**Content:**

[Equestrian 3](#_Toc122420856)

[Database Tables Description 4](#_Toc122420857)

[Entities 4](#_Toc122420858)

[Regions 4](#_Toc122420859)

[Countries 5](#_Toc122420860)

[Cities 5](#_Toc122420861)

[Arenas 5](#_Toc122420862)

[Disciplines 6](#_Toc122420863)

[Events 6](#_Toc122420864)

[Industries 6](#_Toc122420865)

[Sponsors 7](#_Toc122420866)

[Sponsors-Events 7](#_Toc122420867)

[Salutation 7](#_Toc122420868)

[Activities 8](#_Toc122420869)

[Clubs 8](#_Toc122420870)

[Contestants 8](#_Toc122420871)

[Categories 9](#_Toc122420872)

[Boxes 9](#_Toc122420873)

[Registrations 9](#_Toc122420874)

[Types 10](#_Toc122420875)

[Breeds 10](#_Toc122420876)

[Gender 10](#_Toc122420877)

[Age-Groups 11](#_Toc122420878)

[Horses 11](#_Toc122420879)

[Rewards 12](#_Toc122420880)

[Ranking 12](#_Toc122420881)

[Audits 12](#_Toc122420882)

[Sequences 13](#_Toc122420883)

[Stored Procedures 13](#_Toc122420884)

[Stored Procedure: SP\_INSERT\_REGION 14](#_Toc122420885)

[Stored Procedure: SP\_INSERT\_COUNTRY 14](#_Toc122420886)

[Stored Procedure: SP\_INSERT\_CITY 14](#_Toc122420887)

[Stored Procedure: SP\_INSERT\_ARENA 14](#_Toc122420888)

[Stored Procedure: SP\_INSERT\_EVENT 14](#_Toc122420889)

[Stored Procedure: SP\_INSERT\_SPONSOR 15](#_Toc122420890)

[Stored Procedure: SP\_INSERT\_EVENT\_SPONSOR 15](#_Toc122420891)

[Stored Procedure: SP\_INSERT\_CLUB 15](#_Toc122420892)

[Stored Procedure: SP\_INSERT\_ CONTESTANT, 15](#_Toc122420893)

[Stored Procedure: SP\_INSERT\_HORSE 16](#_Toc122420894)

[Stored Procedure: SP\_INSERT\_REGISTRATION 16](#_Toc122420895)

[Stored Procedure: SP\_INSERT\_RANK 16](#_Toc122420896)

[Stored Procedure: SP\_DELETE\_CONTESTANT 17](#_Toc122420897)

[Stored Procedure: SP\_UPDATE\_REGS 17](#_Toc122420898)

[Functions 17](#_Toc122420899)

[User Defined Function: UDF\_BOOKED\_BOXES 17](#_Toc122420900)

[User Defined Function: UDF\_GET\_CLUBS\_BUDGET 17](#_Toc122420901)

[User Defined Function: UDF\_GET\_HORSE\_OWNER 18](#_Toc122420902)

[User Defined Function: UDF\_GET\_RIDER\_IDCARD 18](#_Toc122420903)

[User Defined Function: UDF\_GET\_RIDER\_NAME\_AGE 18](#_Toc122420904)

[User Defined Function: UDF\_RIDERS\_PER\_CITY 18](#_Toc122420905)

[Triggers 18](#_Toc122420906)

[Trigger: TR\_BEFORE\_INSERT\_HORSE 18](#_Toc122420907)

[Trigger: TR\_BEFORE\_INSERT\_RANK 18](#_Toc122420908)

[Trigger: TR\_CHECK\_BOXES\_COUNT 19](#_Toc122420909)

[Trigger: TR\_SET\_NULL\_OR\_DELETE\_RIDER 19](#_Toc122420910)

[Trigger: TR\_CONTESTANT\_AUDIT 19](#_Toc122420911)

[Useful Select Statements 19](#_Toc122420912)

[Abbreviations 19](#_Toc122420913)

# Equestrian

**A Database** is an organized collection of structured data, corresponding to the real world, stored in tables, files, docs, etc. In most common scenario data is organized in tables and we call that type: Relational Database – the most widely used database model.

**RDBMS** is a software for creating, maintaining, manipulating, administrating, and controlling relational databases. It takes the role of an interface between users, applications and RDBs.

**Databases** is used for organizing data and optimizing its size; to control the accesses; to prevent data violations and disruption; to control data validation; to control data accuracy; to provide data integrity, security, and privacy; to remove redundancy, and many more.

**Oracle SQL Developer** is a GUI database programming and administration tool

This document describes a sample Oracle Database named Equestrian, reached through Oracle SQL Developer.

It contains 23 tables + 1 in addition, created for auditing purposes.

The aim of the project is to describe and reflect the process of holding Horse Race Tournaments and Competitions according to the Equestrian Federation.

**Equestrian ER Diagram**

Diagram, schematic

Description automatically generated

To access the database, you should first open SQL Developer and login with

User: \*\*\*\*\*\*\*\*\*\*\* and password: \*\*\*\*\*\*\*\*, like shown below:

**Login Screen**

Graphical user interface

Description automatically generated

# Database Tables Description

Tables are **database objects that contain all the data in a database and usually called entities.** They represent a binary structure of records/entities (rows) and attributes (columns). Each table contains information for a specific object (something we collect data for). Tables in RDB have names, Primary Key, and Foreign Key. All tables in one RDB are logically connected via specific relationships.

All entities in a database are created under set of Data Rules, Conditions, and constraints:

* **Primary Key Constraint** - a rule applied to a table attribute, obliged it to contain unique and unrepeatable values for each record, where the value cannon be Null.
* **Foreign Key Constraint** – a rule set to a table attribute, giving it the right to apply control over data entry, reduce the repeatability and establish relationships with other tables.
* **Unique Constraint -** a rule applied to a table attribute, saying that all values must be unique, where Null is also allowed.
* **Check Constraint:** Validation of entered data, whether it meets specific condition set for the attribute.
* **Type of data:** TEXT (CHAR | VARCHAR), NUMBER, DATE, TIMESTAMP, etc. – Datatype defines the value range; how much memory data will take, and what operations could be executed with it.
* **? NULL / NOT NULL** **Constraint** and many more.

# Entities

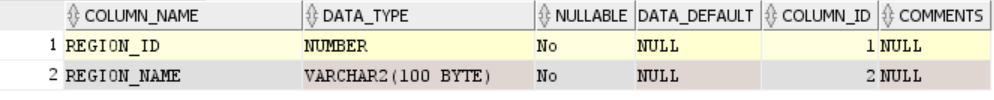
## Regions

The entity has two attributes:

Region ID, datatype NUMBER as a Primary Key and

Region Name, datatype VARCHAR2(100) that must be Unique.

There is a sequence created for the Region ID, named SQ\_Region\_ID. For Inserting records into Regions, you should use a Stored Procedure: SP\_Insert\_Region, described below further at this document.



## Countries

This entity has bellow four attributes:

Country ID, datatype NUMBER as a Primary Key,

Country Code, datatype CHAR (2) that must be Unique and cannot be NULL,

Country Name, datatype VARCHAR2(100) that must be Unique and cannot be NULL and

Region ID, datatype NUMBER as a Foreign Key to Regions.

There is also a sequence created for the Country ID, named SQ\_Country\_ID.

For Inserting records into Countries, you should use a Stored Procedure: SP\_Insert\_Country, described below further at this document.

Table

Description automatically generated

## Cities

This entity has three attributes:

City ID, datatype NUMBER as a Primary Key,

City Name, datatype VARCHAR2(100) that cannot be NULL and

Country ID, datatype NUMBER as a Foreign Key to Countries.

There is a sequence created for the City ID, named SQ\_City\_ID. For Inserting records into Cities, you should use a Stored Procedure: SP\_Insert\_City, described below further at this document.

Table

Description automatically generated

## Arenas

Entity ARENAS has four attributes:

Arena ID, datatype NUMBER as a Primary Key,

Arena Name, datatype VARCHAR2(50) that cannot be NULL,

Boxes, datatype NUMBER which cannot be NULL and

City ID, datatype NUMBER as a Foreign Key to Cities.

There is a sequence created for the Arena ID, named SQ\_Arena\_ID. For Inserting records into Arenas, you should use a Stored Procedure: SP\_Insert\_Arena, described below further at this document.

Table

Description automatically generated

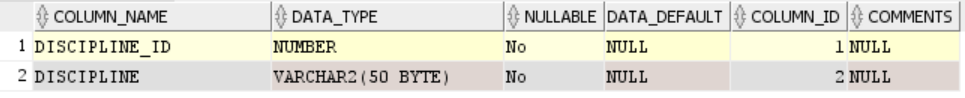
## Disciplines

The entity has two attributes:

Discipline ID, datatype NUMBER as a Primary Key and

Discipline, datatype VARCHAR2(50) that cannot be NULL and must be Unique.

There is a sequence created for the Discipline ID, named SQ\_ Discipline\_ID.



To insert a record into Disciplines, use below DML statement:

INSERT INTO Disciplines VALUES (SQ\_DISCIPLINE\_ID.nextval, 'Discipline Name');

## Events

It contains six attributes:

Event ID, datatype NUMBER as a Primary Key,

Event Name, datatype VARCHAR2(100) which cannot be NULL,

Discipline ID, datatype NUMBER as a Foreign Key to Disciplines,

Event Date, datatype DATE,

About Event, datatype VARCHAR2(255) which cannot be NULL and

Arena ID, datatype NUMBER as a Foreign Key to Arenas.

There is a sequence created for the Event ID, named SQ\_Event\_ID. For Inserting records into Events, you should use a Stored Procedure: SP\_Insert\_Event, described below further at this document.

Table

Description automatically generated

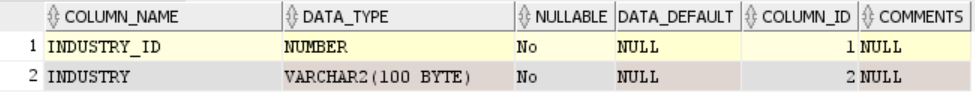
## Industries

It contains two attributes:

Industry ID, datatype NUMBER as a Primary Key and

Industry, datatype VARCHAR2(100) that cannot be NULL and must be Unique.

There is a sequence created for the Discipline ID, named SQ\_ Industry \_ID.



To insert a record into Industries, use below DML statement:

**INSERT INTO Industries** **VALUES (SQ\_INDESTRY\_ID.nextval, '** **Industry Name');**

Also check the Stored Procedure named SP\_Isert\_Sponsor, described below further at this document.

## Sponsors

It contains six attributes:

Sponsor ID, datatype NUMBER as a Primary Key,

Sponsor Name, datatype NVARCHAR2(100) which cannot be NULL,

Contact Name, datatype NVARCHAR2(100) which cannot be NULL,

Phone, datatype VARCHAR2(15) which cannot be NULL and must be unique,

Industry ID, datatype NUMBER as a Foreign Key to Industries and

About Sponsor, datatype VARCHAR2(255).

There is a sequence created for the Sponsor ID, named SQ\_Sponosr\_ID. For Inserting records into Sponsors, you should use a Stored Procedure named SP\_Isert\_Sponsor, described below further at this document.

Table

Description automatically generated

## Sponsors-Events

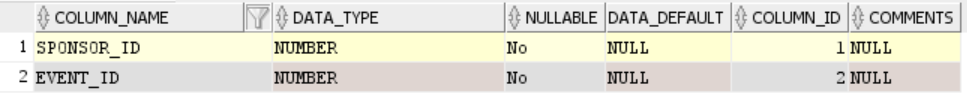
It’s a mapping table, also called association table, bridge table, cross-reference table, intermediary table, intersection table, join table, junction table, link table, map table and few more.[[1]](#footnote-1)

Sponsor-Event contains two attributes as a composite primary key[[2]](#footnote-2).

Sponsor ID, datatype NUMBER as a Primary Key and Foreign Key to Sponsors,

Event ID, datatype NUMBER as a Primary Key and Foreign Key to Events.

For Inserting records into Sponsors\_Events, you should use a Stored Procedure named SP\_Isert\_Event\_Sponsor, described below further at this document.



## Salutation

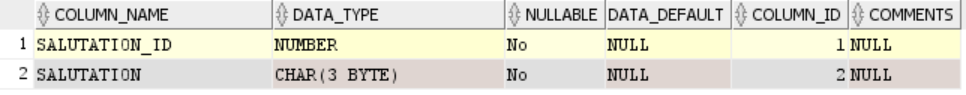
It contains two attributes:

Salutation ID, datatype NUMBER as a Primary Key and

Salutation, datatype CHAR (3) that cannot be NULL and must be Unique.

There is no need of a Sequence because of the limited records.

Anyway, if needed, use a simple INSERT INTO DML Statement.



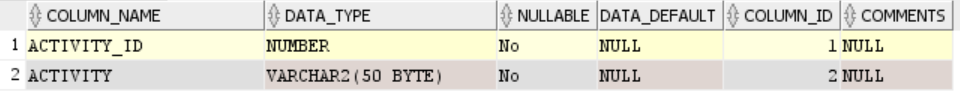
## Activities

It contains two attributes:

Activity ID, datatype NUMBER as a Primary Key and

Activity, datatype VARCHAR2(50) that cannot be NULL and must be Unique.

There is a sequence created for the Activity ID, named SQ\_ Activity \_ID.



To insert a record into Activities, use below DML statement:

**INSERT INTO Activities VALUES (SQ\_ACTIVITY\_ID.nextval, 'Activity Name');**

## Clubs

It contains six attributes:

Club ID, datatype NUMBER as a Primary Key,

Club Name, datatype NVARCHAR2(100) which cannot be NULL,

Chairman, datatype NVARCHAR2(100) which cannot be NULL,

Activity ID, datatype NUMBER as a Foreign Key to Activities,

Budget, datatype NUMBER which cannot be NULL and

City ID, datatype NUMBER as a Foreign Key to Cities.

There is a sequence created for the Sponsor ID, named SQ\_Club\_ID. For Inserting records into Sponsors, you should use a Stored Procedure named SP\_Isert\_Club, described below further at this document.

Table

Description automatically generated

## Contestants

It contains eleven attributes:

Rider ID, datatype NUMBER as a Primary Key,

ID Card, datatype CHAR (10) which cannot be NULL and must be Unique,

First Name, datatype NVARCHAR2(100) which cannot be NULL,

Last Name, datatype NVARCHAR2(100) which cannot be NULL,

Salutation ID, datatype NUMBER as a Foreign Key to Salutations,

Email, datatype VARCHAR2(100) which cannot be NULL and must be Unique,

Phone, datatype VARCHAR2(15) which cannot be NULL and must be Unique,

Birth Date, datatype DATE which cannot be NULL,

Category ID, datatype NUMBER as a Foreign Key to Categories,

City ID, datatype NUMBER as a Foreign Key to Cities and

Club ID, datatype NUMBER as a Foreign Key to Clubs.

Table

Description automatically generated

There is a sequence created for the Contestant ID, named SQ\_Contestant\_ID. For Inserting records into Sponsors, you should use a Stored Procedure named SP\_Isert\_Contestant, described below further at this document.

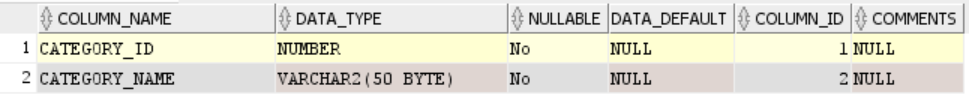
## Categories

It contains two attributes:

Category ID, datatype NUMBER as a Primary Key and

Category Name, datatype VARCHAR2(50) that cannot be NULL and must be Unique.

There is a sequence created for the Category ID, named SQ\_Category\_ID.



To insert a record into Categories, use below DML statement:

**INSERT INTO Categories VALUES (SQ\_CATEGORY\_ID.nextval, ‘Category Name');**

## Boxes

It contains three attributes:

Box ID, datatype NUMBER as a Primary Key,

Box Type, datatype VARCHAR2(50) which cannot be NULL and must be Unique and

Price Per Dey in EUR, datatype NUMBER (5,2) which cannot be NULL.

There is a sequence created for the Box ID, named SQ\_BOX\_ID.

Table

Description automatically generated

To insert a record into Boxes, use below DML statement:

**INSERT INTO Boxes VALUES (SQ\_BOX\_ID.nextval, ‘Box Type’, ‘Price per Day’);**

## Registrations

It contains six attributes:

Reg ID, datatype NUMBER as a Primary Key,

Rider ID, datatype NUMBER as a Foreign Key to Contestants,

Event ID, datatype NUMBER as a Foreign Key to Events,

Box ID, datatype NUMBER as a Foreign Key to Boxes,

Reg Date, datatype DATE which cannot be NULL and

Cancel Date, datatype DATE DEFAULT NULL.

Table

Description automatically generated

There is a sequence created for the Reg ID, named SQ\_Reg\_ID. For Inserting records into Sponsors, you should use a Stored Procedure named SP\_Isert\_Registration, described below further at this document.

## Types

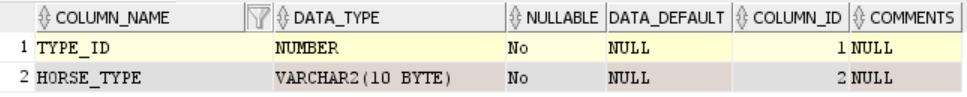
It contains two attributes:

Type ID, datatype NUMBER as a Primary Key and

Horse Type, datatype VARCHAR2(10) that cannot be NULL and must be Unique.

There is no need of a Sequence because of the limited records.

Anyway, if needed, use a simple INSERT INTO DML Statement.



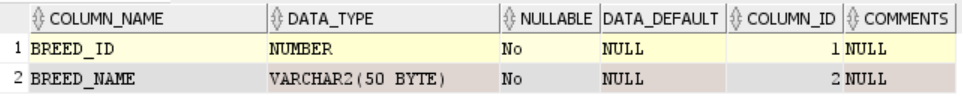
## Breeds

It contains two attributes:

Breed ID, datatype NUMBER as a Primary Key and

Breed Name, datatype VARCHAR2(50) that cannot be NULL and must be Unique.

There is a sequence created for the Breed ID, named SQ\_Breed\_ID.



To insert a record into Breeds, use below DML statement:

**INSERT INTO Breeds VALUES (SQ\_BREED\_ID.nextval, ‘Breed Name')**

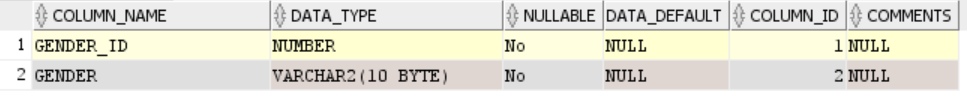
## Gender

It contains two attributes:

Gender ID, datatype NUMBER as a Primary Key and

Gender Name, datatype VARCHAR2(10) that cannot be NULL and must be Unique.

There is no need of a Sequence because of the limited records.



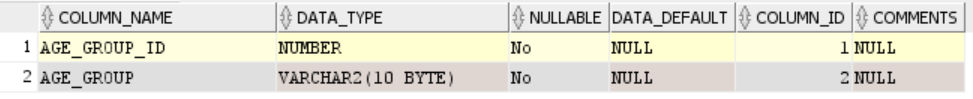
Age-Groups

It contains two attributes:

Age-Group ID, datatype NUMBER as a Primary Key and

Age-Group, datatype VARCHAR2(10) that cannot be NULL and must be Unique.

There is a sequence created for the Age Group ID, named SQ\_AgeGroup\_ID.



To insert a record into Age Groups if neded, use below DML statement:

**INSERT INTO Age\_Groups VALUES (SQ\_AgeGroup\_ID.nextval, ‘Age Group')**

## Horses

It contains ten attributes:

Horse ID, datatype NUMBER as a Primary Key,

Reg Number, NUMBER which cannot be NULL and must be Unique,

UELN, NUMBER which cannot be NULL and must be Unique,

Horse Name, datatype NVARCHAR2(50) which cannot be NULL,

Type ID, datatype NUMBER as a Foreign Key to Types,

Breed ID, datatype NUMBER as a Foreign Key to Breeds,

Gender ID, datatype NUMBER as a Foreign Key to Gender,

Birth Date, datatype DATE which cannot be NULL,

Age Group ID, datatype NUMBER as a Foreign Key to Age Groups,

Rider ID, datatype NUMBER as a Foreign Key to Riders.

Table

Description automatically generated

There is a sequence created for the Horse ID, named SQ\_Horse\_ID, same as for the Horse Reg Number. For Inserting records into Horses, you should use a Stored Procedure named SP\_Isert\_Horse, described below further at this document.

## Awards

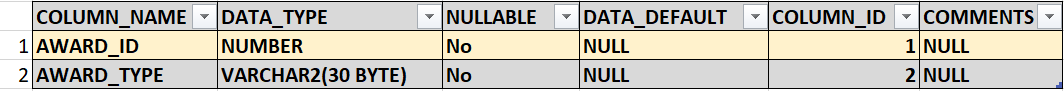
It contains two attributes:

Award ID, datatype NUMBER as a Primary Key and

Award Type, datatype VARCHAR2(30) that cannot be NULL and must be Unique.

There is no need of a Sequence because of the limited records.

Anyway, if needed, use a simple INSERT INTO DML Statement.



## Awarding

It contains five attributes and 2-columns composite PK:

Horse ID, datatype NUMBER as a Primary Key and Foreign Key to Horses,

Event ID, datatype NUMBER as a Primary Key and Foreign Key to Events,

Total Score, datatype NUMBER which cannot be NULL,

Award ID, datatype NUMBER as a Foreign Key to Awards and

Money Prize in EUR, datatype NUMBER (15,2) DEFAULT 0.

Table

Description automatically generated

For Inserting records into Awarding, you should use a Stored Procedure named SP\_Isert\_Award, described below further at this document.

## Audits

It contains five attributes:

Audit ID, datatype NUMBER as a Primary Key,

Table Name, datatype VARCHAR2(225),

Transaction Name, VARCHAR2(10),

User Name, datatype VARCHAR2(30) and

Transaction Date, datatype DATE.

Table

Description automatically generated

Filling in the table is an auto process, described down below.

# Sequences

Sequence is a database object, that represents a sequence of integers that can be used by any table or view. A Sequence's values can be accessed using the NEXTVAL. Below are all Sequences created to database, Horse Race Tournament and should be used anytime when a new record is about to be inserted:

To call any Sequence use its name and then .nextval, like: **SEQUENCE\_NAME.nextval**

**SQ\_Region\_ID START WITH 1;**

**SQ\_Country\_ID START WITH 1;**

**SQ\_City\_ID START WITH 1;**

**SQ\_Arena\_ID START WITH 1;**

**SQ\_Event\_ID START WITH 1;**

**SQ\_Indestry\_ID START WITH 1;**

**SQ\_Sponsor\_ID START WITH 1;**

**SQ\_Club\_ID START WITH 1;**

**SQ\_Contestant\_ID START WITH 1;**

**SQ\_Horse\_ID START WITH 1;**

**SQ\_Breed\_ID START WITH 1;**

**SQ\_Reg\_ID START WITH 1;**

**SQ\_Audit\_ID START WITH 1;**

**SQ\_Horse\_RegNumb START WITH 1;**

**SQ\_Discipline\_ID START WITH 5;**

**SQ\_Category\_ID START WITH 5;**

**SQ\_Box\_ID START WITH 3;**

**SQ\_AgeGroup\_ID START WITH 5;**

**SQ\_Activity\_ID START WITH 4;**

# Stored Procedures

A **Procedure** in PL/SQL is a subprogram unit that consists of a group of PL/SQL statements that can be called by name. Each procedure in PL/SQL has its own unique name by which it can be referred to and called. This subprogram unit in the Oracle database is stored as a database object.

They are compiled once and stored in an executable form. As a result, procedure calls are quick and efficient. Executable code is automatically cached and shared among users. This lowers memory requirements and invocation overhead.

Below are described several procedures created to use for our database:

## Stored Procedure: SP\_INSERT\_REGION

The role of this procedure is to support the Insert into Regions transaction.

It accepts Region Name as a TEXT for parameter.

Stored Procedure should be used for inserting a new record into Regions as follow:

**EXECUTE** SP\_Isert\_Region (‘Region Name’);

## Stored Procedure: SP\_INSERT\_COUNTRY

The role of this procedure is to support the Insert into Countries transaction.

It accepts three parameters:

Country Code as a TEXT – exact 2 letters,

Country Name as a TEXT and

Region Name as a TEXT.

Stored Procedure should be used for inserting a new record into Countries as follow:

**EXECUTE** SP\_Isert\_Country(‘Country Code’, ‘Country Name’, ‘Region Name’);

## Stored Procedure: SP\_INSERT\_CITY

The role of this procedure is to support the Insert into Cities transaction.

It accepts two parameters:

City Name as a TEXT and

Country Code as a TEXT

Stored Procedure should be used for inserting a new record into Cities as follow:

**EXECUTE** SP\_Isert\_City (‘City Name’, ‘Country Code’);

## Stored Procedure: SP\_INSERT\_ARENA

The role of this procedure is to support the Insert into Arenas transaction.

It accepts three parameters:

Arena Name as a TEXT,

Number of boxes as a NUMBER and

City Name as a Text.

Stored Procedure should be used for inserting a new record into Arena as follow:

**EXECUTE** SP\_Isert\_Arena (‘Arena Name’, Numb of Boxes, ‘City Name’);

## Stored Procedure: SP\_INSERT\_EVENT

The role of this procedure is to support the Insert into Events transaction.

It accepts five parameters:

Event Name as a TEXT,

Discipline ID as a NUMBER – between 1 and 4,

Event Date as a DATE,

About Event as a TEXT and

Arena ID as a NUMBER.

Stored Procedure should be used for inserting a new record into Events as follow:

**EXECUTE** SP\_Isert\_Event(‘Event Name’, Discipline ID,

**TO\_DATE** (‘Event Date’,’dd-mm-yyyy’), ‘About Event’, Arena ID);

## Stored Procedure: SP\_INSERT\_SPONSOR

The role of this procedure is to support the Insert into Sponsors transaction.

It accepts five parameters:

Sponsor Name as a TEXT,

Contact Name as a TEXT – First Name and Last Name

Phone – up to 15 characters that needs to match any of the **Phone Patterns**

(Listed at the end of the document).

Industry as a TEXT and

About Sponsor as a TEXT.

Stored Procedure should be used for inserting a new record into Sponsors as follow:

**EXECUTE** SP\_Isert\_Sponsor (‘Sponsor Name’, ‘Contact Name’, ‘Phone’, ‘Industry’

‘About Sponsor’);

## Stored Procedure: SP\_INSERT\_EVENT\_SPONSOR

The role of this procedure is to support the Insert into Sponsors\_Events transaction.

It accepts two parameters:

Sponsor ID as a NUMBER and

Event ID as a NUMBER.

Stored Procedure should be used for inserting new records into Sponsors\_Events as follow:

**EXECUTE** SP\_Isert\_Event\_Sponsor(Sponsor ID, Event ID)

## Stored Procedure: SP\_INSERT\_CLUB

The role of this procedure is to support the Insert into Clubs transaction.

It accepts five parameters:

Club Name as a TEXT,

Chairman Name as a TEXT - First Name and Last Name,

Activity Name as a TEXT,

Budget as a NUMBER and

City Name as a TEXT

Stored Procedure should be used for inserting a new record into Clubs as follow:

**EXECUTE** SP\_Isert\_City (‘Club Name’, ‘Chairman’, ‘Activity’, Budget, ‘City’);

## Stored Procedure: SP\_INSERT\_ CONTESTANT,

The role of this procedure is to support the Insert into Contestants transaction.

It accepts ten parameters:

ID Card – exact 10 digits,

First Name as a TEXT,

Last Name as a TEXT,

Salutation – either ‘Mr.’ or ‘Ms.’,

Email as a TEXT,

Phone – up to 15 characters,

Birth Date as a Date,

Category Name – any of: "M/F", "Amateur", "Teens 14" or "Teens 18",

City Name as a TEXT and

Club Name as TEXT.

Stored Procedure should be used for inserting a new record into Contestants as follow:

**EXECUTE** SP\_Isert\_Contestant (‘ID Card’, ‘First Name’, ‘Last Name’, ‘Salutation’, ‘Email’, ‘Phone’, **TO\_DATE** (‘Birth Date’, ‘dd-mm-yyyy’), ‘Category Name’, ‘City Name’, ‘Club Name’ );

## Stored Procedure: SP\_INSERT\_HORSE

The role of this procedure is to support the Insert into Horses transaction.

It accepts seven parameters:

UELN – exact 15 digits,

Horse Name as a TEXT,

Horse Type – either: ‘Light’, ‘Heavy’, ‘Wild’ or ‘Pony’,

Breed as TEXT,

Gender – either ‘Stallion’ or ‘Mare’,

Birth Date as a Date and

Rider ID as a NUMBER.

Stored Procedure should be used for inserting a new record into Horses as follow:

**EXECUTE** SP\_Isert\_Horse (‘UELN’, ‘Horse Name’, ‘Horse Type, ‘Breed, ‘Gender, **TO\_DATE** (‘Birth Date’, ‘dd-mm-yyyy’), Rider ID );

## Stored Procedure: SP\_INSERT\_REGISTRATION

The role of this procedure is to support the Insert into Registrations transaction.

It accepts four parameters:

Rider ID as a NUMBER,

Event ID as a TEXT,

Box Type as a TEXT – either ‘With Straw’ or ‘Without Straw’ and

Reg Date as a DATE.

Stored Procedure should be used for inserting a new record into Registrations as follow:

**EXECUTE** SP\_Isert\_Registration(Rider ID, ‘Event Name’, ‘Box Type’, **TO\_DATE** (‘Reg Date’, ‘dd-mm-yyyy’));

## Stored Procedure: SP\_INSERT\_AWARD

The role of this procedure is to support the Insert into Awarding transaction.

It accepts four parameters:

Horse ID as a NUMBER,

Event ID as a NUMBER,

Total Score as a NUMBER and

Award ID as a NUMBER.

Stored Procedure should be used for inserting a new record into Awarding as follow:

**EXECUTE** SP\_Isert\_Award(Horse ID, Event ID, Total Score, Award ID);

## Stored Procedure: SP\_DELETE\_CONTESTANT

The role of this procedure is to support the Delete transaction on Contestants. It accepts parameter: Rider Id as a NUMBER. Stored Procedure should be used anytime wherever a contestant should be removed from the DB. Follow below code of execution:

**EXECUTE** SP\_Delete\_Contestant(Rider ID);

## Stored Procedure: SP\_UPDATE\_REGS

The role of this procedure is to support the Update transaction on Registrations.

It accepts two parameters:

Rider Id as a NUMBER and

Date of cancelations as a DATE.

Stored Procedure should be used anytime in case of a reg cancelation where a cancel date should be updated properly. Usually Cancel Date should be SYSDATE, so below code might be used:

**EXECUTE** SP\_Update\_Regs(Reg\_ID, SYSDATE);

# Functions

A function is **a subprogram stored in an Oracle database that returns a value**. It is a stored PL/SQL block just like procedures but there is a difference. A function always returns a value whereas a procedure may or may not return a value.

Below are described few functions created to use for our database:

## User Defined Function: UDF\_BOOKED\_BOXES

It accepts two parameters: Box type as a Text and Event name as a Text, where returns how many boxes are booked from the given box type and for the given event. The returned value is a NUMBER. For calling the function use the code as shown below:

**SELECT** udf\_booked\_boxes ('Box Type', 'Event Name')

**FROM** Dual;

## User Defined Function: UDF\_GET\_CLUBS\_BUDGET

It accepts no parameters and returns the total sum of all club’s budgets as a NUMBER. For calling the function use the code as shown below:

**SELECT** udf\_get\_clubs\_budget

**FROM** Dual;

## User Defined Function: UDF\_GET\_HORSE\_OWNER

It accepts one parameter: Horse ID as a NUMBER and returns the name of his owner along with his/her phone number as a Text. For calling the function use the code as shown below:

**SELECT** udf\_get\_horse\_owner(horse\_id)

**FROM** Dual;

## User Defined Function: UDF\_GET\_RIDER\_IDCARD

It accepts one parameter: Rider ID as a NUMBER and returns Rider’s name along with his/her phone ID Card as a Text. Result has a format like: F. Name Last Name, ID CARD: 123456\*\*\*\*.

For calling the function use the code as shown below:

**SELECT** udf\_get\_rider\_idcard (rider\_id)

**FROM** Dual;

## User Defined Function: UDF\_GET\_RIDER\_NAME\_AGE

It accepts one parameter: Rider ID as a NUMBER and returns Rider’s name along with his/her агв as a Text. Result has a format like: First Name Last Name <age>.

For calling the function use the code as shown below:

**SELECT** udf\_get\_rider\_ name\_age (rider\_id)

**FROM** Dual;

## User Defined Function: UDF\_RIDERS\_PER\_CITY

It accepts one parameter: City Name as a Text and returns how many riders the DB has per city. The returned value is a NUMBER.

For calling the function use the code as shown below:

**SELECT** udf\_riders\_per\_city('City Name')

**FROM** Dual;

# Triggers

Triggers are procedures that are stored in the database and implicitly run, or fired, when something happens. Traditionally, triggers supported the execution of a PL/SQL block when an INSERT, UPDATE, or DELETE occurred on a table or view.

Below are described few triggers created to use for our database:

## Trigger: TR\_BEFORE\_INSERT\_HORSE

The role of this trigger is to automatically assign an appropriate Horse Age Group, based on a calculation over provided horse’s Birth Date and the other parameters described at SP\_Isert\_Horse.

## Trigger: TR\_BEFORE\_INSERT\_AWARD

The role of this trigger is to automatically calculate and assign an appropriate Money Award for the winners from the first three places. It’s a percentage of the total budget of all clubs.

## Trigger: TR\_CHECK\_BOXES\_COUNT

The role of this trigger is mainly to control Arena’s boxes level and throw an error when there are no free boxes left.

## Trigger: TR\_SET\_NULL\_OR\_DELETE\_RIDER

Role of the trigger is to support the DB integrity. After a contestant being removed from the DB for some reason, this trigger will be automatically update entity: Horses and set NULL over the old Rider ID, also remove all related records form entity: Registrations.

## Trigger: TR\_CONTESTANT\_AUDIT

This trigger is created to apply an auditing over entity: Contestants, where same can be raised to any other entity from the DB if needed. The trigger will automatically generate a record to entity: Audit every time after Update, Delete or Insert being executed on entity: Contestants.

# Useful Select Statements

A SELECT statement consists of a query. It is used to select data from a database. The data returned is stored in a result table, called the result-set. The SELECT statement is so named because the typical first word of the query construct is SELECT.

And here is a file showing some interesting and useful Select Statements related to our database:



# Abbreviations

|  |  |
| --- | --- |
| **Abbreviation:** | **Meaning:** |
| DB | Database |
| DBMS | Database Management System |
| RDBMS | Relational Database Management System |
| GUI | Graphic User Interface |
| SQL | Structured Query Language |
| PL/SQL | Programming Language to Structured Query Language |
| DDL | Data Definition Language covers: Create, Drop. Alter, Truncate |
| DML | Data Manipulation Language covers: Insert, Update, Delete |
| DCL | Data Control Language covers: Grant, Revoke |
| TCL | Transaction Control Language covers: Commit, Rollback, Save Point |
| DQL | Data Query Language: **SELECT** Statement for Reading and Extracting data |
| PK | Primary Key |
| FK | Foreign Key |
| UQ | Unique |
| CHK | Check |
| SP | Stored Procedure |
| TR | Trigger |
| UDF | User Defined Function |

**Phone Patterns**

**Standard Phone Patterns:**

+359 123456789

1-718-444-1122

718-444-1122

(718)-444-1122

17184441122

7184441122

718.444.1122

1718.444.1122

1-123-456-7890

1 123-456-7890

1 (123) 456-7890

1 123 456 7890

1.123.456.7890

+91 (123) 456-7890

18005551234

1 800 555 1234

+1 800 555-1234

+86 800 555 1234

1-800-555-1234

1 (800) 555-1234

(800)555-1234

(800) 555-1234

(800)5551234

800-555-1234

800.555.1234

18001234567

1 800 123 4567

1-800-123-4567

+18001234567

+1 800 123 4567

+1 (800) 123 4567

1(800)1234567

+1800 1234567

1.8001234567

1.800.123.4567

+1 (800) 123-4567

18001234567

1 800 123 4567

+1 800 123-4567

+86 800 123 4567

1-800-123-4567

1 (800) 123-4567

(800)123-4567

(800) 123-4567

(800)1234567

800-123-4567

800.123.4567

1231231231

123-1231231

123123-1231

123-123 1231

123 123-1231

123-123-1231

(123)123-1231

(123)123 1231

(123) 123-1231

(123) 123 1231

+99 1234567890

+991234567890

(555) 444-6789

555-444-6789

555.444.6789

555 444 6789

18005551234

1 800 555 1234

+1 800 555-1234

+86 800 555 1234

1-800-555-1234

1.800.555.1234

+1.800.555.1234

1 (800) 555-1234

(800)555-1234

(800) 555-1234

(800)5551234

800-555-1234

800.555.1234

(003) 555-1212

(103) 555-1212

(911) 555-1212

18005551234

1 800 555 1234

+86 800-555-1234

1 (800) 555-1234

1. **Table that solves a complex many to many relationships between objects in a database**  [↑](#footnote-ref-1)
2. **Composite** **Primary Key** **is** **made up of a combination of columns** [↑](#footnote-ref-2)