

# Databases SQL

Hans Philippi

February 19, 2024

## SQL: Structured Query Language

- IBM, seventies: SEQUEL
- First standard: ANSI 1986
- Updated in 1992, 1999, 2003
- Data definition language (DDL)
- Constraint definition language (DDL)
- Data query language (DQL)
- Data manipulation language (DML)
- Collection model: bags/multisets instead of sets

## SQL DDL

```
CREATE TABLE Book (  
  bookid          integer          not null,  
  title           varchar(100)     not null,  
  author          varchar(100)     not null,  
  price           float,  
  date_of_purchase date,  
  publisher_id     varchar(6),  
  CONSTRAINT Book_pk PRIMARY KEY (bookid),  
  CONSTRAINT Book_fk_Publisher FOREIGN KEY (publisher_id)  
    REFERENCES Publisher(publisher_id)  
);
```

INSERT command

```
INSERT INTO Book VALUES  
  (9876543210, 'The name of the rose', 'Umberto Eco',  
   '11.33', NULL, 'Warner');
```

INSERT command

```
CREATE TABLE Eco_Titles (title varchar(100));
```

```
INSERT INTO Eco_Titles  
  SELECT title FROM Book  
  WHERE author = 'Umberto Eco';
```

## UPDATE and DELETE command

```
UPDATE Artist  
  SET name = 'TAFKAP'  
  WHERE name = 'Prince';
```

```
DELETE FROM Article  
  WHERE author = 'Diederik Stapel';
```

# SQL DQL: query language

Basic template SQL query

```
SELECT <attribute_list>  
FROM <table_list>  
WHERE <condition>
```

Condition:

- comparison attribute with value
- complex conditions, possibly with subqueries

Basic structure has been extended with non-essential syntactic sugar

Example database (inspired by imdb.com)

Movie (*filmid*, title, year, rating, genre, dirid, company, length)

Actor (*pid*, name, birth\_year)

Cast (*filmid*, *pid*, *character*)

Director (*pid*, name, birth\_year)

Series (*filmid*, title)

Episode (*filmid*, *epno*, year, rating, dirid, company)



$Q_1$  : give names of all actors (*projection*)

```
SELECT name  
FROM Actor
```

$Q_2$  : give all production companies (*projection*)

```
SELECT DISTINCT company  
FROM Movie
```

$Q_3$  : give all info about directors born before 1940 (*selection*)

```
SELECT *  
FROM Director  
WHERE birth_year < 1940
```

Q<sub>4</sub> : all info about actors with year of birth unknown

```
SELECT *  
FROM Actor  
WHERE birth_year IS NULL
```

$Q_5$  : names of actors who played character 'Tarzan'  
(*selection - projection - natural join*)

```
SELECT name FROM Actor, Cast
WHERE Actor.pid = Cast.pid
AND character = 'Tarzan'
```

Q<sub>5</sub> : actors who played character 'Tarzan'

```
SELECT name FROM Actor
WHERE pid IN (
    SELECT pid FROM Cast
    WHERE character = 'Tarzan' )
```

```
SELECT name FROM Actor
WHERE EXISTS (
    SELECT * FROM Cast
    WHERE character = 'Tarzan'
    AND Actor.pid = Cast.pid
)
```

$Q_5$  : actors who played character 'Tarzan'

algebra inspired syntactic sugar

$\pi_{name}(\sigma_{character='Tarzan'}(Actor \bowtie Cast))$

```
SELECT name
FROM Actor NATURAL JOIN Cast
WHERE character = 'Tarzan'
```

$Q_6$  : names of actors who did not play in a movie since 1920

```
SELECT name FROM Actor
WHERE pid NOT IN (
    SELECT pid FROM Cast, Movie
    WHERE Cast.filmid = Movie.filmid
    AND year >= 1920
)
```

```
SELECT name FROM Actor
WHERE NOT EXISTS (
    SELECT * FROM Cast, Movie
    WHERE Cast.filmid = Movie.filmid
    AND Actor.pid = Cast.pid
    AND year >= 1920
)
```

$Q_7$  : give names of actors who are also directors

```
(SELECT name FROM Actor)
INTERSECT
(SELECT name FROM Director)
```

SQL also knows UNION and EXCEPT

Member (mno, name, address)

Book (bno, author, title, publisher)

Loan (mno, bno, ldate, rdate)

-- Q101

```
SELECT name FROM Member M, Loan L, Book B  
WHERE author = 'James'
```



```
-- Q102
SELECT name FROM Member M
WHERE EXISTS (
    SELECT * FROM Loan L, Book B
    WHERE B.bno = L.bno AND B.author = 'James'
)
```

```
-- Q103
SELECT name FROM Member M, Loan L
WHERE M.mno = L.mno
AND EXISTS (
    SELECT * FROM Loan L, Book B
    WHERE B.bno = L.bno AND B.author = 'James'
)
```

-- Q104

```
SELECT name FROM Member M
WHERE mno IN (
    SELECT mno FROM Loan L, Book B
    WHERE B.bno = L.bno AND B.author = 'James'
)
```

-- Q105

```
SELECT name FROM Member M
WHERE mno IN (
    SELECT mno FROM Loan L, Book B
    WHERE B.bno = L.bno AND M.mno = L.mno
    AND author = 'James'
)
```

$Q_8$  : actors born in 1980 who never played in thrillers

```
SELECT name FROM Actor A
WHERE birth_year = 1980
AND NOT EXISTS (
    SELECT *
    FROM Cast C, Movie M
    WHERE C.filmid = M.filmid
    AND C.pid = A.pid
    AND genre = 'thriller'
)
```

Q<sub>9</sub> : actors born in 1980 who played in thrillers only

```
SELECT name FROM Actor A
WHERE birth_year = 1980
AND NOT EXISTS (
    SELECT *
    FROM Cast C, Movie M
    WHERE C.filmid = M.filmid
    AND C.pid = A.pid
    AND genre <> 'thriller'
)
```

# SQL DQL: universal quantification

$Q_{10}$  : actors who played in every episode of 'Twin Peaks'

*Rephrase as:*

The actors

for whom there exists no episode of Twin Peaks  
such that they do not play in that episode

$Q_{10}$  : actors who played in every episode of 'Twin Peaks'

*Rephrase as:*

$$\{ \langle a.name \rangle \mid a \in Actor \wedge \\ \forall e \in Episode, s \in Series ( \\ (s.title = 'Twin Peaks' \wedge e.filmid = s.filmid) \Rightarrow \\ \exists c \in Cast (c.filmid = e.filmid \wedge c.pid = a.pid) \\ ) \\ \}$$

SQL does not support universal quantification, so you will have to rewrite this expression using the equivalence of

$$\forall x(P(x) \Rightarrow Q(x))$$

and

$$\neg \exists x(P(x) \wedge \neg Q(x))$$



$$\{ \langle a.name \rangle \mid a \in Actor \wedge \\ \forall e \in Episode, s \in Series ( \\ (s.title = 'Twin Peaks' \wedge e.filmid = s.filmid) \Rightarrow \\ \exists c \in Cast (c.filmid = e.filmid \wedge c.pid = a.pid) \\ ) \\ \}$$

becomes

$$\{ \langle a.name \rangle \mid a \in Actor \wedge \\ \neg \exists e \in Episode, s \in Series ( \\ (s.title = 'Twin Peaks' \wedge e.filmid = s.filmid) \wedge \\ \neg \exists c \in Cast (c.filmid = e.filmid \wedge c.pid = a.pid) \\ ) \\ \}$$

Q<sub>10</sub> : actors who played in every episode of 'Twin Peaks'

```
SELECT name FROM Actor A
WHERE NOT EXISTS (
    SELECT * FROM Episode E, Series S
    WHERE E.filmid = S.filmid
    AND title = 'Twin Peaks'
    AND NOT EXISTS (
        SELECT * FROM Cast C
        WHERE C.filmid = E.filmid
        AND E.pid = A.pid
    )
)
```

```
-- Q106
SELECT name FROM Member M
WHERE mno NOT IN (
    SELECT mno FROM Loan L, Book B
    WHERE B.bno = L.bno AND author = 'James' )
```

```
-- Q107
SELECT name FROM Member M
WHERE NOT EXISTS (
  SELECT * FROM Book B
  WHERE author = 'James'
  AND NOT EXISTS (
    SELECT * FROM Loan L
    WHERE L.bno = B.bno
    AND L.mno = M.mno ))
```

```
-- Q108
SELECT name FROM Member M
WHERE NOT EXISTS (
    SELECT * FROM Book B
    WHERE author = 'James'
    AND NOT EXISTS (
        SELECT * FROM Loan L
        WHERE L.bno = B.bno ))
```

```
-- Q109
SELECT name FROM Member M
WHERE NOT EXISTS (
    SELECT * FROM Loan L
    WHERE L.mno = M.mno
    AND NOT EXISTS (
        SELECT * FROM Book B
        WHERE L.bno = B.bno
        AND author = 'James' ))
```

$Q_{11}$  : all pairs of movies with the same genre

```
SELECT X.title, Y.title  
FROM Movie X, Movie Y  
WHERE X.genre = Y.genre
```

$Q_{12}$  : the longest movie

```
SELECT * FROM Movie
WHERE length >= ALL
      (SELECT length FROM Movie)
```

$Q_{13}$  : give all movies except the shortest one(s)

```
SELECT * FROM Movie
WHERE length > ANY
      (SELECT length FROM Movie)
```



Q<sub>14</sub> : give all thrillers from 2012 ordered by length (longest first)

```
SELECT * FROM Movie
WHERE genre = 'thriller' AND year = 2012
ORDER BY length DESC, title ASC
```

Q<sub>15</sub> : how many thrillers were made in 2012

```
SELECT count(*) FROM Movie
WHERE genre = 'thriller' AND year = 2012
```

$Q_{16}$  : give for each actor the average rating of movies (s)he played in

```
SELECT pid, name, avg(rating)
FROM Movie M, Cast C, Actor A
WHERE M.filmid = C.filmid AND C.pid = A.pid
GROUP BY pid, name
```

aggregate functions:

COUNT, SUM, MIN, MAX, AVG

$Q_{16'}$  : give for each actor the average rating of movies (s)he played in, if this average exceeds 7

```
SELECT pid, name, avg(rating)
FROM Movie M, Cast C, Actor A
WHERE M.filmid = C.filmid AND C.pid = A.pid
GROUP BY pid, name
HAVING avg(rating) > 7
```

Q<sub>17</sub> : give the actor with the highest average movie rating

```
SELECT pid, name, avg(rating)
FROM Movie M, Cast C, Actor A
WHERE M.filmid = C.filmid AND C.pid = A.pid
GROUP BY pid, name
HAVING avg(rating) >= ALL (
    SELECT avg(rating)
    FROM Movie M, Cast C
    WHERE M.filmid = C.filmid
    GROUP BY pid
)
```

Q<sub>18</sub> : the length of 'Avatar' in hours instead of minutes

```
SELECT length/60  
FROM Movie  
WHERE title = 'Avatar'
```

Q<sub>19</sub> : give the directors whose name ends with 'Coen'

```
SELECT *  
FROM Director  
WHERE name LIKE '%Coen'
```

- <http://www.w3schools.com/sql>
- <https://en.wikipedia.org/wiki/SQL>