DATABASE FOUNDATIONS

ORACLE ACADEMY

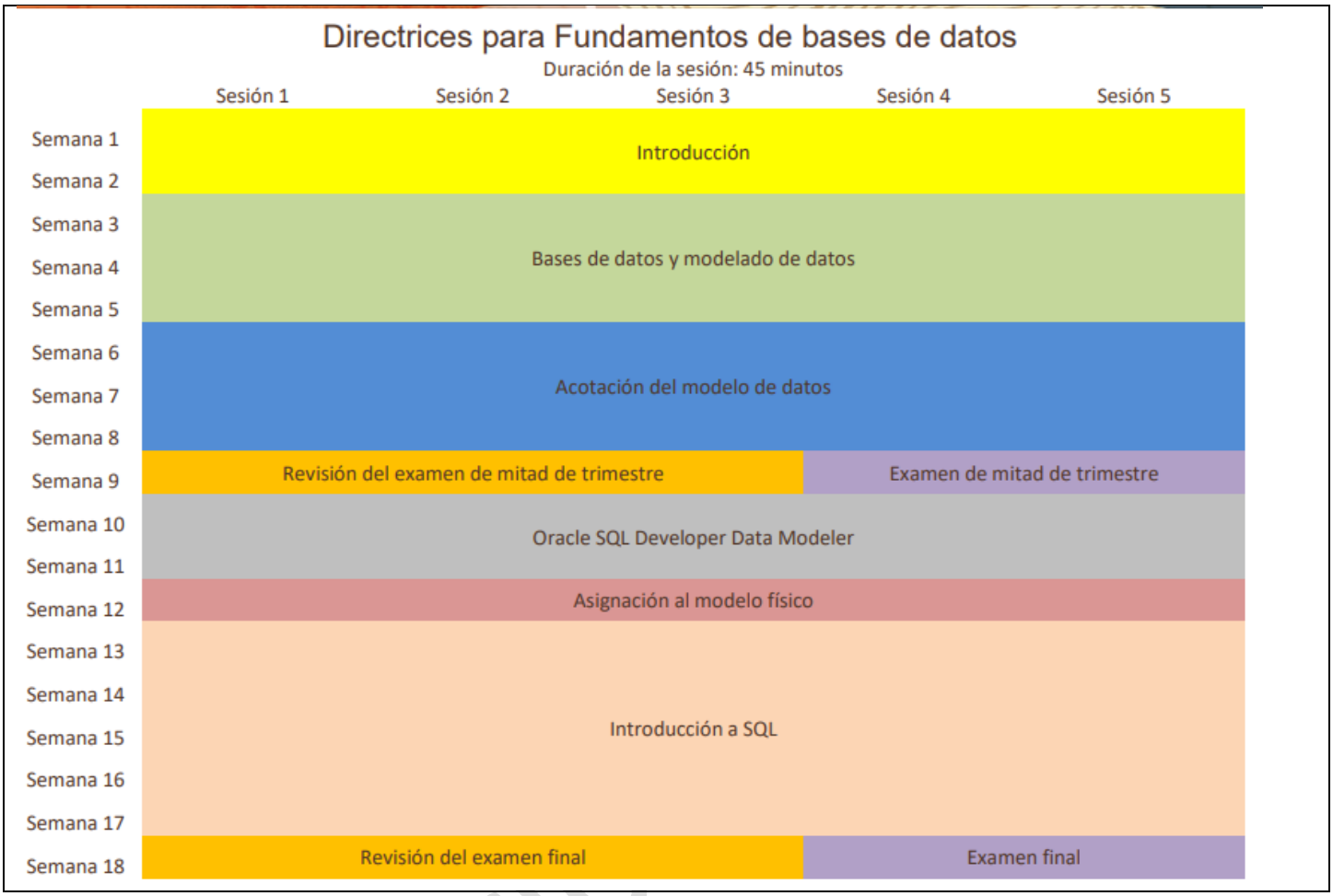
6 DE MAYO DE 2025

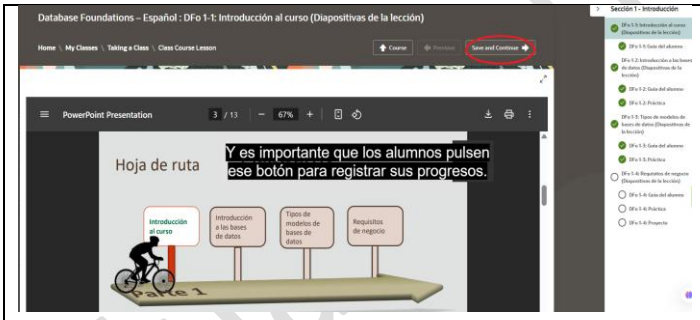
https://academy.oracle.com/

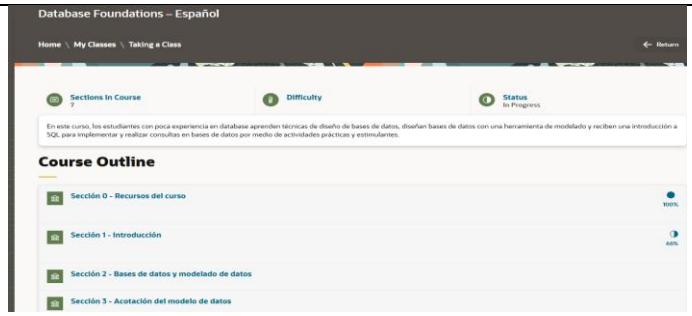
HTTPS://GITHUB.COM/ISC-UPA/2025-2-ISC05-DB

Contenido

1. Introduction

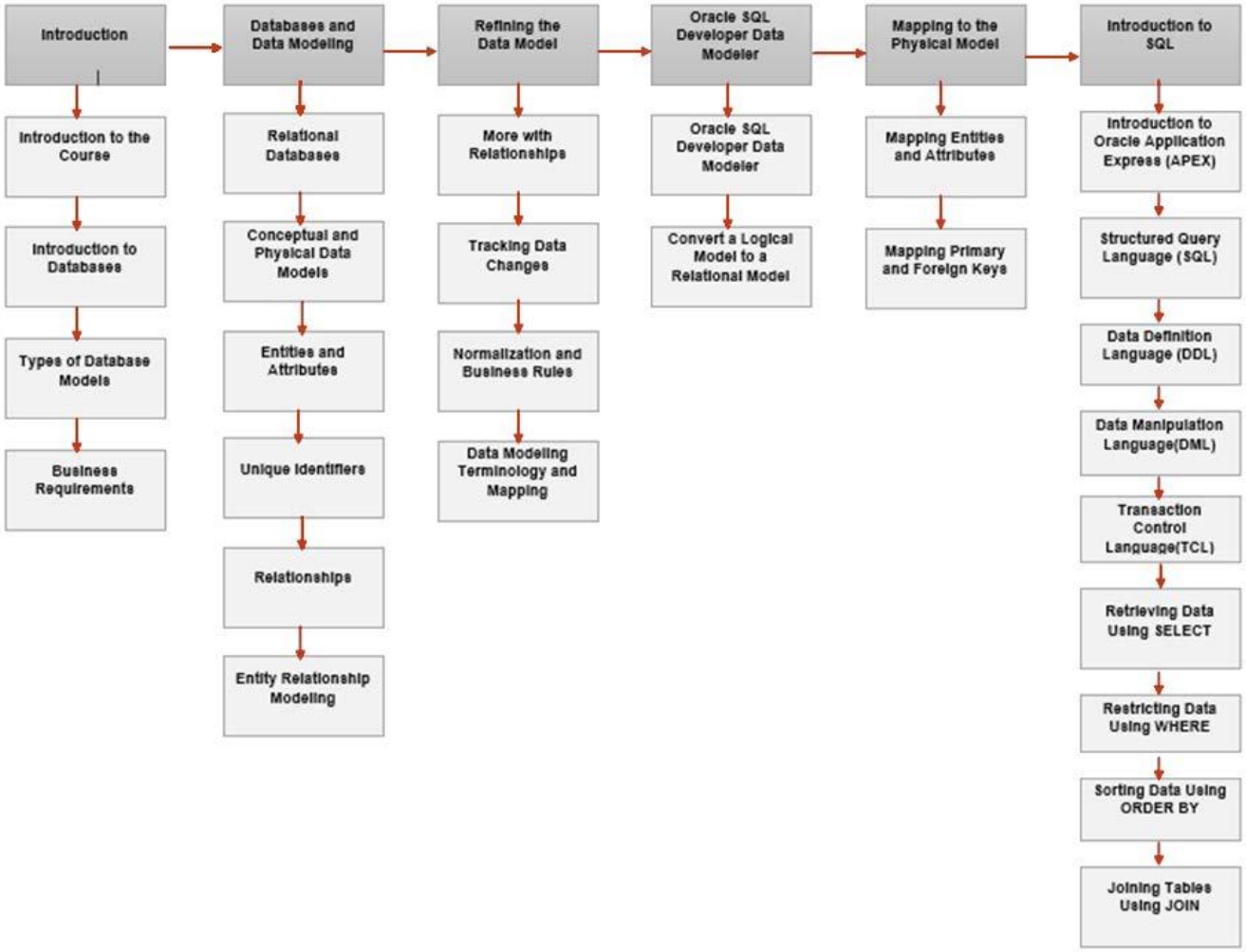






→

1.1. Introduction



Technological Requirements:

Oracle SQL Developer or Oracle APEX application
Oracle Data Modeler

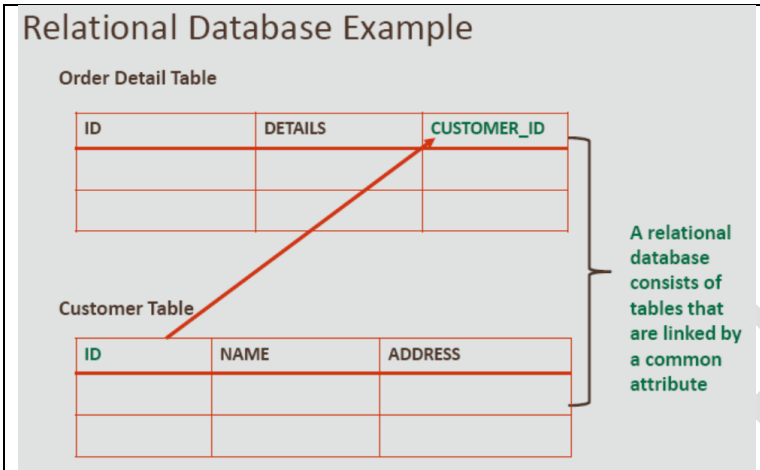
→

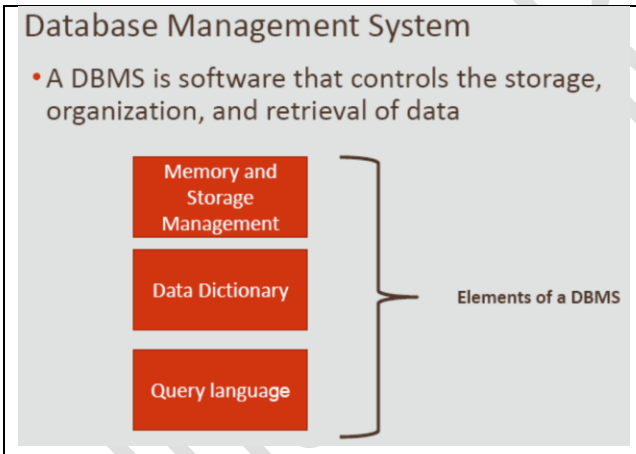
1.2. Introduction to Databases

Data vs Information.
Data: Collected facts about a topic or item
Information: The result of combining, comparing, and performing calculations on data.

Introduction to Relational Databases

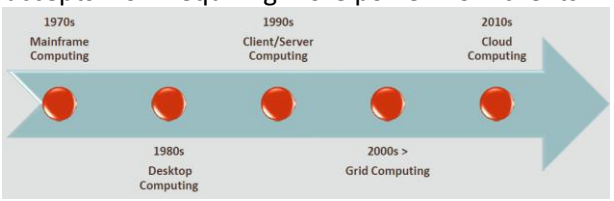
- A relational database stores information in tables with rows and columns
- A table is a collection of records
- A row is called a record (or instance)
- A record is a collection of fields
- A column is referred to as a field (or attribute)





Key Computing Terms

Hardware: physical parts of a computer
Software: instructions to tell hardware what to do
Operating system : software that directly controls the hardware
Application: performs specific task
Client : workstation used by end users
Server : accepts work requiring more power from clients

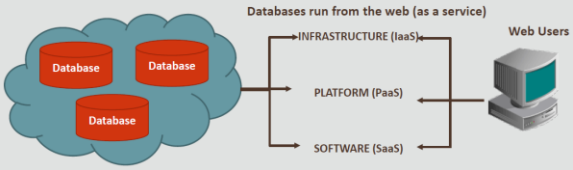


2000s: Grid Computing (Shared Processing)

- In the grid-computing model, all of an organization's computers in different locations can be utilized just like a pool of computing resources

- Grid computing builds a software infrastructure that can run on a large number of networked servers
- A user makes a request for information or computation from his or her workstation and that request is processed somewhere in the grid as efficiently as possible

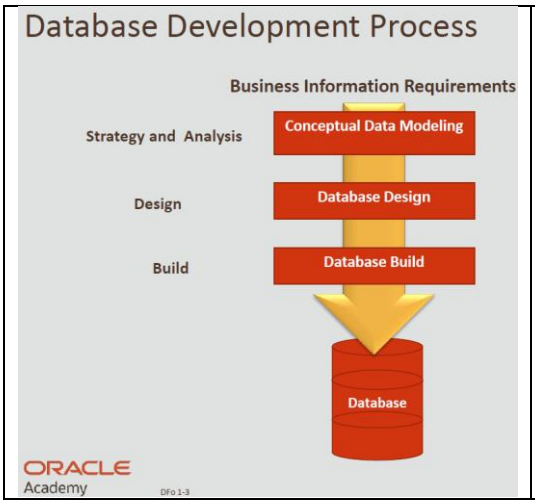
2010s: Cloud Computing (Internet Based Processing)

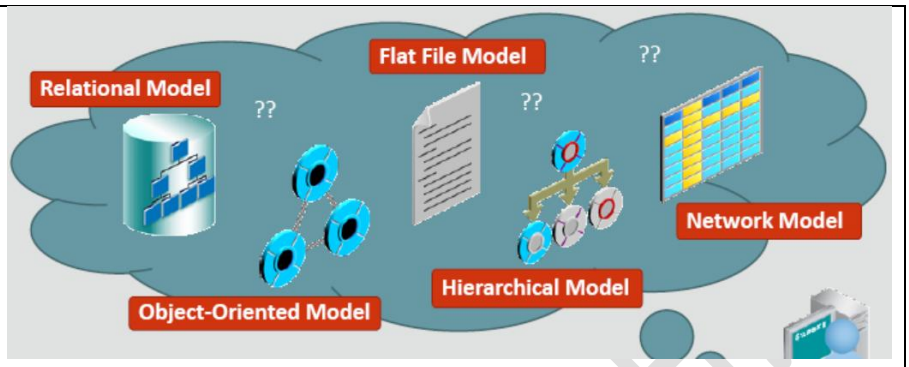


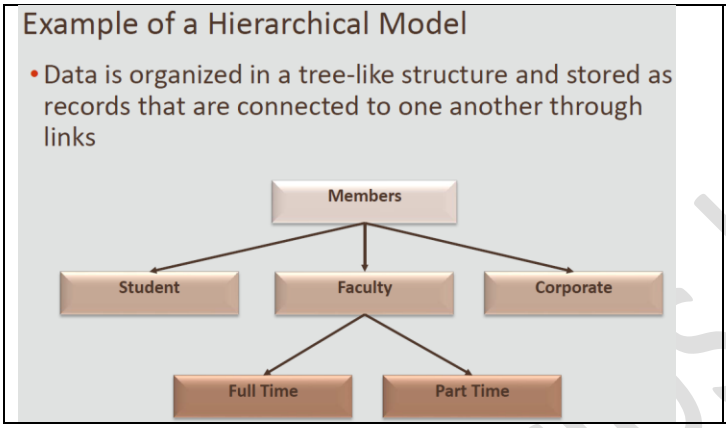
- Cloud computing allows the delivery of computing services over the Internet
- The three main categories of cloud services are:
  - IaaS – Allows you to rent cloud based servers, storage, operating systems etc
  - PaaS – Gives access to an online environment for developing and testing software without any setup or management costs
  - SaaS – Delivers software direct from the Internet. Users normally access it through a web browser

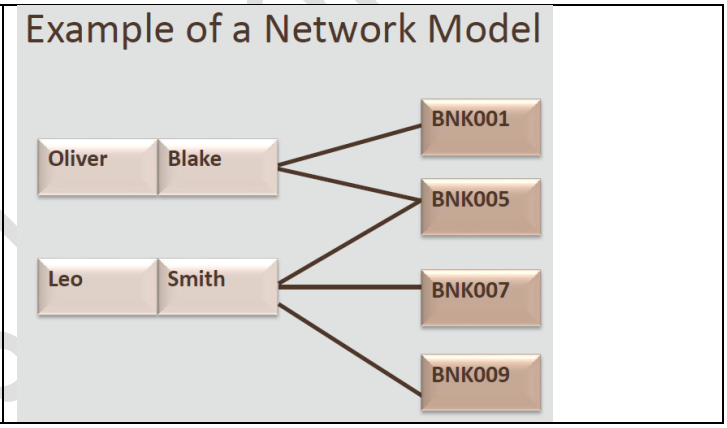
→

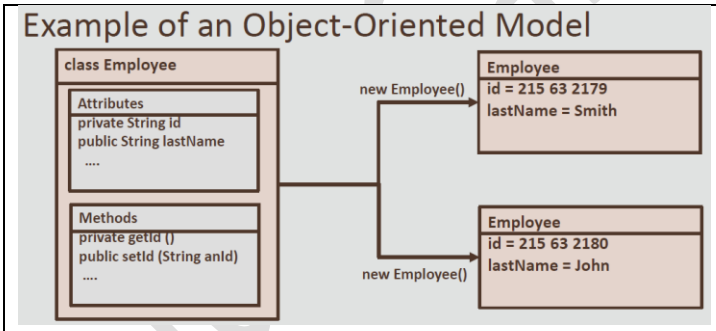
1.3. Types of Database Models





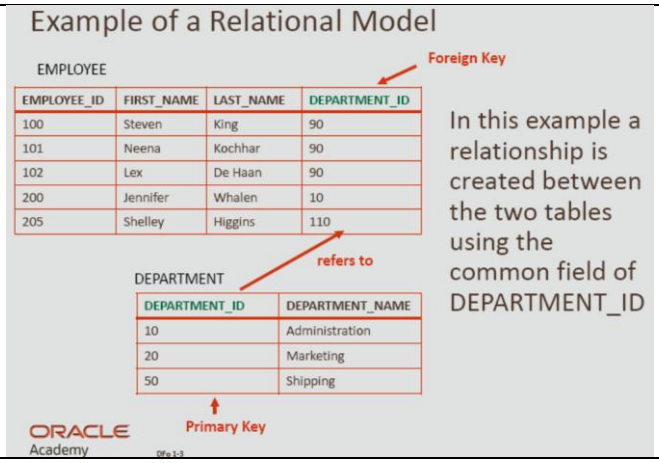






Los nombres de clases son en Singular

Los nombres de las tablas son en Plural



→

1.4. Business Requirements

Case Scenario: Need a Database Solution

	STUDENT_ID	SPORT_1	PRICE_1	SPORT_2	PRICE_2
Record 1	ST0001	Tennis	$100	Badminton	$150
Record 2	ST0002	Soccer	$175	Tennis	$100
Record 3	ST0003	Cycling	$200	Badminton	$150
.....	.....	.....	.....	.....	.....

Case Scenario: Possible Database Solution

Student Details Table

ID	FIRST_NAME	LAST_NAME
ST0001	Sean	Smith

Sport Details Table

ID	NAME	PRICE
TN001	Tennis	$100

Participant Details Table

STUDENT_ID	SPORT_ID	SEMESTER_DETAILS
ST0001	TN001	Fall2017

Flat file was split into three tables eliminating issues related to:

- Redundancy
- Data entry anomalies
- Inconsistency

Importance of Business Rules

It is important to identify and document business rules when designing a database

Business rules:

- Allow the developer/architect to understand the relationship and constraints of the participating entities
- Help you understand the standardization procedure that an organization follows when handling huge data
- Should be simple and easy to understand
- Must be kept up-to-date

Note: Not all business rules can be modeled in a database, but must be documented

Case Scenario: Identifying Key Business Rules, Problems, and Assumptions

- Business rule: Used to understand business processes and the nature, role, and scope of the data
- Assumption: Can be defined as a fact or a statement that has been taken for granted
- Problem: Can be defined as a situation or scenario that requires attention and a possible solution to alleviate the situation

Example:

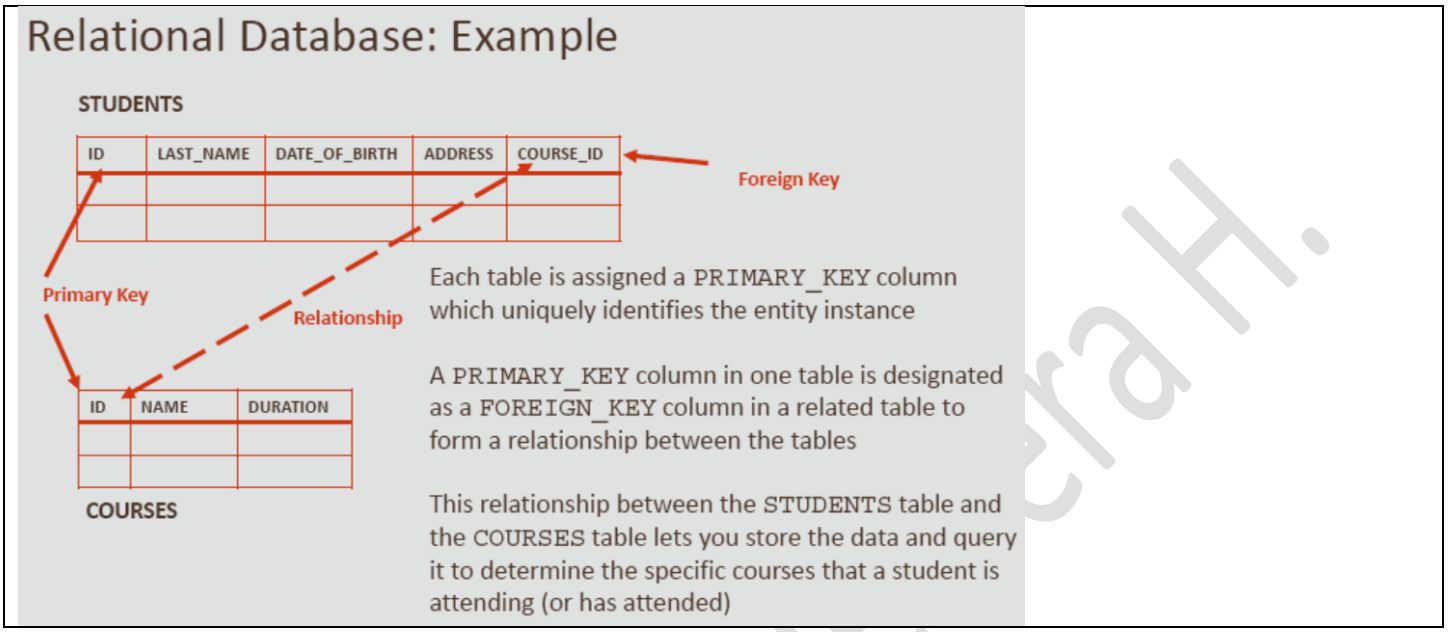
Note	Business Rule	Assumption	Problem
To ensure that new book arrivals happen on the 21st of every month.			
Librarian cannot easily identify DVDs that are seriously overdue (more than two weeks late).			
Our current system probably uses Oracle Database 10g and is on UNIX.			

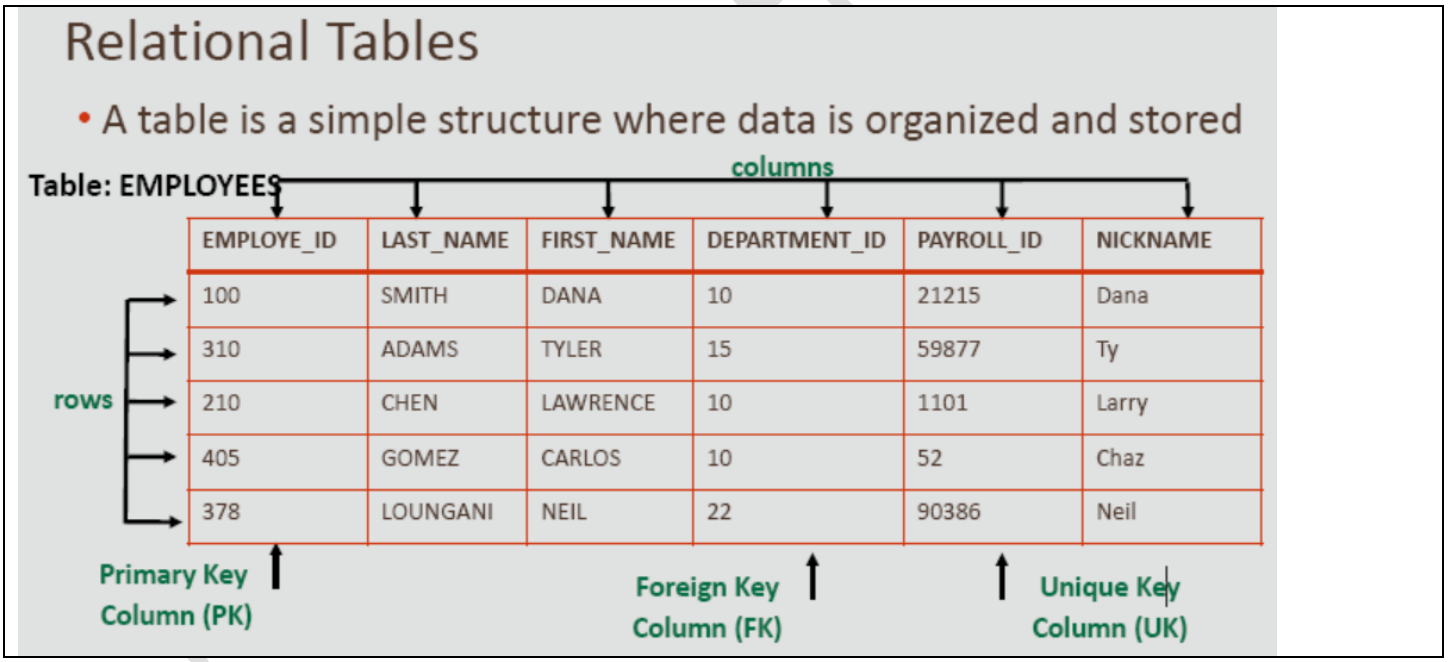
Identify the statements as a business rule, a problem, or an assumption.

→

2. Databases and Data Modeling

2.1. Relational Databases





Rules for Relational Database Tables

- Each table has a distinct name
- Each table may contain multiple rows
- Each table has a value to uniquely identify the rows
- Each column in a table has a unique name
- Entries in columns are single values
- Entries in columns are of the same kind
- Order of rows and columns is insignificant

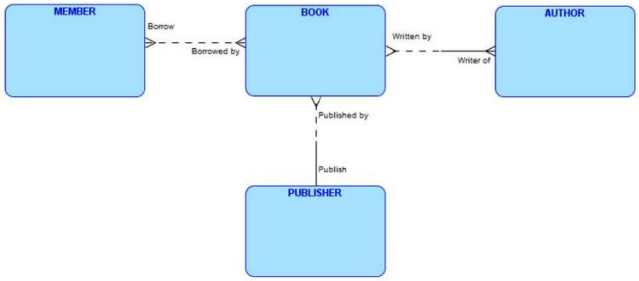
Key Terms

Table –A basic storage structure
Column–attribute that describes the information in the table
Primary Key –the unique identifier for each row
Foreign Key –a column that refers to a primary key column in another table
Row–data for one table instance
Field –the one value found at the intersection of a row and column

→

2.2. Conceptual and Physical Data Models

Conceptual data Model



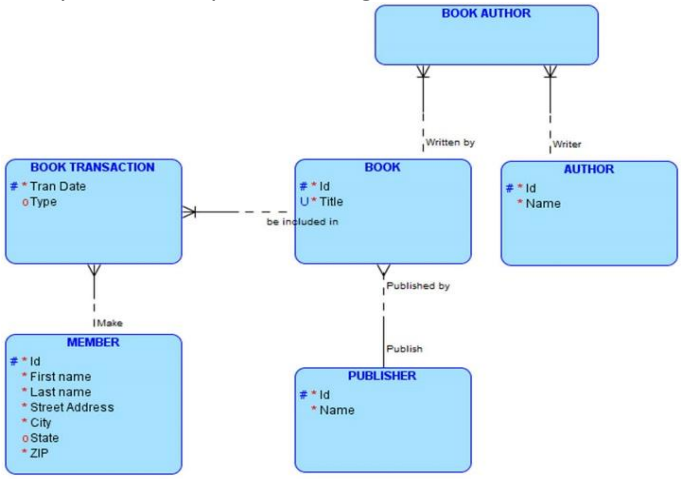
Identifies :

- important entities
- relationships among entities

Does not specify :

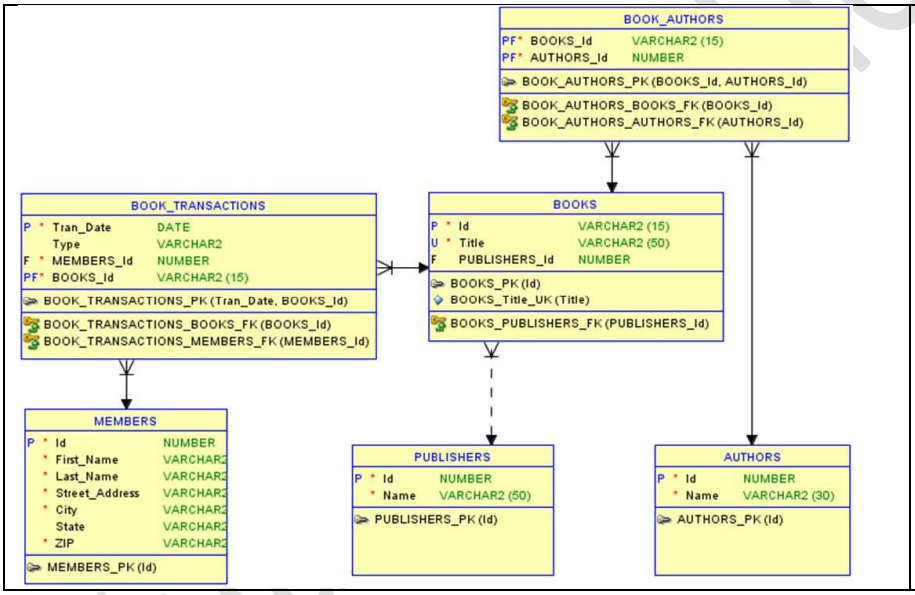
- attributes
- unique identifiers (primary key)

Logical data Model (ERM o ERD)
Entity Relationship Model/Diagram



Exists: attributes (# key, * not null , o optional)
Does not exist: N:M relationship

Physical data Model: Entities -> Relationships -> Attributes -> Constrains



Is a extesion to a logical data model:

- Atrributes data types and precision
- Identifies indexes
- Primary keys (P)
- Foreign keys (F)
- Unique keys (U)

Modeling Performed :
Entities → Tables
Attributes → Columns
Relationships → Foreign keys

→

2.3. Entities and Attributes

Identify UID(#), mandatory(*), optional(o), volatile or derivate(age), and nonvolatile(birthDate) attributes

Entity Types

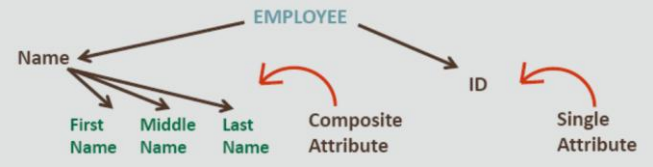
An entity can be classified as one of the following types:

Name	Description	Example
Prime	Exists independently	CUSTOMER, INSTRUCTOR
Characteristic	Exists because of another (prime) entity	ORDER, CLASS OFFERING
Intersection	Exists because of two or more entities	ORDER ITEM, CLASS ENROLLMENT

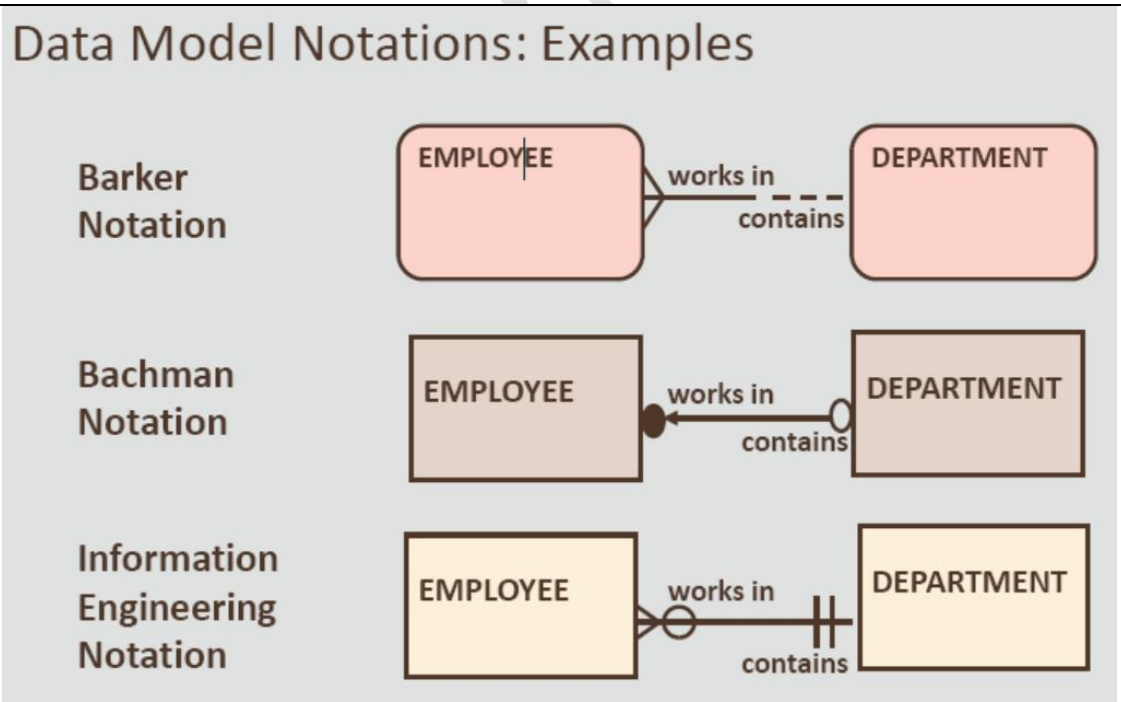
Tipos de Entidad:
- Principal
- Caracteristica
- Interseccion

Entidades:
- Fuertes
- Débiles

Single(atomic) and Composite(subparts) Attributes



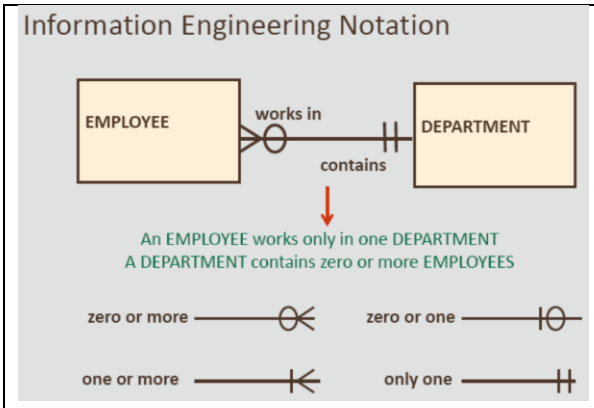
Single-Valued and Multi-Valued Attributes



Continua: obligatorio
Discontinua: opcional

Black: debe
White: puede
Arrow: many

Circulo: opcional
Pipe: obligatorio

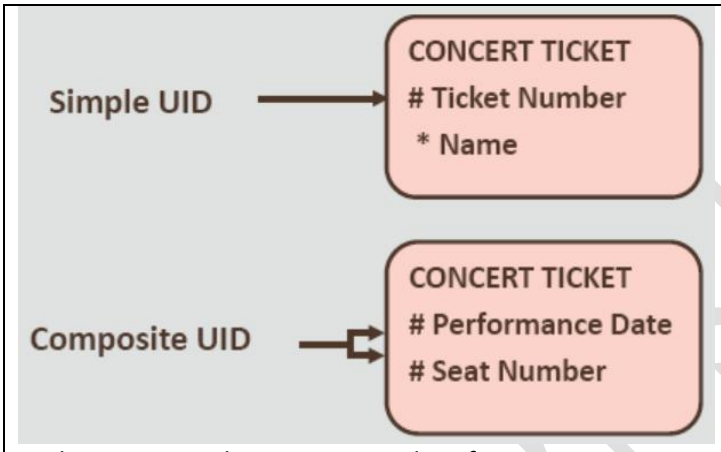


Data Model Notations

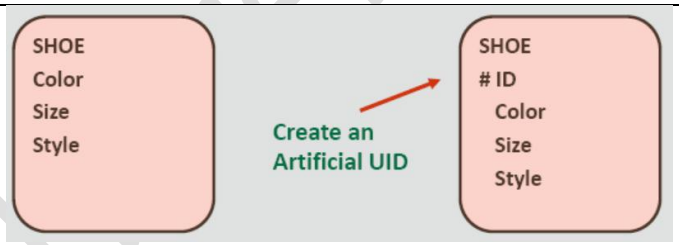
Notation (Read left to right)	Barker Notation	Bachman Notation	Information Engineering
Zero or one			
Only one			
Zero or more			
One or more			
Primary Key/Unique key	#	P	

Note: Barker notation is used for this course

2.4. Unique Identifiers



Each entity must have a unique identifier
Otherwise, it is not an entity

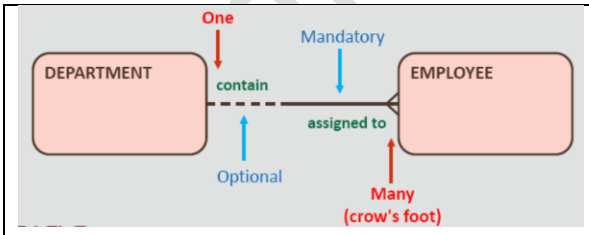


Candidate unique identifiers or secondary UIDs, so define primary key

It has a non-Null
It has a value that never changes

Artificial UIDs do not occur in the natural world but are created for identification purposes in a system
Example Composite UID: Bank_No and Account_No.

2.5. Relationships



Type: 1 : M
Cardinality: one and only one in one direction and one or more in other direction

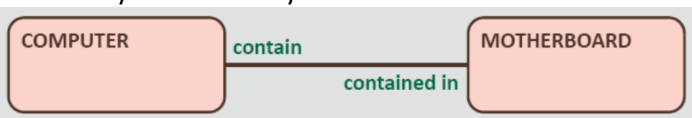
Optional: Use "may be" or "may."
Mandatory: Use "must be" or "must."
Line: Use "one and only one."
Crow's feet: Use "one or more."

Relationship Types or Cardinality

Many-to-one (M : 1) or one-to-many (1 : M)
Many-to-many (M : M)
One-to-one (1 : 1)

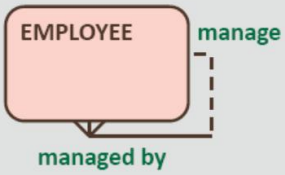
Type: Many to Many
Cardinality: one or more in both directions

Type 1 : 1
Cardinality: one and only one in both directions



Recursive Relationships

- A recursive relationship is a relationship with an entity and itself



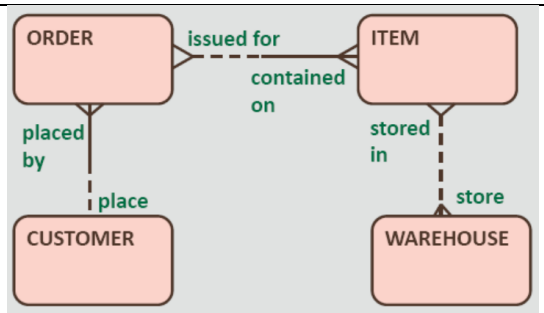
- Business rules:
  - Each EMPLOYEE may manage one or more EMPLOYEE
  - Each EMPLOYEE must be managed by one and only one EMPLOYEE

	EMPLOYEE_ID	LAST_NAME	MANAGER_ID
1	101	Kochhar	100
2	201	Hartstein	100
3	124	Mourgos	100
4	149	Zlotkey	100
5	102	De Haan	100
6	200	Whalen	101
7	205	Higgins	101
8	103	Hunold	102
9	104	Ernst	103
10	107	Lorentz	103
11	142	Davies	124
12	144	Vargas	124
13	143	Matos	124
14	141	Rajs	124
15	176	Taylor	149
16	174	Abel	149
17	178	Grant	149
18	202	Fay	201
19	206	Gietz	205
20	100	King	(null)

Relationship Matrix: Mapping the Contents

	CUSTOMER	ITEM	ORDER	WAREHOUSE
CUSTOMER			place	
ITEM			contained on	stored in
ORDER	placed by	issued for		
WAREHOUSE		store		

A relationship matrix can be used to collect initial information about the relationships among a set of entities



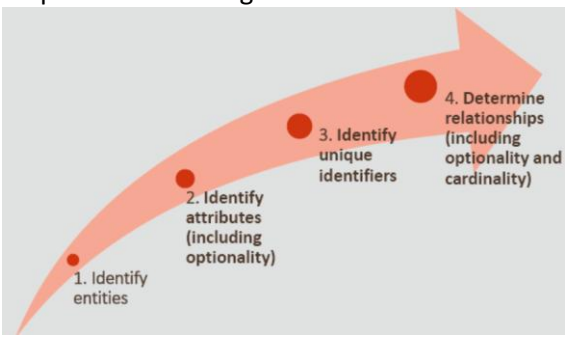
→

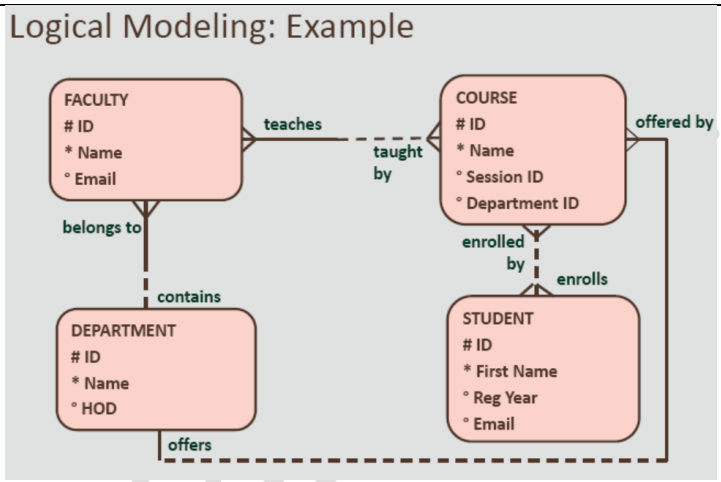
2.6. Entity Relationship Modeling (ERDs)

DB roles: designers, database administrators, and application developers

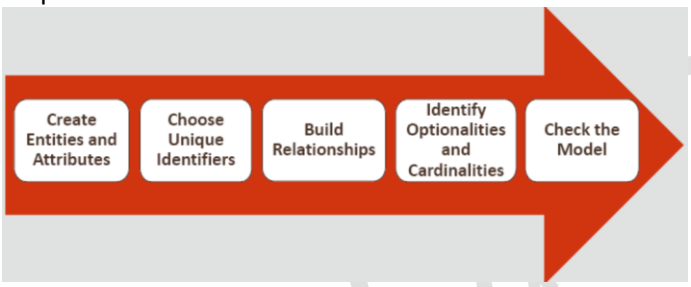
Logical Modeling:
Includes all entities, attributes, UIDs and relationships as well as optionality and cardinality of these ítems

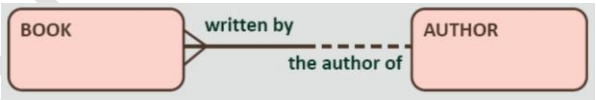
Steps to Create a Logical Model





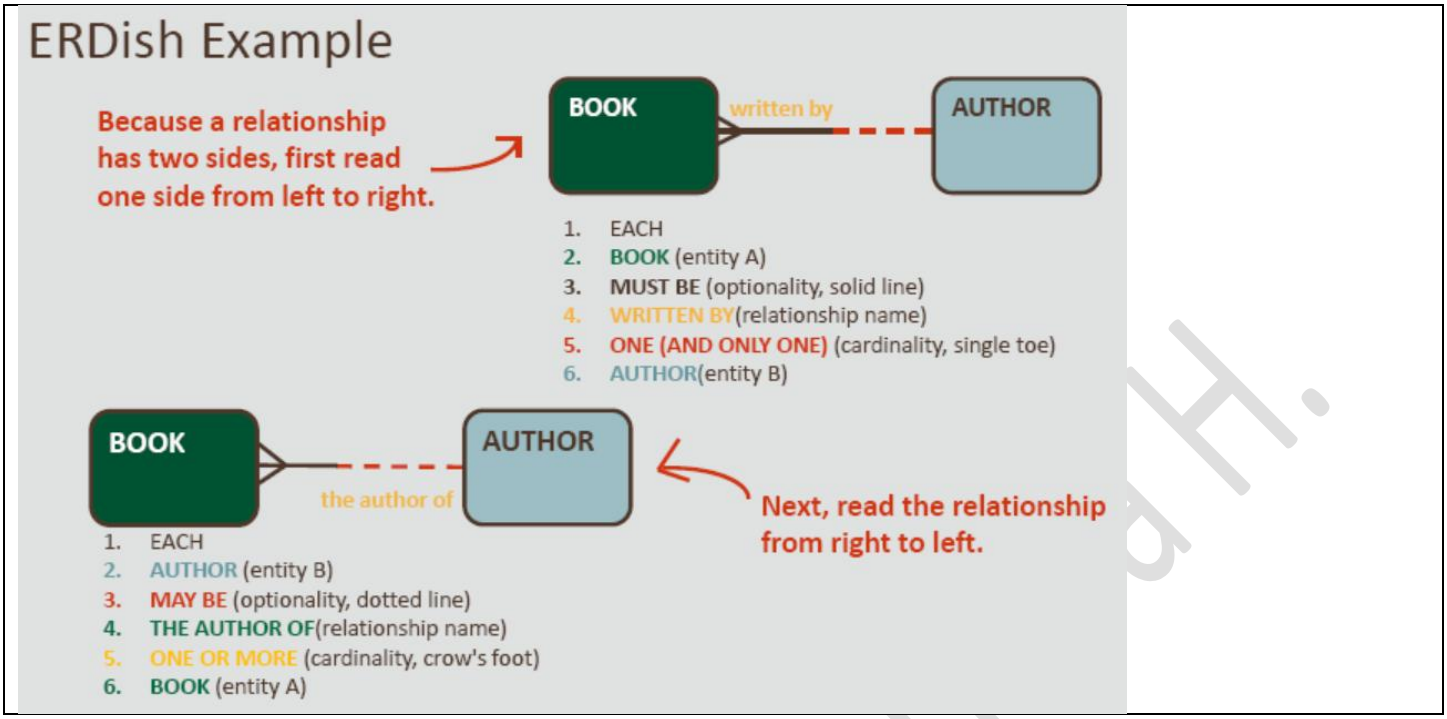
Steps to Build an ERD

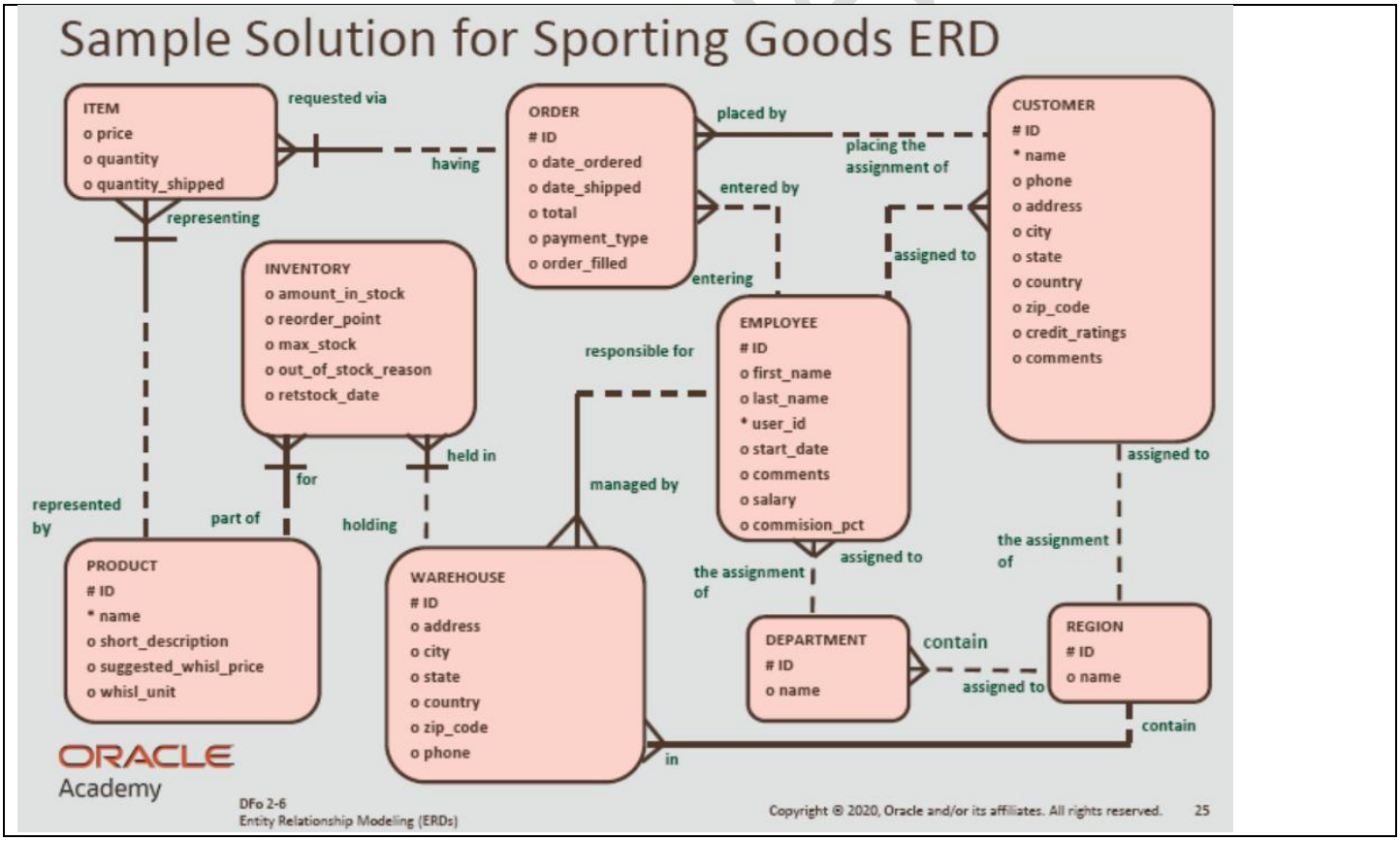




Components of ERDish

- EACH
- Entity A
- OPTIONALITY (must be/may be)
- RELATIONSHIP NAME
- CARDINALITY (one and only one/ one or more)
- Entity B

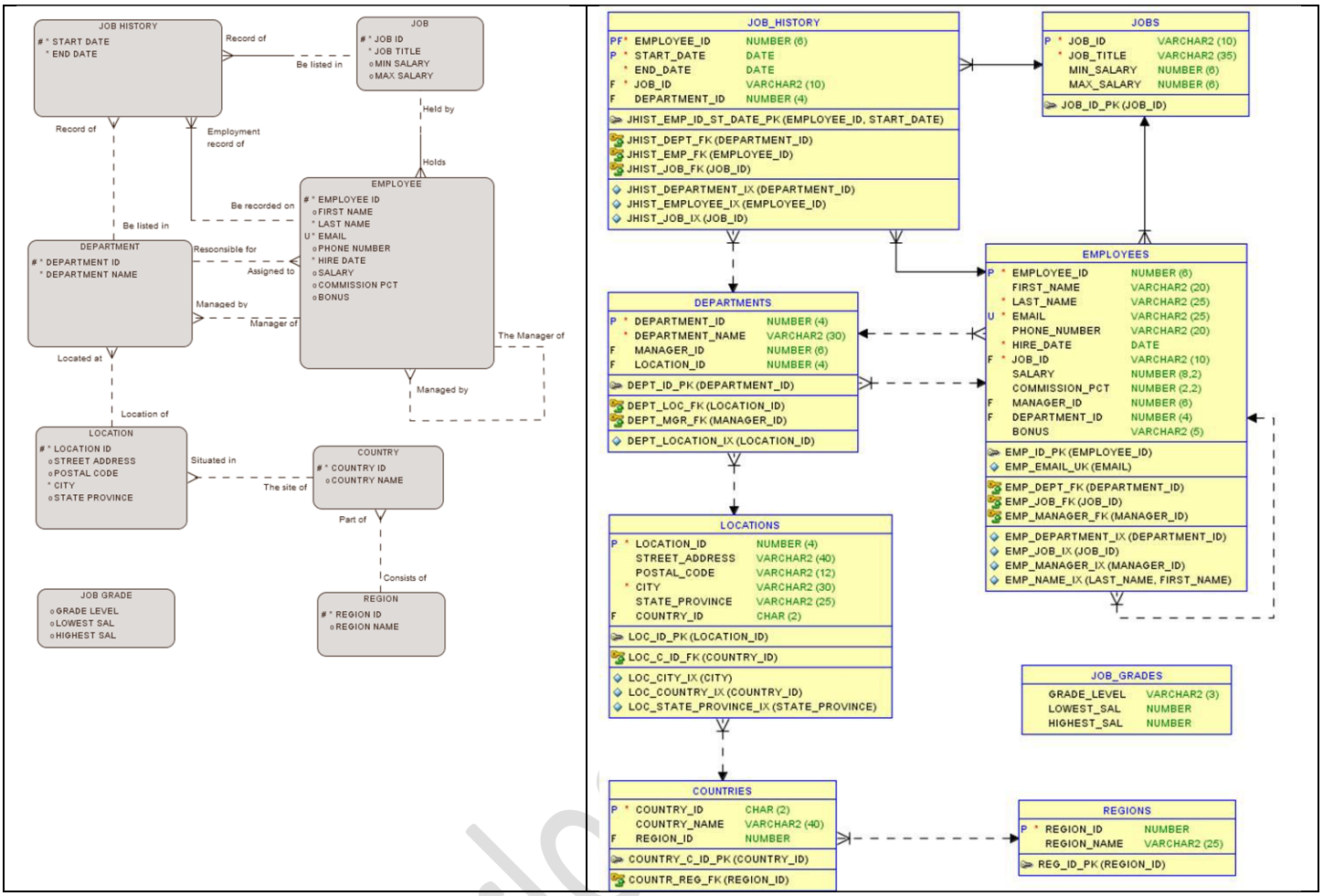

ERDish Example
Because a relationship has two sides, first read one side from left to right.
BOOK
written by
AUTHOR
1. EACH
2. BOOK (entity A)
3. MUST BE (optionality, solid line)
4. WRITTEN BY(relationship name)
5. ONE (AND ONLY ONE) (cardinality, single toe)
6. AUTHOR(entity B)
BOOK
the author of
AUTHOR
Next, read the relationship from right to left.
1. EACH
2. AUTHOR (entity B)
3. MAY BE (optionality, dotted line)
4. THE AUTHOR OF(relationship name)
5. ONE OR MORE (cardinality, crow's foot)
6. BOOK (entity A)


Sample Solution for Sporting Goods ERD
ITEM
o price
o quantity
o quantity_shipped
requested via
having
representing
ORDER
ID
o date_ordered
o date_shipped
o total
o payment_type
o order_filled
placed by
placing the assignment of
entered by
entering
CUSTOMER
ID
* name
o phone
o address
o city
o state
o country
o zip_code
o credit_ratings
o comments
assigned to
INVENTORY
o amount_in_stock
o reorder_point
o max_stock
o out_of_stock_reason
o retstock_date
for
held in
responsible for
EMPLOYEE
ID
o first_name
o last_name
* user_id
o start_date
o comments
o salary
o commision_pct
assigned to
represented by
part of
holding
managed by
the assignment of
assigned to
the assignment of
PRODUCT
ID
* name
o short_description
o suggested_whisl_price
o whisl_unit
WAREHOUSE
ID
o address
o city
o state
o country
o zip_code
o phone
DEPARTMENT
ID
o name
contain
assigned to
REGION
ID
o name
contain
in
ORACLE
Academy
DFo 2-6
Entity Relationship Modeling (ERDs)
Copyright © 2020, Oracle and/or its affiliates. All rights reserved.
25

→

Logical Data Model

Physical data Model

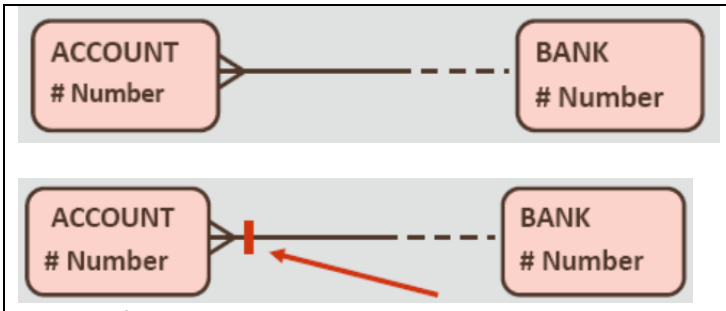


→

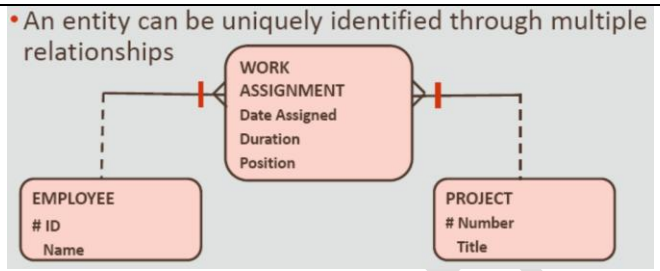
3. Refining the Data Model

3.1. More with Relationships

3.1.1. Identifying (| Barred) Relationships

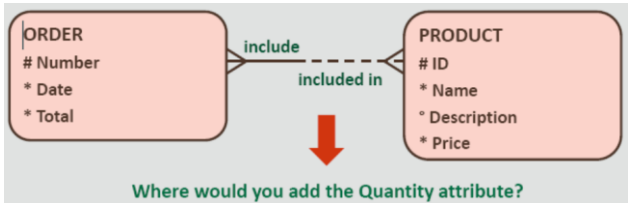


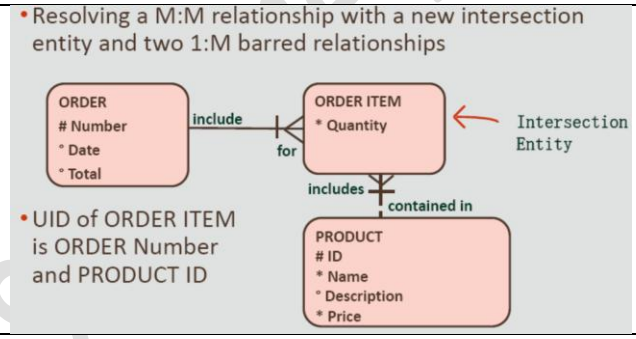
The UID for the ACCOUNT :
BANK Number , ACCOUNT Number



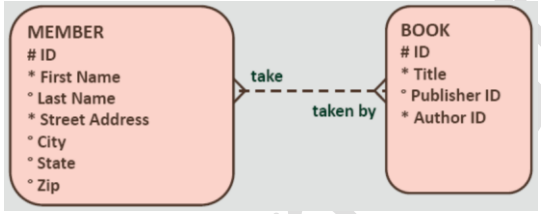
The UID for WORK ASSIGNMENT:
EMPLOYEE ID , PROJECT Number

3.1.2. M:M Relationships

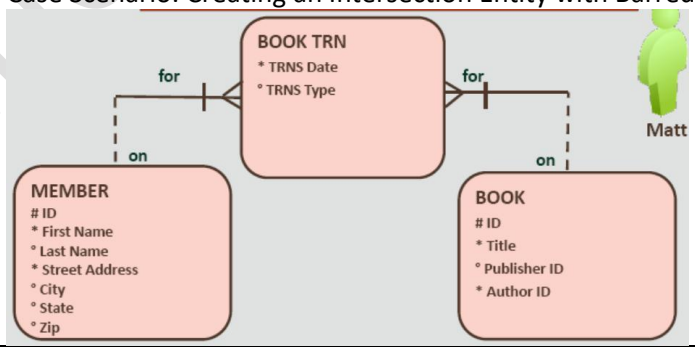




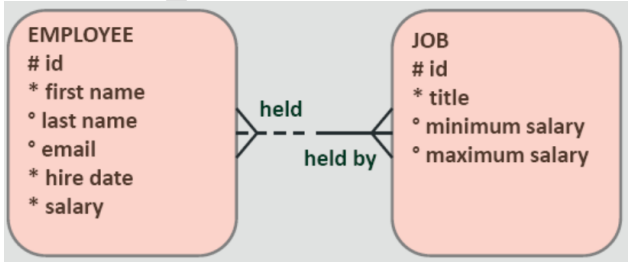
Case Scenario: Resolving M:M Relationships



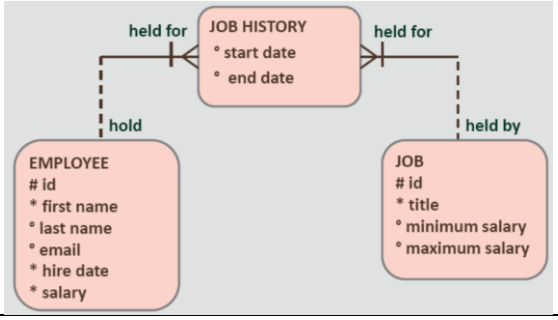
Case Scenario: Creating an Intersection Entity with Barred



Resolving M:M Relationships: Example 2

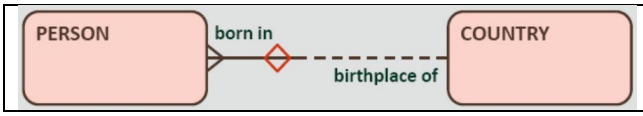


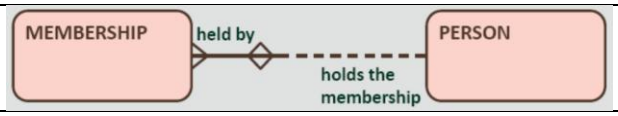
Resolving M:M Relationships Using Intersection



→

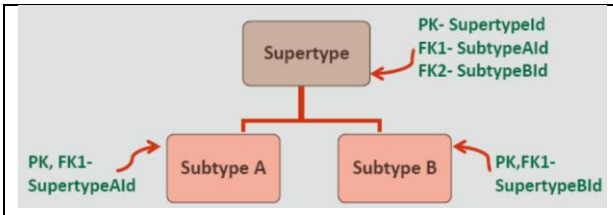
3.1.3. Non-Transferable Relationships ◇



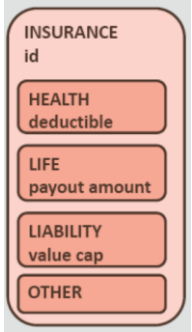


3.1.4. Supertype and Subtype Entities

- Supertype has a parent-child relationship with one or more subtypes
- Subtype is a subgrouping of the entity in an entity type which has attributes that are distinct from those in other subgroupings



Drawing a Subtype

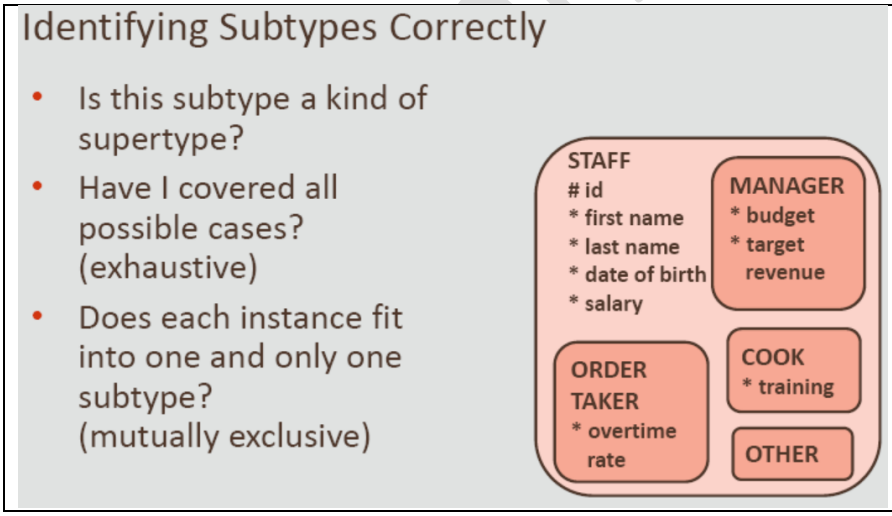


Characteristics of a Subtype

A subtype:

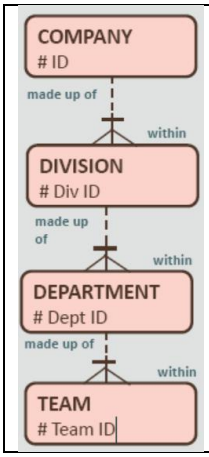
- Inherits all attributes of the supertype
- Inherits all relationships of the supertype
- Usually has its own attributes or relationships
- Is drawn within the supertype
- Never exists alone
- Has identical primary keys of the supertype and subtype

Example: An employee must be:
full time, part time, or other



→

3.1.5. Modeling Hierarchical Data

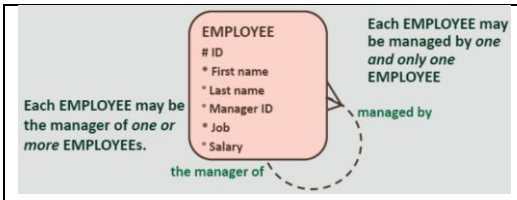


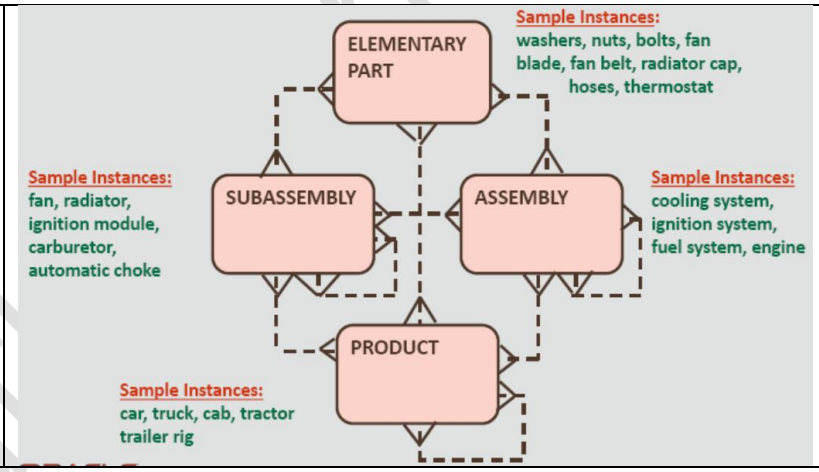
Resolver: La jerarquía del Hotel

En un hotel se tienen varias recamaras en cada departamento o suite(1,2 o 3), cuenta con 2 edificios, uno de 15 pisos y otro de 20 pisos.

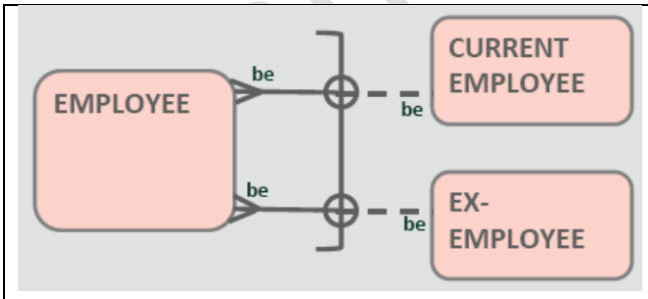
3.1.6. Recursive Relationships

- A recursive relationship is always modeled with a loop.
- A recursive relationship is one where an entity instance is related to another instance in the same entity





3.1.7. Arc Relationship ⌒



An arc is an exclusive relationship group, which is defined such that only one of the relationships can exist for any instance of an entity
Exor

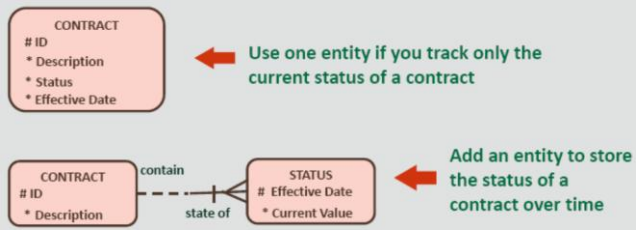
A supertype entity and its subtypes can be modeled as an arc relationship

→

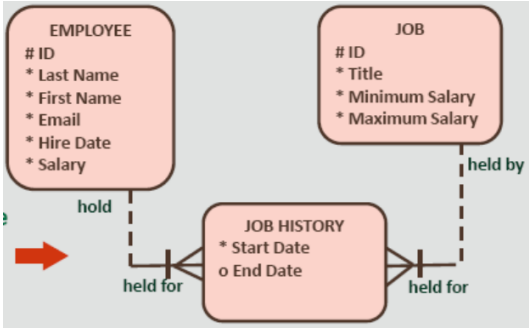
3.2. Tracking Data Changes

Objetive: Keep track of data that changes over time

Tracking Attribute Values

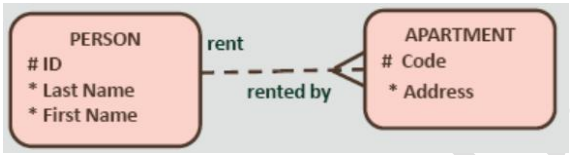


Creating Intersection Entities

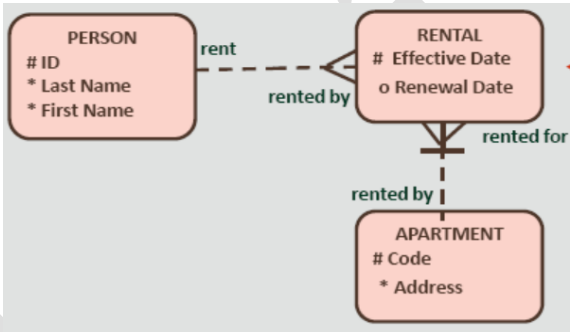


Accommodating a Changing Relationship

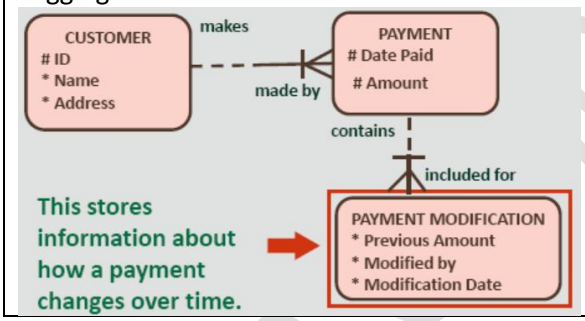
This does not allow tracking of the date that an apartment is rented.



Add an entity to capture the rental relationship over time

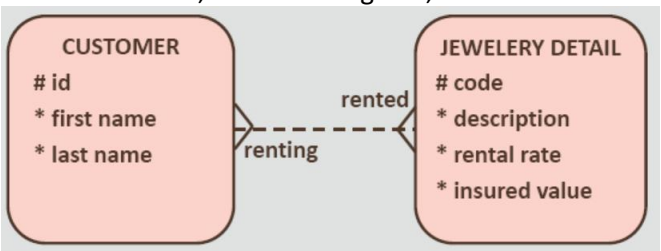


Logging Modified Information



Model Rental over Time: **Scenario**

Cuando se rentó, cuando lo regreso, cuanto le cobraron



→

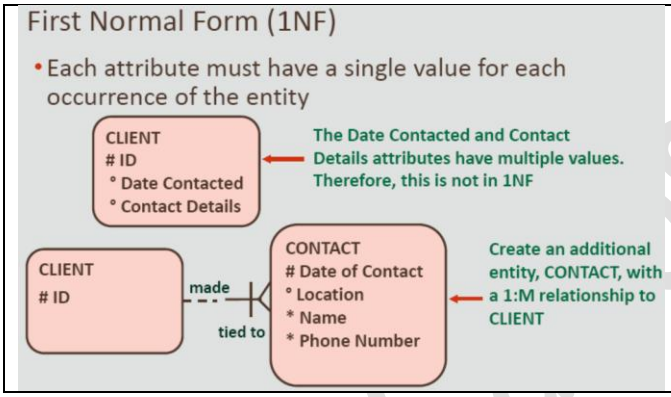
3.3. Normalization and Business Rules

Normalization: Is the process of organizing the attributes and tables of a relational database to minimize redundancy.

No duplicate content, Increase the integrity of data.

Rule	Description
First Normal Form (1NF)	All attributes must be single-valued. (no multi-valued = atomic)
Second Normal Form (2NF)	An attribute must be dependent on its entity's entire UID. No existen dependencias parciales
Third Normal Form (3NF)	No non-UID attributes can be dependent on another non-UID attribute. Ningún atributo no UID puede depender de otro atributo no UID. No existen dependencias transitivas.

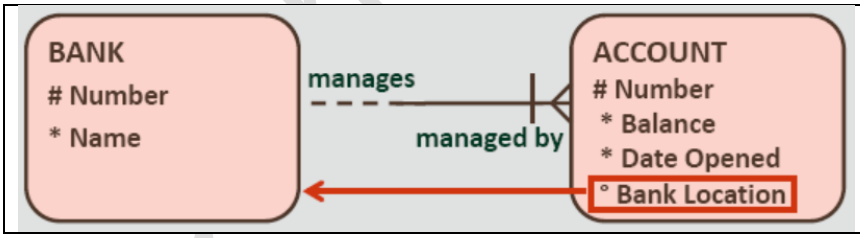
1NF : If an attribute is multi-valued, create an additional entity and relate it to the original entity with a 1:M relationship



Scenario:

Por cada estudiante, se desea registrar los lenguajes de programación que conoce.

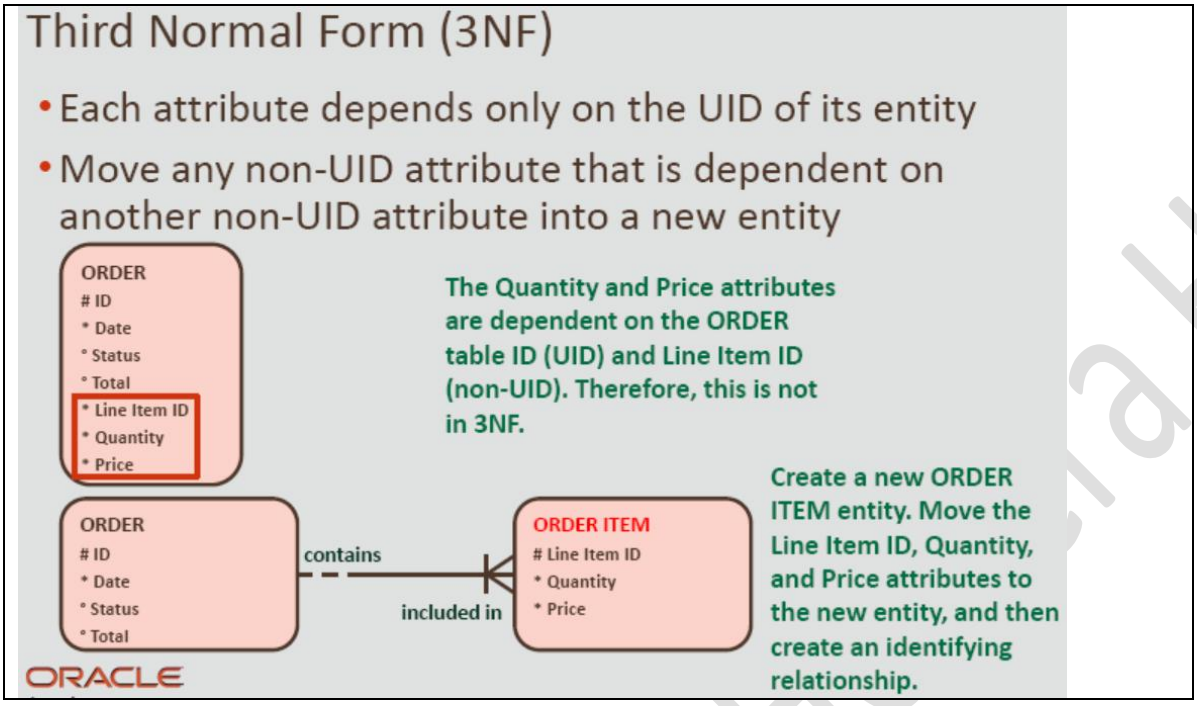
2NF : If an attribute is not dependent on the entire UID, create an additional entity with the partial UID



The Bank Location attribute is dependent on BANK,not on ACCOUNT. Therefore, this is not in 2NF.
Move the attribute to the BANK entity

3NF : You need to move any non-UID attribute that is dependent on another non-UID attribute into a new entity

A transitive dependency exists when any attribute in an entity is dependent on any other non-UID attribute in that entity



Business Rules

A business rule is a statement that defines or constrains some aspect of the business

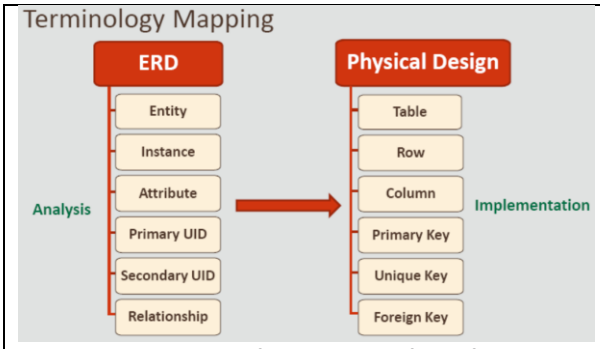
There are two types of business rules:

- Structural: These rules can always be diagrammed in the ERD
- Procedural: Some procedural business rules cannot be diagrammed. But must still be documented.
  Example: Event A must happen before event B
  La aprobación debe estar firmada por el director.
  La tienda no acepta entregar pedido al día siguiente, si se recibe después de las 3pm.
  El libro debe entregarse antes de las 4 dias.
  No se permite prestar otro libro, si debe libros con fecha expirada.
  Las horas extras deberán pagarse al 1.5 veces la tarifa.

Written documentation

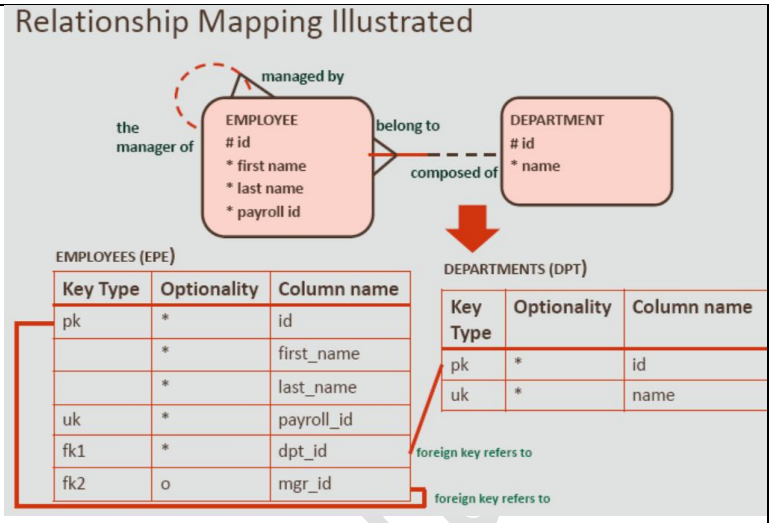
- Procedures
- Standards
- Operations manuals

3.4. Data Modeling Terminology and Mapping

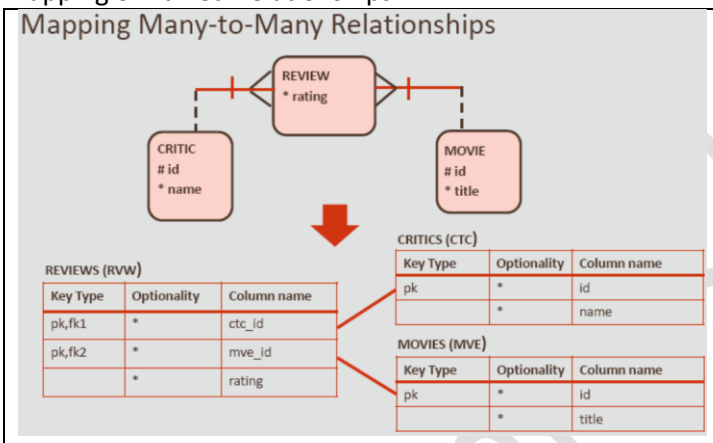


Attributes are classified as one of the following:
- Not null (mandatory): Indicated by the (*)
- Optional (nulls allowed): Indicated by the (o)

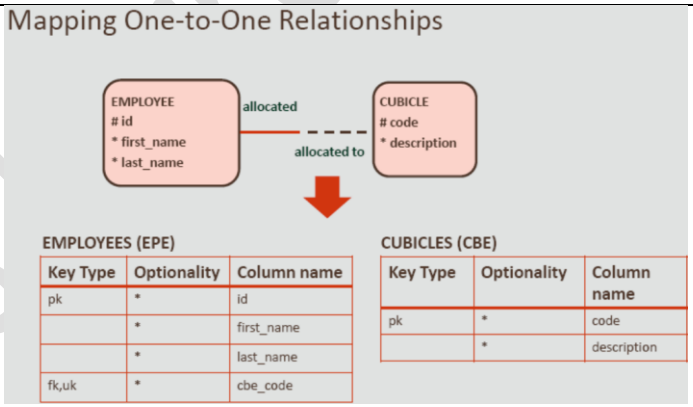
Key Type: pk, uk, fk



Mapping of Barred Relationships

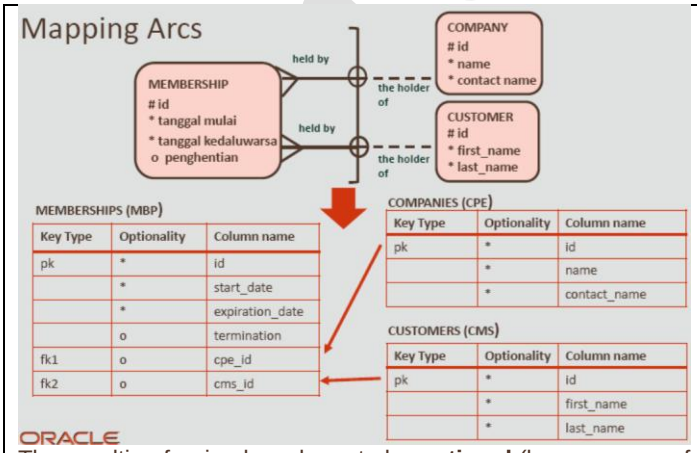


In this case, the foreign-key column plays a double role because it is also part of the primary key



When transforming a 1:1 relationship, you create a foreign key and a unique key

3.4.1. Mapping Arcs



The resulting foreign keys have to be **optional** (because one of them will always be blank).
- CHECK (cpe_id is not null AND cms_id is null)
- OR (cpe_id is null AND cms_id is not null)

For entity names that contain more than one word, take the:
- First character of the first word
- First character of the second word
- Last character of the last word

Example: PRIVATE HOME gets a short name of PHE.

For entity names that contain one word but more than one syllable, take the:
- First character of the first syllable
- First character of the second syllable
- Last character of the last syllable

Example: EMPLOYEE gets a short name of EPE and CLIENT gets a short name of CET.

For entity names that contain one syllable but more than one character, take the:
- First character
- Second character
- Last character

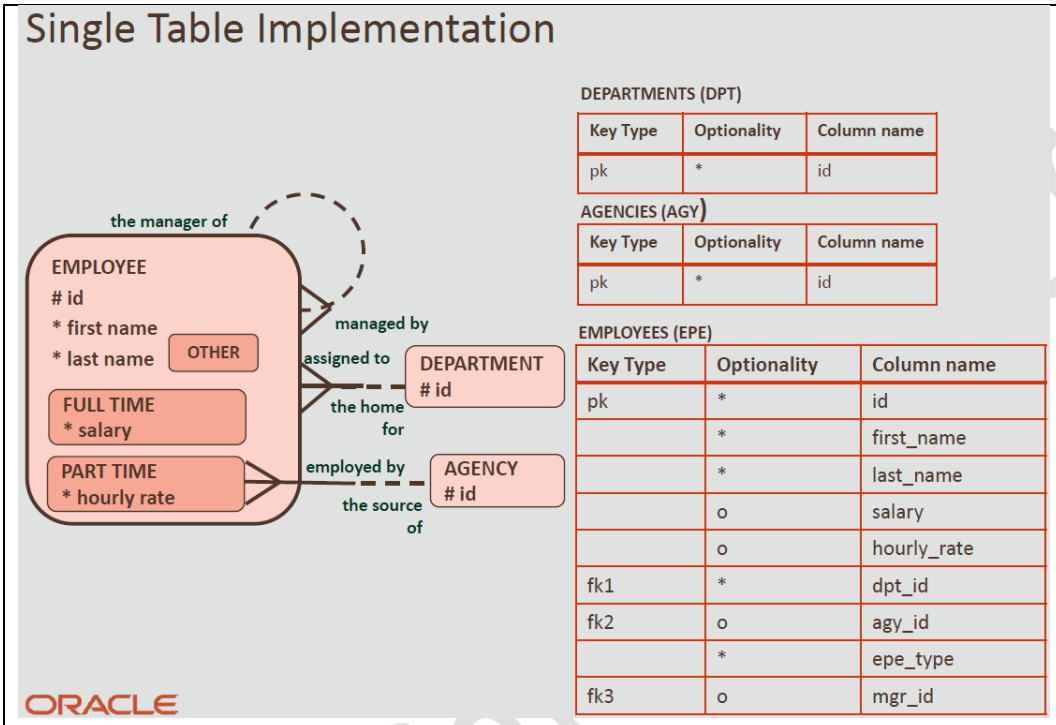
Example: FLIGHT gets a short name of FLT.

3.4.2. Mapping Supertype/Subtypes

Supertype/subtype entities can be mapped in multiple ways:

- Single table implementation : one table is created regardless of number of subtypes, used when most of the attributes and relationships are shared and therefore at the supertype level
- Two table implementation : a table is created for each of the subtypes (so there can be more than two tables), used when subtypes have little in common and few shared attributes and relationships

3.4.2.1. Single Table Implementation



DEPARTMENTS (DPT)

Key Type	Optionality	Column name
pk	*	id

AGENCIES (AGY)

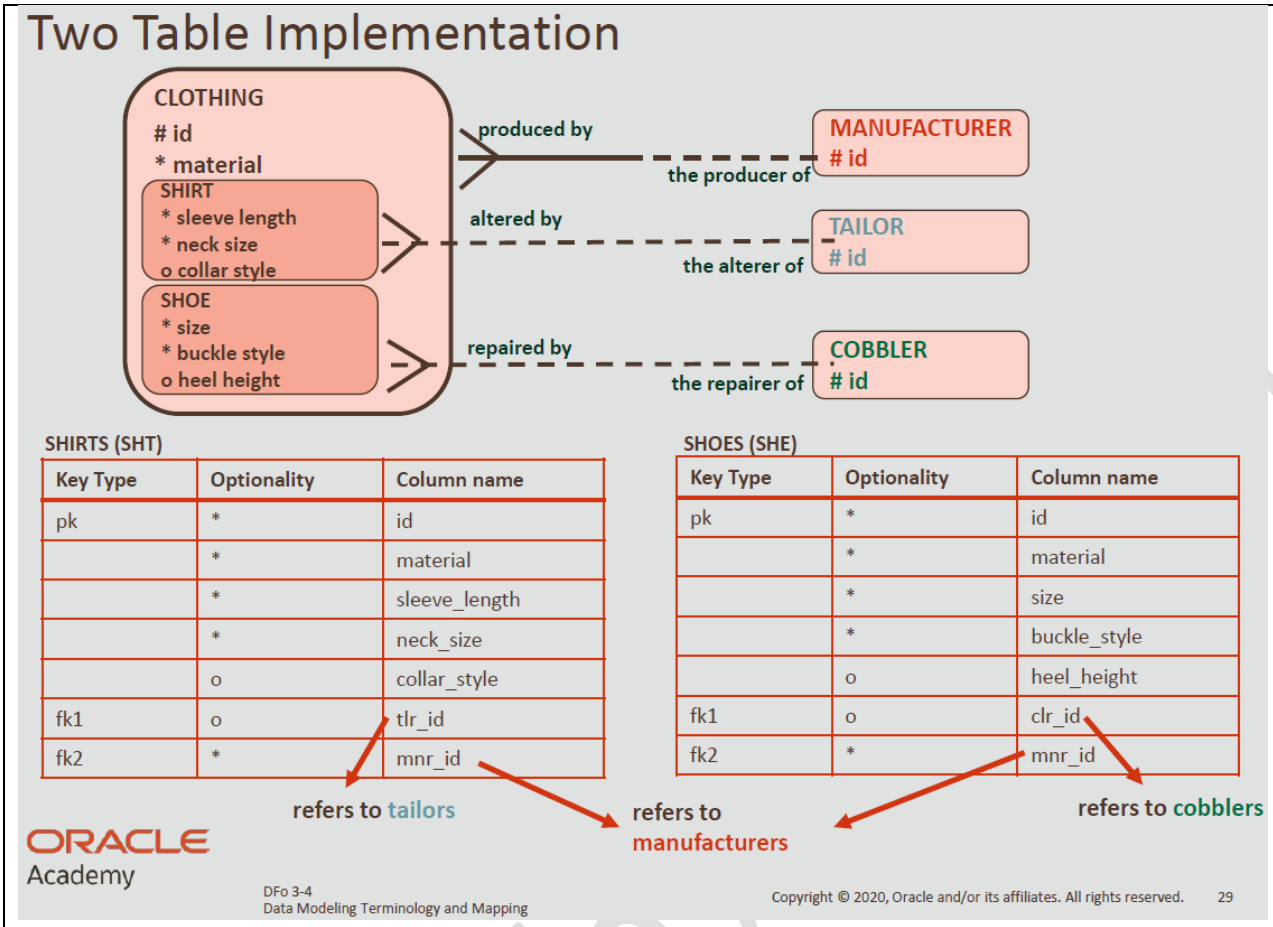
Key Type	Optionality	Column name
pk	*	id

EMPLOYEES (EPE)

Key Type	Optionality	Column name
pk	*	id
	*	first_name
	*	last_name
	o	salary
	o	hourly_rate
fk1	*	dpt_id
fk2	o	agy_id
	*	epe_type
fk3	o	mgr_id

A mandatory column should be created to act as a discriminator column to distinguish between the different subtypes of the entity. For this example: **epe_type** (full time or part time)

3.4.2.2. Two Table Implementation



For relationships at the subtype levels, the foreign key is implemented in the table it is mapped to, original optionality is retained. (tailors, cobblers)

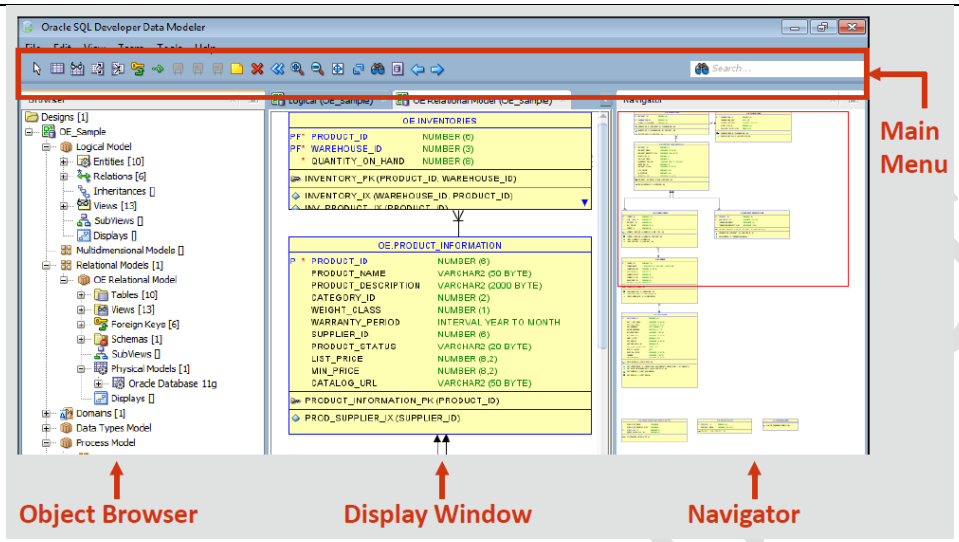
→

4. Oracle SQL Developer Data Modeler

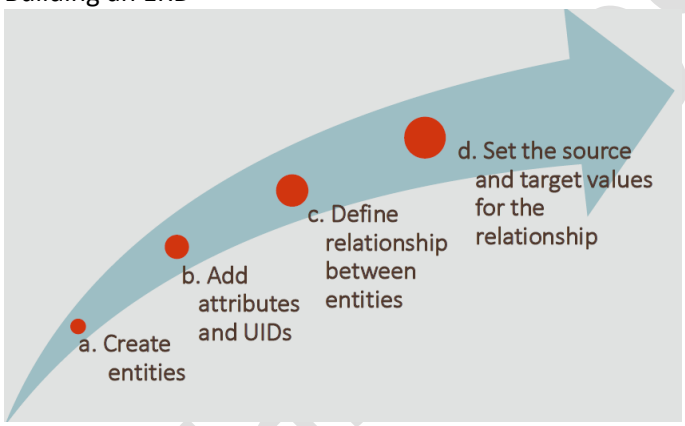
4.1. Oracle SQL Developer Data Modeler

Use Oracle SQL Developer Data Modeler to create:

- Entities, attributes, and UIDs with correct optionality and cardinality
- Supertype and subtype entities
- Arc, hierarchical, barred, and recursive relationships

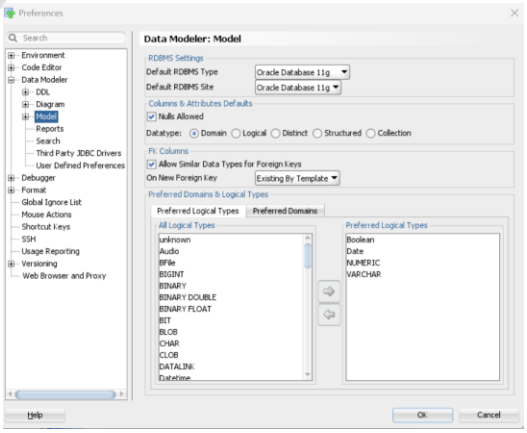


Building an ERD



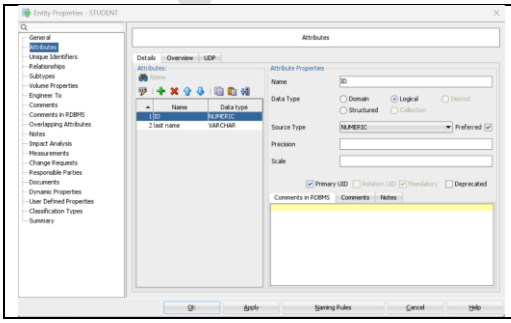
Tools -> Preferences -> Data Modeler -> Model

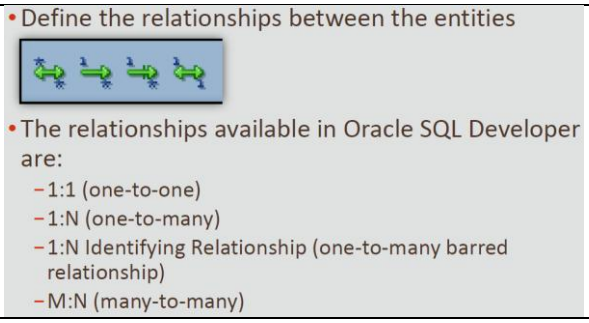
Boolean, Date, NUMERIC, VARCHAR



Entity -> Attributes -> Attributes Details -> Preferred

Set Primary and Secondary UIDs

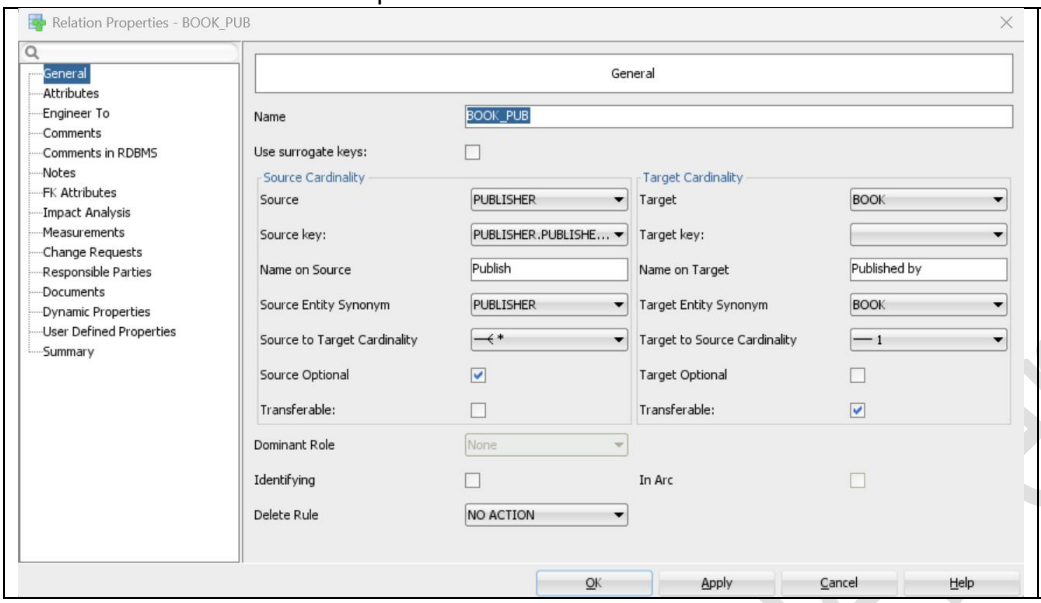




To define the relationships between entities in Oracle SQL Developer, perform the following steps:

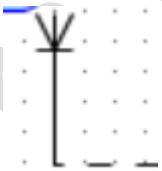
a. Click a relationship type on the toolbar
b. Click the source entity(PK) and then click the target entity(FK) . The relationship is created

Double-Click on the relationship established



Show labels:
Right click on the whitespace
Show -> Labels

Change the path
Right click on the relationship
Select Add elbow



Creating the Supertype Entity

Select the table that is subtype (Full Time) General properties -> Super Type (select Employee)	

Creating the Arc Relationship

Perform the following steps: a. Hold the ctrl key and select the intersecting entity and both relationships on which you want to create the Arc relationship b. Click the New Arc icon in the toolbar. The exclusive relationship is created with the arc	

Creating the Barred Relationship

To add a barred relationship select Identifying Relationship from the toolbar, and click the source and target entities to add the relationship between the entities	

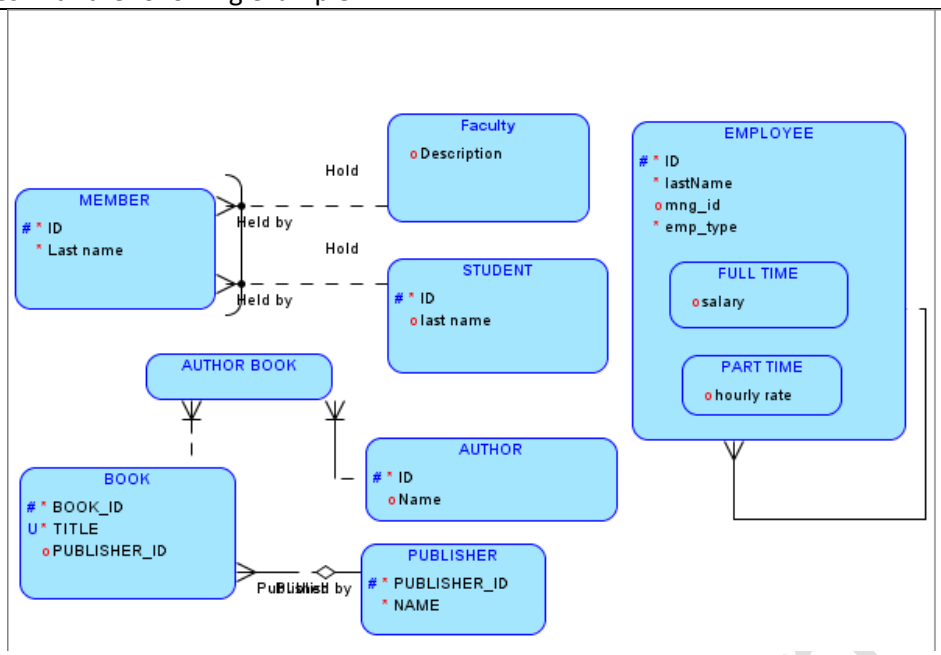
Creating the Hierarchical Relationship

The UIDs for a set of hierarchical entities can be propagated through multiple relationships by making the relationships Identifying

Creating the Recursive Relationship

To add a Recursive Relationship, select the required relationship from the toolbar as normal, then click on the entity to make it the source, and click on the same entity a second time to make it the target.

Practices with the following example:



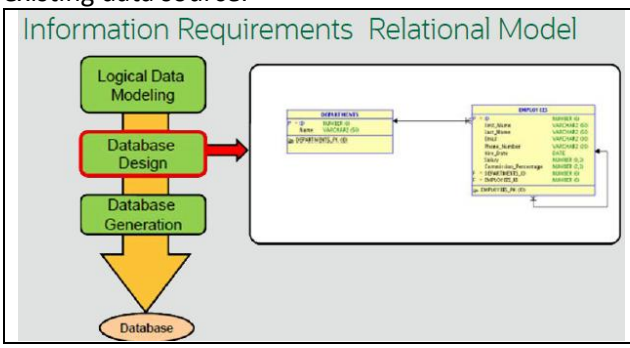
→

4.2. Convert a Logical Model to a Relational Model

Objectives:

- Describe how to convert a logical model to a relational model
- List the steps to convert a logical model to a relational model
- List the steps to convert a relational model to a logical model

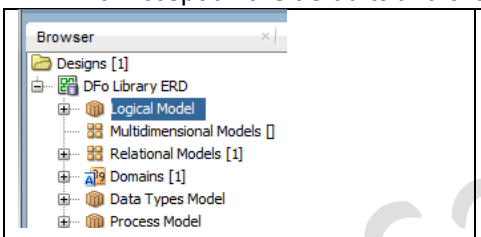
In Oracle SQL Developer Data Modeler a physical model is represented by a Relational Model.
Forward engineering is the process of transforming a logical data model to a relational model.
Reverse engineering is the process of creating a conceptual or logical model by extracting the information from an existing data source.

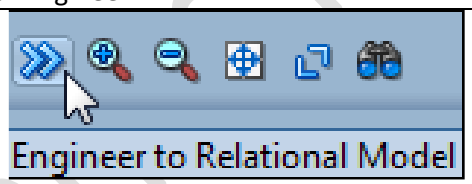


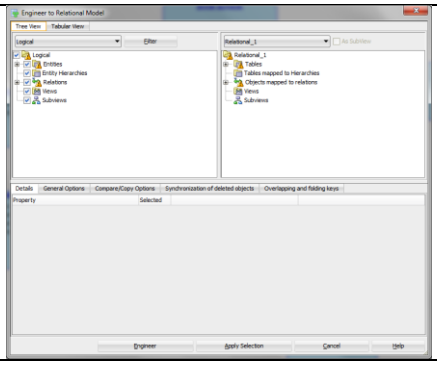
ANALYSIS	DESIGN
Logical Model	Relational Model
Entity	Table
Attribute	Column
Instance	Row
Primary UID	Primary Key
Secondary UID	Unique Constraint
Relationship	Foreign Key
Business Constraints	Check Constraints

Logical Model to a Relational Model:

a. Select the logical model.
b. Click the Engineer to Relational Model icon
c. Accept all the defaults and click Engineer



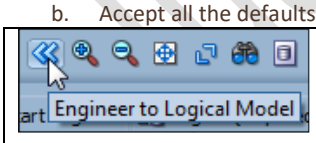


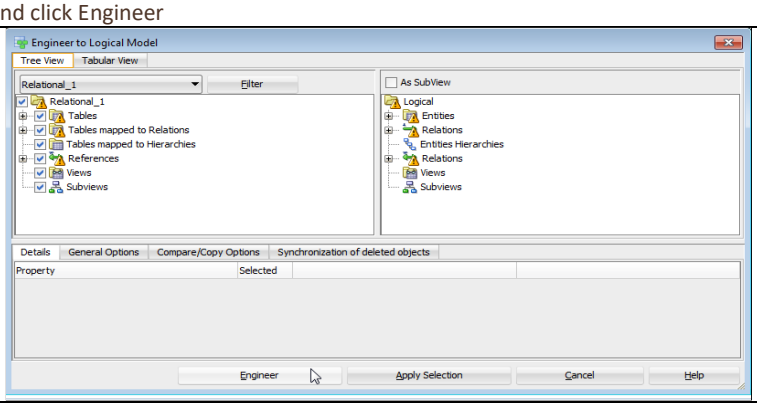


Reverse engineering from a Relational to a Logical Model

Allows an ERD to be created from an existing Physical design

a. Click the Engineer to Logical Model icon
b. Accept all the defaults and click Engineer





→

5. Mapping to the Physical Model

5.1. Mapping Entities and Attributes

In Oracle SQL Developer Data Modeler a Physical model is known as a Relational Model.

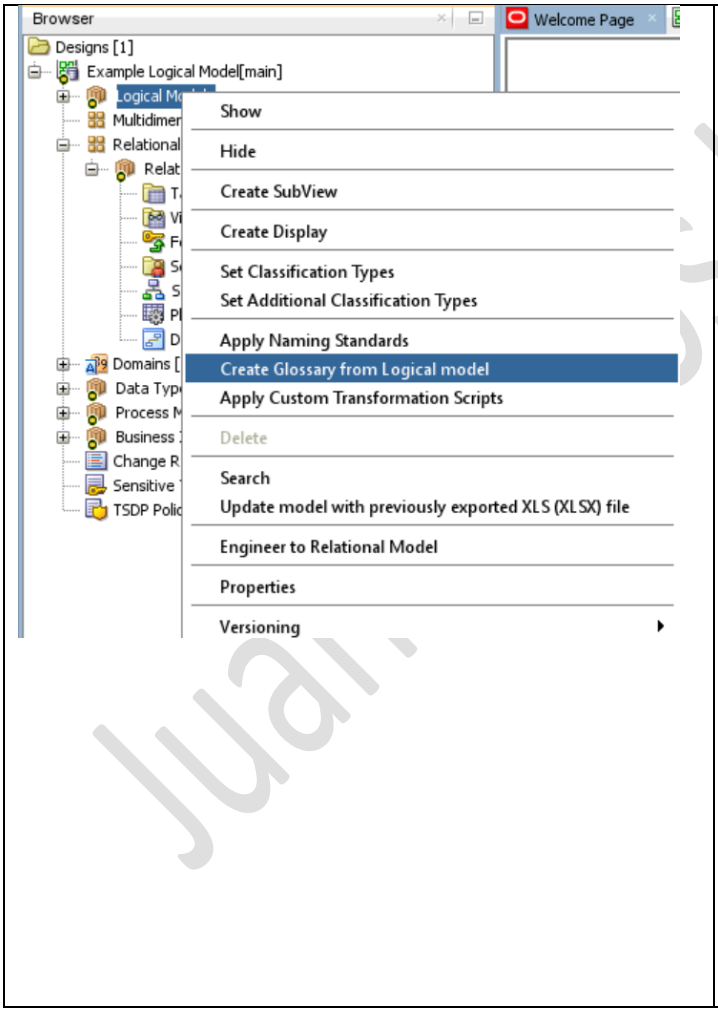
An Entity Relationship Model (ERM) does not highlight the physical and database constraints. It is essential to transform the ERM into a relational model that can serve as the basis for defining the physical implementation of the database.

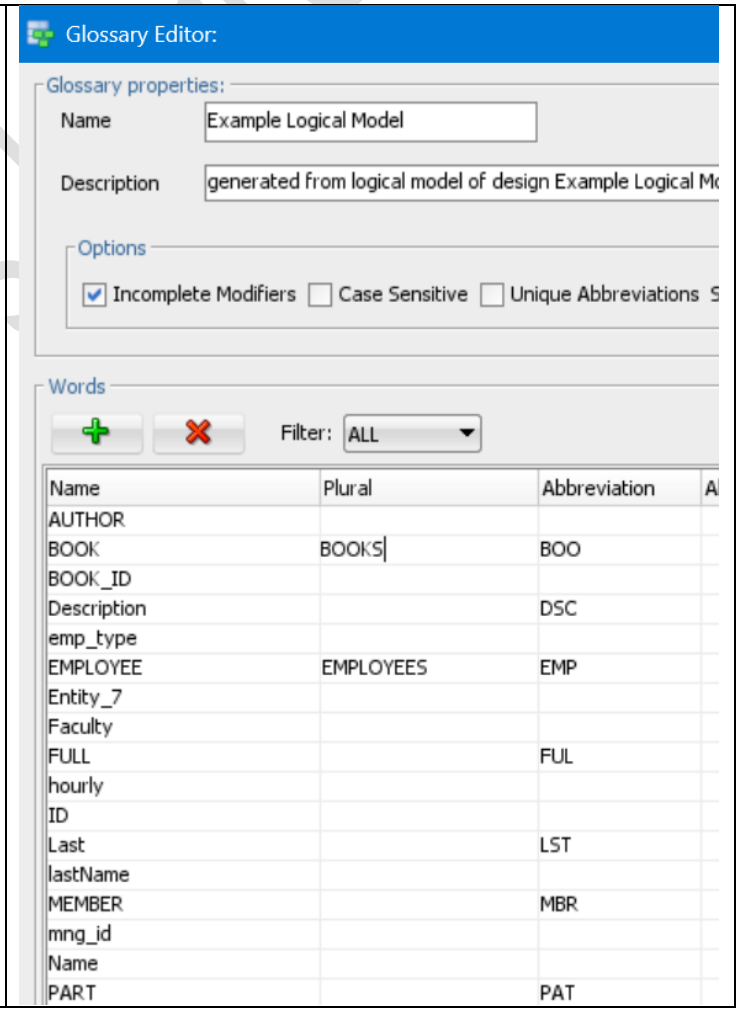
Table short names (abbreviations)
A unique abbreviation for every table is a very useful element for the naming of foreign key columns or foreign key constraints.

Applying Naming Standards from Logical to a Relational Model

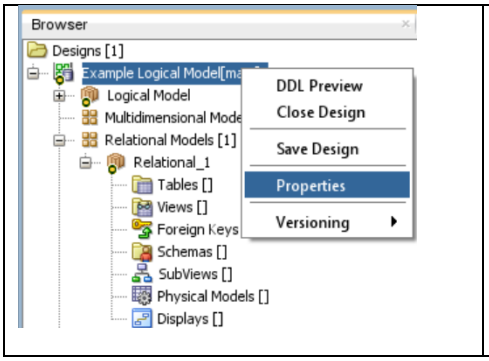
Logical Model (ERD)		Physical model (Relational Model)
Entity		Table
EMPLOYEE	to	EMPLOYEES
Last name	to	Last_name

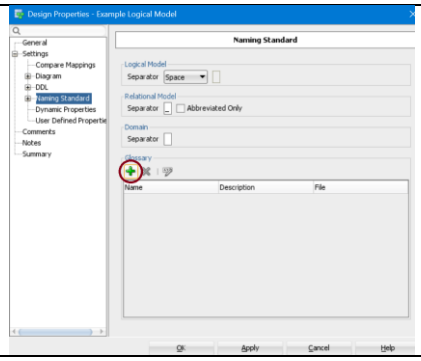
Create a Glossary from a Logical Model

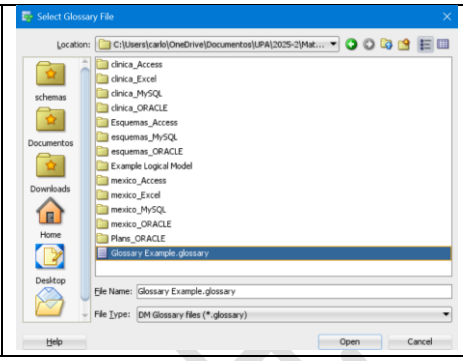




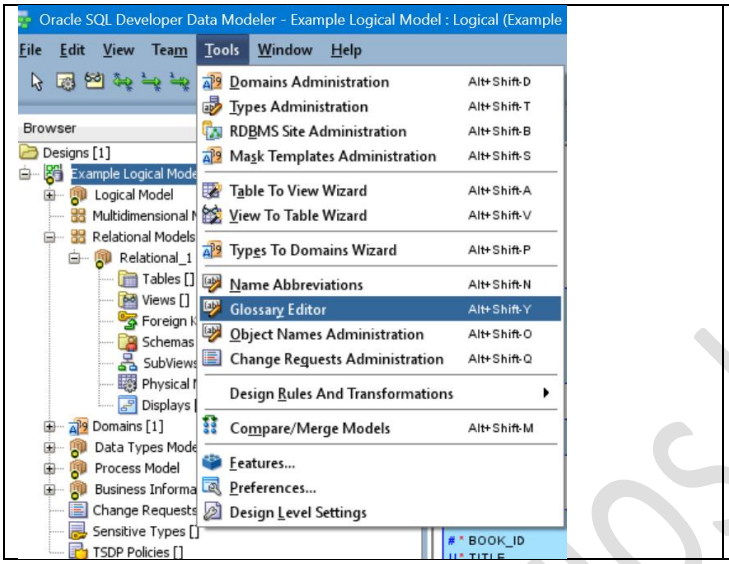
Adding the Glossary as the Naming Standard

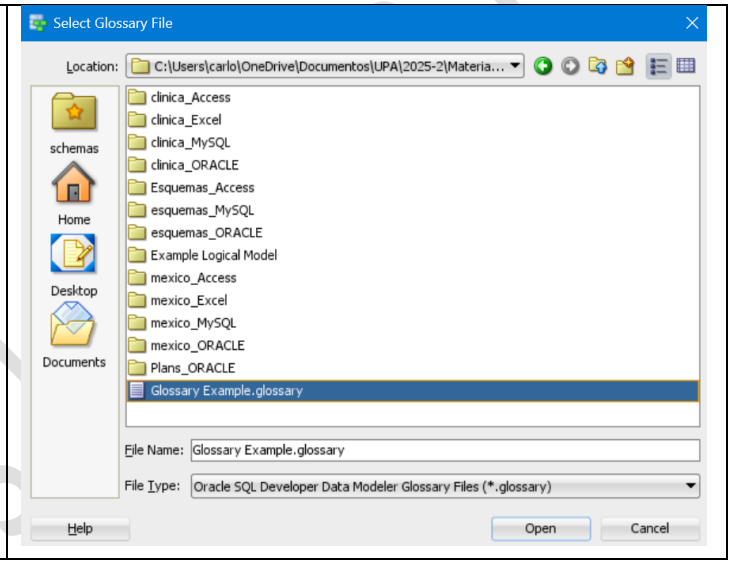






Or

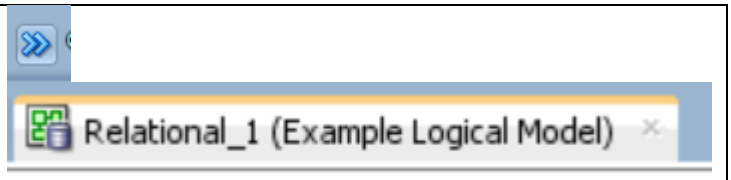


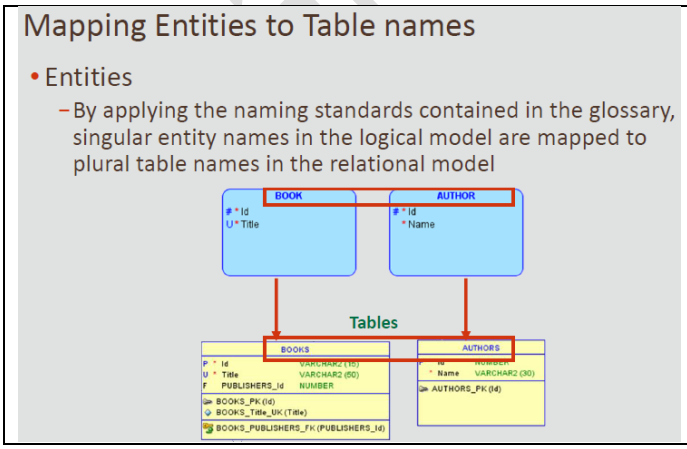


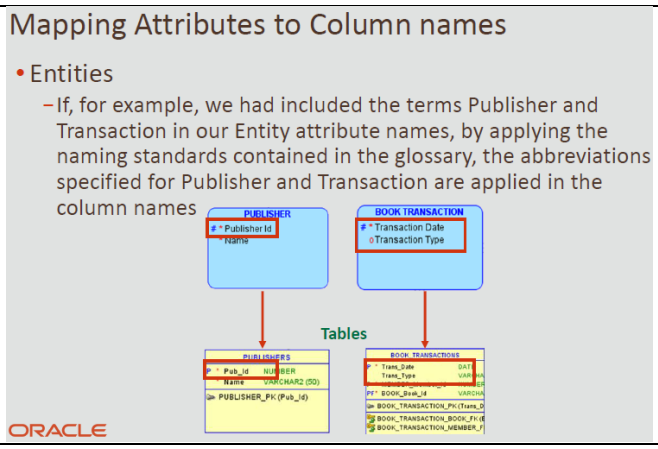
Applying the Naming Standard

To Apply the Glossary as the naming standard,

a) Engineer the Logical model once more
b) lick the relational tab to view the results







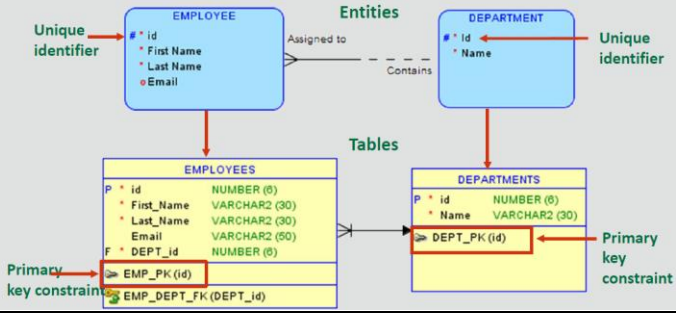
5.2. Mapping Primary and Foreign Keys

Decide on naming conventions for:

- Primary key constraint names
- Foreign key constraint names
- Foreign key column names
- Map subtypes to tables

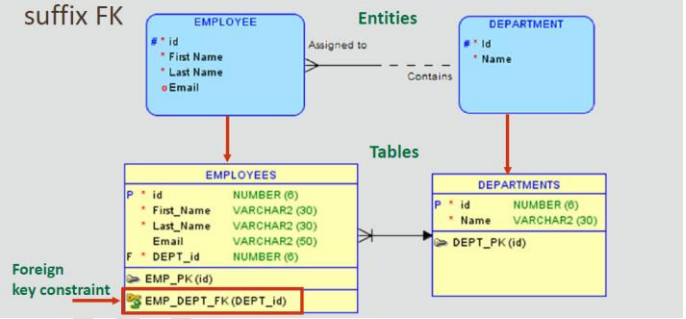
Naming Conventions for Primary Key Constraints

- Primary key constraints are named using the short table name, an underscore and the suffix PK



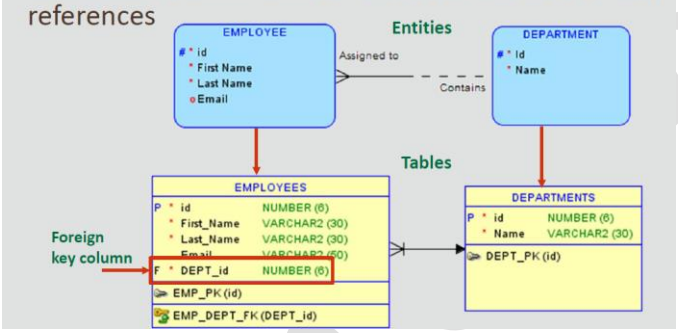
Naming Conventions for Foreign Key Constraints

- Foreign key constraints are named using the short table names of both tables, an underscore and the suffix FK



Naming Conventions for Foreign Key Columns

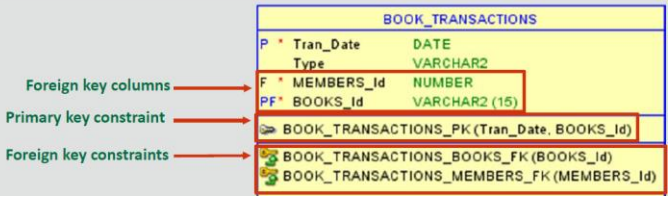
- Foreign key columns are named using the short table name and column name of the table the foreign key references



Sugerencia Personal: En ambas tablas
IdEmployee = ID_EMPLOYEE = employee_id

Apply naming standards in Oracle SQL Developer Data Modeler

- By default, constraint names are created using the full table name in Oracle SQL Developer Data Modeler. This can lead to constraint names that are very long, difficult to manage, and may exceed the maximum permitted number of characters in SQL



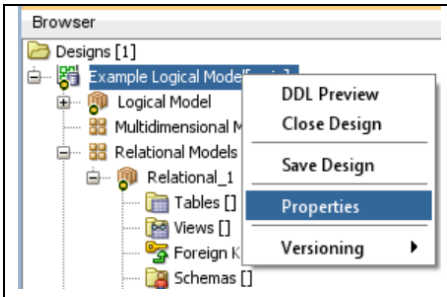
Create Name Abbreviations

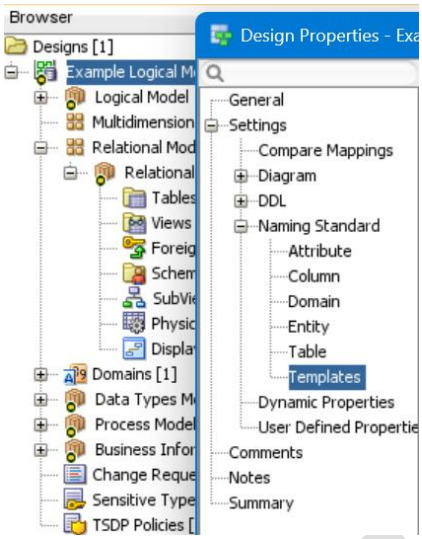
- To apply naming standards to constraint names, firstly create a .csv file in a spreadsheet application
- In the first column specify the full (plural) table names, and in the second column the abbreviation to use.
  Save as **.csv** file and note the location, example: constrain_names.csv

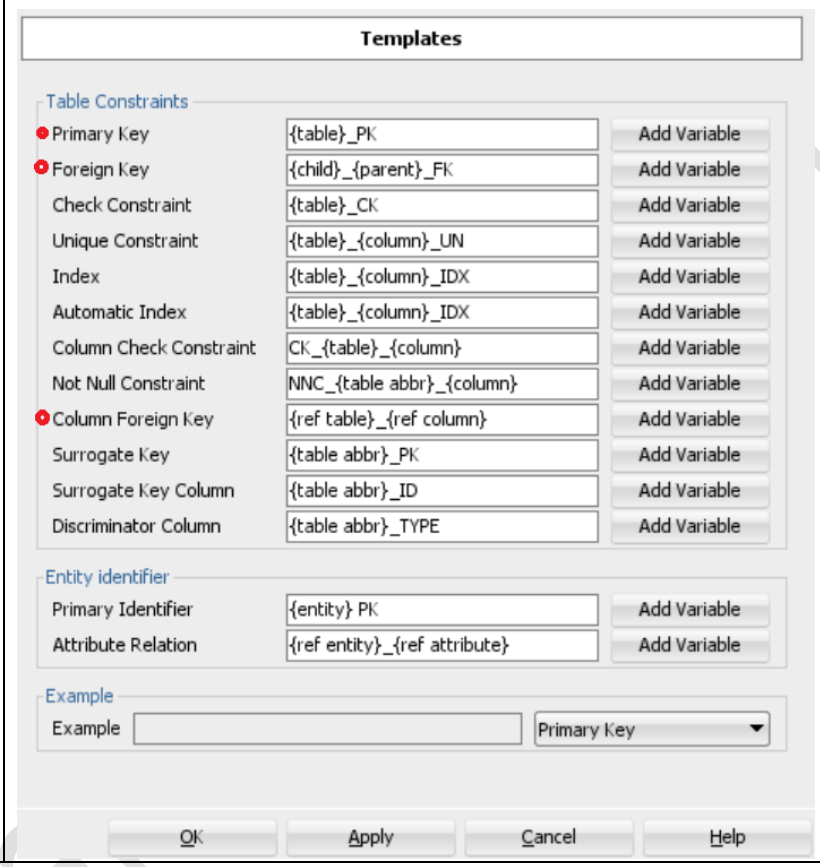
	A	B
1	PUBLISHERS	PUB
2	BOOKS	BK
3	AUTHORS	ATHR
4	MEMBERS	MEM
5	TRANSACTIONS	TRN

Defining Naming Templates

- To apply our abbreviations to our model, we need to set the pattern in the Naming template
- To access the Templates page:
    a. Right-click the name of the Design in the Object Browser
    b. Select Properties
    c. expand Settings and Naming Standard
    d. Select Templates

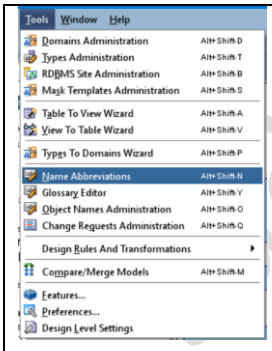


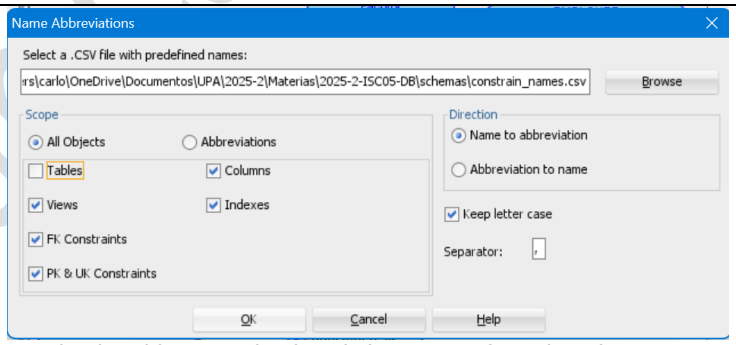




{ref column}

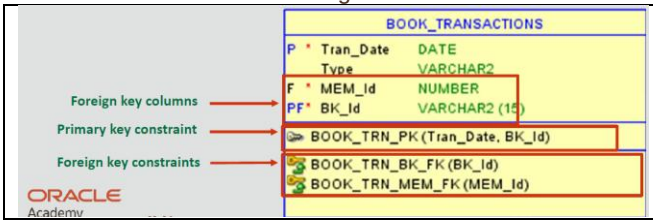
Applying Naming Abbreviations





Un-check Tables to maintain existing names from the Glossary

The Relational model is now engineered with constraint names that follow our naming standards



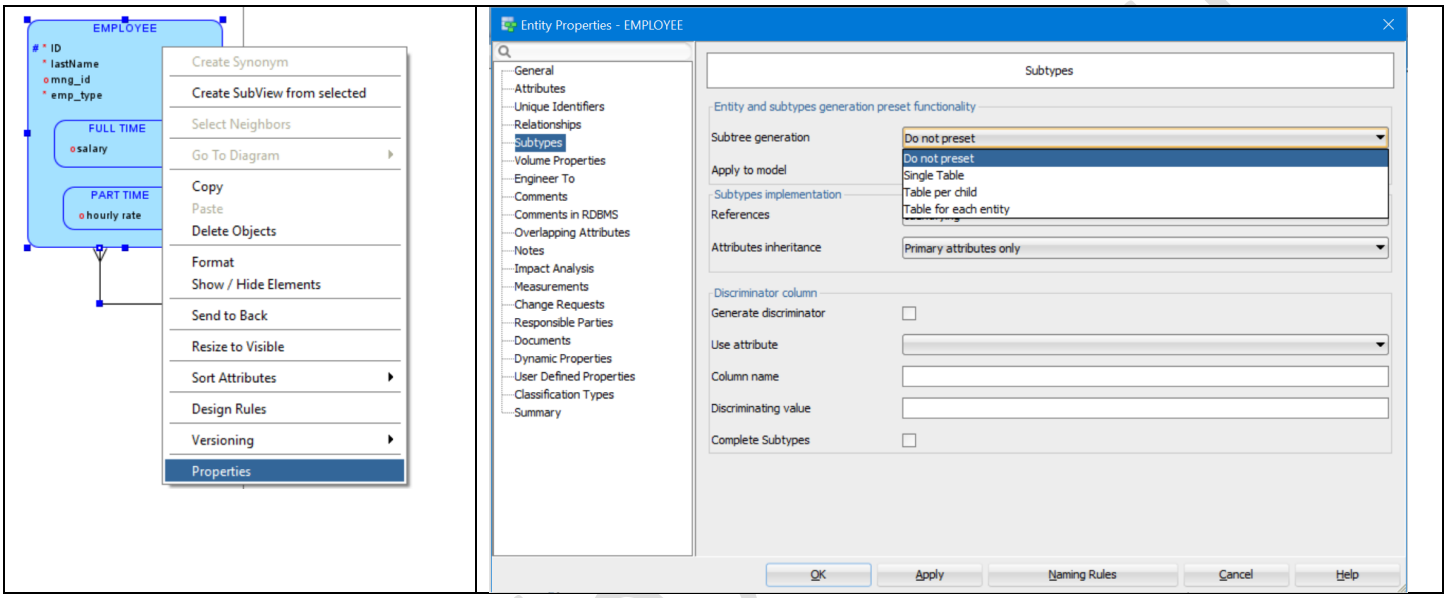
Mapping Subtypes to Tables

subtypes can be mapped to tables in a number of ways:

- One table (single table implementation)
- Two table (table for each child (subtype) implementation)

To select how subtypes are engineered in Oracle SQL Developer Data Modeler:

a) Double click the Super type entity to edit properties
b) Select Subtypes
c) Select required method for generating subtypes from the Subtree generation dropdown box options



→

6. Introduction to SQL

6.1. Introduction to Oracle Application Express

Oracle APEX has these components:

- SQL Workshop: To learn SQL
- Application Builder: To design an application
- Object Browser

Upload and run a script in APEX
File zip -> 2 – Course -> Code Files -> DFo_Section_6_1 -> obl Sports.zip (obl Sports.ddl)
Create table…
Alter table…

SQL workshop

(Zip - obl Sports Script) and must first be extracted: chose the obl Sports.ddl
run the script
DESCRIBE customers;

6.2. Structured Query Language (SQL)

Types of SQL Commands

- DDL(Data Definition Language) –defines database structures (CREATE, ALTER, drop, rename)
- DML(Data Manipulation Language) –manipulates data (INSERT, UPDATE, DELETE)
- DQL(Data Query Language) –SELECTs data
- DCL(Data Control Language) –controls user access (Grant, revoke)
- TCL (Transactional Control Language) –manages database transactions (begin|savepoint, commit, rollback)

Three tools are:

- SQL*Plus (consola)
- SQLDeveloper
- APEX (Oracle Application Express)

→

6.3. Data Definition Language (DDL)

Object	Description
Table	Is the basic unit of storage; consists of rows
View	Logically represents subsets of data from one or more tables
Sequence	Generates numeric values
Index	Improves the performance of some queries
Synonym	Gives an alternative name to an object

CREATE TABLE [schema.]table (column datatype [DEFAULT expr][, ...]);	CREATE TABLE dept(deptno NUMBER(2), dname VARCHAR2(14), loc VARCHAR2(13), create_date DATE DEFAULT SYSDATE);

```
Describe dept
```

Table	Column	Data Type	Length	Precision	Scale	Primary Key	Nullable	Default	Comment
DEPT	DEPTNO	NUMBER	-	2	0	-		-	-
	DNAME	VARCHAR2	14	-	-	-		-	-
	LOC	VARCHAR2	13	-	-	-		-	-
	CREATE_DATE	DATE	7	-	-	-		SYSDATE	-

Data Types

Data Type	Description
VARCHAR2(size)	Variable-length character data (A maximum size must be specified; minimum size is 1.) Maximum size: 32767 bytes
CHAR(size)	Fixed-length character data of length (size) bytes. (Default and minimum size is 1; maximum size is 2,000)
NUMBER(p,s)	Variable-length numeric data. Precision is p, and scale is s. (Precision is the total number of decimal digits, and scale is the number of digits to the right of the decimal point; precision can range from 1 to 38, and scale can range from -84 to 127.)
DATE	Date and time values to the nearest second between January 1, 4712 B.C, and December 31, 9999 A.D.
LONG	Variable-length character data (up to 2 GB)
CLOB	A character large object (CLOB) containing single-byte or multibyte characters. Maximum size is (4 GB -1) * (DB_BLOCK_SIZE); stores national character set data.
NCLOB	A CLOB containing Unicode characters. Both fixed-width and variable-width character sets are supported, both using the database national character set. Maximum size is (4 GB -1) * (database block size); stores national character set data.
RAW (Size)	Raw binary data of length size bytes. You must specify size for a RAW value. Maximum size: 32767 bytes if MAX_SQL_STRING_SIZE = EXTENDED 4000 bytes if MAX_SQL_STRING_SIZE = LEGACY
LONG RAW	Raw binary data of variable length up to 2 GB.
BLOB	A binary large object. Maximum size is (4 GB -1) * (DB_BLOCK_SIZE initialization parameter (8 TB to 128 TB)).
BFILE	Binary data stored in an external file (up to 4 GB).
ROWID	Base 64 string representing the unique address of a row in its table. This data type is primarily for values returned by the ROWID pseudocolumn.

Date Data Types

Data Type	Description
TIMESTAMP	Enables storage of time as a date with fractional seconds. It stores the year, month, day, hour, minute, the second value of the DATE data types, and the fractional seconds value. There are several variations of this data type, such as WITH TIMEZONE and WITH LOCALTIMEZONE.
INTERVAL YEAR TO MONTH	Enables storage of time as an interval of years and months. Used to represent the difference between two datetime values in which the only significant portions are the year and month.
INTERVAL DAY TO SECOND	Enables storage of time as an interval of days, hours, minutes, and seconds; used to represent the precise difference between two datetime values.
TIMESTAMP WITH TIME ZONE	Variant of TIMESTAMP that includes a time zone region name or time zone offset in its value.

```
CREATE TABLE table_ts(
c_id NUMBER(6),
c_ts TIMESTAMP
);
```

```
INSERT INTO table_ts
VALUES(1, '01-JAN-2003 2:00:00');
```

```
CREATE TABLE time_table(
start_time TIMESTAMP,
duration_1 INTERVAL DAY (6) TO SECOND (5),
duration_2 INTERVAL YEAR TO MONTH
);
```

Data Integrity Constraints

Constraints	Description
NOT NULL	The column cannot contain a null value (constraint only at the column level)
UNIQUE	The values for a column or a combination of columns must be unique for all rows in the table
PRIMARY KEY	The column (or a combination of columns) must contain the unique AND IS NOT NULL value for all rows
FOREIGN KEY REFERENCES	The column (or a combination of columns) must establish and enforce a reference to a column or a combination of columns in another (or the same) table
CHECK	A condition must be true

Name a constraint (otherwise, the Oracle server generates a name in the SYS_Cn format). Example SYS_C0014370

Defining Constraints

CREATE TABLE [schema.]table (column datatype [DEFAULT expr] [**column_constraint**], ... [**table_constraint**][,...]);	• Column-level constraint syntax: column datatype [CONSTRAINT constraint_name] constraint_type, • Table-level constraint syntax: Column datatype, ... [CONSTRAINT constraint_name] constraint_type (column, …),

• Column-level constraint:

```
CREATE TABLE employees(
  employee_id  NUMBER(6)CONSTRAINT emp_emp_id_pk
                        PRIMARY KEY,
  first_name   VARCHAR2(20),
  ...
);
```

• Table-level constraint:

```
CREATE TABLE employees(
  employee_id  NUMBER(6),
  first_name   VARCHAR2(20),
  ...
  job_id       VARCHAR2(10),
  CONSTRAINT emp_emp_id_pk PRIMARY KEY (employee_id)
);
```

FOREIGN KEY Constraint

• Defined at the table level :

```
CREATE TABLE employees(
  employee_id      NUMBER(6),
  last_name        VARCHAR2(25),
  email            VARCHAR2(25),
  salary           NUMBER(8,2),
  commission_pct   NUMBER(2,2),
  hire_date        DATE,
  ...
  department_id    NUMBER(4),
  CONSTRAINT emp_dept_fk FOREIGN KEY (department_id)
     REFERENCES departments(department_id)
);
```

• Defined at the column level:

```
CREATE TABLE employees(
  employee_id      NUMBER(6),
  last_name        VARCHAR2(25),
  email            VARCHAR2(25),
  salary           NUMBER(8,2),
  commission_pct   NUMBER(2,2),
  hire_date        DATE,
  ...
  department_id    NUMBER(4) CONSTRAINT emp_dept_fk
                   REFERENCES departments(department_id)
);
```

FOREIGN KEY Constraint: Keywords

- FOREIGN KEY: Defines the column in the child table at the table-constraint level
- REFERENCES: Identifies the table and column in the parent table
- ON DELETE CASCADE: Deletes the dependent rows in the child table when a row in the parent table is deleted
- ON DELETE SET NULL: Converts dependent foreign key values to null

CHECK Constraint

```
CREATE TABLE employees(
employee_id NUMBER(6),
last_name VARCHAR2(25),
salary NUMBER(8,2) CONSTRAINT emp_salary_min CHECK (salary > 0),
commission_pct NUMBER(2,2),
hire_date DATE DEFAULT SYSDATE,
email VARCHAR2(25),
CONSTRAINT hire_date_min CHECK (hire_date >'01-JAN-2018')
);
```

ALTER TABLE Statement

Use the ALTER TABLE statement to change the table structure:

- Add a column
- Modify an existing column definition
- Define a default value for the new column
- Drop a column
- Rename a column
- Change a table to read-only status

ALTER TABLE table ADD (column data type [DEFAULT expr] [, column data type]...);	ALTER TABLE employees ADD termination_date DATE;
ALTER TABLE table MODIFY (column data type [DEFAULT expr] [, column data type]...);	ALTER TABLE employees MODIFY first_name VARCHAR2(30);
ALTER TABLE table DROP (column [, column] …);	ALTER TABLE employees DROP (termination_date);

SET UNUSED Option (Exclusive Oracle)

```
ALTER TABLE   <table_name>
SET   UNUSED(<column_name> [ , <column_name>]);
OR
ALTER TABLE  <table_name>
SET   UNUSED COLUMN <column_name> [ , <column_name>];
```

```
ALTER TABLE dept
SET UNUSED (dname);
```

```
ALTER TABLE <table_name>
DROP  UNUSED COLUMNS;
```

```
ALTER TABLE dept
DROP UNUSED COLUMNS;
```

Read-Only Tables

- Put a table in read-only mode, which prevents DDL or DML changes during table maintenance
- Put the table back into read/write mode

```
ALTER TABLE dept READ ONLY;
  --perform table maintenance and then
  --return table back to read/write mode

ALTER TABLE dept READ WRITE;
```

Dropping a Table

DROP TABLE dept; -- Observar el Directorio PURGE, la tabla todavia existe

→

6.4. Data Manipulation Language (DML)

A DML statement is executed when you:

- Add new rows to a table (INSERT)
- Modify existing rows in a table (UPDATE)
- Remove existing rows from a table (DELETE)

A **transaction** consists of a collection of DML statements that form a logical unit of work

INSERT Statement Syntax

INSERT INTO table [(column [, columna ...])] VALUES (value [, value...]); -- only one row is inserted at a time CREATE TABLE copy_departments AS (SELECT * FROM departments); -- does not copy all constraints (only NOT NULL)	INSERT INTO copy_departments VALUES (40, 'Advertising', 201, 1800, SYSDATE); INSERT INTO copy_departments VALUES (100, 'Finance', NULL, NULL, TO_DATE('Dec 7, 2002', 'MON DD, YYYY')); INSERT INTO copy_departments (department_id, department_name) VALUES(30,'Purchasing');

Default: DD-MON-RR
Tools -> preferences -> DB -> NLS
alter session set nls_date_format = 'dd-mm-yyyy'

-- MySQL: Dia de la semana (1-7, 1=Dom)
select dayofweek(sysdate()) from dual;
 	
(columna, position, [length])
select substr('UNIVERSIDAD', 7,4) FROM DUAL;
select substr('UNIVERSIDAD', 7) FROM DUAL;
	-- Oracle: Dia de la semana (1-7, 1=Dom)
	select to_char(sysdate, 'D') from dual;

UPDATE Statement Syntax

UPDATE table SET column = value [, column = value, ...] [WHERE condition]; **-- Si se omite afecta todos los registros**	UPDATE copy_employees SET department_id = 50, Salary = 12000, COMMISSION_PCT = NULL WHERE employee_id = 113;

DELETE Statement

DELETE [FROM] table [WHERE condition]; **-- Si se omite afecta todos los registros**	DELETE FROM copy_departments WHERE department_name = 'Purchasing';

TRUNCATE Statement

Removes all rows from a table, leaving the table empty and the table structure intact

TRUNCATE TABLE table_name;	TRUNCATE TABLE copy_employees;

→

6.5. Transaction Control Language (TCL)

COMMIT, ROLLBACK, and SAVEPOINT are not supported in **Oracle Application Express**, due to the way Oracle Application Express manages connections to the database.

Transactions give you more flexibility and control when you change data, and they ensure data consistency in the event of user process failure or system failure.

All or nothing
For example, a transfer of funds between two accounts should include the debit in one account and the credit to another account in the same amount. Both actions should either fail or succeed together; the credit should not be committed without the debit.

Database Transactions

A database transaction consists of one of the following statements:

- DML statements that represent one consistent change to the data
- One DDL statement
- One TCL statement

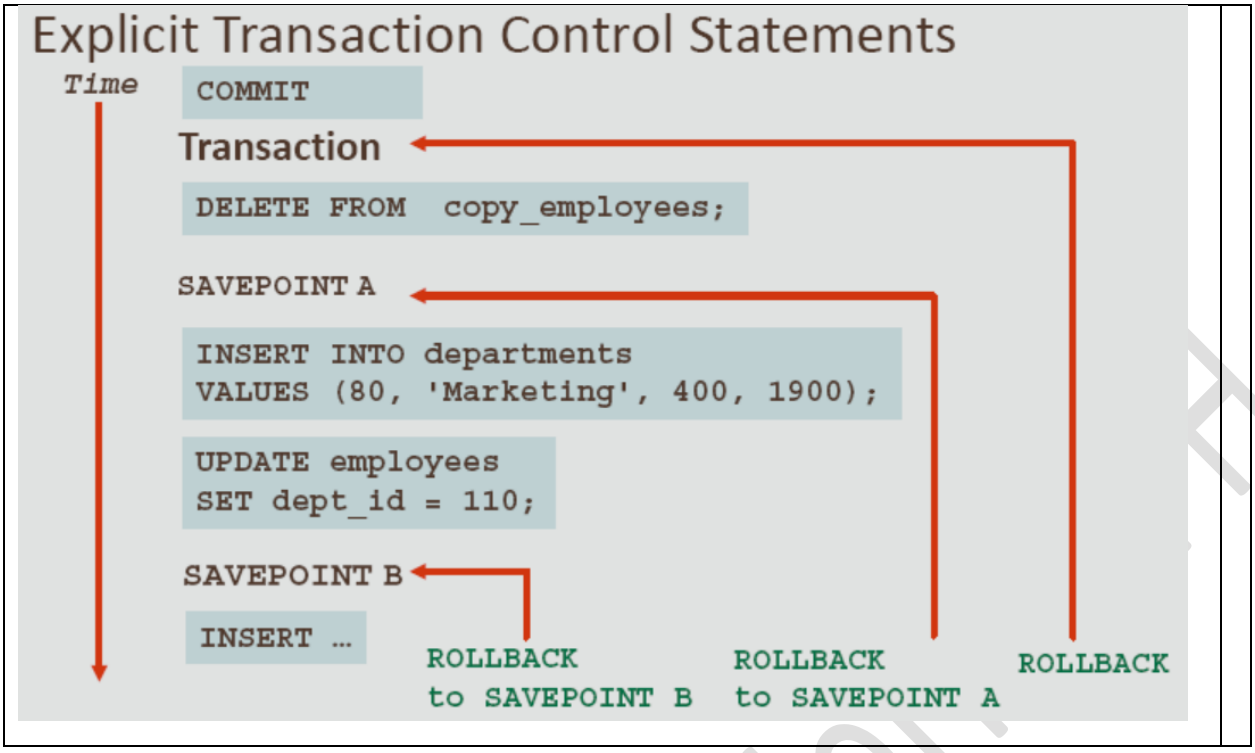
Database Transactions: Start and End

A transaction begins when the first DML SQL statement is executed
It ends with one of the following events:

- A COMMIT or ROLLBACK statement is issued
- A DDL or TCL statement executes (automatic commit)
- The user exits SQL software being used
- The system crashes

Transaction Control Statements

Statement	Description
COMMIT	Ends the current transaction by making all pending data changes permanent.
SAVEPOINT name	Marks a savepoint within the current transaction.
ROLLBACK	Ends the current transaction by discarding all pending data changes.
ROLLBACK TO SAVEPOINT name	Rolls back the current transaction to the specified savepoint, thereby discarding any changes and/or savepoints that were created after the savepoint to which you are rolling back. If you omit the TO SAVEPOINT clause, the ROLLBACK statement rolls back the entire transaction. Because savepoints are logical, there is no way to list the savepoints that you created.



Implicit Transaction Processing

State of the Data Before COMMIT or ROLLBACK

State of the Data After COMMIT

Read Consistency

→

6.6. Retrieving Data Using SELECT

SELECT {* | [DISTINCT] *column* | *expression* [*alias*],...}
FROM *table;*

Arithmetic Expressions

Operator Precedence

Defining a Null Value

- Null is a value that is unavailable, unassigned, unknown, or inapplicable
- Null is not the same as zero or a blank space
- Any arithmetic expression containing a null value will evaluate to null

```
SELECT last_name, 12*salary*commission_pct AS "Comision Anual"
FROM a_employees;
```

Concatenation Operator

```
SELECT 'Hello ' || first_name || ' ' || last_name "Complete Name"
FROM a_employees;
```

```
                          Complete Name
                          Hello Ellen Abel
                          Hello Curtis Davies
```

Alternative Quote (q) Operator

select your own quotation mark delimiter: { }, [ ], ( )

```
SELECT department_name || q'{ Department's Manager Id: }' || manager_id
AS "Department and Manager"
FROM a_departments;
```

```
                     Department and Manager
                Marketing Department's Manager Id: 201
                Shipping  Department's Manager Id: 124
```

Duplicate Rows

```
SELECT DISTINCT department_id
FROM employees;
```

DESC[RIBE] tablename

```
Name            Null?    Type
--------------- -------- ------------
EMPLOYEE_ID     NOT NULL NUMBER(6)
FIRST_NAME               VARCHAR2(20)
```

→

6.7. Restricting Data Using WHERE

This lesson covers the following objectives:

Describe the rules of precedence for operators in an expression

Limit rows with:

- WHERE clause
- Comparison operators using =, <=,>=, <>,>,<, !=,^=, BETWEEN,IN,LIKE and NULL conditions
- Logical conditions using AND,OR and NOT operators

```
SELECT *|{[DISTINCT] column|expression [alias],...}
FROM table
[WHERE logical expression(s)];
```

Character Strings and Dates

- Character strings and date values are enclosed in single quotation marks
- The default date display format is DD-Mon-YYYY

```
select last_name, job_id, hire_date
from a_employees
where hire_date = '07-JuN-1994'  -- no sensitivo
  and job_id   != 'AC_MGR';      --    sensitivo
```

Comparison Operators

Operator	Meaning	
=	Equal to	
>	Greater than	
>=	Greater than or equal to	
<	Less than	
<=	Less than or equal to	
<> ¡=	Not equal to	
BETWEEN…AND…	Between two values (inclusive)	between 18 and 30
IN (set) NOT IN	Match any of a list of values	in (101, 200, 300)
LIKE	Match a character pattern	like 'A%' '_o%'
IS NULL	Is a null value	manager_id IS NULL

Pattern Matching: LIKE Operator

- % denotes zero or more characters
- _ denotes one character

Combining Wildcard Characters

```
SELECT employee_id, last_name, job_id
FROM a_employees
WHERE job_id LIKE '%SA\_%' ESCAPE '\';
```

Defining Conditions Using the Logical Operators

A logical condition combines the result of two component conditions to produce a single result based on those conditions or if using NOT it inverts the result of a single condition.

Operator	Meaning
AND	Returns TRUE if both component conditions are TRUE
OR	Returns TRUE if either component condition is TRUE
NOT	Returns TRUEif the condition is FALSE Returns FALSE if the condition is TRUE

```
SELECT last_name, job_id
FROM a_employees
WHERE job_id NOT IN ('IT_PROG', 'ST_CLERK');
```

Rules of Precedence

Precedence	Operator
1	Arithmetic operators
2	Concatenation operator
3	Comparison conditions
4	IS [NOT] NULL, LIKE, [NOT] IN
5	[NOT] BETWEEN
6	Not equal to
7	NOT logical operator
8	AND logical operator
9	OR logical operator

Find the mistake:

```
SELECT last_name, job_id, salary
FROM a_employees
WHERE job_id = 'SA_REP'
OR    job_id = 'AD_PRES'
AND salary > 15000;
```

→

6.8. Sorting Data Using ORDER BY

Sort the retrieved rows with the ORDER BY clause:

- ASC: Ascending order (default)
- DESC: Descending order

Null values are displayed last in ascending order and first in descending order
You can sort by column alias.
Sorting by using the column's numeric position.
Sorting by multiple columns.

```
SELECT expr
 FROM  table
 [WHERE condition(s)]
 [ORDER BY {column, expr, numeric_position} [ASC|DESC]];
```

The order of execution of a SELECT statement is as follows:

FROM clause: locates the table that contains the data
WHERE clause: restricts the rows to be returned
SELECT clause: selects from the reduced data set the columns requested
ORDER BY clause: orders the result set

Using ROWNUM for Top-N-Analysis

```
SELECT employee_id, last_name, hire_date
FROM a_employees
WHERE extract(year from hire_date) <= 1990
--where to_char(hire_date, 'YYYY') <= 1990
--and ROWNUM <= 2;
```

Se presentan en el orden que fueron insertados

	EMPLOYEE_ID	LAST_NAME	HIRE_DATE
1	100	King	17-JUN-87
2	101	Kochhar	21-SEP-89
3	200	Whalen	17-SEP-87
4	103	Hunold	03-JAN-90

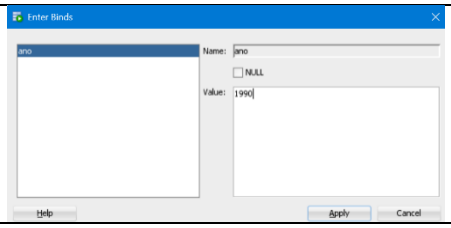
No se pueden usar al mismo tiempo:
ROWNUM y ORDER BY

```
SELECT employee_id, last_name, hire_date
FROM a_employees
WHERE to_char(hire_date, 'YYYY') <= 1990
  and ROWNUM <= 2;
ORDER BY hire_date; --No funciona el ORDER BY

100     King     17-JUN-87
101     Kochhar 21-SEP-89
```

```
select *
from (SELECT employee_id, last_name, hire_date
      FROM a_employees
      WHERE extract(year from hire_date) <= 1990
      ORDER BY hire_date)
where ROWNUM <=2;

100     King     17-JUN-87
200     Whalen  17-SEP-87
```

Substitution Variable in WHERE with colon(:)

```
SELECT employee_id, last_name, hire_date
FROM a_employees
WHERE extract(year from hire_date) <= :ano
order by 2;
```



varchar: IT_PROG

date: 07-jun-94

number: 1990

6.9. Joining Tables Using JOIN

Types of Joins

- Join with the ON Clause
- Join with the USING Clause
- Natural join with the NATURAL JOIN clause
- { LEFT | RIGHT | FULL} OUTER JOIN
- CROSS JOIN
- Without join

```
SELECT table1.column, table2.column
FROM   table1
    [JOIN table2 ON (table1.column_name= table2.column_name)]|
    [JOIN table2 USING (column_name)] |
    [NATURAL JOIN table2] |
    [LEFT | RIGHT | FULL OUTER JOIN table2
                ON (table1.column_name= table2.column_name)]|
    [CROSS JOIN table2];
```

Without JOIN

```
SELECT table1.column, table2.column, table3.column
FROM   table1, table2, table3
WHERE table1.column = table2.column
  And table2.column = table3.column
  And ciudades.idCiudad = 'Ags';
```

Qualifying Ambiguous Column Names

```
SELECT department_id, d.department_name, d.manager_id department_head,
        employee_id, e.first_name, e.manager_id Report_To
FROM a_departments d JOIN a_employees e USING(department_id)
-- FROM a_departments d JOIN a_employees e on d.department_id = e.department_id
order by 1,employee_id;
```

DEPARTMENT_ID	DEPARTMENT_NAME	DEPARTMENT_HEAD	EMPLOYEE_ID	FIRST_NAME	REPORT_TO

```
Without join
SELECT D.department_id, d.department_name, d.manager_id department_head,
       employee_id, e.first_name, e.manager_id Report_To
FROM a_departments d, a_employees e
where d.department_id = e.department_id
order by 1,employee_id;
```

Join 3 tables

```
SELECT employee_id, city, department_name "Nombre Depto"
FROM a_employees e
   JOIN a_departments d     ON d.department_id = e.department_id
   inner JOIN a_locations l ON d.location_id   = l.location_id
WHERE CITY IN ('Seattle', 'Toronto');

SELECT employee_id, city, department_name "Nombre Depto"
FROM a_employees e, a_departments d, a_locations l
WHERE d.department_id = e.department_id
  and d.location_id   = l.location_id
  and CITY IN ('Seattle', 'Toronto');
```

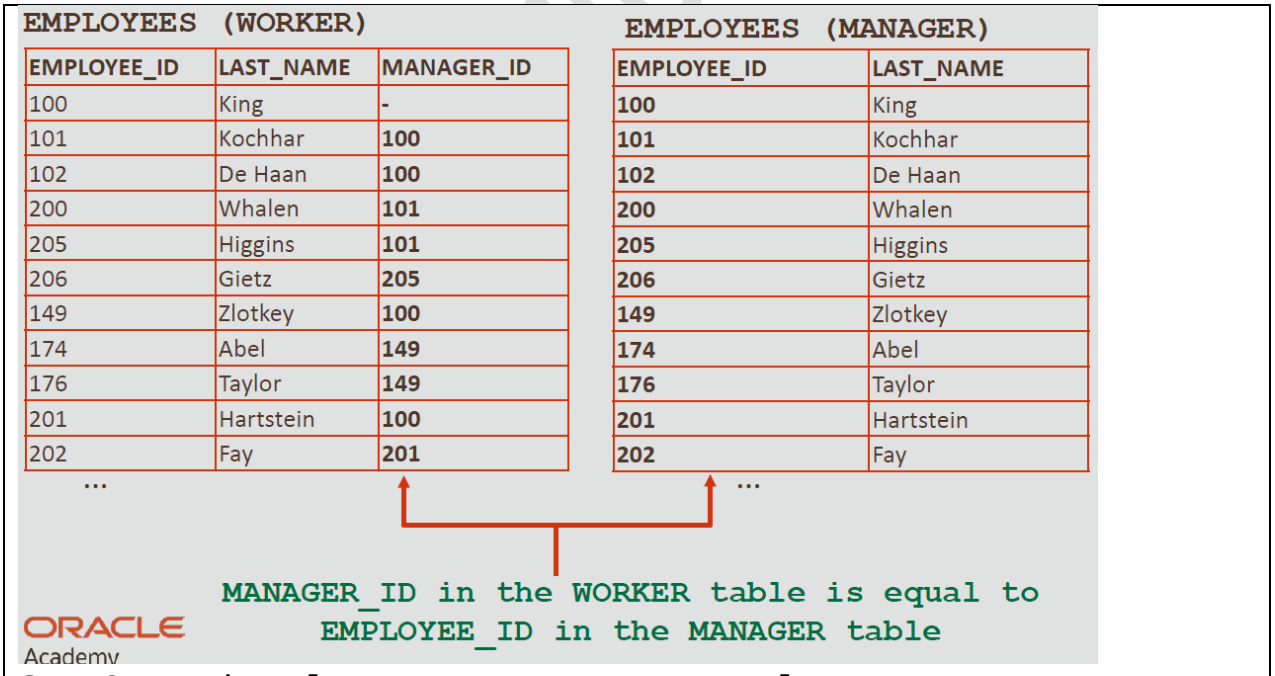
Creating Natural Joins

- The NATURAL JOIN clause is based on all columns in the two tables that have the same name and the same data type
- It selects rows from the two tables that have equal values in all matched columns
- If columns with the same names have different data types, an error is returned

Mismo Nombre, mismo Tipo

```
SELECT department_id, department_name, location_id, city
FROM departments NATURAL JOIN locations;
```

Joining a Table to Itself




```
SELECT worker.last_name emp, manager.last_name mgr
FROM a_employees worker JOIN a_employees manager
  ON worker.manager_id = manager.employee_id;
```

Retrieving Records with Nonequijoins

```
SELECT employee_id, e.last_name, e.salary, j.grade_level
FROM a_employees e JOIN a_job_grades j
ON e.salary BETWEEN j.lowest_sal AND j.highest_sal
```

OUTER Joins

```
SELECT d.department_id, d.department_name, employee_id, e.last_name
FROM a_departments d  LEFT OUTER JOIN a_employees e
ON (d.department_id = e.department_id);

SELECT d.department_id, d.department_name, employee_id, e.last_name
FROM a_departments d  RIGHT OUTER JOIN a_employees e
ON (d.department_id = e.department_id);

SELECT d.department_id, d.department_name, employee_id, e.last_name
FROM a_departments d  FULL OUTER JOIN a_employees e
ON (d.department_id = e.department_id);

Cartesian Product OR Cross Joins
Departments = 8  rows
Employees   = 20 rows

SELECT d.department_id, d.department_name, employee_id, e.last_name
FROM a_departments d  CROSS JOIN a_employees e
ORDER BY 1,3;  -- 160 rows

SELECT d.department_id, d.department_name, employee_id, e.last_name
FROM a_departments d, a_employees e
ORDER BY 1,3;  -- 160 rows
```

→

```
select m.employee_id mng, m.last_name , w.employee_id emp, w.last_name empleado
from a_employees m inner join a_employees w
  on m.employee_id = w.manager_id
order by m.employee_id, w.employee_id;
```

MNG	LAST_NAME	EMP	EMPLEADO

```
select w.employee_id emp, w.last_name empleado, m.employee_id mng, m.last_name
from a_employees w inner join a_employees m
  on w.manager_id = m.employee_id
order by m.employee_id, w.employee_id;
```

EMP	EMPLEADO	MNG	LAST_NAME