001</td>
<td>L003</td>
<td>dinA4</td>
<td>100</td>
<td>0.05</td>
</tr>
</tbody>
</table>
“Complex” entities

- The supervisor only supervises projects that are proposed by a company.
Representation: aggregation

- Possible representations
  - And much more
    - Maybe they are useful for specific cases
Conclusions

- Differences between E-R and EER (Extended Entity-Relationship)?
  - Reaching the object-oriented approach

- Structures
  - Entities, relationships

- The first definition provided by P. Chen has had several extensions and representations.
  - Extended E-R (EER)
    - More relationships (e.g., Generalization)

- Non DBMS for the EER model
Conclusions

• It implements the common abstraction mechanisms
  • classification, aggregation (general), generalization
  • Domain constraints (not being used yet), identification and correspondence between classes

• Many notations
  • Not exactly a standard
  • Our notation, even less standard

• Many CASE tools
  • Although they are just “picture tables"
  • Some of them provide “translation" to relational model
References

• Important

• Others
  • Fundamentos de sistemas de bases de datos: Elmasri, Ramez, ed.5,
    • (pág. 51) Cap. 3, Modelado de datos con el modelo Entidad-Relación (ER)
    • (pág. 89)Cap. 4, El modelo Entidad-Relación mejorado (EER)
  • http://personales.unican.es/zorrillm/BasesDatos/02 - Modelos de datos ER-UML-relacional.pdf
Logical Design

Relational model:
entity, generalization and binary relationship

34014 - DATA BASE DESIGN (2013-14)
Degree in Computer Engineering
University of Alicante
logical design

Transformation from EER to relational
relational model

- Candidate key (or superkey)
  - Primary Key (PK) No duplicates
  - Alternate Key (AltK) No null values

- THERE IS ONLY ONE PK BUT SEVERAL AltK

- Foreign key

- Non-null values
one entity

A(a0, a1, a2, a3)
PK: a0
NNV: a2
one entity
compound primary key

A( a0, a1, a2, a3)
PK: (a0,a1)
NNV.: a2
one entity
with multivalued attributes

A( ao, a1, a2)
PK: ao
NNV: a2

M(a0,a3)
PK:(ao, a3)
FK: ao→A
We cannot capture that the generalization is TOTAL in the relational model (tables) only \textit{Partial} and \textit{Overlapping} generalizations are well-represented.
binary relationship 1:m

A( a0, a1, rB)
PK: a0
FK: rB → B

B( b0, b1)
PK: b0
binary relationship 1:m
with existence constraint

A( a0, a1, rB)
PK: a0
FK: rB \rightarrow B
NNV: rB

B( b0, b1)
PK: b0
binary relationship 1:m
with existence constraint

A( a0, a1, rB)
PK: a0
FK: rB \rightarrow B

B( b0, b1)
PK: b0

existence constraint is lost from B to R
binary relationship 1:m
when there is a compound primary key

A( a0, a1, rB0, rB1)
PK: a0
FK: (rB0,rB1) \rightarrow B

B( b0, b1,b2)
PK: (b0,b1)
binary relationship 1:m
with attribute

- A( a0, a1, rB, r1 *)
  PK: a0
- B( b0, b1)
  PK: b0
  FK: rB \rightarrow B

- A( a0, a1)
  PK: a0
- B( b0, b1)
  PK: b0
  FK: rA \rightarrow A
  FK: rB \rightarrow B
- R(rA, rB, r1)
  PK: rA
  NNV: rB

* There will be values of r1 if values of rB exist
binary relationship m:m

A(a0, a1)
PK: a0

B(b0, b1)
PK: b0

R(rA, rB, r1)
PK: (rA, rB)
FK: rA \rightarrow A
FK: rB \rightarrow B
binary relationship m:m
with multivalued attribute

A(a0, a1)
PK: a0
B(b0, b1)
PK: b0

R( rA, rB, r1)
PK: (rA, rB)
FK: rA \rightarrow A
FK: rB \rightarrow B

R2( rA, rB, r2)
PK: (rA, rB, r2)
FK: (rA, rB) \rightarrow R
binary relationship 1:1

A(a0, a1)  
PK: a0

B(b0, b1)  
PK: b0

R( rA, rB)  
PK: rA  
AltK: rB  
FK: rA \rightarrow A  
FK: rB \rightarrow B
binary relationship 1:1
with a existence constraint

A( a0, a1)
PK: a0

B( b0, b1, rA)
PK: b0
AltK: rA
FK: rA → A
binary relationship 1:1
with two existence constraints

R( a0, a1, b0, b1)
PK: a0
AltK: b0
Logical Design

Relational model:
reflexive and ternary relationships

34014 - DATA BASE DESIGN (2013-14)
Degree in Computer Engineering
University of Alicante
logical design

Transformation from EER to relational
relational model

• Candidate key (or superkey)
  • Primary Key (PK) No duplicates
  • Alternate Key (AltK) No null values

• THERE IS ONLY ONE PK BUT SEVERAL AltK

• Foreign key

• Non-null values
reflexive m:m

A(a0, a1)
PK: a0

R( rA1, rA2)
PK: (rA1, rA2)
FK: rA1 \rightarrow A
FK: rA2 \rightarrow A
reflexive 1:m

\[ A(a_0, a_1, rA) \]
PK: \( a_0 \)
FK: \( rA_1 \rightarrow A \)
reflexive 1:1

A(a0, a1)
PK: a0

R( rA1, rA2)
PK: rA1
AltK: rA2
FK: rA1 \rightarrow A
FK: rA2 \rightarrow A
ternary 1:1:1

A
- a0
- c1
- c2

PK: a0

B
- b0
- b1
- b2

PK: b0

R
- rA
- rB
- rC

PK: (rA, rB)
AltK: (rA, rC)
AltK: (rB, rC)
FK: rA \rightarrow A
FK: rB \rightarrow B
FK: rC \rightarrow C

C
- c0
- c1
- c2

PK: c0

A(a0, a1)

B(b0, b1)
ternary 1:1:m

A
a0
c1
c2

B
b0
b1
b2

C
c0
c1
c2

A(a0, a1)
PK: a0

B(b0, b1)
PK: b0

C(c0, c1)
PK: c0

R( rA, rB, rC)
PK: (rA, rB)
AltK: (rA, rC)
FK: rA \rightarrow A
FK: rB \rightarrow B
FK: rC \rightarrow C
ternary 1:m:m

A(a0, a1)
PK: a0

B(b0, b1)
PK: b0

C(co, c1)
PK: co

R(rA, rB, rC)
PK: (rA, rB)
FK: rA \rightarrow A
FK: rB \rightarrow B
FK: rC \rightarrow C
NNV: rC
ternary m:m:m

A(a0, a1)
PK: a0

B(b0, b1)
PK: b0

C(c0, c1)
PK: c0

R(rA, rB, rC)
PK: (rA, rB)
FK: rA → A
FK: rB → B
FK: rC → C
NNV: rC

Existence constraint from B to R is lost
ternary m:m:m

- **A**
  - a0
  - c1
  - c2

- **B**
  - b0
  - b1
  - b2

- **C**
  - c0
  - c1
  - c2

- **R**
  - rA, rB, rC

**A(a0, a1)**
PK: a0

**B(b0, b1)**
PK: b0

**C(c0, c1)**
PK: c0

**R( rA, rB, rC)**
PK: (rA, rB, rC)
FK: rA → A
FK: rB → B
FK: rC → C
Logical Design

Relational model:
identification dependency and aggregation

34014 - DATA BASE DESIGN (2013-14)
Degree in Computer Engineering
University of Alicante
logical design

Transformation from EER to relational
relational model

- Candidate key (or superkey)
  - Primary Key (PK) No duplicates
  - Alternate Key (AltK) No null values

- THERE IS ONLY ONE PK BUT SEVERAL AltK

- Foreign key

- Non-null values
identification dependency

B( b0, b1)  
PK: b0

A( a0, a1, rB)  
PK: (a0, rB)  
FK: rB → B
aggregation

A(a0, a1) FK: a0
B(b0, b1) FK: b0
C(c0, c1) FK: c0

R1( rA, rB)
PK: (rA, rB)
FK: rA -> A
FK: rB -> B

R2( rA, rB, rC)
PK: (rA, rB)
AltK: rC
FK: (rA, rB) -> R1
FK: rC -> C
aggregation

A
PK: a0

B
PK: b0

C
PK: c0

R1
PK:

R2

0..N

1..1

0..1

A(a0, a1)
B(b0, b1)
C(c0, c1)

R1( rA, rB, rC)
PK: (rA, rB)
AltK: rC
FK: rA → A
FK: rB → B
FK: rC → C
aggregation

A(a0, a1)
PK: a0

B(b0,b1)
PK: b0

C(co,c1)
PK: co

R1( rA, rB, rC)
PK: (rA, rB)
FK: rA → A
FK: rB → B
FK: rC → C
aggregation

A(a0, a1) PK: a0
B(b0, b1) PK: b0
C(c0, c1) PK: c0

A0
A1

R1
0..1
0..1

R2
0..1
0..N

C
C0
C1

R1( rA, rB, rC)
PK: rA
AltK: rB
FK: rA → A
FK: rB → B
FK: rC → C
aggregation

A(a0, a1)
PK: a0

C(c0, c1)
PK: c0

B(b0, b1, rA, rC)
PK: b0
FK: rA \rightarrow A
FK: rC \rightarrow C

Column rC only has value if rA has value

Option 1
aggregation

A(a0, a1)
PK: a0

B(b0, b1)
PK: b0

C(c0, c1)
PK: co

R1( rB, rA, rC)
PK: rB
FK: rB \rightarrow B
FK: rA \rightarrow A
FK: rC \rightarrow C
NNV: rA

Option 2
aggregation

A(a0, a1)
PK: a0
C(c0, c1)
PK: c0
B(b0, rA, rC)
PK: b0
FK: rA \rightarrow A
FK: rC \rightarrow C
NNV: rA
Conceptual design exercise
Hiking association

A hiking local association wants to build a database to store information about a group of hikers from whom we know the member ID (which identifies each member), their national ID number (e.g., NIF), name, address, phone number and email.

The association has a calendar of outings in such a way that for given a date, only an outing can be planned. In each outing, a unique route is done. Also, the outing can be guided by several instructors. Hikers can go to the outing in their own vehicle, by bus or other means of transport. When the weather forecast is known for a certain outing, the expected average temperature is stored, as well as if rain is expected or not and the wind speed. Each outing has two prices, one for those using their own vehicle and another for those using public transport.

There is information about many routes. These routes are identified by means of a code, and also we can know their name, kilometres of the route and level of difficulty. Furthermore, we need to know which routes are circular (the starting and final point is the same) and which of them take more than one day (being required to know how many nights must be spent).

Association counts on instructors. We know their ID number, name, address and phone number, as well as their salary. There are several instructor categories: speleologists, climbers, birdwatchers and botanists. All instructors in each category have the same salary. From each instructor, we must know his/her category (only one). The following information is required:

- Speleologists: we must know the cave which they visited and if they are experts in underwater caves.
- Climbers: we must know their personal records.
- Birdwatchers and botanists: we must know the courses they have taken.

Hikers can enroll in several outings and when the booking of the outing is done, they have to make a deposit (which must be known) and also we must know if they will use their own vehicle or not.

The association collects pictures of each outing. These pictures are sent by the hikers. For each outing, the pictures are numbered consecutively. Several pictures are selected to be used at the end of each year to make a memories album.

The association has material (such as sleeping bags, torchlights, sticks, air beds, etc.) which can be borrowed by hikers. Each material has a code and the date is known in which it was acquired and its condition. A hiker books the material for certain dates (independently of the outings). One hiker can only book one material at the most per date, and in one date a material can only be booked by a hiker.

About the material, the weight and the material from which the stick is made from must be known, and also the minimum temperature allowed by the sleeping bags.

Note: although two outings are for the same route, they do not necessarily have the same cost (since it can depend on the instructors’ salary, transport, etc.)
A transport company wants to build a DB to store some information. You must design the EER conceptual schema that best fits with the following requirements. The company has a set of trucks from which the following information is known: number plate (which identifies the truck), brand, purchase date, and last service date. These trucks are loaded by using mechanical ramps. These ramps are identified by a code and their brand and the purchase date are known. The company knows which employees are authorized to load trucks. According to the policy of the company, several employees can work in the same ramp, and an employee can work in several ramps. However, a truck for each ramp in which it is loaded always has the same employee, and an employee in one ramp can only load the same truck. We know the DNI, name and birthday of the employees. We must know the country of the trucks with a foreign plate, as well as the service tracking with its corresponding dates, total cost and the garage. We know the name (which identifies the garage) and phone number of every garage. A service is done to a truck at the most once a month. Services are evaluated by employees that are designated by the company; an employee can evaluate several services and one service can be evaluated by several employees. There are employees that validate the evaluation of the service that are known as supervisors. The company wants to know the employees that carry out each service (with marks from 1 to 10), as well as the corresponding supervisor and the date of the validation. An evaluation only has one validation. For trucks with national plate, we must know the drivers assigned to the truck for each day. A driver is assigned only to one truck each day. Drivers may drive different trucks on different days. We know the license number of the driver, as well as the name and a list of kinds of license the driver has and the day the driver obtained each of them. Finally, the company has car parks to leave the trucks during the mandatory resting month. Car parks are identified by a number and a city. There is a list of available places of parkings per month. Trucks are always assigned to the same car park in the same month. Obviously, two trucks cannot be parked in the same place at the same time.
Hotel case study

Exercise 1

34014 - DATA BASE DESIGN (2013-14)
Degree in Computer Engineering
University of Alicante
The hotel scenario

- A hotel hired us to design a database to store all the required information for the business
- Design the database taking into account requirements stated in the following slides
Requirements (1)

- The hotel has many rooms, which are identified by a number
- Each of these rooms have one of the following categories:
  - Single
  - Double
  - Double with balcony
  - Suite
Requirements (2)

- Rooms of a specific category have the same description which must be known, as well as the minimum and maximum surface area corresponding to each room category.
Requirements (3)

- Each room category must be known
Requirements (and 4)

- Room prices depend on its category and season in which the booking is done
- There are three seasons
  - Low, middle, high
- Every category room has the same price per season
- There are four prices to be considered
  - Only accommodation
  - Breakfast
  - Half board
  - Full board
Hotel case study

Exercise 2

34014 - DATA BASE DESIGN (2013-14)
Degree in Computer Engineering
University of Alicante
Requirements (1)

- The hotel offers a set of activities, identified by a code, which have a description.
- Activities are performed on different days and hours, but one activity performed on a specific day is only performed once (in one hour).
- Two activities performed at the same time are not offered, i.e., in one hour of one day, only one activity can be performed.
Requirements (2)

- Activities can belong to three different kinds: only for children, only for adults, for everyone

- A list of alternatives must be available in such a way that an activity can be replaced with other of the same kind
  - Only the alternative activities must be known, the order is not relevant
Requirements (and 3)

- Every date can only belong to one season that must be known
Solution of exercise 1
Hotel case study

Exercise 3

34014 - DATA BASE DESIGN (2013-14)
Degree in Computer Engineering
University of Alicante
requirements (1)

- The hotel has customers of which we must know their ID number, name and phone number. ID number identifies each customer.
- We may also know the customer’s city, province and country.
requirements (2)

- Customers book rooms
- Bookings are identified by a code and date of booking
- Each booking code can be related to several rooms on several dates
- Each time that a room is booked for a specific date, it may be known if an extra bed is required and, also, if any meals are required for this day in this room
requirements (3)

- Meals can be breakfast, half board, and full board
- Flexibility in booking a meal is required
  - E.G., a customer may book a room for three days, having half board for two days and only breakfast the last one
- If it is known that a room is booked for a specific day, then the corresponding booking code must be known
requirements (4)

- Bookings can be done by travel agencies, by Internet, by phone or in the hotel
  - Bookings done by travel agencies require knowing ID number of the travel agency as well as the name and phone number, employee at the desk in charge of the booking
  - Bookings done by phone or in the hotel require knowing the employee at the desk in charge of the booking
  - Bookings done by Internet require knowing a booking reference (several booking codes can have the same booking reference)
requirements (5)

- A room must only be booked for the same day for one customer
- Invoicing is not considered in our scenario, but we want to know the services used by a customer who stays in the hotel
  - Price of the service is always the same regardless of the season and the room type
- We need to know the services used per day in each booked room in order to include them in the invoice
  - We need to know the quantity of services used
requirements (6)

- There are other services provided by the hotel, regardless of the customer staying there
  - These services are performed by a group of employees: a service can be performed by more than one employee, and an employee can perform several services
  - From each service, we know an identity code, some observations and a price (mandatory)
  - When a customer wants some of these services, a date must be arranged with one of the employees, pointing out the required service
  - Arrangements consist of a specific day and a “moment”
    - A day is divided in six “moments”, numbered from 1 to 6
    - There must be a control of an employee not making an arrangement with more than one customer at a time
requirements (and 7)

- ID number, name, address, city, phone number and education of the hotel staff are known
- Hotel staff work on reception, in entertainment, in the restaurant, cleaning/maintenance or doing service tasks
- Each employee only works in one of these categories
- The hotel must have ONLY one cleaning/maintenance employee available each day
- For each proposed activity, we must know who the entertainment employee in charge of it is
- CIF, name, address and phone are known from the travel agencies
solution: part 2
Computer booking scenario

Exercise

34014 - DATA BASE DESIGN (2013-14)
Degree in Computer Engineering
University of Alicante
Computer booking scenario

- Let's assume we already have three entities
  - STUDENT (id, name)
  - TURN (code)
  - COMPUTER (code, purchase-date, model)

- Not overlaying groups
- Imagine all the computers in the same room
- We need to capture two issues related to booking turns with computers in practical lessons in certain subjects
  - For booking the student must book both: turn and computer (not only turn or only computer)
First issue

- One student can book turns to make assignments with computers. He or she can book several turns and several computers. We are interested in knowing, in any booked turn, the computer that is booked. There are two constraints: a student in a turn can only book one computer, and a computer in a turn cannot be booked by more than one student.
Second issue

- A student can book a turn and a computer to do an exam. In this case, **booking more than one turn is not allowed**, neither booking more than one **computer**. Two students in the same turn cannot book the same computer.
Working with Oracle
Data Type

- Char(n)
- Varchar(n)
- Number(p,s)  \[ \text{Integer} = \text{number}(38) \]
- Date
Functions for ORACLE DATE data type

- **TO_CHAR(date [, format])**
  Converts a DATE type date to a VARCHAR2 value in the format specified in “format”.

- **TO_DATE(string [, format])**
  Converts the character string “string” of CHAR type to a DATE type value in the format specified in “format”.

- **SYSDATE**
  Returns the current system date.

- **ADD_MONTHS(date,n)**
  Returns the date specified by n months more.

- **MONTHS_BETWEEN(date1,date2)**
  Returns the months between date1 and date2.
For the functions TO_CHAR and TO_DATE

<table>
<thead>
<tr>
<th>ELEMENT</th>
<th>MEANING</th>
</tr>
</thead>
<tbody>
<tr>
<td>- / ' . ; : 'texto'</td>
<td>Punctuation marks and fixed text that is reproduced in the result</td>
</tr>
<tr>
<td>D</td>
<td>Day of the week (1-7)</td>
</tr>
<tr>
<td>DAY</td>
<td>Name day of the week</td>
</tr>
<tr>
<td>DD</td>
<td>Day of the month (1-31)</td>
</tr>
<tr>
<td>DDD</td>
<td>Day of the year (1-366)</td>
</tr>
<tr>
<td>DY</td>
<td>Reduced name day of the week (LUN, MAR, MIE...)</td>
</tr>
<tr>
<td>MM</td>
<td>Month (1-12)</td>
</tr>
<tr>
<td>MON</td>
<td>Reduced name of the month</td>
</tr>
<tr>
<td>MONTH</td>
<td>Complete name of the month</td>
</tr>
<tr>
<td>Q</td>
<td>Quarter of the year (1-4)</td>
</tr>
<tr>
<td>YYYYY</td>
<td>4-digit year</td>
</tr>
<tr>
<td>Y,YYY</td>
<td>Year with thousand point</td>
</tr>
<tr>
<td>YY</td>
<td>2-digit year</td>
</tr>
</tbody>
</table>
Examples

• Teachers who entered the first half of any year.
  
  ```sql
  select * from profesores
  where to_char(ingreso,'MM') <= 6
  ```

• Teachers who have entered today.
  
  ```sql
  select * from profesores where ingreso = sysdate
  ```

• Teachers who have entered the first quarter of 2010.
  
  ```sql
  select * from profesores
  where to_char(ingreso,'MM') <= 3 and to_char(ingreso,'YYYY') = 2010
  ```
What have you already seen in FBD?
SELECT [ DISTINCT ] columnsList
FROM tableList
[ WHERE row condition ]
[ GROUP BY columnsList for which you want to group ]
[ HAVING group condition ]
[ ORDER BY columnsList [ ASC | DESC ] ]

[ ] means that is optional
JOIN review

– When we refer to **JOIN**, it means that we combine rows of two or more tables in a single query.
– Such rows are concatenated following a specific criterion.
– Depending on the condition that is used there are **different types of join**.
JOIN Review

• If the tables are related without condition, Oracle returns the Cartesian product of those tables.

• When a condition exists and the resulting rows are only those ones that satisfy the condition, we talk about **inner join (or simple join)**. This is the most common way to link the tables.

• Sometimes it is useful to extend the results of these joins, and then work with **outer joins**. By working with outer joins, Oracle will return all rows that satisfy the join condition over those rows of the table marked (LEFT, RIGHT) for which we have found no rows that have made the condition of the join.

Depending on which of the tables we want to extend the result, we distinguish:

- extend the result with the rows of the table that appears first in the FROM, we will use **LEFT [OUTER] JOIN** in the FROM clause.
- extend the result with the rows of the table that appears second in the FROM, we will use **RIGHT [OUTER] JOIN** in the FROM clause.
- extend the result to rows from both tables, you must use **FULL [OUTER] JOIN** in the FROM clause.
Let’s see some examples with the following tables: USUARIO and PEDIDO.

- **usuario** (email, nombre, apellidos ... )  
  Primary key: email

- **pedido** (numPedido, usuario, fecha date )  
  Primary Key: numpedido  
  Foreing Key: usuario → usuario
### JOIN Review

**select email from usuario**

<table>
<thead>
<tr>
<th>EMAIL</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="mailto:acdlv@bitoben.mus.es">acdlv@bitoben.mus.es</a></td>
</tr>
<tr>
<td><a href="mailto:acg@hotmail.com">acg@hotmail.com</a></td>
</tr>
<tr>
<td><a href="mailto:acl@dlsiu.us.es">acl@dlsiu.us.es</a></td>
</tr>
<tr>
<td><a href="mailto:acn@hotmail.com">acn@hotmail.com</a></td>
</tr>
<tr>
<td><a href="mailto:adf@lolipop.com">adf@lolipop.com</a></td>
</tr>
<tr>
<td><a href="mailto:adlmm@ua.es">adlmm@ua.es</a></td>
</tr>
<tr>
<td><a href="mailto:adrm@dlsiu.us.es">adrm@dlsiu.us.es</a></td>
</tr>
<tr>
<td><a href="mailto:aeb@colegas.com">aeb@colegas.com</a></td>
</tr>
<tr>
<td><a href="mailto:afg@colegas.com">afg@colegas.com</a></td>
</tr>
<tr>
<td><a href="mailto:agg@gmail.com">agg@gmail.com</a></td>
</tr>
<tr>
<td><a href="mailto:agl@dlsiu.us.es">agl@dlsiu.us.es</a></td>
</tr>
<tr>
<td><a href="mailto:agt@lamail.ar">agt@lamail.ar</a></td>
</tr>
<tr>
<td><a href="mailto:alm@lolipop.com">alm@lolipop.com</a></td>
</tr>
<tr>
<td><a href="mailto:ama@lolipop.com">ama@lolipop.com</a></td>
</tr>
</tbody>
</table>

**27 selected rows**

---

**Select usuario from pedido**

<table>
<thead>
<tr>
<th>USUARIO</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="mailto:deg@lamail.ar">deg@lamail.ar</a></td>
</tr>
<tr>
<td><a href="mailto:jccf@eps.ua.es">jccf@eps.ua.es</a></td>
</tr>
<tr>
<td><a href="mailto:svv@colegas.com">svv@colegas.com</a></td>
</tr>
<tr>
<td><a href="mailto:rbc@bitoben.mus.es">rbc@bitoben.mus.es</a></td>
</tr>
<tr>
<td><a href="mailto:mav@colegas.com">mav@colegas.com</a></td>
</tr>
<tr>
<td><a href="mailto:jme@lolipop.com">jme@lolipop.com</a></td>
</tr>
<tr>
<td><a href="mailto:pge@colegas.com">pge@colegas.com</a></td>
</tr>
<tr>
<td><a href="mailto:bmm@agwab.com">bmm@agwab.com</a></td>
</tr>
<tr>
<td><a href="mailto:amd@colegas.com">amd@colegas.com</a></td>
</tr>
<tr>
<td><a href="mailto:jmem@colegas.com">jmem@colegas.com</a></td>
</tr>
<tr>
<td><a href="mailto:mps@agwab.com">mps@agwab.com</a></td>
</tr>
<tr>
<td><a href="mailto:adlmm@ua.es">adlmm@ua.es</a></td>
</tr>
<tr>
<td><a href="mailto:hrdcj@colegas.com">hrdcj@colegas.com</a></td>
</tr>
<tr>
<td><a href="mailto:acl@dlsiu.us.es">acl@dlsiu.us.es</a></td>
</tr>
</tbody>
</table>

**157 selected rows**
select email, nombre, numpedido from usuario, pedido where email=usuario

<table>
<thead>
<tr>
<th>EMAIL</th>
<th>NOMBRE</th>
<th>NUMPEDIDO</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="mailto:amd@colegas.com">amd@colegas.com</a></td>
<td>ALEJANDRA</td>
<td>1</td>
</tr>
<tr>
<td><a href="mailto:rpv@hotmail.com">rpv@hotmail.com</a></td>
<td>RAMIRO</td>
<td>2</td>
</tr>
<tr>
<td><a href="mailto:jmem@colegas.com">jmem@colegas.com</a></td>
<td>JUAN MANUEL</td>
<td>7</td>
</tr>
<tr>
<td><a href="mailto:jptg@colegas.com">jptg@colegas.com</a></td>
<td>JUAN PABLO</td>
<td>9</td>
</tr>
<tr>
<td><a href="mailto:jccf@eps.ua.es">jccf@eps.ua.es</a></td>
<td>JUAN CARLOS</td>
<td>11</td>
</tr>
<tr>
<td><a href="mailto:mraj@colegas.com">mraj@colegas.com</a></td>
<td>MARIA ROSA</td>
<td>14</td>
</tr>
<tr>
<td><a href="mailto:rbc@bitoben.mus.es">rbc@bitoben.mus.es</a></td>
<td>RUTH</td>
<td>15</td>
</tr>
</tbody>
</table>

27 Selected rows

select email, nombre, numpedido from usuario left join pedido on email=usuario

<table>
<thead>
<tr>
<th>EMAIL</th>
<th>NOMBRE</th>
<th>NUMPEDIDO</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="mailto:jmem@colegas.com">jmem@colegas.com</a></td>
<td>JUAN MANUEL</td>
<td>7</td>
</tr>
<tr>
<td><a href="mailto:jptg@colegas.com">jptg@colegas.com</a></td>
<td>JUAN PABLO</td>
<td>9</td>
</tr>
<tr>
<td><a href="mailto:jccf@eps.ua.es">jccf@eps.ua.es</a></td>
<td>JUAN CARLOS</td>
<td>11</td>
</tr>
<tr>
<td><a href="mailto:mraj@colegas.com">mraj@colegas.com</a></td>
<td>MARIA ROSA</td>
<td>14</td>
</tr>
<tr>
<td><a href="mailto:rbc@bitoben.mus.es">rbc@bitoben.mus.es</a></td>
<td>RUTH</td>
<td>15</td>
</tr>
<tr>
<td><a href="mailto:deg@lamail.ar">deg@lamail.ar</a></td>
<td>DAVID</td>
<td>18</td>
</tr>
<tr>
<td><a href="mailto:jmf1@colegas.com">jmf1@colegas.com</a></td>
<td>JAVIER</td>
<td>19</td>
</tr>
<tr>
<td><a href="mailto:rpg@colegas.com">rpg@colegas.com</a></td>
<td>RAUL</td>
<td>20</td>
</tr>
<tr>
<td><a href="mailto:gvs@colegas.com">gvs@colegas.com</a></td>
<td>GUILLERMO</td>
<td>21</td>
</tr>
</tbody>
</table>

158 selected rows
JOIN Review

select email, nombre, numpedido
from usuario, pedido
where email=usuario

right join pedido
on email=usuario

EMAIL               NOMBRE       NUMPEDIDO
------------------ ----------- --------------
amd@colegas.com    ALEJANDRA    1
rpv@hotmail.com     RAMIRO       2
jmem@colegas.com    JUAN MANUEL  7
jptg@colegas.com    JUAN PABLO   9
jccf@eps.ua.es      JUAN CARLOS  11
mrraj@colegas.com   MARIA ROSA   14
rbc@bitoben.mus.es  RUTH         15
.
.
.

27 Selected rows

27 selected rows
## COUNT review

**EMPLEADO (DNI, NOMBRE, ESPECIALIDAD)**  
Primary key: DNI

<table>
<thead>
<tr>
<th>DNI</th>
<th>NOMBRE</th>
<th>ESPECIALIDAD</th>
</tr>
</thead>
<tbody>
<tr>
<td>11111111A</td>
<td>Juan Martínez</td>
<td>1</td>
</tr>
<tr>
<td>22222222B</td>
<td>María Pérez</td>
<td></td>
</tr>
<tr>
<td>33333333C</td>
<td>Ana Escudero</td>
<td>1</td>
</tr>
<tr>
<td>44444444D</td>
<td>Pedro Pérez</td>
<td>2</td>
</tr>
<tr>
<td>55555555M</td>
<td>Arturo Álvarez</td>
<td>3</td>
</tr>
<tr>
<td>66666666L</td>
<td>Luisa Olmo</td>
<td>3</td>
</tr>
</tbody>
</table>

- **SELECT count(*)**  
  FROM empleado  
  
  $\text{count(*)}$  
  
  6

- **SELECT count(dni)**  
  FROM empleado  
  
  $\text{count(dni)}$  
  
  6

- **SELECT count(especialidad)**  
  FROM empleado  
  
  $\text{count(especialidad)}$  
  
  5

- **SELECT count(distinct especialidad)**  
  FROM empleado  
  
  $\text{count(distinct especialidad)}$  
  
  3

- **SELECT count(distinct dni)**  
  FROM empleado  
  
  $\text{count(distinct dni)}$  
  
  6
COUNT and SUM are not the same!!!

<table>
<thead>
<tr>
<th>DNI</th>
<th>NOMBRE</th>
<th>ESPECIALIDAD</th>
</tr>
</thead>
<tbody>
<tr>
<td>11111111A</td>
<td>Juan Martínez</td>
<td>1</td>
</tr>
<tr>
<td>22222222B</td>
<td>María Pérez</td>
<td></td>
</tr>
<tr>
<td>33333333C</td>
<td>Ana Escudero</td>
<td>1</td>
</tr>
<tr>
<td>44444444D</td>
<td>Pedro Pérez</td>
<td>2</td>
</tr>
<tr>
<td>55555555M</td>
<td>Arturo Álvarez</td>
<td>3</td>
</tr>
<tr>
<td>66666666L</td>
<td>Luisa Olmo</td>
<td>3</td>
</tr>
</tbody>
</table>

SELECT count(especialidad) FROM empleado

5

SELECT sum(especialidad) FROM empleado

10
### GROUP BY – HAVING review

```sql
SELECT especialidad, count(*)
FROM empleado
GROUP BY especialidad
HAVING count(*) >= 2
```

<table>
<thead>
<tr>
<th>especialidad</th>
<th>count(*)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
</tr>
</tbody>
</table>

---

<table>
<thead>
<tr>
<th>DNI</th>
<th>NOMBRE</th>
<th>ESPECIALIDAD</th>
</tr>
</thead>
<tbody>
<tr>
<td>11111111A</td>
<td>Juan Martínez</td>
<td>1</td>
</tr>
<tr>
<td>22222222B</td>
<td>María Pérez</td>
<td>2</td>
</tr>
<tr>
<td>33333333C</td>
<td>Ana Escudero</td>
<td>1</td>
</tr>
<tr>
<td>44444444D</td>
<td>Pedro Pérez</td>
<td>2</td>
</tr>
<tr>
<td>55555555M</td>
<td>Arturo Álvarez</td>
<td>3</td>
</tr>
<tr>
<td>66666666L</td>
<td>Luisa Olmo</td>
<td>3</td>
</tr>
</tbody>
</table>
We strongly suggest you to review the materials of Fundamentals of Databases (subqueries, operators, ...)
Revision of the SELECT statement using the TiendaOnLine DB

<table>
<thead>
<tr>
<th>Table</th>
<th>Columns</th>
<th>Type</th>
<th>PK/FK</th>
</tr>
</thead>
<tbody>
<tr>
<td>provincia</td>
<td>codp, nombre</td>
<td>varchar(2), varchar(25)</td>
<td>PK</td>
</tr>
<tr>
<td>localidad</td>
<td>codm, pueblo, provincia</td>
<td>varchar(4), varchar(50), varchar(2)</td>
<td>PK, FK</td>
</tr>
<tr>
<td>articulo</td>
<td>cod, nombre, pvp, marca, imagen, urlImagen, especificaciones</td>
<td>varchar(7), varchar(45), decimal(7,2), varchar(15), blob, varchar(100), text</td>
<td>PK, FK</td>
</tr>
<tr>
<td>usuario</td>
<td>email, nombre, apellidos, dni, telefono, calle, calle2, codpos, pueblo, provincia, nacido</td>
<td>varchar(50), varchar(35), varchar(55), varchar(12), varchar(45), varchar(45), varchar(5), varchar(4), varchar(2), date</td>
<td>PK, FK</td>
</tr>
<tr>
<td>direnvio</td>
<td>email, calle, calle2, codpos, pueblo, provincia</td>
<td>varchar(50), varchar(45), varchar(45), varchar(5), varchar(4), varchar(2)</td>
<td>PK, FK</td>
</tr>
<tr>
<td>memoria</td>
<td>cod, tipo</td>
<td>varchar(30), varchar(15)</td>
<td>PK, FK</td>
</tr>
<tr>
<td>objetivo</td>
<td>cod, tipo, montura, focal, apertura, especiales</td>
<td>varchar(7), varchar(15), varchar(15), varchar(10), varchar(10), varchar(35)</td>
<td>PK, FK</td>
</tr>
<tr>
<td>pack</td>
<td>cod, nombre, empresa, logo</td>
<td>varchar(7), varchar(45), varchar(60), blob</td>
<td>FK</td>
</tr>
<tr>
<td>plienea</td>
<td>pack, articulo</td>
<td>varchar(7), varchar(7)</td>
<td>FK</td>
</tr>
<tr>
<td>stock</td>
<td>articulo, disponible, entrega</td>
<td>varchar(7), int(11), varchar(35)</td>
<td>PK</td>
</tr>
<tr>
<td>cesta</td>
<td>articulo, usuario, fecha</td>
<td>varchar(7), varchar(50), datetime</td>
<td>PK, FK</td>
</tr>
<tr>
<td>pedido</td>
<td>numPedido, usuario, fecha</td>
<td>int(11), varchar(50), datetime</td>
<td>PK, FK</td>
</tr>
<tr>
<td>lineped</td>
<td>numPedido, linea, articulo, precio, cantidad</td>
<td>int(11), int(11), varchar(7), decimal(9,2), int(11)</td>
<td>PK, FK, FK</td>
</tr>
</tbody>
</table>
We will use SQLDeveloper with Orabbdd connection

USER: dbdalumno

PASSWORD: dbdalumno

1. Show the email, name and surname of the users ordered by their surname and name
2. All the information (code and name) of the provinces which have users
3. All the information (code and name) of the provinces which have users, without repeated information and ordered by name
4. Articles that do not have brand (marca)
5. E-mail of the users from Murcia who do not have telephone. The following message should be shown for each user that is shown in the output "No tiene teléfono".
6. Date and user of an order (pedido), code, name, brand, pvp and selling price of the articles that are televisions and belong to the order number 1.
7. E-mail of the users whose postal code is not either 02012, 02018 or 02032.
8. Order number, date and name and surname of the user that has requested something for the orders made by any user with surname “MARTINEZ”.
9. Code, name and brand for the most expensive article.
10. Code, name and pvp of the most expensive camera from the “réflex” ones.
11. Brands that do not belong to any TV in our database.
12. Code, name, type and brand of the cameras belonging to Nikon, LG or Sigma.
13. Code and name of the articles. Moreover, if the article is a camera, you have to show its resolution and sensor tool.
14. Show the baskets (cestas) of 2010 together with the name of the article they refer to and its pvp.
15. Show all the information of the articles. If any of them appears in a basket of 2010, you have to show also the information of the basket.
16. Number of users in our database.
17. Number of distinct provinces that we know that we have users.
18. Biggest screen size of the televisions.
19. Birth date of the oldest user
20. Obtain the total price per line (línea de pedido) for the order 1. In the output, the information about line number, article, and the total price should be shown.
21. Order number, date, name and surname of the user whose the total price of his/her order lines is the highest one. (Número de pedido, fecha y nombre y apellidos del usuario de las líneas de pedido cuyo total en euros es el más alto)
22. How many articles of each brand are there?
23. Dni, name, surname and e-mail of the users who made more than one order.
24. Orders (order number and user) that contain more than four different articles.
25. Obtain the code and name of the provinces that have more than 50 users (we refer to the province of the user, not the province stated the delivery address).
26. Number of articles that are not memories, TVs, lenses (objetivos), cameras or packs

27. In how many orders have been requested each article? You should also show the articles not requested in any order, if any. You have to show the code and name of the article, together with the times that has been included in an order (only if it has been included).

28. Order number and user of the orders with a cost higher than 4000 euros.

29. Code and price of the 'Samsung' articles that have pvp and have not been included in any order.

30. Code of the articles that are included in a basket (cesta) or in an order line (línea de pedido).

31. E-mail and name of the users that make none or only one order.

32. E-mail and name of the users that have not ordered any camera.

33. Current date
Session 2 – CREATE TABLE, DROP TABLE, ALTER TABLE, INSERT, UPDATE y DELETE

Next, 3 statements are shown. These statements are very useful when working with ORACLE. In our case, SQLDeveloper allows us to obtain the same information when browsing through the menu options.

---

**TABLES DEFINED BY USERS**

SELECT table_name FROM user_tables / SELECT * FROM user_tables

**INFORMATION ABOUT A TABLE**

DESC nombre_tabla

**INFORMATION about the RESTRICTIONS of a TABLE**

With SELECT * FROM user_constraints we obtain all the restrictions defined by a user.

As a result, we obtain a table with several columns. From all the columns, the ones that are more interesting are:

<table>
<thead>
<tr>
<th>OWNER</th>
<th>CONSTRAINT_NAME</th>
<th>CON</th>
<th>TABLE_NAME</th>
<th>SEARCH_CONDITION</th>
<th>R_OWNER</th>
<th>R_CONSTRAINT_NAME</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>usuBD2</td>
<td>SYS_C006505</td>
<td>R</td>
<td>T4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>usuBD2</td>
<td>SYS_C006504</td>
<td>P</td>
<td>T4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>usuBD2</td>
<td>PKT</td>
<td>P</td>
<td>T</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>usuBD2</td>
<td>SYS_C006502</td>
<td>C</td>
<td>T</td>
<td>c between 2 and 4</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Where owner is the user that has defined the restriction.
- Constraint_name is the name of the restriction. It will be a name given by the user, or by the system (in case the user has omitted the name). When the system has provided the name for a restriction, SYS appears in the name.
• C is a column where the following values can be found: P, to indicate a primary key restriction; R, to indicate a foreign key restriction, and C, to indicate a CHECK restriction.

• Table_name is the table to which the restrictions are defined.

• Search_condition is the condition for a CHECK restriction.

• R_Constraint_name in the foreign key restrictions refers to the name of the primary key restriction of the referred table.

In order to know the restrictions of a table, we can execute a SELECT statement in the table together with conditions using the operators we know.

```
SELECT * FROM user_constraints WHERE CONSTRAINT_NAME LIKE 'L%'
```
CREATE TABLE (for creating a table)
where
- **tableName**
  - is the name of the table.
- **column**
  - is the name of one column of the table.
- **datatype**
  - specifies the column data type. Such type has to be one of the supported data types for the system (you can find them in help, but we will usually use `char(n)`, `varchar(n)`, `number(n,m)`, `integer`, `date`)

- Table-constraint /Column-constraint

Table restrictions (table-constraint) or columns restrictions (column-constraint). We can define: PRIMARY KEY (for each table), UNIQUE, FOREIGN KEY, DEFAULT (for each column), and CHECK. All of them can be put in the same statement, according to the syntax shown on the next page. Moreover, the following issues have to be taken into account:

- The columns defined as PRIMARY KEY cannot contain null values and their values cannot be repeated.
- The combination of values for the columns containing a UNIQUE restriction cannot be repeated. The columns which have this type of restriction can contain null values.
- When defining the referential integrity, the column data types involved in a foreign key, as well as the ones in the referred columns must be compatible. When the restriction is defined at a column level, the keywords “FOREIGN KEY” and the enumeration of the columns are omitted.
- The DEFAULT restriction specifies the value for the column when we do not provide a specific value when inserting a new row in the table.
- The CHECK restriction narrows the possible values that a column can have. Multiple CHECK restrictions can be defined for the same table. However, only one can be defined for each column. When there is more than one CHECK restriction for a column, all of them get evaluated. The CHECK restriction that affects more than one column must be defined at a table level.
- In the syntax scheme shown next, when it says “CONSTRAINT constraint”, it refers to the fact that we can give a name to the restrictions. If we do not provide such name, the system automatically will give it a default name for the constraint. When a user provides the names for the restrictions, it would be clearer and easier to work with the database created. It is suggested that the user defines an homogeneous criterion to define the name for the constraints.
Table_constraint

CONSTRAINT constraint

UNIQUE column

PRIMARY KEY column

Foreign_key_clause

CHECK condition

Foreign_key_clause

FOREIGN KEY column REFERENCES table

ON DELETE CASCADE
Example:

To illustrated the syntax, we show several constraints that were defined differently in each table. In this manner, you can learn the different ways to define them.

CREATE TABLE pieza (numpieza number(10) CONSTRAINT cppiezas PRIMARY KEY,
                      nompieza varchar(30),
                      preciovent number(6) CONSTRAINT chprecioven CHECK (preciovent >0));

CREATE TABLE vendedor (numvend number(5),
                        nomvend varchar(30), nombrecomer varchar(30), direccion varchar(30),
                        telefono varchar(15), poblacion varchar(20), provincia varchar(20),
                        PRIMARY KEY (numvend));

CREATE TABLE preciosum (numvend integer, numpieza varchar(16) CONSTRAINT fkprepie REFERENCES pieza,
                        preciouunit number(6), diassum number(3) DEFAULT 2 CONSTRAINT chdiassum CHECK (diassum>0),
                        CONSTRAINT fkpreven FOREIGN KEY (numvend) REFERENCES vendedor,
                        CONSTRAINT cppreciosum PRIMARY KEY (numpieza, numvend));

DROP TABLE nomtable

In many cases, when we want to drop a table, we should follow an appropriate order, as it is possible that the dropping is not allowed so the referential integrity of the database is preserved.
ALTER TABLE (for modifying the structure and restrictions of a table)

Once a table has been created, it is possible to modify its structure with the statement ALTER TABLE. This statement allows us to add or delete a new column in the table, as well as to add or delete restrictions for the columns.

Add_column_options

Column datatype expr DEFAULT Column_constraint

Table_constraint

ADD (Add_column_options)

MODIFY (Modify_column_options)

RENAME TO New_table_name

DROP CONSTRAINT Constraint_name

DROP COLUMN Column_name

Constraint name
Example:

ALTER TABLE VENDEDOR ADD fnacimiento date;

ALTER TABLE PIEZA MODIFY nompieza not null;

Let’s suppose that a table called “DELUJO” exists with a column called “numpieza”

ALTER TABLE DELUJO ADD constraint lujopieza foreign key(numpieza) references pieza;
**INSERT** (for inserting rows in a table)

If we do not specify any list of columns, it is assumed that we are going to insert values for all the columns of the table.

It is important to take into account that the order of the list of values to be inserted must be the same as the order of the columns, if we enumerate them, or the order of the columns defined in the table.

```
INSERT INTO pieza VALUES ('A25D', 'DESTORNILLADOR MULTIPLE', 1500);
INSERT INTO pieza(numpieza) VALUES ('A29C');
INSERT INTO subpiezas(num, precio) SELECT numpieza, preciovent FROM pieza WHERE preciovent>150;
```

**DELETE** (to delete rows)

If we do not specify any condition (WHERE), all the rows of the table (or view) will be deleted.

```
DELETE FROM pieza WHERE numpieza='A29D'
```
DELETE FROM pieza WHERE nompieza LIKE ‘%DESTORNILLADOR%’

**UPDATE** (for modifying the date of a table)

As what happens for the DELETE statement, if we do not specify any condition, the modification will be applied to all rows of the table.

Example

```sql
UPDATE piezas
SET preciovent= 2000
WHERE numpieza='A29D';

UPDATE vendedor
SET direccion='OLMO,19',
   telefono='666567686'
WHERE numvend= 2;

UPDATE PIEZA
SET preciovent=(select max(preciovent)
   FROM pieza)
WHERE numpieza='A-1001-L';
```

```
UPDATE vendedor
SET telefono='65656566565'
WHERE nombrecomer LIKE ‘HARW%’;
```
Session 2 - Exercises
CREATE TABLE, DROP TABLE, ALTER TABLE, INSERT, UPDATE, DELETE

IMPORTANT for ALL the LAB SESSIONS
The exercises will be done using SQL Developer, with Orabbdd connection.

The user for each student given that his/her email address is abc12@alu.ua.es is the following:

- His/her user will be dbdabc12
- His/her passwd will be dbd

The statements that you execute will not be saved from one session to another. This means that each student should save his/her statements. In case the users should be removed and created again, students that have been saving the statements could easily rebuild the database using their statements. Exceptionally, students who could not save their statements, could use the ones the teacher has.
1.- Create the following tables. Some of the data types are given, but some of them must be deduced from the structure provided.

**TEMPORADA(nombre: varchar(5))**
Primary key: nombre

**CATEGORIA(nombre:varchar(2), descripción: varchar(80), supMin:number(4,2), supMax:number(4,2))**
Primary key: nombre
Create the table, taking into account that:
“Nombre” only allows the following values: D, DT, I o S

**PVPTEMPORADA(categoría, temporada, pSA, pAD, pMP, pPC)**
Primary key: (categoría, temporada)
Foreign key: categoría → CATEGORIA
Primary key: temporada → TEMPORADA
The prices pSA (only accommodation), pAD (breakfast), pMP (half board) y pPC (full board) are all number(3)

**HABITACIÓN (número: number(3), categoría)**
Primary key: número
Foreign key: categoría → CATEGORIA
N.N.V.: categoría

2.- Once the tables have been created, make the following changes:
- The column “descripción” must have a value.
- “Temporada” only allows the following values: “BAJA”, “MEDIA” or “ALTA”
- The default value for the type of the room (“categoría”) is D.
- The default price for the option “Only accommodation” in any season “TEMPORADA” is 60.

3.- To simulate an alternative key, the combination of the following two restrictions is used: UNIQUE and NOT NULL. You have to create the tables below. What differences are there when creating them?

**T1(a number(2), b number(2), c number(2), d number(2))**
Primary key: (a, b)

**T2(a number(2), b number(2), c number(2), d number(2))**
Primary key: a
Alternative key: b

**T3(a number(2), b number(2), c number(2), d number(2))**
Primary key: a
Alternative key: (b, c)

**T4(a number(2), b number(2), c number(2), d number(2))**
Primary key: a
Alternative key: b
4.- Insert the following data. If any insertion cannot be performed, please explain and justify the reason.

- Insert the double category (D), with description “SECADOR, MINIBAR, AMPLIA, ADMITE SUPLETORIA, with minimum surface of 15.
- Insert the individual category (I), with description “SECADOR”, with maximum surface of 12.
- Insert the double category (DT), with description “SECADOR, MINIBAR, AMPLIA, DOS CAMAS, ADMITE SUPLETORIA, TERRAZA AL PARQUE”, with minimum surface of 15.
- Insert the three seasons (temporadas) in the table “TEMPORADA.
- Insert a new season, EXTRA, in the table “TEMPORADA”.

5.- Insert in the table “pvptemporada” all the categories combined with all the possible seasons. More specifically, when inserting data for the double rooms (D) in the 3 seasons, you should change the price from SA to 65.

6.- Delete the category SUITE (S).

7.- Delete the category DOBLE (D).

8.- The prices for the rooms from “DOBLE with TERRACE” category (DT) for only accommodation (SA) are 100 euros in low season (“temporada baja”), 115 in middle season ( “temporada media”) and 140 in high season ( “temporada alta”).

9.- Insert room number 7, without category.

10.- Create 4 different tables with the definition you want, but each of them should contain a column with the definition given in the table below. In this manner, you will be able to complete the table, understanding what happens in each proposed situation.

<table>
<thead>
<tr>
<th>The column is defined as</th>
<th>NULL and has defined a default value</th>
<th>NULL and does not have a default value</th>
<th>NOT NULL and has defined a default value</th>
<th>NOT NULL and does not have a default value</th>
</tr>
</thead>
<tbody>
<tr>
<td>When doing</td>
<td>NULL</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>


<table>
<thead>
<tr>
<th>INSERT to the specified column …</th>
<th>DEFAULT</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>I do not provide a name for the column</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
In order to show messages, we can use:

**MESSAGES**

`raise_application_error (error_number, message)`

For example:
`raise_application_error(-20101,'Total number of students reached')`

This function shows the message and generates an error, stopping the running process.

`dbms_output.put_line(message)`

This function puts the message in a buffer and continues with the running process.

In order for the messages to be shown, we need to run the following statement first:

**SET SERVEROUTPUT ON FORMAT**

We will use the “||” operator, when we want to link several messages. It can be used in both functions: `raise_application_error` and `dbms_output.put_line`
A SQL block is composed of three sections: 1) declarative; 2) executable, and 3) instruction management (warnings and error conditions). From these sections, only the one that contains the code to be executed is required (i.e., it is mandatory).

\[
\begin{align*}
&\text{[ DECLARE} \\
&\quad -- \text{declarations} \\
&\text{BEGIN} \\
&\quad -- \text{statements} \\
&\text{[EXCEPTION} \\
&\quad -- \text{exception handling} \\
&\text{END;}
\end{align*}
\]

**VARIABLES AND CONSTANTS DEFINITION**

In the declarative section of the SQL block, we can define variables and constants. For instance:

```sql
DECLARE
    total number(6,2);
    name varchar(30):= 'ROB';
    maximum CONSTANT number(4):=9999;
```

Moreover, in the declarative section, the **variables can be initialized** to a specific value, but it is not mandatory to do so. In contrast, **constants must be initialized** in this section.
VARIABLES AND CONSTANTS DEFINITION

If we need that the data type to be the same as the one that was defined for a column, %TYPE can be used.

The %TYPE attribute provides the data type of a variable or column of the database.

DECLARE
Value1 number(4,2);
AuxValue Value1%TYPE;
AuxCategory room.category%TYPE;

The variable “AuxValue” is of the same data type than “Value1” and “AuxCategory” has the same data type of the column “category” in the table “room”.

When using the %TYPE attribute, we do not need to know in advance the exact data type for a variable or column. Moreover, the advantage of using %TYPE is that whenever the column data type changes, the data type of the variable will automatically change too.

If we need to define a structure similar to a row in a table, we can use %ROWTYPE

The %ROWTYPE attribute obtains a register type that represents the row of a table. The fields in such register and the corresponding columns of the table have the same name and data type.

DECLARE
pvp pvptemporada%ROWTYPE;

SET VALUES FOR VARIABLES

By default, variables are initialized as NULL.
There are two ways to assign values to variables: through an expression, or through a SELECT statement.

aux_value1:=25;
SELECT category INTO auxcategory FROM room WHERE ....
aux_value2:=total-10;
SELECT cod, description INTO auxcod, auxdesc FROM ACTIVITIES WHERE ...
aux_name:='ROBERT'
CURSORS

Oracle employs working areas in order to run SQL statements, and stores the processed information. The use of cursors allows us to give a name to a working area, so we can access the information processed in such working area.

When a cursor is declared, we must give it a name, and assign a specific query.

```
CURSOR cursor_name IS select_statement;
```

For instance:

```
DECLARE
    CURSOR c1 IS SELECT num, superf FROM habitación h, pvptemporada p WHERE h.categoria=p.categoria;
BEGIN
    OPEN c1;
    FETCH c1 INTO auxnum, auxsuperf, auxsupMin;
    …
END;
```

After processing this, it is like we had a table called c1 and we could go row by row.

Working with OPEN, FETCH and CLOSE

(although this manner of dealing with cursors is also valid, we strongly recommend to work with cursors in an easier way, that is using CURSORS within FOR LOOPS. This will be explained in the next section)

When working with cursors, the following commands can be used: OPEN, FETCH and CLOSE.

OPEN
Opens the cursor, that is, it runs the query and identifies the result (the rows selected after the query). When running OPEN, the rows are not returned.

```
DECLARE
    auxnum    habitación.num%type;
    auxsuperf habitación.superf%type;
    auxsupMin categoría.supMin%type;
    CURSOR c1 IS SELECT num, superf, supMin FROM habitación , categoría WHERE categoría=nombre;
BEGIN
    OPEN c1;
    …
    FETCH c1 INTO auxnum, auxsuperf, auxsupMin;
    …
END;
```

FETCH
It allows to return the rows from the result. Each time we run FETCH, the cursor goes to the next row in the result.

CLOSE
It closes the cursor. Once it has been closed, it can be opened again

For each column that is returned by the query assigned to the cursor, we should have a variable after the INTO keyword, with a compatible data type.
Each cursor has four attributes associated to it: %FOUND, %ISOPEN, %NOTFOUND, %ROWCOUNT

After a cursor is opened with OPEN, and before the first FETCH is performed:

- The attribute %FOUND contains NULL
- The value of %ROWCOUNT is 0

From this moment and onwards, every time we run FETCH:

- %FOUND will return TRUE if the last FETCH returned a row, or otherwise FALSE.
- %ROWCOUNT will contain the number of rows to which FETCH was already run.

The opposite attribute to %FOUND is %NOTFOUND.

The attribute %ISOPEN returns:

- true if the cursor is open.
- false if it is not open.

For instance:

If c1 is a cursor that we have defined, we could use expressions, such as:

IF c1%FOUND THEN …
IF c1%ROWCOUNT >4 THEN …
When working with cursors, we can simplify the code by using a FOR loop instead of OPEN, FETCH and CLOSE. The FOR loop:

- Opens the cursor,
- Performs the FETCH operations repeatedly,
- And closes the cursors when all the rows have been processed.

In the loop, besides opening the cursor, a variable is also declared (regc1, in the following example).

- This variable can only be used inside the loop.
- Its data type is registered, and its fields have the same name and data type as the columns of the SELECT query that is in the definition of the cursor.
- If any of the columns were calculated, it would be necessary for the SELECT query to have an alias.

For example:

```sql
DECLARE
  CURSOR c1 IS SELECT num, superf, supMin FROM habitación, categoría WHERE categoría = nombre;
BEGIN
  FOR regc1 IN c1 LOOP
    IF regc1.sup < regc1.supMin THEN
      Dbms_output.put_line('Habitación ' || regc1.num || ' dimensión INCORRECTA ***');
      INSERT INTO MALCATEGORIA VALUES(regc1.num, regc1.supMin - regc1.superf);
    ELSE
      Dbms_output.put_line('Habitación ' || regc1.num || ' dimensión correcta');
    END IF;
  END LOOP;
END;
```

We assume that MALCATEGORIA is already created MALCATEGORIA(num: number(3), dif: number(4,2))

In the DECLARE block, together with the definition of the variables and constants, the definition of cursors is also made.

In the first iteration:
- The cursor is open (OPEN),
- We access the first row (FETCH) and
- It is placed in regc1 (which was declared in the FOR).

Each new iteration means the access to the next row (FETCH) and place it in regc1.

When going through all the rows, all the iterations finished and the cursor is automatically closed (CLOSE).
For working within the BEGIN-END section, we can use the same control structures as for other programming languages.

**CONDITIONAL STRUCTURE**

- IF ... THEN...
  - ELSE ...
  - END IF;

**INTERATIVE STRUCTURES**

- FOR i IN min .. max LOOP
  - ...
  - END LOOP;
- WHILE condition LOOP
  - ...
  - EXIT WHEN ...;
  - END LOOP;
- LOOP
  - ...
  - EXIT WHEN ...;
  - END LOOP;
- LOOP
  - IF condition THEN
    - ...
    - EXIT;
  - END IF;
  - END LOOP;
Session 3 - Exercises
MESSAGES.
CONSTANTS, VARIABLES AND CURSORS.

1.- Create the following tables, some of the data types are indicated, but others should be deduced. Note: P.K. means Primary Key

CALENDARIO (fecha: date, temporada varchar(5))
  P.K.: fecha
HORA (hora: number(2))
  P.K.: hora
ACTIVIDAD (código:char(4), descripción varchar(100))
  P.K.: código
  N.N.V.: descripción
HORARIO (fecha, hora, actividad)
  P.K. (fecha, hora)
  Alternative Key: (fecha, actividad)
  Foreign key: fecha → CALENDARIO
  Foreign key: hora → HORA
  Foreign key: actividad → ACTIVIDAD
ActNIÑOS(actividad)
  P.K.: actividad
  Foreign key: actividad → ACTIVIDAD
ActADULTOS(actividad)
  P.K.: actividad
  Foreign key: actividad → ACTIVIDAD
ActTodos(actividad)
  P.K.: actividad
  Foreign key: actividad → ACTIVIDAD
SustNIÑOS(activ1, activ2)
  P.K.: (activ1, activ2)
  Foreign key: activ1 → ActNIÑOS
  Foreign key: activ2 → ActNIÑOS
SustADULTOS(activ1, activ2)
  P.K: (activ1, activ2)
  Foreign key: activ1 → ActADULTOS
  Foreign key: activ2 → ActADULTOS
SustTODOS(activ1, activ2)
  P.K: (activ1, activ2)
  Foreign key: activ1 → ActTODOS
  Foreign key: activ2 → ActTODOS
2.- Add two columns to the table HABITACIÓN. These columns are: its surface (superficie) (number(4,2)) and its flat (piso) (number(2)).

3.- The “superficie” of room number 1 is 17, the “superficie” of room number 2 is 14, and “superficie” of room number 3 is 18.

4.- Create the table MALCATEGORIA (num: number(3), dif: number(4,2)).

Now, you have to create a SQL block, relying on a cursor so that the surface of the rooms (superficie) is revised and it has to control that such surface is no lower than the minimum surface of its category (category). A list containing the room number will be shown, indicating whether its dimension is correct or not. Moreover, those rooms which have a surface lower than the minimum of their category will be included in the table MALCATEGORIA, with its number and the difference with respect to the minimum surface. *(You can check the example in today’s session statements).*

5.- Using a cursor, you must obtain a list for the adult activities with the following information:
   - Code of the activity which was substituted,
   - Description
   - Message: ‘conflicto X sustituciones en la fecha Y’, where X is the number of activities that coincide in the date and Y is the date.

   The adult activities can be substituted by more than two adult activities, and at least three of them can coincide at the same date.

   **Note:**
   Table “sustADULTOS”, “activ1” represents the activity that is substituted and “activ2” the activity that substitutes. It can be assumed that one activity cannot be substituted by itself (we will look at this later during the course).

   **For example:**
   - If activity A10 is for adults and substitutes A11, A12, and A13, but these never coincide with the date, A10 will not appear in the output list.
   - If activity A20 is for adults and substitutes activities A11, A12, A13 and A14, and from these, three of them coincide with the date 21/09/2012 and only two of them coincide with the date 22/09/2012, A20 will appear in the output list, indicating that A20 has 3 substitutions on 21/09/2012.
   - If activity A30 is for adults and substitutes activities A11, A12, A13, and A14, and from these, three of them coincide with the date 21/09/2012 and four of them with the date 22/09/2012, A30 will appear twice in the output list, indicating that it has 3 substitutions on the date 21/09/2012, and 4 substitutions on the date 22/09/2012.
Session 4: CREATE PROCEDURE

CREATE [OR REPLACE] PROCEDURE Procedure_name

IS BEGIN Executable section

EXCEPTION Exception section

END ;

When defining the parameters, we must indicate: param_name param_type data_type (without length)
As a param_type, we can use:
- IN(by default). It also allows to provide values when we call the procedure. This value cannot be changed. We can assign a default value
  
  CREATE PROCEDURE EXAMPLE(entrada in Lumber default 10) IS ...  

- OUT for returning values.
- IN OUT allows to provide values when calling the procedure and return values afterwards.

For dropping a procedure     DROP PROCEDURE procedure_name
Examples:

create or replace procedure escribir(auxcad in varchar2) is
    begin
        dbms_output.enable;
        dbms_output.put_line(auxcad);
    end;

It can be directly executed

exec escribir('Hola');

Or within another procedure

create or replace procedure obtenerpreciobase(numhab in number, posiblereserva date) is
    auxprecio number(3);
    begin
        select pSA into auxprecio
        from habitación h, pvptemporada p, calendario c
        where numero=numhab and fecha=posiblereserva and
            h.categoria=p.categoria and p.temporada=c.temporada
        escribir('El precio base para la habitación '||numhab||' en esa fecha es '||auxprecio);
    end;

exec obtenerpreciobase(1,'29/03/2013');
When creating a procedure, function or trigger (we will see these latter two in the following sections) with SQL-Developer, you must click on it on the left side of the screen, and select the COMPIL option. If you see that there have been any errors in the procedure, you should correct them; otherwise it cannot be executed.
create or replace procedure escribir(suceso in varchar2) is
begin
  dbms_output.put_line(suceso);
end;

create or replace procedure escribir(suceso in varchar2) is
begin
  dbms_output.put_line(suceso);
  end;

create or replace procedure escribir(suceso in varchar2) is
begin
  dbms_output.put_line(suceso);
  end;

create or replace procedure escribir(suceso in varchar2) is
begin
  dbms_output.put_line(suceso);
  end;

create or replace procedure escribir(suceso in varchar2) is
begin
  dbms_output.put_line(suceso);
  end;

create or replace procedure escribir(suceso in varchar2) is
begin
  dbms_output.put_line(suceso);
  end;

create or replace procedure escribir(suceso in varchar2) is
begin
  dbms_output.put_line(suceso);
  end;

create or replace procedure escribir(suceso in varchar2) is
begin
  dbms_output.put_line(suceso);
  end;

create or replace procedure escribir(suceso in varchar2) is
begin
  dbms_output.put_line(suceso);
  end;

create or replace procedure escribir(suceso in varchar2) is
begin
  dbms_output.put_line(suceso);
  end;

create or replace procedure escribir(suceso in varchar2) is
begin
  dbms_output.put_line(suceso);
  end;

create or replace procedure escribir(suceso in varchar2) is
begin
  dbms_output.put_line(suceso);
  end;

create or replace procedure escribir(suceso in varchar2) is
begin
  dbms_output.put_line(suceso);
  end;

create or replace procedure escribir(suceso in varchar2) is
begin
  dbms_output.put_line(suceso);
  end;

create or replace procedure escribir(suceso in varchar2) is
begin
  dbms_output.put_line(suceso);
  end;

create or replace procedure escribir(suceso in varchar2) is
begin
  dbms_output.put_line(suceso);
  end;

create or replace procedure escribir(suceso in varchar2) is
begin
  dbms_output.put_line(suceso);
  end;

create or replace procedure escribir(suceso in varchar2) is
begin
  dbms_output.put_line(suceso);
  end;

create or replace procedure escribir(suceso in varchar2) is
begin
  dbms_output.put_line(suceso);
  end;

create or replace procedure escribir(suceso in varchar2) is
begin
  dbms_output.put_line(suceso);
  end;

create or replace procedure escribir(suceso in varchar2) is
begin
  dbms_output.put_line(suceso);
  end;

create or replace procedure escribir(suceso in varchar2) is
begin
  dbms_output.put_line(suceso);
  end;

create or replace procedure escribir(suceso in varchar2) is
begin
  dbms_output.put_line(suceso);
  end;

create or replace procedure escribir(suceso in varchar2) is
begin
  dbms_output.put_line(suceso);
  end;

create or replace procedure escribir(suceso in varchar2) is
begin
  dbms_output.put_line(suceso);
  end;

create or replace procedure escribir(suceso in varchar2) is
begin
  dbms_output.put_line(suceso);
  end;

create or replace procedure escribir(suceso in varchar2) is
begin
  dbms_output.put_line(suceso);
  end;

create or replace procedure escribir(suceso in varchar2) is
begin
  dbms_output.put_line(suceso);
  end;

create or replace procedure escribir(suceso in varchar2) is
begin
  dbms_output.put_line(suceso);
  end;

create or replace procedure escribir(suceso in varchar2) is
begin
  dbms_output.put_line(suceso);
  end;

create or replace procedure escribir(suceso in varchar2) is
begin
  dbms_output.put_line(suceso);
  end;

create or replace procedure escribir(suceso in varchar2) is
begin
  dbms_output.put_line(suceso);
  end;

create or replace procedure escribir(suceso in varchar2) is
begin
  dbms_output.put_line(suceso);
  end;

create or replace procedure escribir(suceso in varchar2) is
begin
  dbms_output.put_line(suceso);
  end;

create or replace procedure escribir(suceso in varchar2) is
begin
  dbms_output.put_line(suceso);
  end;

create or replace procedure escribir(suceso in varchar2) is
begin
  dbms_output.put_line(suceso);
  end;

create or replace procedure escribir(suceso in varchar2) is
begin
  dbms_output.put_line(suceso);
  end;

create or replace procedure escribir(suceso in varchar2) is
begin
  dbms_output.put_line(suceso);
  end;

create or replace procedure escribir(suceso in varchar2) is
begin
  dbms_output.put_line(suceso);
  end;

create or replace procedure escribir(suceso in varchar2) is
begin
  dbms_output.put_line(suceso);
  end;

create or replace procedure escribir(suceso in varchar2) is
begin
  dbms_output.put_line(suceso);
  end;

create or replace procedure escribir(suceso in varchar2) is
begin
  dbms_output.put_line(suceso);
  end;

create or replace procedure escribir(suceso in varchar2) is
begin
  dbms_output.put_line(suceso);
  end;

create or replace procedure escribir(suceso in varchar2) is
begin
  dbms_output.put_line(suceso);
  end;

create or replace procedure escribir(suceso in varchar2) is
begin
  dbms_output.put_line(suceso);
  end;

create or replace procedure escribir(suceso in varchar2) is
begin
  dbms_output.put_line(suceso);
  end;

create or replace procedure escribir(suceso in varchar2) is
begin
  dbms_output.put_line(suceso);
  end;

create or replace procedure escribir(suceso in varchar2) is
begin
  dbms_output.put_line(suceso);
  end;

create or replace procedure escribir(suceso in varchar2) is
begin
  dbms_output.put_line(suceso);
  end;

create or replace procedure escribir(suceso in varchar2) is
begin
  dbms_output.put_line(suceso);
  end;

create or replace procedure escribir(suceso in varchar2) is
begin
  dbms_output.put_line(suceso);
  end;

create or replace procedure escribir(suceso in varchar2) is
begin
  dbms_output.put_line(suceso);
  end;

create or replace procedure escribir(suceso in varchar2) is
begin
  dbms_output.put_line(suceso);
  end;

create or replace procedure escribir(suceso in varchar2) is
begin
  dbms_output.put_line(suceso);
  end;

create or replace procedure escribir(suceso in varchar2) is
begin
  dbms_output.put_line(suceso);
  end;

create or replace procedure escribir(suceso in varchar2) is
begin
  dbms_output.put_line(suceso);
  end;

create or replace procedure escribir(suceso in varchar2) is
begin
  dbms_output.put_line(suceso);
  end;

create or replace procedure escribir(suceso in varchar2) is
begin
  dbms_output.put_line(suceso);
  end;

create or replace procedure escribir(suceso in varchar2) is
begin
  dbms_output.put_line(suceso);
  end;

create or replace procedure escribir(suceso in varchar2) is
begin
  dbms_output.put_line(suceso);
  end;

create or replace procedure escribir(suceso in varchar2) is
begin
  dbms_output.put_line(suceso);
  end;
create or replace

procedure escribir(auxcad in varchar2) is
begin
    dbas_output.enable;
    dbas_output.put_line(auxcad);
end;

Project: C:\Archivos de programa\sqldeveloper\sqldeveloper\system\oracle.sqldeveloper.1.2:

PROCEDURE ESCRIBIR@Orabdd

Errore(4,1): PL/SQL
Errore(4,3): PL/SQL

ESCRIBIR Compiled (with errors)
ESCRIBIR Compiled (with errors)
ESCRIBIR Compiled (with errors)
ESCRIBIR Compiled
Session 4 - Exercises
CREATE PROCEDURE.

0. Execute the file “crea-hotel.txt” that you will find in Campus Virtual and be sure that you have created the procedure called escribir (see the statements in session 4).
1. Modify the table calendreservas by adding the column alimentación. This column has a char(2) datatype and it only allows the following values: SA, AD, MP, PC.
Execute the file “datos-hotel.txt” that you will find in Campus Virtual.
2. Create a procedure for last week’s exercise 4. The wording of such exercise is as follows:

Create a SQL block, relying on a cursor so that the surface of the rooms (superficie) is revised and it has to control that such surface is no lower that the minimum surface of its category (category). A list containing the room number will be shown, indicating whether its dimension is correct or not. Moreover, those rooms whose surface is lower than the minimum of their category will be included in the table MALCATEGORIA, with its number and the difference with respect to the minimum surface. (You can check the example in last week’s session statements – session 3).

You should improve the code in a way that when there is no information about the surface extension (superficie) in a room or category, a message saying that circumstance will be shown. The information has to be shown sorted by the room ID (código).

Note: Before executing the procedure, you should delete all the rows in table MALCATEGORIA, in order to avoid conflicts with last week’s results.

3. Create a procedure where, given a date, a list showing the hour, activity ID (código), and description of the activities planned on that date is obtained.
4. Create a procedure where, given an adult activity, the ID (codigo) and description of the equivalent activities is shown.
5. Create a new table called ADAPTADAS. In this table we will represent a room subtype. It has a single column that is the room number.
6. Create a procedure called CompletaAdaptadas. With this procedure, we have to insert in table ADAPTADAS all rooms that are on the ground floor (piso 0 - cero) and they are bigger than 25 m². Moreover, after inserting, the following message has to be shown: “X adapted rooms have been inserted”, where X is the total number of rooms that have been inserted in ADAPTADAS.
7. Create a procedure in which the room prices for a season are revised. Given the season as input the price for accommodation only (pSA) for each room type will be checked. If such price is known, the price for breakfast included, AD, will be updated, adding 10% to the price of SA. Also, the price for half-board, MP, will be
updated by adding 20% to the price of SA, and the price of full board, PC, will be updated by adding 25% to the price of SA.

(Original wording for the exercise in Spanish: Crea un procedimiento en el que se revisen los precios de las habitaciones de una temporada dada como entrada. Para esa temporada se mirará el precio de sólo alojamiento (pSA) en cada tipo de habitación. En cada tipo de habitación, si se conoce ese precio de SA, se actualizará el precio de AD para ese tipo a un 10% más del precio para SA, el de MP a un 20% más del de SA y el de PC a un 25% más del de SA.)

8. Create a procedure that returns the ID (código) and description of the activities that do not have equivalent/replacing activities.
When defining the parameters we must indicate: *parameter_name parameter_type data_type (without length)*

As a *parameter_type*, we can use:
- IN (by default). It also allows providing values when we call the procedure. This value cannot be changed. We can assign a default value

```
CREATE FUNCTION EXAMPLE (entrada in number default 10) RETURN varchar IS ...
```

- The OUT and IN OUT are not normally used with functions.

For dropping a function
```
DROP FUNCTION function_name
```
For directly executing a function:

Exec escribir(function_name (param1));  (although the input parameter of “escribir” is varchar data type, the transformation from number or date to varchar is automatically done)

SELECT function_name(param1) FROM dual;

Besides, we can call a function from

- A procedure
- A SELECT statement: as a column in the SELECT, in a condition of WHERE and HAVING clause, as well as in the GROUP BY.
- An INSERT statement within the VALUES part.
- An UPDATE statement within the SET clause.

Examples (assuming that in the table “empleados” there is a column called fechanac):

create or replace function edad (elnif char) return number
is
aux number(2);
beging
select (to_char(sysdate,'yyyy')-to_char(fechanac,'yyyy')) into aux
from empleados where nif=elnif;
return(aux);
end;

Another option: create or replace function edad(elnif empleado.nif%type) return number

The compilation errors work the same as the procedures ones.
A function can be directly executed

Select edad('11111111A') from dual;

or

select count(*), edad(nif) from empleados group by edad(nif);

or within a procedure

create or replace procedure listadoporedad(laedad number) is
cursor emp is select e.nif,nombre, direccion
from empleado e, empanimacion an
where e.nif= an.nif;
begin
escribir('EMPLEADOS DE ANIMACIÓN MENORES de '||laedad||' :
');
for x in emp loop
  if edad(x.nif)<laedad then
    escribir('Nombre: '||x.nombre||'   direccion   '||x.direccion);
  end if;
end loop;
end;

exec listadoporedad(25);

In the definition of a cursor we have to assign a name to the columns in which alias from tables are employed, or columns that are aggregated functions. In this example we could have given another name to the column, not necessarily “dni”. It could have been “empleado” or any other representative name.
Session 5 - Exercises
CREATE FUNCTION.

1. Create a function that given a room number and a date, it returns the price of such room on that date for “accommodation only”.

2. Create a function that given a room category, a date and a type of accommodation (régimen alimenticio), it returns the corresponding price for the room and accommodation type.

3. Create a function that given a NIF from a service employee (empleado de servicios) and a date, it indicates how many expected appointments he or she has for that day.

4. Create a function called “free” that given an employee, a date and a time, it returns “YES” if the employee does not have appointments for this date and time, and “NO”, otherwise.

5. Create a function called “gastodiario” in which given a room number and a date, it returns the total amount spent on services for that room on that date.

6. Create a function called “gastototal” in which given a booking number, it returns the total amount spent on services for that booking.

7. Create a function that given a treatment code (código de tratamiento), it returns the total number of clients that have booked an appointment for that treatment.

8. Create a function that given a booking code made via telephone, it returns the name of the employee that made such booking.

9. Create a function that returns the name of the employee that has been on cleaning duty fewer times (el que menos guardia de limpieza tiene). If there is a draw (empate) between two employees, the function should return first the name of the employee, taking into account the alphabetical order.
A view is a “logic” table that is associated to a SELECT statement, that is, it does not physically contain its own data. In contrast, every time a reference to the view is made, the associated SELECT statement is executed. If we execute SELECT * FROM view_name in two different moments, the result obtained may vary. This is due to the fact that the content of the tables associated to the view could change, and therefore, the result of the SELECT.

We can use views for security reasons, for simplifying the structure to a specific user (it allows you to work with a “table”, but indeed you are working with a view associated to a SELECT from several tables with complex conditions), ...

WITH CHECK OPTION forces all the insertions and updates made through the view to fulfill the conditions stated in the SELECT statement.

Given a view, we can make operations, such as SELECT, INSERT, UPDATE y DELETE. The use of these operations is subjected to the constraints of the view when they were created. For instance, we cannot use INSERT in a view that was created with an aggregated function (count(*), sum(column_name),...) or, for example, if the primary key of the table is not in the view as well,....
For dropping views

```
DROP VIEW View_name
```

Example:

```
CREATE VIEW tratamientocaro AS
SELECT * FROM tratamiento WHERE precio > 60
```

Since we did not create the view WITH CHECK OPTION constraint, we could insert the following:

```
INSERT INTO tratamientocaro (Note: TRATAMIENTO (código, observaciones, precio))
VALUES ('EXMAS', 'Exfoliación y masaje completo', 30)
```

If we do not violate any constraint (primary key, not null value, check, …) the insertion will be done into the base table of the view, i.e., tratamiento.
Since we have not stated the constraint WITH CHECK OPTION, the conditions of the SELECT statement will not be evaluated. Only the table associated to the view and its constraints will be taken into account.

If we run

```
SELECT * from tratamiento
```
the new row will be shown in the result, because it has been inserted in the table “TRATAMIENTO”.

If we run

```
SELECT * from tratamientocaro
```
the new row will not appear in the result, because in this case, the select associated to the view is executed, and, so far, there is no row that fulfills the condition.

A good exercise will be to assess what happens with the insertions when, in the SELECT statement associated to the view, the FROM section includes more than one table, or when in the SELECT section contains aggregated functions.
Session 6 - Exercises
CREATE VIEW.

Do the following exercises, putting only WITH CHECK OPTION when indicated. In some cases, the statement you propose can fail (not necessarily a syntactic error) and you should understand why it fails.

1) Create a view called `individuales` that contains the number of individual rooms.

2) Create a view called `individuales_check` that contains the number of individual rooms. Use WITH CHECK OPTION when you create the view.

3) Create a view called `individuales_cat` that contains the number and category of individual rooms.

4) Create a view called `individuales_cat_check` that contains the number and category of individual rooms. Use WITH CHECK OPTION when you create the view.

5) Perform the following insertions.
   a. Insert through the view “`individuales``room number 100.
   b. Insert through the view “`individuales_check``room number 110.
   c. Insert through the view “`individuales_cat``room number 120 of category S.
   d. Insert through the view “`individuales_cat_check``room number 130 of category S.
   e. Insert through the view “`individuales_cat_check``room number 140 of category I.

Do you understand what happens in the previous insertions?

6) Create a view called “`dobles_check``that contains the room numbers of category D. Create it using WITH CHECK OPTION (similar to exercise 2 with I).

   Insert room number 150 through the view “`dobles_check``. Is there any difference with respect to what happened in exercise 5.b?

7) Create a view called “`clientes_alicante``that contains the name, phone number and city (localidad) for customers from the Alicante province.

   Insert into the view “`Rodrigo López`, who is from ”`Alcoy``, and his telephone number is “`646565444``”.

8) Create a view called “`clientes_alicante_nif``that contains the nif, name, city (localidad) and phone number for customers from the Alicante province.

   Insert into the view the customer “`Rodrigo López`, who is from ”`Alcoy`, with phone number “`646565444``and nif “`999999992``”.

   Insert into the view the customer ”`Miguel Bermejo`, who is from ”`Villena`, with phone number “`646555555``and nif “`216688999P``”.

9) Create a view called “`contadorsustituciones``, where for each activity the following is shown: its identifier (código), its description (descripción), its type (NIÑOS,
ADULTOS, TODOS), and total number of equivalent activities (actividades equivalentes) that are in the data base.

Insert into the view the activity PAC with description “Horse riding” (paseo a caballo), type “ADULTS” and that has 5 equivalent activities.
Session 7 - Exercises

Execute `SET SERVEROUTPUT ON FORMAT;`

1. Create the following table:

   ```sql
   CREATE TABLE GRUPO (
     num number(2),
     nombre varchar(50),
     grupomin number(2)
   )
   PRIMARY KEY (num);
   
   The values of "grupomin" must be between 1 and 5.
   ```

2. Add a column in the table "ACTIVIDAD" and make it foreign key to table GRUPO.

3. Modify the data type of the column "descripción" of table "CATEGORIA". It must be `varchar(70)`.

4. Create a view that shows the number and name (número y nombre) of all the groups, and for those where their activities are known. The total number of activities belonging to the group must be also shown.

5. Create a view that shows the name of the season (temporada), as well as the total number of days that belong to that season.

6. Create a function that returns the name of the season (temporada) that has more assigned days.

7. Create a function that shows the most frequent hour in which activities, which has the description “Campeonato”, take place.

8. Create a procedure where, given a date, the following information is obtained: each room category together with the price of half board (media pension) for staying in rooms of such category on the given date.

9. Add a column in the table “ACTIVIDAD” called “precio”. It has to be number (2), and 0 by default.

10. Create a procedure where, given two values “X” and “Y” the column “precio” of table “ACTIVIDAD” is modified according to the following criterion: if the activity is due to last for at least X days, its price will be increased by “Y” units. Moreover, a list containing the code and name of the activity together with the final price will also be shown (the procedure can be executed at any time and several times, this price does not have to be necessarily “Y”).

Note: The entity DÍA is represented in the table CALENDARIO.
Session 8
MANAGING EXCEPTIONS

In most programming languages, EXCEPTIONS are employed for managing errors at running time. Oracle also offers a way to define exceptions. These definitions lead to the fact that when an error occurs, the flow of the program goes to the corresponding exception, and this takes control of the program. Actions to be followed are included in the EXCEPTION section defined at the end of the BEGIN-END block.

```
BEGIN
...
EXCEPTION
  WHEN <exception_name> THEN
  <block of statements>;
  WHEN <exception_name> THEN
  <block of statements>;
...
  [WHEN OTHERS THEN <block of statements >;]
END;
```

There are two types of exceptions:
The ones predefined in the system, and the ones defined by the user.

**PREDEFINED EXCEPTIONS**

They are those ones that are automatically triggered when specific types of errors occur. Next, the most frequent ones are shown:

- **too_many_rows**: it happens when SELECT ... INTO returns more than one row.
- **no_data_found**: it happens when SELECT... INTO does not return any row.
- **value_error**: it happens when there is an arithmetic or conversion error.
- **zero_divide**: it can be used when there may be a division by zero.
- **dupval_on_index**: it happens when we try to store a value that would create duplicates in the primary key or in a column defined as UNIQUE.
- **invalid_number**: it happens when we try to convert a string into a numeric value.

As they are already defined in Oracle, they do not have to be declared in the DECLARE section.
EXCEPTION DEFINED BY THE USER

The user defines this type of exception. In order to work with this kind of exceptions, we have to follow three steps:

1. Define the exception name in the DECLARE section. The syntax is:

   ```
   Exception_name EXCEPTION
   ```

2. Launch the exception. It is done by using RAISE;

3. Define the actions to be done when the exception is launched. This is done in the EXCEPTION section.

   ```
   WHEN exception_name THEN ...
   ```

Example 1:

Create a procedure where given the nif of an employee as input, his/her name and phone number are shown.

```sql
create or replace PROCEDURE mostrar_empleado(elnif empleado.nif%type) is
    auxnombre empleado.nombre%type;
    auxtelefono empleado.telefono%type;
BEGIN
    select nombre, telefono into auxnombre, auxtelefono
    from empleado where nif=elnif;
    escribir('Name '|auxnombre|'| phone '|auxtelefono);
EXCEPTION
    WHEN no_data_found then
        escribir('There is not any employee with that nif');
END;
```
Example 2:
Create a function where given a room number as input, its category is returned.

```sql
create or replace FUNCTION tipo_habitacion (elnumero habitacion.numero%type) return habitacion.categoria%type
is
auxcategoria habitacion.categoria%type;
begin
select categoria into auxcategoria
from habitacion
where numero=elnumero;
return(auxcategoria);
EXCEPTION
  when no_data_found then
    escribir('There is not any room with that number');
    return null;
end;
```
Example 3:
Create a procedure where given a nif of an employee and a number X, include this employee as a receptionist, as long as there are not already more than X receptionists.
If when we register this employee as a receptionist, he/she is not registered as an employee, he/she will be also registered as an employee (and as a receptionist) and a message will be shown indicating that more data is needed in order to complete it.

create or replace PROCEDURE pon_recepcion(elnif empleado.nif%type, maximo number) is
    auxnombre empleado.nombre%type;
    auxtelefono empleado.telefono%type;
    total number(3);
    pasa_tope exception;
BEGIN
    select count(*) into total from emprecepcion;
    if total>maximo then raise pasa_tope;
end if;
    select nombre, telefono into auxnombre, auxtelefono
        from empleado where nif=elnif;
    escribir('Name '||auxnombre||' is registered as a receptionist');
    insert into emprecepcion values(elnif);
EXCEPTION
    WHEN no_data_found then
        insert into empleado(nif) values(elnif);
        insert into emprecepcion values(elnif);
        escribir('This employee did not exist. It has been registered as a receptionist. The employee data must be completed');
    WHEN pasa_tope then
        escribir('Maximum number of receptionists reached.');
END;
Session 8 - Exercises
Functions, procedures and exceptions.

NOTE: For each exercise, you must deduce whether you need to create a function or a procedure.

1) Given a service code (código de servicio), obtain its price, using exceptions to show a message if such code does not exist for a service.

2) Given a category name and type of stay (régimen alimenticio), obtain the average price of such category and type among all seasons (temporadas).

3) Obtain a list containing each category together with its average price (among all seasons) for only accommodation (SA), accommodation plus breakfast (AD), half-board (MP) and full-board (PC). You must take the previous function as a basis to do this exercise.

4) Given a date and a category, obtain the total number of rooms that have been booked in that date and category.

5) Given a booking identifier (código de reserva) and a room number, obtain the total expenses for the services that have been associated to that room in the specified booking.

6) Given a booking code as input, and taking the code you developed for the previous exercise as a basis, obtain the total expenses for the services associated to that booking code.

7) Create a table called clienteVIP with a single column, NIF. Such column is the primary key of the table and also foreign key to the table CLIENTE.

8) Given a customer’s NIF and a year, obtain the total expenses for services that such client has accumulated through all his/her bookings of the given year.

9) Given X value as input, and taking the code from the previous exercise as a basis, insert in the table clientesVIP, the NIF of all the customers who spent more than X on services associated to the rooms they booked in the current year.
Session 9
CREATE A TRIGGER

Creating a trigger associated to an operation for manipulating data in a table T means to define several actions that will be automatically executed when a particular operation is done in table T. It can be useful for completing the values of the derived columns (i.e., which has a value calculated taking the value of other columns) or for establishing complex restrictions (i.e., they cannot be established through CHECK restrictions).
**When do the actions associated to a trigger take place?**
We can indicate that these actions take place just before the operation in table T (BEFORE), or just after (AFTER).

**How many times are these actions executed?**
- If we indicate FOR EACH ROW: once for each row that is affected by the operation triggered by the TRIGGER.
- If we do not indicate anything, or if we indicate FOR EACH STATEMENT, the actions will be executed only once for each operation.

Within the trigger we can use conditions for executing different codes depending on the type of statement that activated the trigger.

- IF INSERTING returns true if the trigger was activated by an INSERT statement
- IF DELETING returns true if the trigger was activated by a DELETE statement
- IF UPDATING returns true if the trigger was activated by an UPDATE statement
- IF UPDATING(column_name) returns true if the trigger was activated by an UPDATE statement and column_name was updated.

When **INSERT** or **UPDATE** operations are performed, Oracle **automatically** controls one variable, :new that it is used to refer to the new inserted rows or to the rows after being modified.

For instance, if we are inserting in “empleados” and we want to refer to the “dni” of each inserted row, within FOR EACH ROW, we will refer to dni using :new.dni

When performing **DELETE** or **UPDATE** operations, for referring to the values that were deleted, we will use :old. (:old.dni)
Trigger for controlling that the generalization for EMPLEADO is disjoint.

As an example we show the trigger that controls the employees that we insert into emprecepcion are not already in tables empanimacion, emplimpieza, emprestaurante and empservicios. For capturing that the generalization is disjoint we should define similar triggers in the other tables: empanimacion, emplimpieza, emprestaurante and empservicios.

create or replace trigger disj_emprecepcion
before insert or update of nif
on emprecepcion
for each row
declare cuantos number(1);
begin
    cuantos:=0;
    select count(*) into cuantos from empanimacion where nif=:new.nif;
    if (cuantos=1)
        then raise_application_error(-20601,'El nif '||:new.nif|| ' ya es empleado de animación');
    end if;
    select count(*) into cuantos from emplimpieza where nif=:new.nif;
    if (cuantos=1)
        then raise_application_error(-20601,'El nif '||:new.nif|| ' ya es empleado de limpieza');
    end if;
    select count(*) into cuantos from emprestaurante where nif=:new.nif;
    if (cuantos=1)
        then raise_application_error(-20601,'El nif '||:new.nif|| ' ya es empleado de restaurante');
    end if;
    select count(*) into cuantos from empservicios where nif=:new.nif;
    if (cuantos=1)
        then raise_application_error(-20601,'El nif '||:new.nif|| ' ya es empleado de servicios');
    end if;
end;
Several observations

- We should not execute the code of a trigger in combination with other statements (create table, insert,...). If so, we may obtain syntax errors.
- Within a trigger we cannot execute sentences such as CREATE TABLE.
- When using IF UPDATING(column_name), the name of the column should be written with single quotes.
- If we specify WHEN condition, if we use the variables :new and :old, such variable will be written without ":".

---

**Example 2**

Let's suppose that we have the following table

```
COMPROBAR_ESTUDIOS(NIF char(9), estudios varchar(50))
```

C.P.: NIF
Clave ajena: NIF→ EMPLEADO

When the data for an employee is inserted and it has value for the column “estudios”, we want to insert in the table COMPROBAR_ESTUDIOS the NIF and the studies that the employee indicates.

```sql
CREATE OR REPLACE TRIGGER control_estudios
AFTER INSERT ON empleado
FOR EACH ROW
WHEN (new.estudios IS NOT NULL)
BEGIN
    INSERT INTO comprobar_estudios VALUES (:new.nif, :new.estudios);
END;
```

Check what happens when we put BEFORE instead of AFTER.
Session 9 - Exercises
Create a trigger.

1) Create a trigger to control the prices for each room category in each season fulfilling the following: pSA<pAD<pMP<pPC.
   If it is not like this, the operation that makes this incorrect should be avoided. Moreover, a message indicating what is wrong should be shown.
   
   *You must check whether the trigger works ok in case any value is null.*

2) Create a trigger to control that the SUITE rooms (room of type SUITE) are only on the fifth floor.

3) Create a trigger to control the following issue: when a person books an individual room, such room cannot have a suppletory bed (cama supletoria).

4) When a customer books a meeting with a service employee for a specific treatment, you should control if this employee does that treatment and, if not, the booking has to be avoided for such treatment. The following message has to be shown: "Employee X is not in charge of treatment Y" (where X and Y are the name of the employee and the name of the treatment, respectively).

5) Create a trigger to control the following situation: when a cleaning employee has to be on duty and he/she is going to be on duty for another 5 days in the same month, the following message should be shown:

"X is quite busy in the month of Y" (where X refers to the name of the employee and Y is the name of the month).

Original exercise in Spanish: Cuando a un día se le vaya a asignar de guardia un empleado de limpieza que ya vaya a estar de guardia otros 5 días del mismo mes, se mostrará un mensaje indicando "X está bastante argado de guardias el mes Y" (donde en X figurará el nombre del empleado y en Y el del mes).
Session 10
CREATE A TRIGGER II

Some observations about the syntax we learnt in the previous session.

Execution order:

If there are several triggers for the same statement, the order for executing them is:
1. BEFORE for the statement
2. BEFORE for the row
3. AFTER for the row
4. AFTER for the statement

Mutant table:

It is a table that is being modified by an INSERT, an UPDATE or a DELETE statement. The row type triggers ("for each row") have problems when reading or modifying the rows of a mutant table.
Let’s take the last exercise from the previous lab session:

create or replace trigger control_limpieza
after insert or update on calendario
for each row
declare
cuantos number(2);
auxnombre empleado.nombre%type;
begin
select count(*) into cuantos from calendario
where emplimpieza=:new.emplimpieza and to_char(fecha,'MM')=to_char(:new.fecha,'MM') and
  to_char(fecha,'yyyy')=to_char(:new.fecha,'yyyy');
if (cuantos>5) then
  select nombre into auxnombre from empleado where nif=:new.emplimpieza;
  escribir(auxnombre||'  ya esta bastante cargado el mes de '||to_char(:new.fecha,'Month'));
end if;
end;

If we try to update or insert something into the table CALENDARIO, we get the following error:

Row 6: ORA-04091: la tabla CALENDARIO está mutando, puede que el disparador/la función no puedan verla
ORA-06512: en “dbd.control_limpieza”
ORA-04088: error durante la ejecución del disparador ‘DBDcontrol_limpieza’
ORA-06512: en línea 1

As we can see, we cannot insert or update. Querying the table that suffers the operation from a row type trigger is not always allowed. In this case, the trigger has to be designed differently.
If the trigger was only designed for INSERT, being defined using BEFORE instead of AFTER, it can be solved. As it is also for UPDATE, this does not occur.

We can use an auxiliary table that updates through a trigger of “statement” type -- *for each statement*— (using BEFORE, that is, it would update with the first triggers associated to the statement) and use such information with the row type trigger.

Create table auxlimpieza (niflimpieza char(9),mes char(2), total number(2));

create or replace trigger limpieza_mes
before insert or update on calendario
begin
    delete from auxlimpieza;
    insert into auxlimpieza
        select emplimpieza, to_char(fecha,'MM'), count(*) from calendario
        where emplimpieza is not null
        group by emplimpieza, to_char(fecha,'MM');
end;

create or replace trigger limpieza
before insert or update on calendario
for each row
declare
    cuantos number(2);
    auxnombre empleado.nombre%type;
begin
    select total into cuantos
from auxlimpieza where niflimpieza=:new.emplimpieza and mes=to_char(:new.fecha,'MM')
and to_char(fecha,'yyyy')=to_char(:new.fecha,'yyyy');

if (cuantos>=5) then
    select nombre into auxnombre from empleado where nif=:new.emplimpieza;
    escribir(auxnombre||' ya esta bastante cargado el mes de '||to_char(:new.fecha,'Month'));
end if;
exception
when no_data_found then null;
end;

ACTIVATING/DISACTIVATING TRIGGERS

ALTER TRIGGER trigger_name ENABLE;
ALTER TRIGGER trigger_name DISABLE;
Session 10 - Exercises
Create a trigger II

1) For children, adults and for all activities, we have to control when one activity substitutes another, it will not substitute itself. If this happens (one activity is trying to substitute itself), a message indicating that the operation is not possible should appear and the execution should be aborted.

Spanish translation:
Se debe controlar para las actividades de niños, adultos y para todos, que cuando una actividad sustituye a otra, no esté sustituyéndose a sí misma. Si se diera el caso de intentar sustituirse a sí misma, se mostrará un mensaje indicando que no es posible y se abortaría la operación.

2) When inserting or updating the minimum surface of a room category, we should control that the minimum surface is lower than the maximum, if the maximum surface exists. Otherwise, an error message will be shown, and the operation would not be allowed.

3) When a new activity is inserted on a day that has already programmed 4 activities, the operation will not be allowed and the following message should be shown: “The activity X has to be done another day”, where X contains the description of the activity.

4) When a customer booking is inserted, we have to check if that customer already has another room for the same day at the same hotel. If he/she has already booked rooms, a message with the characteristics of all the rooms that he/she has already booked for the same day should be shown (the characteristics refer to the room numbers, room type, type of meals – *regimen alimentario* and additional bed – *cama supletoria*).

5) Create a table

```
citas_diarias(nif, fecha, total)
clave primaria: (nif, fecha)
total es number(1)
```

This table has to be automatically updated, containing the total number of appointments that an employee has per day. Inserting new appointment, cancelations, and possible employee changes should be taken into account as well.
Session 11

**CREATE AN INDEX** for defining an index in a table

An index is a secondary memory structure that allows direct access to the rows of a table. It decreases the response time for a query, thus improving its performance and optimizing the result. Its handling is made in an intelligent manner. Oracle decides which index is necessary. By default, Oracle always creates an index for each primary key that is defined, as well as for each UNIQUE restriction.

Example:

```
CREATE INDEX ind_prov_cliente ON cliente(provincia);
```

For dropping an index

```
DROP INDEX Index_name;
```

When a statement is executed in Oracle, the optimizer module analyzes several strategies and chooses one of them, which is not necessary the best one, but one that is good enough according to a specific threshold. Therefore, this means that an index that we have defined may not be necessarily used, even though it could lead to the best possible strategy for executing an action. On the other hand, every time an index is defined, this means that there will be an additional cost in the "insert", "update", or "delete" operations associated to a table to maintain the index. For all these reasons, we do not define an index for all columns in a table.

How can we know if an index we have defined is being used in a query?... We have to use **EXPLAIN PLAN**
EXPLAIN PLAN for storing the execution plan of a statement

Insert a row describing each step in the execution plan of a statement in the table PLAN_TABLE. This “insert” is not executed.

The PLAN_TABLE contains:
- Statement_id varchar2(30)  Optional value for identifying execution plans
- Operation varchar2(30)  Name of the internal operation that was performed
- Options varchar2(225)  Operation details
- Object_name varchar2(30)  Table or index name
- Position numeric
  - The first row indicates the cost that the optimizer module has estimated to perform the statement.
  - For the remaining, the relative position with respect to the father is indicated.

EXPLAIN PLAN 'statemnt_name' FOR statement INTO table-name;

Statement identifier. By default NULL.

Table where the output is deposit. By default PLAN_TABLE.

Statement for which we want to know the execution plan: SELECT, INSERT, UPDATE...
Example:

**CASE 1: In the table CLIENTE there is only one index**, the one that corresponds to the primary key of this table. We execute the following statements:

```sql
explain plan
set statement_id='cli1'
for select * from cliente where provincia='Alicante';
```

```sql
select operation, options, object_name, position
from plan_table
where statement_id='cli1'
```

<table>
<thead>
<tr>
<th>OPERATION</th>
<th>OPTIONS</th>
<th>OBJECT_NAME</th>
<th>POSITION</th>
</tr>
</thead>
<tbody>
<tr>
<td>SELECT STATEMENT</td>
<td></td>
<td></td>
<td>3</td>
</tr>
<tr>
<td>TABLE ACCESS</td>
<td>FULL</td>
<td>CLIENTE</td>
<td>1</td>
</tr>
</tbody>
</table>

2 rows selected

**CASE 2: If we define an index for the column "provincia"**

And we now execute:

```sql
explain plan
set statement_id='cli2'
for select * from cliente where provincia='Alicante';
```

```sql
select operation, options, object_name, position
from plan_table
where statement_id='cli2'
```

```sql
create index ind_prov_cliente on cliente(provincia);
```

<table>
<thead>
<tr>
<th>OPERATION</th>
<th>OPTIONS</th>
<th>OBJECT_NAME</th>
<th>POSITION</th>
</tr>
</thead>
<tbody>
<tr>
<td>SELECT STATEMENT</td>
<td></td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>TABLE ACCESS</td>
<td>BY INDEX ROWID</td>
<td>CLIENTE</td>
<td>1</td>
</tr>
<tr>
<td>INDEX</td>
<td>RANGE SCAN</td>
<td>IND_PROV_CLIENTE</td>
<td>1</td>
</tr>
</tbody>
</table>

3 rows selected

Indexes are not used

The index IND_PROV_CLIENTE is used
**GRANT**

It is used to grant privileges. We can define privileges for specific columns, as well as other options that are not going to be learnt in this course.

The privilege list refers to SELECT, INSERT, UPDATE, ALTER, ...

If we do not specify WITH GRANT OPTION, the user that is granted with the privileges can also grant them to other users.

If we state PUBLIC instead of providing a user name, the privileges will be granted to all users.

**REVOKE**

It is used to withdraw the privileges that were granted. As what happens with GRANT, there are many other options that are not taken into consideration in this brief summary.
Session 11 - Exercises
GRANT, REVOKE and CREATE AN INDEX

Understanding how GRANT and REVOKE work

1) Open 2 working sessions, one with your usual user and another with the user the teacher indicates.

2) We will refer to the **usual user** (the one you have been working with in all the previous sessions) as **grantor** ("otorgante"), while the other user will be the **recipient** ("receptor").

The recipient does not have his/her own tables. You can check this in the table tab of the SQLDeveloper, or if you run `Select * from user_tables`.

3) **With the recipient user, execute**

   ```sql
   SELECT * from empleado  (the table does not exist)
   ```

4) **With the recipient user, execute**

   ```sql
   SELECT * from grantor.empleado  (the table does not exist)
   ```

   (remember that in grantor you should write your usual user)

5) Now **from the grantor user**, grant privileges to the recipient in order for him/her to be able to do `SELECT` to his/her EMPLEADO table.

6) **From the recipient session**, execute

   ```sql
   SELECT * from empleado  (the table does not exist)
   ```

   ```sql
   SELECT * from grantor.empleado
   ```

   *If you have correctly granted the privileges, you should get some rows.*

7) **From the recipient session** try to insert the employee "00000007A" whose name is "CARLOS RUIZ" and lives in "calle ESTRELLA 35" in "POLOP". *(You should not be allowed to do the insertion)*

8) **From the grantor session**, grant privileges for inserting in the table EMPLEADO to the recipient user.

9) **From the recipient session** try to insert the employee "00000007A" whose name is "CARLOS RUIZ" and lives in "calle ESTRELLA 35" in "POLOP". *(If you have granted the permission correctly, you should be allowed to do the insertion)*.

10) **From the grantor session**, remove the privilege for inserting into the table EMPLEADO that was given to the recipient user.
11) **From the recipient session**, insert the employee "00000008B" whose name is "ROSA GUARDIOLA" and lives in "calle LUCERO 47" in "SAN JUAN" into the table EMPLEADO from the granter user. *(If you have correctly removed the privileges, you should not be allowed to do the insertion).*

## Working with indexes

12) Create a table called TRATAMIENTO_CARO

```
TRATAMIENTO_CARO(codigo char(5), precio number(3))
```

Primary key: codigo

13) Think, but do not execute, which would be the statement for inserting (into the table TRATAMIENTO_CARO), in only one operation, the code (codigo) and price (precio) for the treatments which has a price higher than 20 euros.

14) Look at what would be the execution plan for the statement from the previous exercise.

15) Define an index for the column "precio" of the table TRATAMIENTO. The execution plan for the statement in exercise 13. Would it be the same as if the index had not been created?