Arquitectura de bases de dades

PRÀCTICA 1:

Extensió *Object-Relational*

Estudiant: *Jordi Bericat Ruz*

Professor col·laborador: *Maria Teresa Bordas Garcia*

Semestre: *Tardor 2020-21 (Aula 1)*

Índex

Pregunta 1

a)

Què són els Objects View d'Oracle? Què ens proporciona la utilització d'un Object View? Quins són els passos a realitzar per crear un Object View?

Els Objects-View[1] d'Oracle s'inclouen a les funcionalitats incorporades a SQL per l'extensió del llenguatge[2] realitzada per a permetre la implantació de bases de dades seguint el model de l'orientació a objectes mitjançant aquest llenguatge declaratiu, així com la seva coexistència amb el model relacional. Concretament:

- D'una banda, ens permeten disposar dels avantatges del paradigma d'orientació a objectes sobre dades existents prèviament a una base de dades relacional (això és, mecanismes com la encapsulació de mètodes en una classe, l'herència, sobrecàrrega / sobre-escriptura de classes, generalització / especialització, etc), fet que alhora possibilita l'accés de lectura (sense possibilitat de modificació o *UPDATE*) de dades de la RDBMS mitjançant qualsevol llenguatge d'alt nivell orientat a objectes (OOL). En altres paraules, gràcies als Object-View podem "mapar" les dades d'una taula implementada utilitzant el model relacional a una altra taula "virtual" (vista) implementada utilitzant el model orientat a objectes.

- De l'altra, ens proporcionen funcionalitats anàlogues a les vistes d'una RDBMS però sobre taules d'objectes d'una ORDBMS, per exemple[3], per a mostrar dades relatives a atributs d'objectes continguts a diferents taules en una mateixa vista (abstracció de la complexitat de l'estructura de la BD), per a restringir l'accés d'escriptura a determinats atributs o la invocació de mètodes, emmagatzemar consultes complexes, o bé optimitzar la seva execució.

Pel que fa als passos a realitzar per a crear una Object-View, estudiarem el primer cas d'ús, ja que aquest inclou el segon de manera implícita:

---

[1] Fonts:

- https://docs.oracle.com/cd/A97335_02/apps.102/bc4j/developing_bc_projects/bc_awhatisavo.htm
- Pàg. 22 del mòdul 2 dels materials de l'assignatura

[2] SQL-99 update -> Desenvolupar (p.16 M2)

[3]

1. En primer lloc, cal disposar d'una o més taules relacionals al ORDMBS. Les podem crear amb la clàusula *CREATE TABLE name ();*
2. Seguidament, caldrà mapar les dades relacionals desitjades a la vista mitjançant la creació d'un tipus com a objecte. A efectes pràctics, això ho podem realitzar mitjançant la clàusula *CREATE TYPE name AS OBECT();*
3. Finalment ja podem mapar les dades relacionals desitjades mitjançant la creació de l'Object View amb la clàusula *CREATE VIEW name OF type AS SELECT statement;*

b)

Quins són els mecanismes que podem utilitzar per establir relacions entre elements de diferents taules o objectes?

En el cas de taules relacionals, podem garantir integritat referencial (i en conseqüència establir relacions) de manera implícita mitjançant la utilització de claus foranies (FK). Ara bé, per a establir relacions entre objectes caldrà emprar punters a objectes (REF's) i a l'hora especificar explícitament que l'objecte referenciat ha d'existir (és a dir, que no pot ser eliminat mentre sigui referenciat).

c)

Quines col·leccions d'objectes podem definir en Oracle sota l'extensió object-relational? Quan utilitzarem cada una d'elles? Com s'emmagatzemen en la base de dades?

A Oracle, sota l'extensió object-relational, podem definir dos tipus de col·leccions; VARRAY's i Nested Tables (taules niuades):

VARRAYS[4]:

- Adients per als següents casos d'ús:
    - Emmagatzemar un nombre fixe d'elements.
    - Realitzar recorreguts en ordre dels elements de la col·lecció.

[4] Informació obtinguda de:

- Pàg. 29 i 36, mòdul 2 dels materials de l'assignatura

  - Obtenir i manipular la col·lecció sencera com a bloc.

- S'emmagatzemen en línia (*inline*) en un sol camp d'una filera d'una taula d'objectes, com un atribut més.

NESTED TABLES[5]:

- Adients per als següents casos d'ús:
  - Emmagatzemar un nombre arbitrari d'elements.
  - Executar consultes sobre una col·lecció de manera eficient.
  - Realitzar operacions massives (*INSERT, UPDATE, DELETE*...) sobre un subconjunt d'elements de la col·lecció
- S'emmagatzemen com una relació individual no referenciable entre la taula niuada i la taula "base".

d)

Quines son les principals característiques de les bases de dades object-relational

Les característiques de les bases de dades "object-relational" es van introduir a la versió SQL-1999 i són les següents:

- **Tipus LOB (Long Object Binary)** → Per tal de donar suport a bd's orientades a objectes.
- **Tipus ROW** → Permet que una columna contingui més d'un atribut
- **Tipus UDT (User Defined Types)** → Definits per l'usuari
- **Taules amb tipus** → Podem establir un tipus UDT a una taula de manera que cada filera es correspondrà amb una instancia d'objecte.
- **Polimorfisme** → Es poden establir relacions d'herencia entre tipus per tal de disposar d'atributs i mètodes dels "supertipus" als "subtipus".
- **Punters a objectes** → El tipus REF proporciona un mecanisme anàleg a les clàus primàries i foràanies a una DB relacional

[5] [5] Informació obtinguda de:

- Pàgs. 31 i 36, mòdul 2 dels materials de l'assignatura

- **Tipus VARRAY (col·leccions)** → Permet implementar atributs multivaluats i associacions "un a molts" definies (1..n)
- **Nested tables** → permeten implementar associacions "un a molts" indefinides (*)

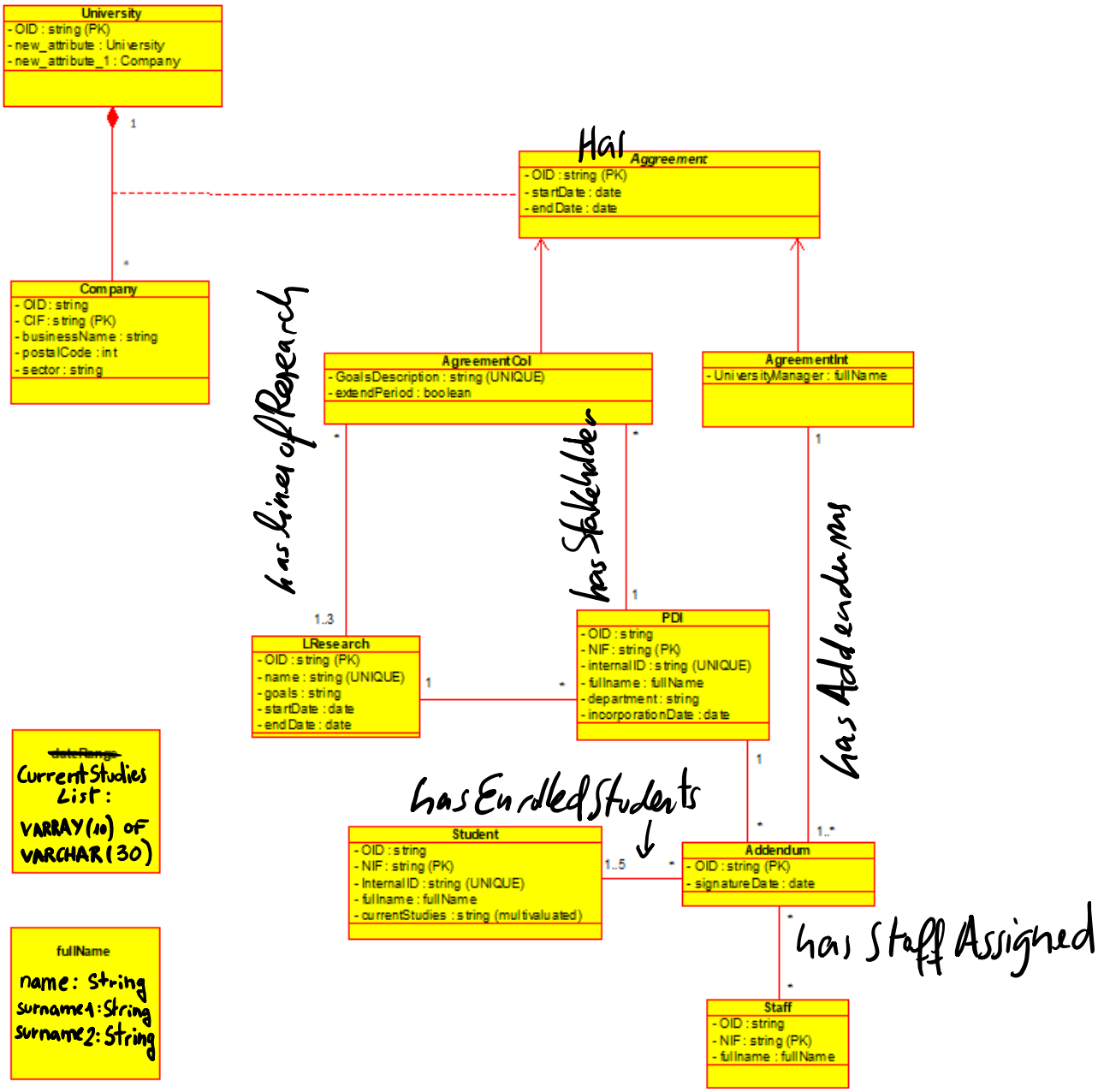
Pregunta 2

a)

L'esquema UML que il·lustri les associacions entre les classes identificades

1. Representació en UML

NOTA IMPORTANT → Veure secció 2.b.5



2. Representació en FN (forma normal)

Representem diagrama UML anterior en la seva FN per a facilitar la posterior aplicació de les funcions de traducció al model *Object-Relational* definides a la pregunta 2-b, subapartat 2. Podeu consultar el glossari de terminologia emprada a l'annex 1.

Classe "University"

```
C'=('University', name, BAS=(' University', 'Company', 1, *, 'Agreement'))
```

Classe "Company"

```
C'=('Company', CIF, businessName, postalCode, sector,
  BAS=('Company', 'University', *, 1, 'Agreement'))
```

* *En aquestes dues classes anteriors (University i Company) la representació en FN de l'associació binària (BAS) també es modifica degut a que aquesta es modela amb una classe associativa, tal i com s'especifica a l'apartat 2.3*

Classe abstracta "Agreement", i les seves especialitzacions: "AgreementCol" i "AgreementInt"

La classe *Agreement* és abstracta i a l'hora representa la relació d'associativitat entre les classes *University* i *Company*. Per tant, al fer la representació del UML en forma normal (FN) procedirem com segueix:

- En verd, la part de FN corresponent al modelat de la generalització/especialització
- En violeta, la part de FN corresponent al modelat de la classe associativa

```
GS=( SPC=('Agreement', startDate, endDate, Ref(University), Ref(Company)),
SBC1=('AgreementCol', goalsDescription, extendPeriod, BAS=('AgreementCol',
        'LResearch', *, 1..3), BAS=('AgreementCol', 'PDI', *, 1)),
SBC2=('AgreementInt', universityManager, BAS=('AgreementInt', 'Addendum', 1,
                                    1..*)))
```

Classe "Addendum"

```
C=('Addendum', signatureDate, BAS=('Addendum', 'AgreementInt', 1..*, 1),
  BAS=(' Addendum', 'PDI', *, 1), BAS=('Addendum', 'Student', *, 1..5),
                    BAS=('Addendum', 'Staff', *, *))
```

Classe "Staff"

```
C=('Staff', NIF, completeName, BAS=('Staff', 'Addendum', *, *))
```

Classe "Student"

```
C=('Student', NIF, internalID, completeName, currentStudies, BAS=('Student',
                                'Addendum', 1..5, *))
```

Classe "PDI"

```
C=('PDI', NIF, internalID, completeName, department, incorporationDate,
BAS=('PDI', 'Addendum', 1, *), BAS=('PDI', 'AgreementCol', 1, *), BAS=('PDI',
                                'LResearch', *, 1))
```

Classe "LResearch"

```
C=('LResearch', name, goals, startDate, endDate, BAS=('LResearch', 'PDI', 1,
                *), BAS=('LResearch', 'AgreementCol', 1..3, *))
```

b)

La implementació de l'esquema UML anterior en forma d'script SQL sobre Oracle DB XE 11g2, utilitzant la orientació a objectes (extensió *object-relational*). També cal lliurar les sentències d'inserció d'informació que permetin validar el correcte funcionament de la implementació.

1. Consideracions prèvies[6]

El diagrama UML modelat a l'apartat anterior és la primera de tres capes en el procés de mapeig d'objectes al paradigma objecte-relacional;

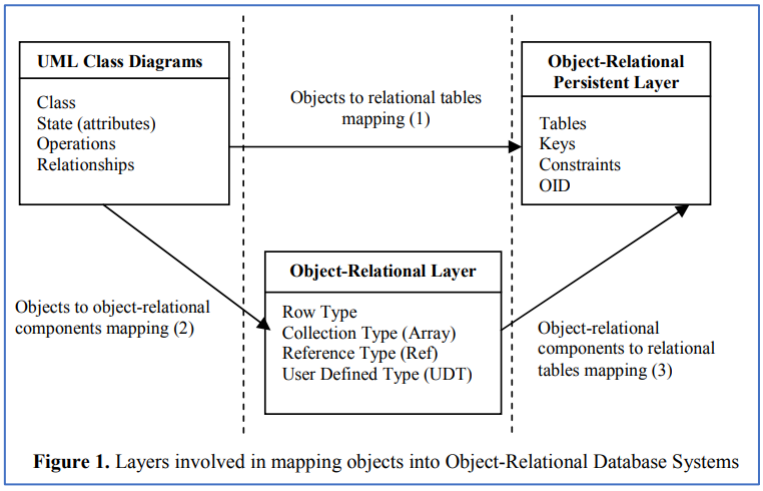


Figure 1. Layers involved in mapping objects into Object-Relational Database Systems

Per tant, el següent pas consisteix en definir el disseny lògic de la BD (segona capa o "*Object-Relational Layer*"), per a finalment poder procedir a la seva implementació al ORDBMS (tercera capa o "Object-Relational Persistent Layer").

[6] *Funcions de traducció de UML a OR i imatges obtingudes del paper: "Mapping UML Class Diagrams into Object-Relational Schemas" (María Fernanda Golobisky and Aldo Vecchietti. Departamento de Sistemas Universidad Tecnológica Nacional. Facultad Regional Santa Fe, Argentina, 2005)*

2. Funcions de transformació UML → OR

Per a realitzar les transformacions / del model UML a la capa OR (Object-Relational) utilitzarem les següents funcions de mapeig entre les capes 1ª i 2ª, on la FN en blau correspon a la capa UML i la FN en taronja correspon al disseny lògic de la capa Object-Relational (trobareu un glossari amb la terminologia emprada a l'annex 1):

2.1. Transformació de classes (C)

```
f1: C → UDT
```

On de manera implícita queden definides les següents transformacions dels components individuals d'una classe:

- Atribut simple → Tipus predefinit (built-in)
- Atribut compost → tipus ROW
- Atribut multivaluat → VARRAY, Nested Table

2.2. Transformació d'associacions binàries (BAS)

Suposant que una classe C1 participa a l'associació binària amb multiplicitat = 1, aleshores la funció de mapeig per a la classe C2 es defineix amb un camp punter (REF) a objecte de la següent manera:

```
f2: BAS=(C1,C2,1,…) → UDT=(C2,…,Ref(C1),…)
```

D'altra banda, si l'associació binària de C1 amb C2 té multiplicitat = 1…*n*, la funció de mapeig per a la classe C2 es definirà amb un camp VARRAY de referències (AT):

```
f3: BAS=(C1,C2,n,…) → UDT=(C2,…,AT(Ref(C1),n),…)
```

Finalment, si l'associació binària de C1 amb C2 té multiplicitat = *, aleshores la funció de mapeig per a la classe C2 es definirà amb un camp de tipus multiset (camp tipus taula)

```
f4: BAS=(C1,C2,*,…) → UDT=(C2,MT(Ref(C1)),…)
```

2.3. Transformació de classes associatives (AC)

Tenint en compte que una classe associativa és a l'hora una associació i una classe (i per tant, té les mateixes propietats d'aquestes darreres, en el cas que ens ocupa el polimorfisme i l'associativitat amb altres classes– Classe abstracta "*Agreement*" del model UML), aleshores podem definir la funció de transformació següent:

```
f5: AC=(Identity, Ref(C1),Ref(C2)....,Ref(Cn), SA, CA, MA, O) →
UDT=('name', Ref(C1),Ref(C2)....,Ref(Cn), BIT, RT, AT, UDT, MM)
```

On observem que:

- L'UDT definit per a la classe associativa conté les referencies a les classes relacionades, així com a d'altres components (tipus built-in, UDT's, VARRAY's, associacions de la classe associativa amb d'altres entitats, etc)

- La relació binària d'associació entre C1 i C2 (BAS) s'ha de corregir quan aquesta es representa mitjançant una classe associativa. Concretament, els UDT que representen cada classe han d'incloure una referència a la classe associativa en comptes de referenciar a la classe amb la que està relacionada, de la següent manera:

```
f2' = BAS=(C1,C2,1,…,AC) → UDT=(C2,…,Ref(AC),…)
f3' = BAS=(C1,C2,n,…,AC) → UDT=(C2,…,AT(Ref(AC),n),…)
f4' = BAS=(C1,C2,*,…,AC) → UDT=(C2,MT(Ref(AC)),…)
```

3. Disseny lògic: Object-Relational Layer (capa 2)

Seguidament podem procedir a aplicar les funcions de transformació definides al punt anterior a les representacions del model UML en FN fetes a l'apartat 2-a. Seguirem la mateixa nomenclatura i codi de colors:

- En verd → representació UML en forma normal
- En Taronja → representació *Object-Relational* en forma normal

Classe "University" → Apliquem f1 i f2

```
C'=('University', name, BAS=(' University', 'Company', 1, *, 'Agreement'))
```

```
UDT=('University', name, MT(Ref(Agreement)))
```

Classe "Company" → Apliquem f1 i f4

```
C'=('Company', CIF, businessName, postalCode, sector,
  BAS=('Company', 'University', *, 1, 'Agreement'))
```

```
UDT=('Company', CIF, businessName, postalCode, sector,
                Ref(Agreement))
```

Classe abstracta "Agreement", i les seves especialitzacions: "AgreementCol" i "AgreementInt"

```
GS=( SPC=('Agreement', startDate, endDate, Ref(University), Ref(Company)),
SBC1=('AgreementCol', goalsDescription, extendPeriod, BAS=('AgreementCol',
        'LResearch', *, 1..3), BAS=('AgreementCol', 'PDI', *, 1)),
SBC2=('AgreementInt', universityManager, BAS=('AgreementInt', 'Addendum', 1,
                                    1..*)))
```

```
UDT=('Agreement', startDate, endDate, Ref(University), Ref(Company)),
        UDT=('AgreementCol', goalsDescription, extendPeriod,
         Ref(PDI),AT(Ref(LResearch),3), UNDER 'Agreement'),
UDT=('AgreementInt', universityManager, MT(Ref(Addendum)), UNDER 'Agreement'))
```

Classe "Addendum"

```
C=('Addendum', signatureDate, BAS=(' Addendum', 'PDI', *, 1), BAS=('Addendum',
            'Student', *, 1..5), BAS=('Addendum', 'Staff', *, *))
```

```
UDT=('Addendum', signatureDate, MT(Ref(Staff)), AT(Ref(Student),5),Ref(PDI))
```

Classe "Staff"

```
C=('Staff', NIF, completeName, BAS=('Staff', 'Addendum', *, *))
```

```
UDT=('Staff', NIF, completeName, MT(Ref(Addendum)))
```

Classe "Student"

```
C=('Student', NIF, internalID, completeName, currentStudies)
```

```
UDT=('Student', NIF, internalID, completeName, currentStudies)
```

Classe "PDI"

```
C=('PDI', NIF, internalID, completeName, department, incorporationDate,
                    BAS=('PDI', 'LResearch', *, 1))
```

```
UDT=('PDI', NIF, internalID, completeName, department, incorporationDate,
                                Ref(LResearch))
```

Classe "LResearch"

```
C=('LResearch', name, goals, startDate, endDate)
```

```
UDT=('LResearch', name, goals, startDate, endDate)
```

4. Mapeig del model UML a OR en Oracle 11g (Capa 3)

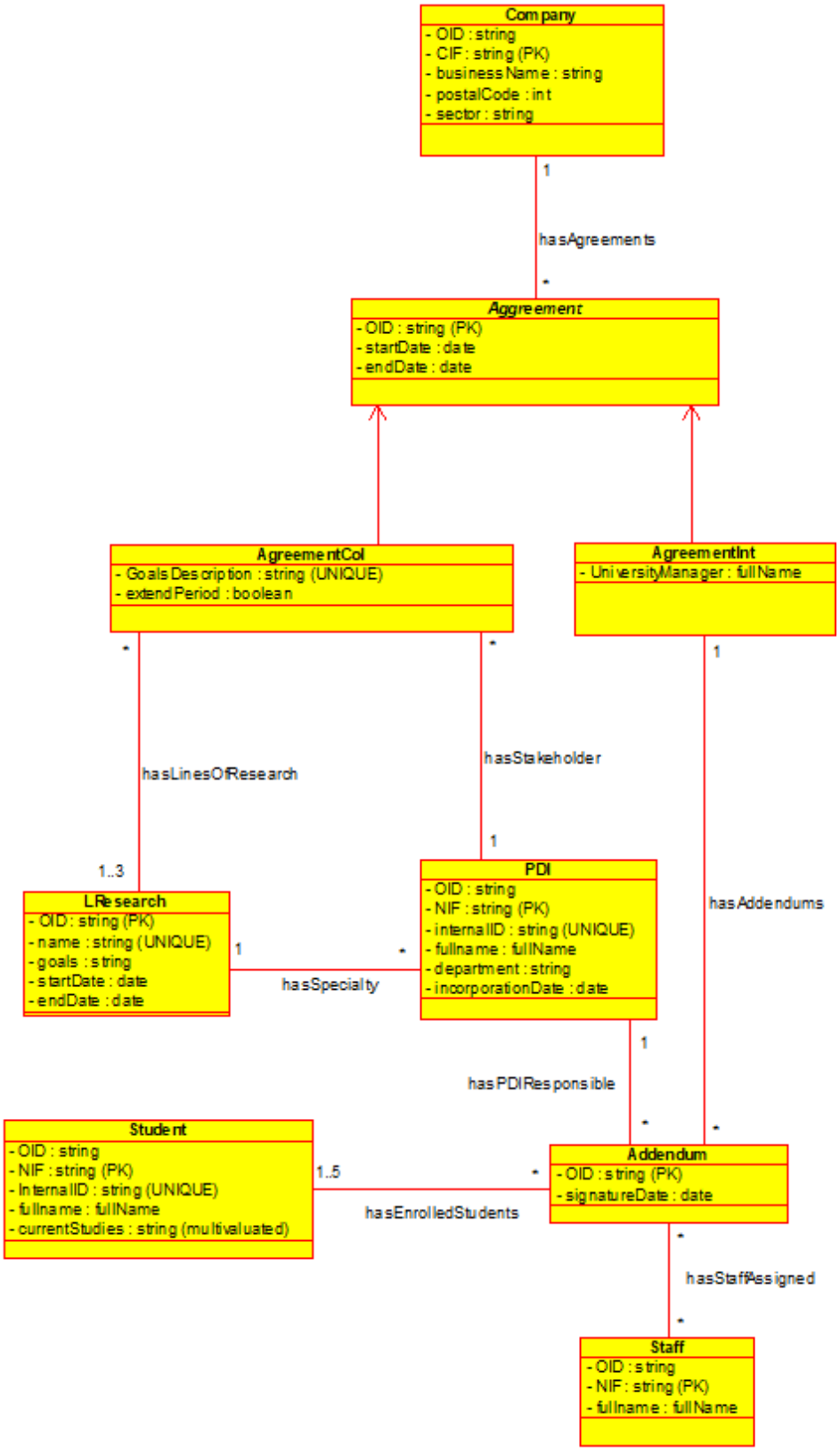
Per a realitzar la implementació al ORDBMS seguirem la següent estratègia[7]:

1. En primer lloc crearem els UDT personalitzats (tipus) que siguin estrictament necessaris per a declarar els atributs de cada UDT
2. Seguidament crearem els UDT (tipus objecte) corresponents a cada classe amb ***CREATE or REPLACE TYPE className AS OBJECT*** sense tenir en compte les associacions entre classes, és a dir, només es crearà l'esquelet de classes junt als seus atributs.
3. De la mateixa manera que al pas anterior, crearem els UDT (tipus objecte) necessaris per a poder referenciar les associacions definides entre UDT's:
    - **Associacions per rang: "1..n"** → Es definiran els tipus AT (VARRAY) com a tipus degut a que aquests no poden ser definits directament des d'un objecte
    - **Associacions múltiples: "0..*" i "1..*"** → La implementació d'associacions múltiples o *multiset* es realitza mitjançant la definició de taules niuades (NESTED TABLES). Concretament, haurem de crear un tipus taula (TABLE TYPE) per a cada associació múltiple i seguidament incloure aquest tipus a la definició d'objecte (UDT)
4. Després realitzarem la implementació de les associacions als tipus objecte UDT (classes). Concretament, haurem de modificar els UDT creats al pas 2 mitjançant "***ALTER TYPE object_type ADD ATTRIBUTE () CASCADE***" de la següent manera:
    - **Associacions úniques: "1"** → Punter (REF) a tipus UDT (classe)
    - **Associacions per rang: "1..n"** → Punter (REF) a tipus VARRAY (vector)
    - **Associacions multiple: "0..*" i "1..*"** → Punter (REF) a tipus TABLE (taula niuada)
5. Finalment, crearem les taules d'objectes (implementació persistent del disseny *Object-Relational* al ORDBMS) mitjançant sentències "***CREATE TABLE table OF object_type***".

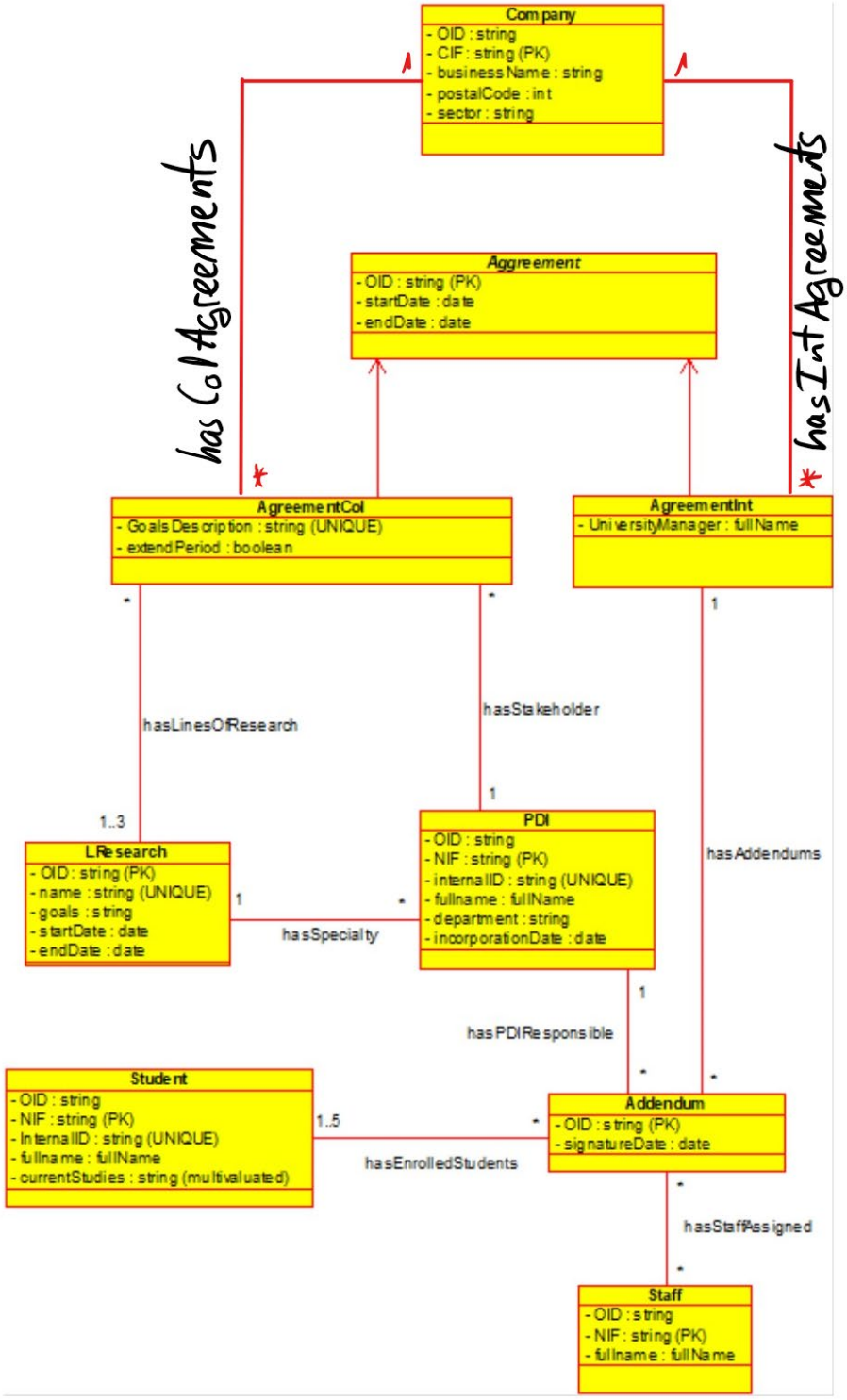
Tanmateix, abans de començar amb el procediment de traducció del disseny lògic a SQL, cal indicar que un cop he iniciat el procés d'implementació m'he trobat amb que la manera com havia modelat la primera versió del meu UML (amb la existència de la classe "*University*" i associada amb "*Company*" mitjançant una classe associativa) podria ser molt més simple, ja que la BD només existeixen dins el domini d'una universitat, és a dir, a la taula "*universities"* només tindríem una fila

[7] Fonts: pag 76, secció 8 document pdf →

amb la universitat que té convenis amb un conjunt d'universitats. El model UML obtingut rere la modificació és el següent:



Observem ara que el nou model UML implica la existència d'una taula niuada dins "*companies*" de superclasse "*Agreement*", la qual pot contenir files de qualsevol dels seus subtipus "*AgreementInt*" i "*AgreementCol*" (els atributs corresponent a aquestes després es podrien obtenir mitjançant la sentencia TREAT). D'altra banda però, tindríem que les files corresponents a la subclasse "*AgreementInt*" també haurien de tenir una columna tipus taula "*hasAgreements*" (*multilevel nested table*), fet que implica que el gestor de base de dades Oracle hauria de proporcionar algun mecanisme per a poder inicialitzar (constructor) aquestes files amb la taula niuada corresponent. Dit d'una altra manera, el que pretenia era crear una taula "*company"* que tingués un atribut que contindria una taula niuada de supertipus "*Agreements*", i per a les fileres d'aquesta que siguin de subtipus "*AgreementInt_tab*", existiria un camp taula amb una taula niuada interior (multinivell) amb les addendes relacionades. Tanmateix, després de realitzar una cerca extensiva a la xarxa, així com una consulta amb la consultora de l'assignatura, he arribat a la conclusió de què no existeix la implementació d'aquest mecanisme. Per tant, he tornat a revisar el model UML per a poder implementar l'associació entre "*companies*" i "*agreements*" mitjançant dues relacions (és a dir, dos camps taula a *companies*, un de les quals contindrà una nested table multinivell) "*hasColAgreements*" i "*hasIntAgreements*":



Finalment, tot i que a apartats anteriors s'ha descrit com traduir el disseny UML original (apartat 2.a.1) al disseny lògic (capa 2), per a després fer la implementació sobre la ORDB (capa 3), ara ens saltarem refer el disseny lògic fet a l'apartat 2.b.3 i passarem a implementar l'SQL directament, ja que les modificacions sobre el disseny lògic de capa 2 són reduïts i es poden fer directament

adaptant el codi SQL de la versió anterior. Trobareu el codi SQL al fitxer zip de la entrega, carpeta SQL:

- **Fitxer "pregunta-2B\1-database.sql"** → SQL amb la estructura de la ORBD
- **Fitxer "pregunta-2B\2-inserts.sql"** → SQL amb els inserts de prova
- **Fitxer "pregunta-2B\3-delete-rows.sql"** → SQL per a eliminar totes les files introduïdes a les taules de la ORBD
- **Fitxer "pregunta-2B\4-selects.sql"** → Queries SQL de mostra per a efectuar algunes operacions de "*unnesting*" sobre la estructura de nested table multicapa creada a la ORDB

c)

Un informe detallat explicant totes i cadascuna de les decisions d'implementació que s'hagin pres durant la implementació.

Decisions predeterminades (basades en les funcions de transformació)

Com ja s'ha comentat a l'apartat anterior, per a implementar el model UML en una ORBD s'han utilitzat tot un seguit de funcions de transformació (veure resposta 2.b.2) que fan de tot el procés una tasca bàsicament mecànica, degut a que el disseny lògic previ realitzat ja contempla les transformacions de UML al model objecte-relacional. Per tant, en primer lloc només farem una repassada a les tècniques d'implementació utilitzades, i seguidament destacarem algunes de les excepcions d'implementació a les quals no s'han utilitzat les funcions de transformació, així com el motiu pel qual s'ha pres la decisió en concret.

Relacions "hasPDIResponsible", "hasSpecialty", "hasStakeholder"

Les relacions amb multiplicitat "1" s'han implementat mitjançant REF's a tipus objecte tal i com s'ha comentat en apartats anteriors (veure resposta 2.b.2, secció 2.2).

Relacions "hasLinesOfResearch" i "hasEnrolledStudents"

Les relacions amb multiplicitat "1..n" s'han implementat mitjançant VARRAYs de punters REF's a objectes tal i com s'ha comentat en apartats anteriors).

Relacions "hasColAgreements", "hasIntAgreements" i "hasAddendums"

Les relacions amb multiplicitat un a molts "1..*" s'han implementat mitjançant NESTED TABLES tal i com s'ha comentat en apartats anteriors. A la resposta 2.b.4 ja s'han detallat amb precisió el motiu de les decisions preses i no hi ha res més a afegir en aquest sentit.

Excepcions (implementacions que no utilitzen les funcions de transformació o bé que són específiques del cas d'ús concret)

Classe "Agreement" i subclasses "AgreementInt" i "AgreementCol"

La decisió d'implementar la classe "*Agreement*" com a generalització de "*AgreementInt*" i "*AgreementCol*", a part d'estar motivada per gaudir de les propietats del polimorfisme en la orientació a objectes (herència), en primera instància també s'havia pres (resposta 2.b.2.4) per tal de poder crear una sola taula "*agreements*" que podria contenir files de qualsevol dels seus subtipus. Això permetria un ventall més ampli de possibilitats d'operar amb la taula, ja que es podrien aprofitar altres característiques OR de l'SQL, com les sentencies IS OF type, TREAT, etc. Tanmateix, com ja s'ha comentat anteriorment, la implementació final en SQL no treu partit d'aquesta funcionalitat al crear-se dues taules per separat per, una per a cada subtipus. Així doncs, la única motivació per a utilitzar una estructura de superclasse i subclasses és organitzativa.

Classe "Staff" i subclasses "Students" i "PDI"

Com en el cas anterior, aquestes classes podrien haver-se organitzat en una estructura amb una superclasse que englobés el atributs comuns, permetés l'ús d'una sola taula per emmagatzemar files de qualsevol dels seus subtipus i en definitiva mostrés una representació lògica de la BD més normalitzada. Tanmateix, per a no fer el codi SQL encara més complexe i donada la proximitat del deadline d'entrega d'aquesta pràctica, finalment he decidit implementar cada classe per separat i no fer ús del polimorfisme.

Relació "hasStaffAssigned"

Tot i que al disseny UML s'indica que la relació "*hasStaffAssigned*" entre les classes "*Agreements*" i "*Staff*" té una multiplicitat "un a molts" (*), la implementació Object-Relational amb SQL s'ha realitzat mitjançant un VARRAY(n), amb "n" essent un nombre elevat d'elements (REF's a objectes tipus "*Staff_ob*"), en comptes de, com seria natural implementar-ho, amb un NESTED TABLE. La motivació que m'ha dut a realitzar aquest canvi en la implementació ha sigut el facilitar la interacció amb la ORBD (quèries, inserts, etc), és a dir, reduir la complexitat de les sentencies de "unnesting" que implica una estructura de taules niuades multi-nivell (*multivel nested tables*) donat que tindríem una profunditat de 3 nivells de taules (*companies.hasIntAgreements_nt.hasAddendums_nt*). Tanmateix, a l'hora d'efectuar aquest canvi s'ha analitzat primer l'impacte d'aquest en el rendiment (performance) de la ORBD, traient com a conclusió que el fet d'obtenir tot (*whole collection at once*) el llistat de personal assignat a una addenda es pot efectuar amb una sola consulta amb millor rendiment respecte a les NESTED TABLES (*row-wise*) ja que les dades del VARRAY

s'emmagatzemen inline, és a dir, ocupant una sola columna per fila com a atribut (*column-wise)*[8].

Integritat referencial: Atributs amb restricció (CHECK CONSTRAINT) NOT NULL a les nested tables

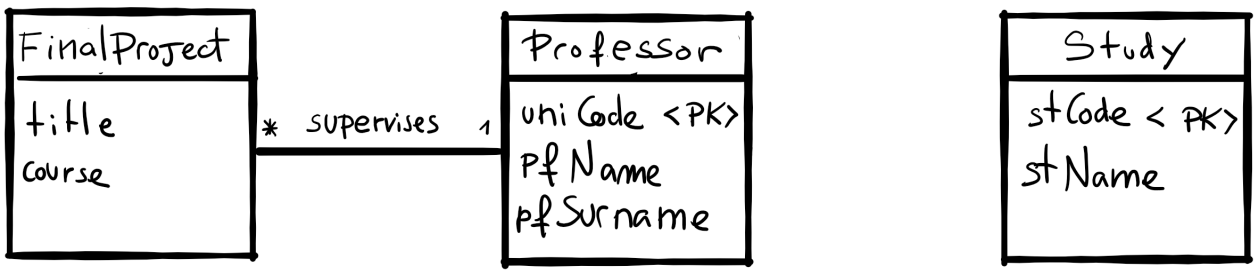
Degut a què l'ORBDMS (SQL) no proporciona cap mecanisme nadiu (*built-in*) per tal de garantir la integritat referencial de les associacions binàries entre taules, quan el camp que desa el punter REF a la "clau forana" es trobi en una nested table, ja que no tindrem manera de garantir que aquest no conté un valor nul, només utilitzant SQL[9]. Una alternativa per a poder establir aquest tipus de restriccions seria la implementació de mètodes de classe que s'encarreguin de realitzar aquesta tasca. Tanmateix, s'ha decidit no implementar cap mètode i només de què en un entorn de producció caldria implementar-los per a garantir la integritat referencial (implementació de FK's al model Object-Relational).

Pregunta 3

a)

sobre Oracle DB XE 11g2, crear una vista que proporcioni per cada un dels estudis, el nom de l'estudi, el codi, el nom i cognom de tots els professors d'aquest estudi amb el títol dels projectes on ha fet de tutor, i el curs d'aquest projecte final. Aquesta informació s'ha de retornar en una línia per cada estudi.

la estructura de l'antiga BD que ens proporcionen correspon al següent model UML:



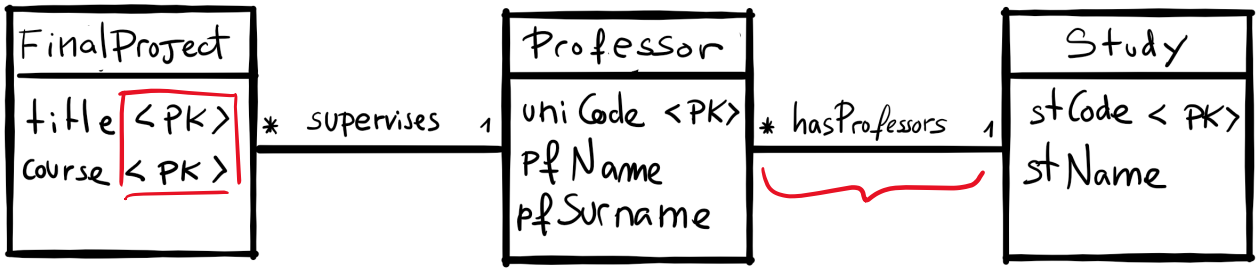
En primer lloc observem que no existeix cap mecanisme (relació) per a enregistrar els professors

[8] Pàg. 33-35, taula 3, M2

[9] Font → https://www.oracletutorial.com/oracle-basics/oracle-check-constraint/

que imparteixen un estudi en concret. Tampoc hi ha cap mecanisme (PK) per a impedir que dos treballs finals d'un mateix curs i professor tinguin el mateix títol. Per tant, abans de mapar la BD relacional al model objecte-relacional, afegirem les modificacions corresponents al següent diagrama UML on:

- Un professor només imparteix uns estudis en concret (no més), però pot haver-hi més d'un professor impartint els mateixos estudis.
- Un projecte final només pot ser supervisat per un professor (i només un)
- Només hi pot haver un projecte final amb mateix títol supervisat per un professor concret durant un curs concret (amb diferents cursos o professors sí que es pot repetir).



Seguidament implementem els canvis a la BD:

```
ALTER TABLE professors ADD (stCode VARCHAR(5) NOT NULL);
/

ALTER TABLE professors ADD CONSTRAINT studyPK FOREIGN KEY(stCode) REFERENCES
Studies (stCode);
/

ALTER TABLE FinalProjects ADD CONSTRAINT finalProjectPK PRIMARY KEY(title,
uniCode, course);
/
```

A partir d'aquest punt ja podem procedir a mapejar la BD relacional al model OR. En primer lloc crearem un tipus que ens servirà de "contenidor" per a assignar els atributs de les taules relacionals:

```
CREATE OR REPLACE TYPE FinalProjects_t AS OBJECT (
  stCode VARCHAR(5),
  stName VARCHAR(200),
  pfName VARCHAR(200),
  pfSurname VARCHAR(200),
  title VARCHAR(150),
  course VARCHAR(15)
);
/
```

Finalment procedim a crear la object-view amb la qual es mapejaran els camps de les taules relacions als seus corresponents camps del tipus objectes *"FinalProjects_t"*

```
CREATE OR REPLACE VIEW
FinalProjects_view OF FinalProjects_t WITH OBJECT OID(stCode) AS
  SELECT s.stCode, s.stName, p.pfName, p.pfSurname, f.title, f.course
  FROM studies s, professors p, finalProjects f
  WHERE f.uniCode = p.uniCode AND p.stCode = s.stCode
;
```

Podem invocar la vista de la següent manera:

```
SELECT * FROM FinalProjects_view;
```

Si inserim algunes files de prova podem comprovar que la vista mostra les dades demanades;

STCODE	STNAME	PFNAME	PFSURNAME	TITLE	COURSE
05607	Database Architecture	Maria Teresa	Bordas Garcia	CoVID19 Open ORBD Project	Tardor 2020/21
05607	Database Architecture	Maria Teresa	Bordas Garcia	Disseny de TAD java que realitza operacions sobre una ORBD	Tardor 2020/21
05577	Disseny de Sistemes Oeratius	Enric	Morancho Llena	Linux kernel driver implementation	Tardor 2020/21

Download CSV
3 rows selected.

A l'arxiu **"pregunta-3A\1-object-view.sql"** del ZIP de l'entrega d'aquesta pràctica trobareu l'script SQL amb la creació i modificació de les taules relacional, inserts de prova i creació de la vista.

b)

Es demana crear un mètode dins del tipus *projstudent* que retorni el títol del projecte més prioritari i una vista que proporcioni la informació següent:

A l'arxiu **"pregunta-3B\1-object-type-method.sql"** del ZIP de l'entrega d'aquesta pràctica trobareu l'script SQL amb la creació dels tipus, mètodes i vistes requerides a l'enunciat, així com un parell d'inserts de prova.

Un cop creada la estructura de la BD i implementats els mètodes, es pot invocar la vista de la següent manera:

```
SELECT * FROM topProjects_view;
```

Obtenint el següent resultat:

UNICODE	STFULLNAME	TOPPROJECT
111111	Freddie Mercury	TFG prioritat 1
222222	Jimmy Hendrix	TFG prioritat 1

Download CSV

2 rows selected.

Annex 1 - Glossari de termes utilitzats a les funcions de traducció UML → OR

Capa 1 (UML):

- C → Class
- GS → Generalization / Specialization
- SPC → Superclass
- SBC → Subclass
- CM → Compositon
- AG → Aggregation
- BAS → Binary Association
- NAS → N-Ary Association
- AC → Association Class
- SA → Single Attribute
- CA → Composite Attribute
- MA → Multivalued Attributed
- O → Operation

Capa 2 (Object-Relational):

- UDT → User Defined Type
- Ref → Reference (Pointer to object)
- AT → Array of references (Array Type)
- MT → Multiset of References (Multiset Type)
- BIT → Built-in Type
- RT → Row Type
- MM → Member Methods