SQL

SQL Queries

Principal form:

SELECT desired attributes
FROM tuple variables — range over relations
WHERE condition about tuple variables

Running example relation schema:

Movie(*title*, *year*, *length*, *inColor*, *studio-Name*, *producerC#*)
StarsIn(*movieTitle*, *movieYear*, *starName*)
MovieStar(*name*, *address*, *gender*, *birthdate*)
MovieExec(*name*, *address*, *cert#*, *netWorth*)
Studio(*name*, *address*, *presC#*)

Example

Consider relation:

Movie(*title*, *year*, *length*, *inColor*, *studio-Name*, *producerC#*)

Find all the movies produced by Disney Studios in 1990…

```
SELECT *
FROM Movie
WHERE studio-Name='Disney' AND year=1990
```

Find all the movies made by Fox that are at least 100 min. long

```
SELECT *
FROM Movie
WHERE studio-Name='Fox' AND length >= 100
```

Star as List of All Attributes

Movie:

title	*year*	*length*	*in-Color*	*studio-Name*	*produceC#*
Star Wars	1977	124	true	Fox	12345
Mighty Ducks	1991	104	true	Disney	67890
Wayne's World	1992	95	true	Paramaount	99999
Spider-Man	2002	121	true	Columbia	12345
Episode I	1999	133	true	Fox	45634
Episode II	2002	142	true	Fox	23456

SELECT *
FROM *Movie*
WHERE *studio-Name*='Disney' **AND** *year*>=1990

title	*year*	*length*	*in-Color*	*studio-Name*	*produceC#*
Mighty Ducks	1991	104	true	Disney	67890

Exercise: Write the above query in relational algebra…

Projection in SQL

Movie:

title	*year*	*length*	*in-Color*	*studio-Name*	*produceC#*
Star Wars	1977	124	true	Fox	12345
Mighty Ducks	1991	104	true	Disney	67890
Wayne's World	1992	95	true	Paramaount	99999
Spider-Man	2002	121	true	Columbia	12345
Episode I	1999	133	true	Fox	45634
Episode II	2002	142	true	Fox	23456

SELECT *title, length*
FROM *Movie*
WHERE *studio-Name*='Disney' **AND** *year*>=1990

title	*length*
Mighty Ducks	104

Exercise: Write the above query in relational algebra…

Renaming columns

Movie:

title	*year*	*length*	*in-Color*	*studio-Name*	*produceC#*
Star Wars	1977	124	true	Fox	12345
Mighty Ducks	1991	104	true	Disney	67890
Wayne's World	1992	95	true	Paramaount	99999
Spider-Man	2002	121	true	Columbia	12345
Episode I	1999	133	true	Fox	45634
Episode II	2002	142	true	Fox	23456

SELECT *title* **AS** *name, length* **AS** *duration*
FROM *Movie*
WHERE *studio-Name*='Disney' **AND** *year*>=1990

name	*duration*
Mighty Ducks	104

Exercise: Write the above query in relational algebra…

Expressions as Values in Columns

Movie:

title	*year*	*length*	*in-Color*	*studio-Name*	*produceC#*
Star Wars	1977	124	true	Fox	12345
Mighty Ducks	1991	104	true	Disney	67890
Wayne's World	1992	95	true	Paramaount	99999
Spider-Man	2002	121	true	Columbia	12345
Episode I	1999	133	true	Fox	45634
Episode II	2002	142	true	Fox	23456

SELECT *title* **AS** *name, length * 0.016667* **AS** *h-duration*
FROM *Movie*
WHERE *studio-Name*='Disney' **AND** *year*>=1990

name	h-*duration*
Mighty Ducks	1

Other tricks…

Movie:

title	*year*	*length*	*in-Color*	*studio-Name*	*produceC#*
Star Wars	1977	124	true	Fox	12345
Mighty Ducks	1991	104	true	Disney	67890
Wayne's World	1992	95	true	Paramaount	99999
Spider-Man	2002	121	true	Columbia	12345
Episode I	1999	133	true	Fox	45634
Episode II	2002	142	true	Fox	23456

SELECT *title* **AS** *name, length * 0.016667* **AS** *h-duration,*
'hrs.' **AS** *inHours*
FROM *Movie*
WHERE *studio-Name*='Disney' **AND** *year*>=1990

name	h-*duration*	*inHours*
Mighty Ducks	1	hrs.

Selection in SQL

Consider relation:
Movie(*title*, *year*, *length*, *inColor*, *studio-Name*, *producerC#*)

Find the length of the movie 'Wayne's World'…

SELECT *length*
FROM *Movie*
WHERE *title* = 'Wayne''s World' **AND** *year* = 1992

Notes:

- Two single-quotes in a character string represent one single quote.
- Conditions in **WHERE** clause can use logical operators **AND**, **OR**, **NOT** and parentheses in the usual way.
- Remember: SQL is *case insensitive*. Keywords like **SELECT** or **AND** can be written upper/lower case as you like. Only inside quoted strings does case matter.

Patterns

- % stands for any string.
- _ stands for any one character.
- "Attribute **LIKE** pattern" is a condition that is true if the string value of the attribute matches the pattern. Also **NOT LIKE** for negation.

Example:
Given relation *Movie*(*title*, *year*, *length*, *inColor*, *studio-Name*, *producerC#*)

find movies with title "Star *something*", where *something* has 4 letters…

SELECT *title*
FROM *Movie*
WHERE *title* **LIKE** 'Star _ _ _ _'

Note: patterns must be quoted, like strings.

Nulls

Used in place of a value in a tuple's component.

- Interpretation is not exactly “missing value.”
- There could be many reasons why no value is present, *e.g.*, “value inappropriate.”

Operations with expressions that evaluate to **NULL**

- The result is always **NULL**

Comparing expressions that evaluate to **NULL** with values

- 3rd truth value **UNKNOWN**.
- A query only produces tuples if the **WHERE**-condition evaluates to **TRUE** (**UNKNOWN** is not sufficient).

Example

title	*year*	*length*	*in-Color*	*studio-Name*	*produceC#*
Mighty Ducks	1991	NULL	true	Disney	67890

SELECT *title*, *year*
FROM *Movie*
WHERE *length* < 90 **OR** *length* >= 90

UNKNOWN **UNKNOWN**

UNKNOWN

Mighty Ducks is not *selected*, even though the **WHERE** condition is a tautology.

MORE FEATURES

3 Valued Logic

Think of true = 1; false = 0, and unknown = 1/2. Then:

- AND = min.
- OR = max.
- NOT(x) = 1 – x.

Using the above we can verify that key laws fail to hold…

Example:

p **OR** (**NOT** *p)* = TRUE

- For 3-valued logic: if p = unknown, then left side = max(1/2,(1–1/2)) = 1/2 ≠ 1.
- There is no way known to make 3-valued logic conform to all the laws we expect for 2-valued logic

Multi-relation Queries

- List of relations in **FROM** clause.
- Relation-dot-attribute disambiguates attributes from several relations

Example:

Consider

Movie(*title*, *year*, *length*, *inColor*, *studio-Name*, *producerC#*)

MovieExec(*name*, *address*, *cert#*, *netWorth*)

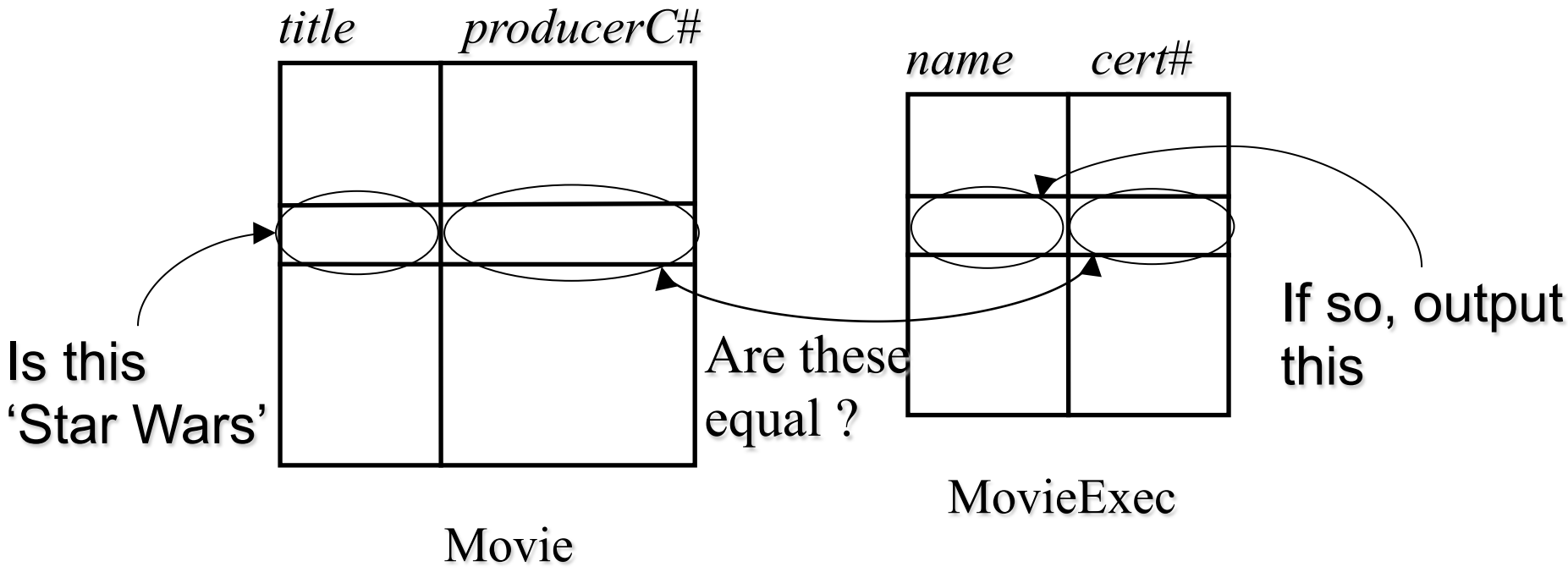
I wonder who is the producer of Star Wars…

SELECT *name*
FROM *Movie*, *MovieExec*
WHERE *title* = 'Star Wars' **AND** *producerC#* = *cert#*

Exercise: Write the above query in relational algebra…

Example

SELECT *name*
FROM *Movie*, *MovieExec*
WHERE *title* = 'Star Wars' **AND** *producerC#* = *cert#*



Formal Semantics of Multi-relation Queries

Same as for single relation, but start with the product of all the

relations mentioned in the **FROM** clause.

Operational Semantics

Consider a tuple variable for each relation in the **FROM**

- Imagine these tuple variables each pointing to a tuple of their relation, in all combinations (*e.g.*, nested loops).
- If the current assignment of tuple-variables to tuples makes the **WHERE** true, then output the attributes of the **SELECT**

Explicit Tuple Variables

Sometimes we need to refer to two or more copies of a relation.

To do that we use *tuple variables* as aliases of the relations.

Example:

Consider *MovieStar*(*name*, *address*, *gender*, *birthdate*)

Now, find two stars that share the same address….

SELECT $Star_1$.*name*, $Star_2$.*name*
FROM *MovieStar* $Star_1$, *MovieStar* $Star_2$
WHERE $Star_1$.*address* = $Star_2$.*address* **AND**
$Star_1$.*name* < $Star_2$.*name*

Note that $Star_1$.*name* < $Star_2$.*name* is needed to avoid producing

(Carrie, Carrie) and to avoid producing a pair in both orders.

Union/Intersection/Difference

Consider

MovieStar(*name*, *address*, *gender*, *birthdate*)
MovieExec(*name*, *address*, *cert#*, *netWorth*)

Find all the female movie stars who are also executives and have a net worth over $10,000,000

(**SELECT** *name*, *address*
FROM *MovieStar*
WHERE *genter* = 'F')
INTERSECT
(**SELECT** *name*, *address*
FROM *MovieExec*
WHERE *netWorth* > 10000000)

Union/Intersection/Difference (cont.)

Consider

MovieStar(*name*, *address*, *gender*, *birthdate*)
MovieExec(*name*, *address*, *cert#*, *netWorth*)

Find all the movie stars who are not executives

(**SELECT** *name*, *address*
FROM *MovieStar*)
EXCEPT
(**SELECT** *name*, *address*
FROM *MovieExec*)

Union/Intersection/Difference (cont.)

Consider

MovieStar(*name*, *address*, *gender*, *birthdate*)
MovieExec(*name*, *address*, *cert#*, *netWorth*)

Find all the names of people that are either movie stars or movie executives

(**SELECT** *name*, *address*
FROM *MovieStar*)
UNION
(**SELECT** *name*, *address*
FROM *MovieExec*)

Subqueries

Result of a select-from-where query can be used in the where-clause of another query.

Example:

Consider *Movie*(*title*, *year*, *length*, *inColor*, *studio-Name*, *prodC#*)

Find movies that have the same length with Star Wars…

SELECT *title*, *year*

FROM *Movie*

WHERE *length* = (**SELECT** *length*

FROM *Movie*

WHERE *title* = 'Star Wars' **AND** *year* = 1977)

- Notice the *scoping rule*: an attribute refers to the most closely nested relation with that attribute.
- Parentheses around subquery are essential.

The IN Operator

Use: “Tuple IN relation” is true iff the tuple is in the relation.

Example:

Find the birthdate for all the stars in Star Wars…

StarsIn(*movieTitle*, *movieYear*, *starName*)
MovieStar(*name*, *address*, *gender*, *birthdate*)

```
SELECT name, birthdate
FROM MovieStar
WHERE name IN (SELECT name
               FROM StarsIn
               WHERE movieTitle = 'Star Wars' AND
                     movieYear = 1977)
```

- Also: **NOT IN**

EXISTS

Use: “EXISTS(relation)” is true iff the relation is nonempty

Example:

Find the movie stars that have unique birthdates...

MovieStar(*name*, *address*, *gender*, *birthdate*)

```
SELECT name
FROM MovieStar MS1
WHERE NOT EXISTS
      (SELECT *
       FROM MovieStar MS2
       WHERE MS2.birthdate = MS1.birthdate AND
             MS2.name <> MS1.name)
```

- A subquery that refers to values from a surrounding query is called a *correlated subquery*.

Quantifiers

ANY and **ALL** behave as existential and universal quantifiers

Beware: in common parlance, “any” and “all” seem to be synonyms, *e.g.*, “I am fatter than any of you” vs. “I am fatter than all of you.” But in SQL…

Example:

Consider *Movie*(*title*, *year*, *length*, *inColor*, *studio-Name*, *prodC#*)

Find the movie(s) with the biggest length…

```
SELECT title, year
FROM Movie
WHERE length >= ALL(SELECT length
                    FROM Movie)
```

Subqueries in From **Clauses**

We can use non-stored relations in the **FROM** clause as the following example illustrates…

Example:

Find the names of the producers of Harrison Ford's movies

SELECT *name*
FROM *MovieExec,* (**SELECT** *producerC#*
FROM *Movie*, *StarsIn*
WHERE *title* = *movieTitle* **AND**
year = *movieYear* **AND**
starname = 'Harrison Ford') Prod
WHERE *cert#* = *Prod.producerC#*

Join-Based Expressions

A number of forms are provided. They can be used either stand-alone (in place of a **SELECT-FROM-WHERE**) or to define a relation in the **FROM**-clause.

- *R* **NATURAL JOIN** *S*
- *R* **JOIN** *S* **ON** *condition*
  e.g., condition: *R.B=S.B*
- *R* **CROSS JOIN** *S*
- *R* **OUTER JOIN** *S*
- Outerjoin can be modified by:
  1. Optional **NATURAL** in front.
  2. Optional **ON** *condition* at end.
  3. Optional **LEFT**, **RIGHT**, or **FULL** (default) before **OUTER**.

Example

Consider

Movie(*title*, *year*, *length*, *inColor*, *studio-Name*, *prodC#*)

MovieExec(*name*, *address*, *cert#*, *netWorth*)

I wonder who is the producer of Star Wars…

```
SELECT name
FROM Movie, MovieExec
WHERE title = 'Star Wars' AND producerC# = cert#
```

```
SELECT name
FROM Movie JOIN MovieExec ON producerC# = cert#
WHERE title = 'Star Wars'
```

Forcing Set/Bag Semantics

- Default for select-from-where is bag; default for union, intersection, and difference is set.
  - Why? Saves time of not comparing tuples as we generate them.
  - But we need to sort anyway when we take intersection or difference. (Union seems to be thrown in for good measure!)
- Force set semantics with **DISTINCT** after **SELECT**.
  - But make sure the extra time is worth it.
- Force bag semantics in Union/Intersection/Difference using **ALL**

Example

Consider

MovieStar(*name*, *address*, *gender*, *birthdate*)
MovieExec(*name*, *address*, *cert#*, *netWorth*)

Find all the names of people that are either movie stars or movie executives

```
(SELECT name, address
FROM MovieStar)
    UNION ALL
(SELECT name, address
FROM MovieExec)
```

Find all the unique names of people that are stars

```
SELECT DISTINCT name
FROM MovieStar
```

Aggregations

SUM, **AVG**, **MIN**, **MAX**, and **COUNT** apply to attributes/columns.

Also, **COUNT**(*) applies to tuples.

Note: Use these in lists following **SELECT**

Example:

Consider *Movie*(*title*, *year*, *length*, *inColor*, *studio-Name*, *prodC#*)

Find the average movie-length during year 2002…

```
SELECT AVG(length)
FROM Movie
WHERE year = 2002
```

Eliminating Duplicates Before Aggregation

Consider *Movie*(*title*, *year*, *length*, *inColor*, *studio-Name*, *prodC#*)

Find the number of different movie-lengths for year 2002

```
SELECT COUNT(DISTINCT length)
FROM Movie
WHERE year = 2002
```

DISTINCT may be used in any aggregation, but typically only

makes sense with **COUNT**

Grouping

Follow select-from-where by **GROUP BY** and a list of attributes.

The relation that is the result of the **FROM** and **WHERE**

clauses is grouped according to the values of these attributes,

and aggregations take place only within a group

Example:

Consider *Movie*(*title*, *year*, *length*, *inColor*, *studio-Name*, *prodC#*)

Find the average movie-length per year…

```
SELECT year, AVG(length)
FROM Movie
GROUP BY year
```

Example

Consider *Movie*(*title*, *year*, *length*, *inColor*, *studio-Name*, *prodC#*)

Find the average movie-length per year for movies produced by

the Fox studios…

```
SELECT year, AVG(length)
FROM Movie
WHERE studio-Name = 'Fox'
GROUP BY year
```

Note: grouping occurs after the selection operation

Restriction on SELECT W/ Aggregation

If any aggregation is used, then *each* element of a SELECT clause must either be aggregated or appear in a group-by clause

Example:

Consider *Movie*(*title*, *year*, *length*, *inColor*, *studio-Name*, *prodC#*)

The following might seem a tempting way to find the movie with the shortest length.

```
SELECT title, year, MIN(length)
FROM Movie
```

But it is illegal in SQL.

Problem: How would we find this movie?

HAVING Clauses

HAVING clauses are selections on groups, just as

WHERE clauses are selections on tuples.

The condition can use the tuple variables or relations

in the **FROM** and their attributes, just like the **WHERE**

can.

- But the tuple variables range only over the group.
- And the attribute better make sense within a group; *i.e.*, be one of the grouping attributes.

Example

Consider

Movie(*title*, *year*, *length*, *inColor*, *studio-Name*, *prodC#*)

MovieExec(*name*, *address*, *cert#*, *netWorth*)

Find the total film length for only those producers who made at least one film prior to 1930.

```
SELECT name, SUM(length)
FROM Movie, MovieExec
WHERE prodC# = cert#
GROUP BY name
HAVING MIN(*) < 1930
```