DML (Data Manipulation Language)

Queries (easy ones) using SQL

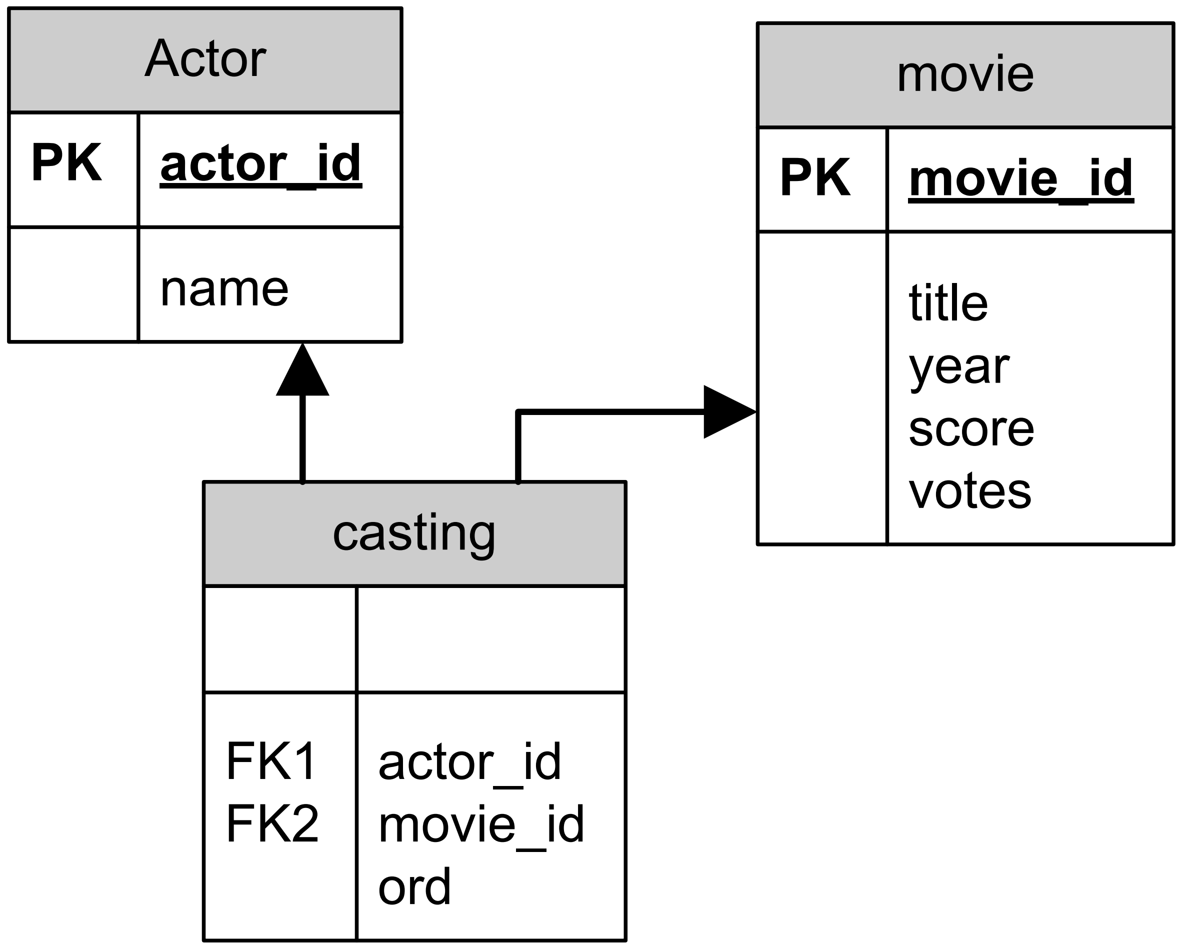
DDL/DML/DCL

- **DDL stands from Data Definition Language:**
  - CREATE - to create objects in the database
  - ALTER - alters the structure of the database
  - DROP - delete objects from the database
  - GRANT - gives user's access privileges to database
  - REVOKE - withdraw access privileges given with the GRANT command
- **DML stands from Data Manipulation Language statements. Some examples:**
  - SELECT - retrieve data from the a database
  - INSERT - insert data into a table
  - UPDATE - updates existing data within a table
  - DELETE - deletes all records from a table, the space for the records remain
  - EXPLAIN PLAN - explain access path to data
  - LOCK TABLE - control concurrency
- **DCL stands from Data Control Language statements. Some examples:**
  - COMMIT - save work done
  - SAVEPOINT - identify a point in a transaction to which you can later roll back
  - ROLLBACK - restore database to original since the last COMMIT
  - SET TRANSACTION - Change transaction options like what rollback segment to use

Movie Database

- Create with data from “The internet Movie- Database” http://www.imdb.com/list
- Made with data prior to 1997
- Only movies with 200 votes (or more)
- Only actors with 2 (or more) movies

Diagram copy and E-R diag (is arelation)



TABLES

```
CREATE TABLE movie(
   movie_ID INTEGER,     -- primary key
   title CHAR(70),       -- movie title
   year DECIMAL(4),      -- Año de estreno (not date!!)
   score FLOAT,          -- average score
   vote INTEGER,         -- Number of votes
   PRIMARY KEY (movie_ID));
--
CREATE TABLE actor (
   actor_ID INTEGER,   -- primary key
   name CHAR(35),      -- actor's name
   PRIMARY KEY (actor_ID));
--
CREATE TABLE casting(
    movie_ID INTEGER, -- reference to movie pk
    actor_ID INTEGER,    -- reference to actor pk
   ORD INTEGER,          -- order
                         -- Star is 1, second in importance 2…
--
   FOREIGN KEY (movie_ID ) REFERENCES MOVIE(movie_ID ),
   FOREIGN KEY (actor_ID ) REFERENCES ACTOR(actor_ID ),
   PRIMARY KEY (movie_ID , actor_ID ));
```

Basic Queries in SQL

```
SELECT
FROM
WHERE
ORDER BY
```

- Show the attributes `(SELECT)` belonging to one or more relation `(FROM)` that satisfied the condition `(WHERE)`, sorted by `(ORDEN BY).`
- The only difference between relation algebra and SQL (in this example) is that SQL does not take care of duplication

Select and Project

<u>Select:</u>

- Find Best movies (score greater than 9 9.0)

```
SELECT *
FROM movie
WHERE score > 9.0 ;
```

<u>Project:</u>

- Find Best movies (score greater than 9 9.0) and print only title and year in which had its premiere

```
SELECT title, year
FROM movie
WHERE score > 9.0 ;
```

Duplicates

- DISTINCT
  - expensive

```
SELECT DISTINCT year
FROM movie
WHERE score > 9.0;
```

Constraining the value of attributes

- “typical” algebraic expressions ‘< ’, ‘>’, ‘= ’, …
- “`like`” allow the use of wildcards (for string)
  - two “wildcards”: ‘_’ (.) and ‘%’ (*)
- Movies (are scores) which start by “Star”

```
SELECT title, score
FROM movie
WHERE title LIKE 'Star%';
```

- Movies with ‘s in the title

```
SELECT title
FROM movie
WHERE title LIKE '%''s%';
```

[NOT] SIMILAR TO

- Similar to LIKE but can handle regular expressions
  - | two possibilities
  - * several (may be zero) repetitions of the previous
  - + several (at least one) repetitions of the previous
  - () group several character to form a group.
  - [...] specify class.

Example: SIMILAR change % by *

- Find movies (and score) starting by ‘Star’ but those of the “Star Treck” saga

```
SELECT title, score
 FROM movie
 WHERE title NOT SIMILAR TO
           '%(S|s)tar [A-z]rek%' AND
           title SIMILAR TO '%Star%';
```

Sort output

- Relation are NEVER sorted
- ORDER BY sorts the screen output :
- Find best movies (score greater than 9.0) and print the result sorted by score.

```
SELECT title, score
FROM movie
WHERE score > 9.0
ORDER BY score;
```

Ordenar la salida

- I want the highest score first:

```
SELECT title, score
FROM movie
WHERE score > 9.0
ORDER BY score DESC;
```

- Worst movie: LIMIT

SORTING

```
SELECT select_list
    FROM table_expression
    WHERE
    ORDER BY column1 [ASC | DESC] [, column2 [ASC | DESC] ...];
```

Rename attributes

- AS is used to rename attributes in the SELECT statement

```
 SELECT title AS titulo, year AS agno, score AS
   puntuacion
 FROM movie
WHERE puntuacion > 9.0;
-- Some Databases will not run the above command
 SELECT title AS titulo, year AS agno, score AS
   puntuacion
 FROM movie
WHERE score > 9.0;
```

Cartesian Product natural,title

- Usually interesting queries involve two or more relation
- Cartesian product is denoted by a , in the FROM field
- 'Pulp Fiction' casting (movie_id=2) use title

```
    SELECT name
    FROM actor, casting
    WHERE movie_id=2 AND
actor.actor_id=casting.actor_id;
```

Natural Join

'Pulp Fiction' casting (movie_id=2)

```
SELECT name
      FROM actor NATURAL INNER JOIN
casting
      WHERE movie_id=2;
```

Cartesian Product

- John Travolta ‘s movies ordered by popularity

```
SELECT title, score
FROM casting natural join movie natural
  join actor
WHERE actor.name='John Travolta'
ORDER BY score desc;
```

Attribute ambiguity

- **Use relation name**
- John Travolta’s movies

```
SELECT title, score
FROM casting,movie,actor
WHERE actor.name='John Travolta'
    AND actor.actor_id=casting.actor_id
    AND casting.movie_id=movie.movie_id
    ORDER BY score desc;
```

Exercise

- If there are two actor with same name and different id they may be an error

```
SELECT Star1.name, Star1.actor_id,
  Star2.name, Star2.actor_id
FROM actor Star1, actor Star2
WHERE Star1.name= Star2.name
AND Star1. actor_id < Star2. actor_id;
```

Exercise-2

- Difference between this query and the previous one

```
SELECT Star1. name, Star1. actor_id , Star2.
  name, Star2. actor_id
FROM actor Star1, actor Star2
WHERE Star1. name = Star2. name
AND Star1. actor_id <> Star2. actor_id ;
```

Relation (query) combination

- Union: union
- Intersect: intersection
- Except: subtraction of sets
- This operation delete duplicates
  - Use ALL to cancel this behaviour: e.g., UNION ALL
- Relation must be compatible
- Actors that appear in Star Trek IV and Star Trek V

```
(SELECT name FROM movie natural join actor natural
 join casting WHERE title LIKE 'Star Trek V:%')
 INTERSECT (SELECT name FROM movie natural join actor
 natural join casting WHERE title LIKE 'Star Trek
 IV:%');
```

Examples copy table

- Movies with less than 5000 votes.

```
SELECT title
FROM movie
WHERE votes > 5000;
```

- Citizen Kane's premiere.

```
SELECT year
FROM movie
WHERE title = 'Citizen Kane';
```

Examples

- Title and score of the ‘Police Academy…’saga.

```
SELECT title, score
FROM movie
WHERE title LIKE 'Police Academy%';
```

- Tile and score of the movies that have the word 'Dog' in the title. (similar to)

```
SELECT title, score
FROM movie
WHERE (title LIKE '%dog%') OR (title
                           LIKE '%Dog%');
```

Examples

- Movies in which 'Harrison Ford' appears but he is not the star.

```
SELECT title
 FROM actor natural join  movie natural
 join casting
 WHERE name = 'Harrison Ford' AND ord <> 1;
```

- 'Alien' cast.

```
SELECT name
FROM actor natural join  movie natural join
casting
WHERE title = 'Alien' ;
```

Examples

- Stars of movies filmed in 1962

```
SELECT title, name
FROM movie natural join casting
  natural join actor
WHERE year=1962 AND
  ord=1;
```

Next Chapter

Year of John travolta’s premieres

- First list with year and movies

```
SELECT title, year
FROM movie natural join casting natural join
actor
WHERE  name= 'John Travolta';
```

- Then group then.

```
SELECT COUNT(*), year
FROM movie natural join casting natural join
actor
WHERE name= 'John Travolta'
    GROUP BY year ORDER BY 1;
```

Title and star for 'Julie Andrews' related movies

- First find movie_id

```
SELECT movie_id
FROM casting natural join actor
WHERE name='Julie Andrews';
```

- Then list title and star

```
SELECT title, name
FROM movie natural join casting natural join
actor
WHERE ord=1
AND movie_id IN (SELECT movie_id
                    FROM casting natural join
actor
                    WHERE
                    name='Julie Andrews');
```