

Database Project

First Deliverable

1 Design Choices

The overall design choice we made was to stick as close as possible to the given data while keeping useful information and reducing redundancy.

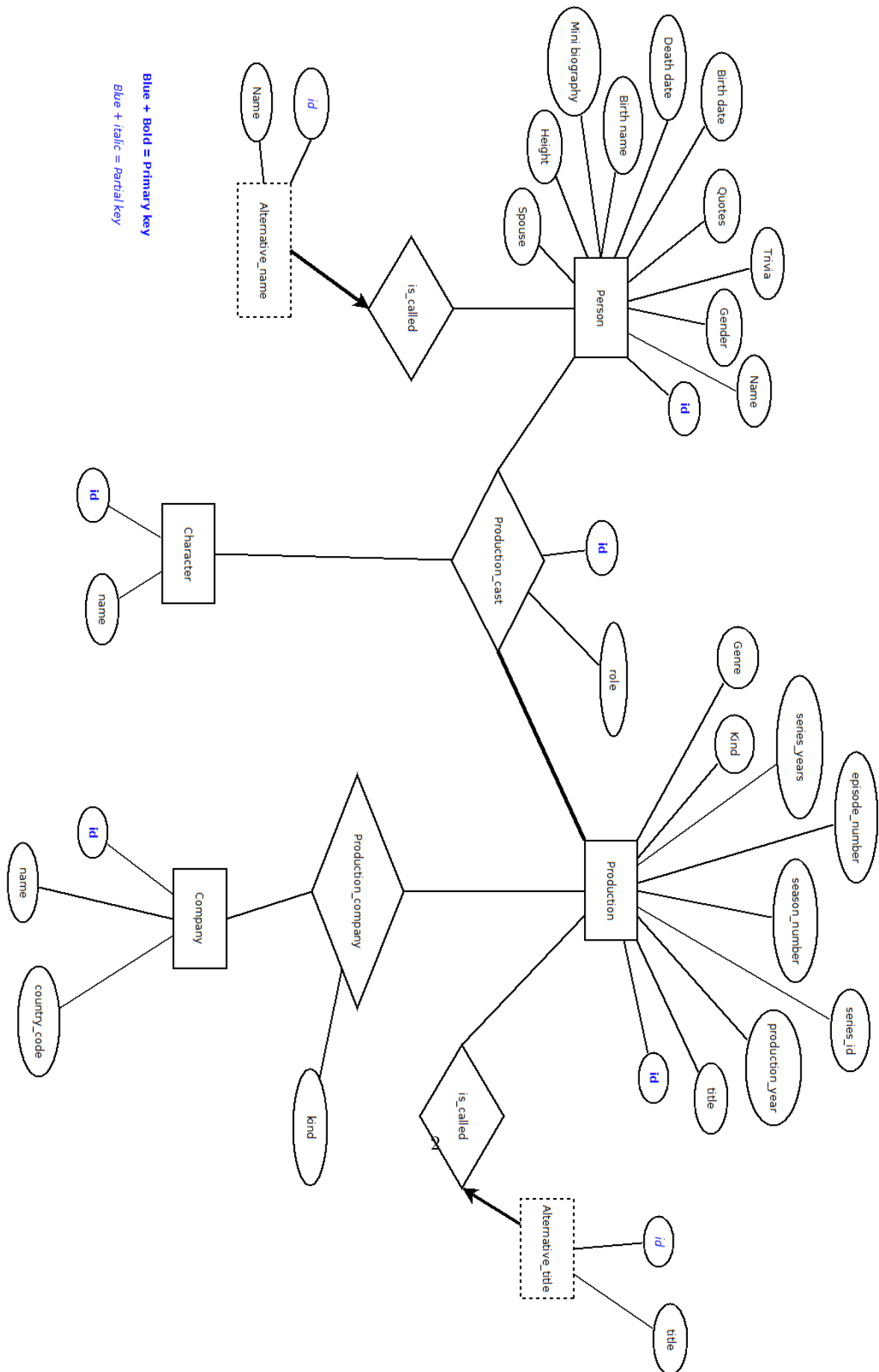
Let's start with real person in the universe, we obviously described them as an entity called Person. This entity is in a weak entity relationship with the Alternative_name entity as it makes no sense to have an alternative name that is not linked to a person. We implemented the weak entity with "ON DELETE CASCADE" such that we never have existing alternative name linked to deleted Person in the database.

We decided to have a 3 way relationship with the Person, Character and Production entities as we felt that linking these entities in a relationship was the best way to form tuples that represent something that produces movies. If a person produces a movies but isn't a character then the character_id field is set to null, this way let's us link directly person to character in the same table. We also link Person and Character to Production_cast with many to many relationship because a person can produce several movies in different production and play many different character also since a character could be played by no real person (in computer generated movies for example) the arrow between Character and Production_cast doesn't need any constraint, Finally anyone in the production cast needs at least one production therefore we connected the relationship with a bold line between Production and Production_Cast. The rest is self explanatory.

The production entity also posses a weak entity relationship with an alternative title the exact same way as Person does with Alternative_name.

And finally we have that the Company entity produces Production. This is seen as a many to many relationship with attributes because a Company can produces several production, but a production can have zero (or many) company that produces it (indie movie or collaborative movies).

2 ER Model



3 QL DDL code for table creation

```
CREATE TABLE Person
```

```
(
    id VARCHAR2(20) NOT NULL,
    Name VARCHAR2(300),
    Gender VARCHAR2(5),
    Trivia VARCHAR2(2000),
    Quotes VARCHAR2(2000),
    Birth_date DATE,
    Death_date DATE,
    Birth_name VARCHAR2(300),
    Mini_biography VARCHAR2(2000),
    Spouse VARCHAR2(300),
    Height FLOAT,

    PRIMARY KEY(id)
)
```

```
CREATE TABLE Characters
```

```
(
    id VARCHAR2(20) NOT NULL,
    name VARCHAR2(200),

    PRIMARY KEY(id)
)
```

```
CREATE TABLE Company
```

```
(
    id VARCHAR2(20) NOT NULL,
    country_code VARCHAR2(20),
    name VARCHAR2(210),

    PRIMARY KEY(id)
)
```

```
CREATE TABLE Production
```

```
(
    id VARCHAR2(20) NOT NULL,
    title VARCHAR2(370),
    production_year DATE,
    series_id CHAR(9), — Here we fix the length to 9.
—No id will be > than 999 999 999. And it's faster than VARCHAR
    season_number CHAR(5),
    episode_number CHAR(10),
    series_years_start DATE
)
```

```
        series_years_end DATE
        kind VARCHAR2(30),
        genre VARCHAR2(30),
        PROD_YEAR VARCHAR2(20)
    — When we don't want to extract the date for the year

    PRIMARY KEY(id)
)

CREATE TABLE Alternative_title
(
    id VARCHAR2(20) NOT NULL,
    title VARCHAR2(330),
    prod_id VARCHAR2(20) NOT NULL,

    PRIMARY KEY (id, prod_id),
    FOREIGN KEY (prod_id) REFERENCES Production(id),
        ON DELETE CASCADE
)

CREATE TABLE Alternative_name
(
    id VARCHAR2(20) NOT NULL,
    Name VARCHAR2(300),
    person_id VARCHAR2(20) NOT NULL,

    PRIMARY KEY (id, person_id),
    FOREIGN KEY (person_id) REFERENCES Person(id),
        ON DELETE CASCADE
)

CREATE TABLE Production_cast
(
    ID NUMBER NOT NULL,
    —auto increments and is our SURROGATE KEY for this table
    production_id VARCHAR2(20) NOT NULL,
    person_id VARCHAR2(20) NOT NULL
    character_id VARCHAR2(20),
    role VARCHAR2(20) NOT NULL,

    PRIMARY KEY (ID),
    FOREIGN KEY (production_id) REFERENCES Production,
    FOREIGN KEY (person_id) REFERENCES Person,
    FOREIGN KEY (character_id) REFERENCES Characters
)
```

```
CREATE TABLE Production_company
(
    production_id VARCHAR2(20) NOT NULL,
    company_id VARCHAR2(20) NOT NULL,
    kind VARCHAR2(50),
    ID VARCHAR2(20) NOT NULL

    PRIMARY KEY (ID),
    FOREIGN KEY (production_id) REFERENCES Production ,
    FOREIGN KEY (company_id) REFERENCES Company
)
```

SECOND DELIVERABLE

4 Parsing Data

We used several "shell" scripts in order to parse the data. The first one was used just to convert "N" into "NULL" strings. For instance :

```
#!/bin/sh
awk -F $'\t' '{
    if ($3 ~ /^\\N/)
        print $1 "\t" $2 "\t" "NULL" "\t" $4
    else
        print
}' PRODUCTION_CAST.csv > PRODUCTION_CASTcorr.csv
```

We also used scripts to put dates in correct format and to split some columns. For example, the starting and ending years of series.

Finally, we also had to parse the height of person in the PERSON table to convert all values in centimetres

5 Interface

5.1 Connection

To connect to the furnished oracle database we have apache configured in our local machines with pdo_oci enabled. We can then query the database using oracle SQL syntax in php pages. For example :

```
1 <?php //Example
2 try{
3     $dbc = new PDO('oci:dbname=diasserv2.sfl.ch:1521/orcl.dias.sfl.ch;charset=CL8MSWIN1251', '*****', '*****');
4     $req = $dbc->query('SELECT NAME FROM PERSON WHERE id=199');
5     $data = $req->fetch();
6     echo $data['NAME']; //will print the name of person with id 199
7 }
8 catch (Exception $e) {
9     die('Error : ' . $e->getMessage());
10 }
11 ?>
```

5.2 Search interface

You can search person, companies, movies and characters through our interface. The following screenshots shows how you can interact with the website to get information and data.

First you query what you're looking for through a text-field form in html

Movies DB

Search

Queries 1

Queries 2

Insert

Search

Search

Then you get information regarding the typed keyword in every category

Movies DB

Search

Queries 1

Queries 2

Insert

Search

Search

Content of the search : "Charlie Chaplin"

People:

Chaplin, Charlie S., [m]

Production:

Charlie Chaplin: A Tramp's Life, **Kind**: [episode] **Genre**: [Documentary]

Charlie Chaplin: Mennesket, klovnen og instruktøren,

Charlie Chaplin, **Kind**: [episode]

Charlie Chaplin, **Kind**: [episode]

Charlie Chaplin, un genio en blanco y negro, **Kind**: [episode]

Charlie Chaplin, **Kind**: [episode]

We Are Charlie Chaplin (Panama City, Panama), **Kind**: [episode]

Charlie Chaplin, **Kind**: [episode]

Today in History: Charlie Chaplin, **Kind**: [episode]

Young Charlie Chaplin,

Charlie Chaplin, **Kind**: [movie]

Charlie Chaplin - Les années suisses, **Kind**: [tv movie] **Genre**: [Biography]

Charlie Chaplin and Mack Sennett, **Kind**: [tv movie] **Genre**: [Short]

Charlie Chaplin Carnival, **Kind**: [movie]

Charlie Chaplin His Life & Work, **Kind**: [video movie] **Genre**: [Documentary]

Movies DB	Search	Queries 1	Queries 2	Insert
<div>Charlie Chaplin Carnival, Kind: [movie]</div> <div>Charlie Chaplin His Life & Work, Kind: [video movie] Genre: [Documentary]</div> <div>Charlie Chaplin Untold Story, Kind: [movie] Genre: [Documentary]</div> <div>Charlie Chaplin's 100 Years: Our Charlie, Kind: [tv movie] Genre: [Documentary]</div> <div>Charlie Chaplin's Body, Kind: [movie] Genre: [Short]</div> <div>Charlie Chaplin's London, Kind: [movie] Genre: [Short]</div> <div>Charlie Chaplin: The Little Tramp, Kind: [tv movie] Genre: [Documentary]</div> <div>Dreamy Dud Sees Charlie Chaplin, Kind: [movie] Genre: [Short]</div> <div>Eine Hommage an Charlie Chaplin, Kind: [video movie] Genre: [Comedy]</div> <div>Introducing Charlie Chaplin, Kind: [movie] Genre: [Comedy]</div> <div>I'm Charlie Chaplin, Kind: [movie] Genre: [Short]</div> <div>Miss Minerva Courtney in Her Impersonation of Charlie Chaplin, Kind: [movie] Genre: [Comedy]</div> <div>The Charlie Chaplin Festival, Kind: [movie] Genre: [Comedy]</div> <div>The Moon Shines Bright on Charlie Chaplin, Kind: [tv movie] Genre: [Drama]</div> <div>99: [wearing a Charlie Chaplin outfit] Thanks., Kind: [movie]</div> <div>Charlie Chaplin Cavalcade, Kind: [movie]</div> <div>Tommy Steele in Search of Charlie Chaplin, Kind: [tv movie] Genre: [Comedy]</div>				
Companies:				
Characters:				
<div>Charlie Chaplin</div> <div>Charlie Chaplin</div>				

Then if youre interested in, let's say, a character named Charlie Chaplin you can click on the hyperlink with the name of the character youre interested in and youll get to a new page that gives all the known information related to that character.

Movies DB	Search	Queries 1	Queries 2	Insert
-----------	--------	-----------	-----------	--------

Charlie Chaplin :

Interpreted by [Allin](#), [Jason](#) in [Chaplin: Home, Sweet Home](#)

You might then be interested in the Chaplin: Home, Sweet Home movie so if you click on the hyperlink youll get to a new page that will give all the known information related to that movie.

Movies DB	Search	Queries 1	Queries 2	Insert
-----------	--------	-----------	-----------	--------

Chaplin: Home, Sweet Home - 2014:

Production Year: 01/04/2014
Genre : Short
Kind : movie

Main Cast:

Allin, Jason:

Character named: Charlie Chaplin
Role: actor

Allin, Jason:

Role: producer

Allin, Jason:

Role: writer

Allin, Jason:

Role: cinematographer

Seems like this was a lot of work for Allin, Jason!

Our interface works the same way for companies and production relationship, for example:

Movies DB	Search	Queries 1	Queries 2	Insert
-----------	--------	-----------	-----------	--------

Warner Bros - Turkey:

Mazlum Kuzey

Affiliation : distributors

Polis Akademisi: Alaturka

Affiliation : distributors

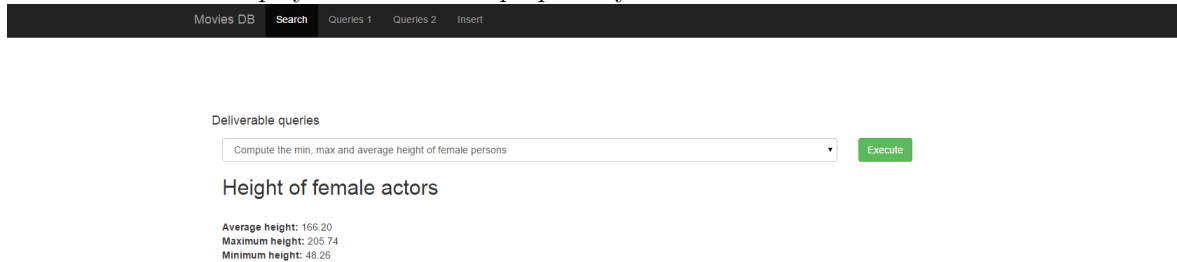
The way it works is we query the databases name fields in the Person, Production, Characters and Company table to get the the name and IDs of what the user is interested in (Note that since name format is not always perfect in our dataset, we try to use regex expression to get as good and precise data as possible but it can always be improved).

Then we have several php pages that given an id gets more information about either a character or a company and everything is linked through hyperlinks in case the user might want to get more

information about related content.

5.3 Predefined queries

For the predefined queries we have a select type of html form that lets you choose which query you want to run and then displays the result in a proper way.



The screenshot shows a web application interface with a dark header bar containing navigation links: "Movies DB", "Search", "Queries 1", "Queries 2", and "Insert". Below the header, there is a section titled "Deliverable queries". It features a dropdown menu with the text "Compute the min, max and average height of female persons" and a green "Execute" button. Below the dropdown, the title "Height of female actors" is displayed. Underneath this title, the results are shown: "Average height: 166.20", "Maximum height: 205.74", and "Minimum height: 48.26".

(Note that our menu still needs refining...)

Since the queries are already precomputed, there is nothing much to say about it. Please refer to the part about the actual SQL query for more information!

5.4 Insertion and deletion

For the insertion of data we use html forms again to add entity instances into the database

Movies DB
Search
Queries 1
Queries 2
Insert

Add a new Character

Name

Person id

Add a new Company

Name

Country

Add a new Person

Name

Gender

Birth Date

Birth name

Trivia

Quotes

Spouse

Biography

To actually insert relationship we have another set of forms below: (note that IDs need to be known in order to insert data for relations and for characters)

Movies DB
Search
Queries 1
Queries 2
Insert

Spouse

Biography

Add a new Production

Title

Type

Add a new Cast member

PersonID

ProductionID

CharacterID

Role

Add a new Production Company affiliation

CompanyID

ProductionID

Affiliation

To delete and remove entities you can either provide the name or the id to a form according to what kind of entity you want to remove. For relations you need to provide id of both entities related. (We could easily improve this part of the project if we spent more time in the web programming aspect of it but this isnt really our interest in this class so this is more a proof of concept since the query to delete data are easy to implement).

Movies DB

Search

Queries 1

Queries 2

Insert

Remove a person

ID

Name

Remove

Remove a Production

ID

Title

Remove

Remove a company

ID

Name

Remove

Remove a production-company affiliation

Production ID

Company ID

Remove

Since everything is linked with delete on cascade and foreign key references. Deleting a person, for example, in the Person table will automatically trigger deletion of any character she played and any production membership she was a part of. An SQL example of our code with an id reference would be: `DELETE FROM PERSONS WHERE ID=$personID` where \$personID is gotten through a form \$_POST variable in php.

6 Queries of deliverable 2

Here is the first set of queries in sql that a user can run in the interface :

```
/*a) Compute the number of movies per year.
Make sure to include tv and video movies.*/
```

```
SELECT prod_year as yearOfProd, COUNT(*)
FROM test_Production
WHERE prod_year != 0
GROUP BY prod_year
ORDER BY prod_year
```

```
/*b) Compute the ten countries with most production companies.*/
SELECT country_code, COUNT(*) as nombreProd
FROM test_Production-company pComp, test_Company comp
WHERE pComp.company_id = comp.id AND country_code is not null
GROUP BY country_code
ORDER BY nombreProd DESC
OFFSET 0 ROWS FETCH NEXT 10 ROWS ONLY
```

—c) Compute the min, max and average career duration. (A career length is implied by
 SELECT MIN(duree) as mini, MAX(duree) as maxi, ROUND(AVG(duree)) as moyenne
 FROM (SELECT MAX(prod_year)-MIN(prod_year)+1 duree
 FROM test_person p, test_prod_cast prodCast, test_production prod
 WHERE prod_year is not null and p.id=prodCast.person_id and prod.id=pr
 GROUP BY p.id) tableDuree

—d) Compute the min, max and average number of actors in a production.
 SELECT MIN(people) as mini, MAX(people) as maxi, ROUND(AVG(people)) as moyenne
 FROM (
 SELECT COUNT(*) as people
 FROM test_person p, test_prod_cast prodCast, test_production prod
 WHERE p.id=prodCast.person_id
 AND prod.id=prodcast.production_id
 AND (prodCast.role LIKE 'actor' OR prodCast.role LIKE 'actress')
 GROUP BY prod.id
) peopleInProd

—e) Compute the min, max and average height of female persons.

```
SELECT MIN(height), MAX(height), AVG(height)
from person
WHERE gender LIKE 'f'
```

—f) List all pairs of persons and movies where the
 —person has both directed the movie and acted in the movie.
 —Do not include tv and video movies.

```
Select name,title
FROM test_production prod, test_person p,
    (SELECT pc1.production_id, pc1.person_id
     FROM test_prod_cast pc1, test_prod_cast pc2
     WHERE pc1.production_id=pc2.production_id
           AND pc1.person_id = pc2.person_id
           AND pc1.role LIKE 'actor'
           AND pc2.role LIKE 'director'
    ) dbles
WHERE prod.id = dbles.production_id
AND p.id = dbles.person_id
```

/*g) List the three most popular character names.*/
 SELECT COUNT(*) as appearances, c.name
 FROM test_prod_cast prodCast
 INNER JOIN characters c
 ON prodCast.character_id=c.id
 GROUP BY character_id, c.name
 ORDER BY appearances DESC
 OFFSET 0 ROWS FETCH NEXT 3 ROWS ONLY

SECOND DELIVERABLE

7 Queries of deliverable 3

The running time of each query on OracleDB can be found in the sequel :

—a) 17.3 minutes

```
With innerQ as (
    SELECT production_id, name, anni
    FROM (
        SELECT production_id,
              name,
              EXTRACT(YEAR FROM birth_date) as anni,
              ROW_Number() Over
        (PARTITION BY production_id order by birth_date) as rownb,
              count(*) over(PARTITION BY production_id) CNT
        FROM production prod, prod_cast prodCast, person p
        WHERE prod.id=prodCast.production_id
              AND p.id = prodCast.person_id
              AND birth_date is not null
    ) tmp
    WHERE (ROWNB = 1 OR ROWNB = CNT))
SELECT p.title, p.id, b.name as Who, b.anni, a.anni-b.anni as AgeDiff
FROM innerQ a, innerQ b, production p
WHERE a.production_id = b.production_id
      AND a.name != b.name
      AND a.anni-b.anni >= 55
      AND a.production_id = p.id
```

—b) 11.9 secondes

```
SELECT *
FROM
(
    SELECT prod_year, count(*) as productivity
    FROM production prod, prod_cast pc, person p
    WHERE prod.id=pc.production_id AND pc.person_id=p.id AND p.id = 3429434
    — id of person is given in request
    AND prod_year <> 0
    GROUP BY prod_year
    ORDER BY productivity DESC
) tmp
WHERE ROWNUM = 1
```

—c) 3 secondes

```
SELECT COUNT(*) as nbreProd, c.name, p.genre
```

```
FROM production p, production_company pComp, company c
WHERE p.id = pComp.production_id
      AND p.prod_year = 2015
      AND pComp.company_id = c.id
GROUP BY c.name, p.genre
ORDER BY p.genre, nbreProd DESC
```

—d) 22.8 minutes

```
SELECT pid, p1.name, plid, p2.name, p2id
FROM person p1, person p2, (
  SELECT pc1.production_id as pid, pc1.person_id as plid,
         pc2.person_id as p2id
  FROM prod_cast pc1, prod_cast pc2
  WHERE pc1.person_id < pc2.person_id
        AND pc1.production_id = pc2.production_id
) tmp
WHERE plid = p1.id AND p2id = p2.id AND p1.name = p2.name
```

—e) 60 secondes

```
SELECT prod_year, avg(cnt)
FROM (
  SELECT production_id, prod_year, count(*) as cnt
  FROM prod_cast prodCast, production prod
  WHERE prodCast.production_id = prod.id and prod_year <> 0
        AND (role LIKE 'actor' or role like 'actress')
  GROUP BY prod_year, production_id
  ORDER BY prod_year
) tmp
GROUP BY prod_year
ORDER BY prod_year
```

—f) 1.5 secondes

```
SELECT ROUND(AVG(nbre)) as NumberOfEpisodes, season_number
FROM (
  SELECT COUNT(*) as nbre, series_id,
         season_number as season_number
  FROM production
  WHERE season_number is not null
  GROUP BY series_id, season_number) tmp
GROUP BY season_number
ORDER BY to_number(season_number)
```

—g) 1.22 secondes

```
SELECT AVG(nbreSeason)
FROM
  (SELECT COUNT(*) as nbreSeason, series_id
  FROM(
    SELECT COUNT(*), series_id, season_number
```

```
FROM production
WHERE kind LIKE 'episode'
GROUP BY series_id, season_number ) as tmp
GROUP BY series_id ) as tmp2
```

—h) 1.06 secondes

```
SELECT COUNT(*) as nbreSeason, series_id
FROM(
    SELECT series_id, season_number
    FROM production
    WHERE kind LIKE 'episode'
    GROUP BY series_id, season_number ) tmp
GROUP BY series_id
ORDER BY nbreSeason DESC
OFFSET 0 ROWS FETCH NEXT 10 ROWS ONLY
```

—i) 51 secondes

```
SELECT series_id, avg(cnt) as episodesPerSeason
FROM (
    SELECT *
    FROM (
        SELECT series_id, COUNT(*) OVER
            (PARTITION BY series_id, season_number ORDER BY series_id) as cnt
        FROM production
        WHERE kind LIKE 'episode'
        ) tmp
    GROUP BY series_id, CNT
    ORDER BY series_id
    ) tmp2
GROUP BY series_ID
ORDER BY episodesPerSeason DESC
```

—j) 44.7 secondes

```
SELECT p.name, prod.title, prod_year as ProductionYear,
    EXTRACT(YEAR from death_date) as deathYear
FROM person p, prod_cast prodCast, production prod
WHERE p.id = prodCast.person_id
AND prodCast.production_id = prod.id
AND (prodCast.role LIKE 'actor' or prodCast.role LIKE 'actress' or
prodCast.role LIKE 'director')
AND EXTRACT(YEAR from death_date) < prod_year
GROUP BY p.name, prod.title, prod_year, EXTRACT(YEAR from death_date)
```

—k) 8.2 secondes


```

SELECT prod_year, rownb as ranking, name
FROM company,
(
  SELECT company_id, prod_year,
  ROWNUMBER() over
  (PARTITION by prod_year order by count(*) DESC ) as rownb
  FROM production_company prodComp, production p
  WHERE p.id = prodComp.production_id
  AND prod_year != 0
  GROUP BY company_id, prod_year
) tmp
where rownb <= 3
AND company.id = company_id
ORDER BY prod_year

```

—l) 21.9 secondes

```

SELECT *
FROM PERSON
WHERE REGEXP_LIKE(TRIVIA, 'opera singer') OR
      REGEXP_LIKE(MINIBIOGRAPHY, 'opera singer')
ORDER BY EXTRACT(YEAR from BIRTHDATE);

```

—n) 18.6 minutes

```

WITH innerQ as (
  SELECT character_id, company_id, cnt, country_code
  FROM company c, (
    SELECT character_id, company_id, cnt,
    row_number() over
    (PARTITION BY company_id order by cnt desc) as rang
  FROM(
    SELECT character_id, company_id, count(*) as cnt
    FROM (
      SELECT DISTINCT pCast.production_id, character_id, company_id
      FROM prod_cast pCast, production_company pComp
      WHERE pCast.production_id = pComp.production_id
      AND character_id is not null
      ORDER BY company_id, production_id
    ) tmp
    GROUP BY company_id, character_id
    ORDER BY company_id, CNT DESC
  ) tmp2
  ) tmp3
  WHERE RANG = 1 AND c.id = company_id
)
SELECT character_id, company_id, country_code
FROM (
  SELECT character_id, company_id, country_code, cnt, row_number() OVER
  (partition by country_code order by cnt DESC) as rang

```

```

FROM innerQ
) tmp
WHERE rang = 1

```

8 Query Optimisation

Query e)

Analysis : Here we do a JOIN between the PROD_CAST and the PRODUCTION table over the production_id and we add a filter on the role such that we only get rows with actors and actress. We then group these rows by year of production and the id and we compute the average of the grouped number of rows per group of production_id and prod_year.

Analysing the execution plan of the query:

Oracle SQL Developer: ProjectDB

Feuille de calcul : Query Builder

```

SELECT prod_year, avg(cnt)
FROM (
  SELECT production_id, prod_year, count(*) as cnt
  FROM prod_cast prodCast, production prod
  WHERE prodCast.production_id = prod.id and prod_year <> 0
  AND (role LIKE 'actor' or role like 'actress')
  GROUP BY prod_year, production_id
  ORDER BY prod_year
) tmp
GROUP BY prod_year
ORDER BY prod_year

```

0.108 secondes

OPERATION	OBJECT_NAME	CARDINALITY	COST
SELECT STATEMENT		138	446357
SORT (ORDER BY)		138	446357
HASH (GROUP BY)		138	446357
VIEW		23672028	311603
HASH JOIN		23672028	311603
Access Predicates		23672028	109802
PRODCAST-PRODUCT			
TABLE ACCESS (FULL)	PRODUCTION	2915061	7512
TO_NUMBERS/PROD			
Filter Predicates			
TABLE ACCESS (FULL)	PROD_CAST	23686470	65127
Filter Predicates			
OR			
ROLE='actor'			
ROLE='actress'			

Messages - Journal

We can see that we could easily index the table accesses to prod_year and role to get better cost. The runtime without any index is about 55 seconds.

We first added an ascending index based on prod_year and production_id since that is the two selected fields on the second SELECT and we got the following execution plan :

The screenshot shows the Oracle SQL Developer interface. The main window displays a query in the 'Feuille de calcul' (Worksheet) tab:

```

SELECT prod_year, avg(cnt)
FROM (
  SELECT production_id, prod_year, count(*) as cnt
  FROM prod_cast, production prod
  WHERE prod_cast.production_id = prod.id and prod_year <> 0
  AND (role LIKE 'actor' or role like 'actress')
  GROUP BY prod_year, production_id
  ORDER BY prod_year
) cnt
GROUP BY prod_year
ORDER BY prod_year

```

The 'Plan d'exécution' (Execution Plan) window shows the following details:

OPERATION	OBJECT_NAME	CARDINALITY	COST	441692
SELECT STATEMENT		138		441692
SORT (ORDER BY)		138		441692
HASH (GROUP BY)		138		441692
VIEW		23672028		306938
HASH (GROUP BY)		23672028		306938
HASH JOIN		23672028		105137
Access Predicates				
INDEX (FAST FULL SCAN)	INDEX2	2915061		2846
Filter Predicates				
TO NUMBER PROC				
TABLE ACCESS (FULL)	PROD_CAST	23686470		65127
Filter Predicates				
OR				
ROLE='actor'				
ROLE='actress'				

Which shows a clear improve in IO cost of the access to prod_year (cost is 7512 without the index and 2846 with it)

We then added an ascending index on the prod_cast table based on the role only to optimise the second (and way more costly) access but we had trouble with it. It seemed to work and to not do a FULL ACCESS in a small subset of the prod.cast table when we added this index in our test table but it doesn't seem to work in with the big table. There is probably a way to make it to work but we couldn't figure it out...

We didn't try to optimize the hash join with an hash index or something as we are not too familiar with it.

Query i)

Analysis: We first create a partition of PRODUCTION that contains all the production with Kind as episodes and then we group these partition by series.id and then we count the avg of how many of these we get to obtain the result. (We interpreted the query this way)

The execution of the query gives a runtime of 71 seconds. Here is the execution plan:

The screenshot shows the Oracle SQL Developer interface. The left pane displays the database schema with tables like ALTERNATIVE_NAME, ALTERNATIVE_TITLE, CHARACTERS, COMPANY, PERSON, PROD_CAST, PRODUCTION, PRODUCTION_COMPANY, TABLE1, TEST_CHARACTERS, TEST_COMPANY, TEST_PERSON, TEST_PROD_CAST, TEST_PRODUCTION, and TEST_SERIES_ID. The main pane shows a SQL query in the Query Builder:

```
SELECT series_id, avg(cnt) as episodesPerSeason
FROM (
  SELECT *
  FROM (
    SELECT series_id, COUNT(*) OVER (PARTITION BY series_id, season_number ORDER BY series_id) as cnt
    FROM production
    WHERE kind LIKE 'episode'
  ) tmp1
  GROUP BY series_id, CNT
  ORDER BY series_id
) tmp2
GROUP BY series_id
ORDER BY episodesPerSeason DESC
```

The bottom pane shows the execution plan for the query. The plan includes a SORT (ORDER BY) operation, followed by a HASH (GROUP BY) operation, and a WINDOW (SORT) operation. The execution plan also shows the cardinality and cost for each operation.

OPERATION	OBJECT_NAME	CARDINALITY	COST
SELECT STATEMENT		66392	23623
SORT (ORDER BY)		66392	23623
HASH (GROUP BY)		66392	23623
VIEW	WM_JNWWW_D	66392	23231
HASH (GROUP BY)		66392	23231
VIEW		1991786	18832
WINDOW (SORT)		1991786	18832
TABLE ACCESS (FULL)	PRODUCTION	1991786	7499

The Messages - Journal pane at the bottom shows two messages: "Tâche annulée :" and "Tâche annulée :".

We can try to optimize the table access again for the kind by making a kind based index on the PRODUCTION table since this is what the execution table (kinda) tells us to do.

We tried adding this index but it didnt affect the query in any way, we are definitely missing something about how to build these indexes... The information given by SQL developer somewhat arent precise enough to understand what happens

Query b)

Analysis: For a given person id our query joins the person and the production_cast and the production table where the id matches and we group the rows by prod_year which means that we count all the production in the same year, next we order these row by their count and then we simply take the first line which is the line with the most production for a given year.

The query takes around 12 seconds to run without any index, and here is the execution plan given by SQL Developer:

The screenshot shows the Oracle SQL Developer interface. On the left, the 'Tables (Filter)' pane lists various tables including ALTERNATIVE_NAME, ALTERNATIVE_TITLE, CHARACTERS, COMPANY, PERSON, PRODUCTION, PRODO_CAST, and PRODUCTION_COMPANY. The main window displays a SQL query in the 'Feuille de calcul' (Query Builder) pane:

```

SELECT
  FROM
  (
    SELECT prod_year, count(*) as productivity
    FROM production prod, prod_cast pc, person p
    WHERE prod_id=pc.production_id AND pc.person_id=p.id AND p.id = 3429434 -- id of person is given in request
    AND prod_year <> 0
    GROUP BY prod_year
    ORDER BY productivity DESC
  )

```

Below the query, the 'Plan d'exécution' (Execution Plan) is shown. It details the operations performed by the database engine, including SELECT STATEMENT, COUNT (STOPKEY), FILTER Predicates, VIEW, SORT (ORDER BY STOPKEY), HASH (GROUP BY), NESTED LOOPS, and HASH JOIN. The plan also includes a table of statistics:

OPERATION	OBJECT_NAME	CARDINALITY	COST
SELECT STATEMENT		1	70561
COUNT (STOPKEY)			
FILTER Predicates			
VIEW		9	70561
SORT (ORDER BY STOPKEY)		9	70561
FILTER Predicates			
HASH (GROUP BY)		9	70561
NESTED LOOPS		9	70559
HASH JOIN		9	70541
Access Predicates			
INDEX (FAST FULL SCAN)	PERSON_PK	1	5440
FILTER Predicates			
TO_NUMBER(P.ID)=3429434			
TABLE ACCESS (FULL)	PRODO_CAST	44046417	64991
INDEX (UNIQUE SCAN)	PRODUCTION_PK	1	1
Access Predicates			
PRODO.ID=PC.PRODUCTION_ID			
TABLE ACCESS (BY INDEX ROWID)	PRODUCTION	1	2
FILTER Predicates			
TO_NUMBER(PROD_YEAR)<>0			

We can see that the default indexes build where used to optimize the query! But we dont really see anything that could be optimized with our current understanding of indexing techniques...