The Database Language SQL

Structured Query Language

- The language with which you can query relational databases.
- It is a standard (long history…)
- Each vendor implements the standard and uses its own extensions/conventions.
- Learn the basic core set of commands, should be able to adapt to any environment easily (this is what manuals are for…)

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

Formal Semantics of Single-Relation SQL Query

- Start with the relation in the **FROM** clause.
- Apply (bag) σ, using condition in **WHERE** clause.
- Apply (extended, bag) π using attributes in **SELECT** clause.

Equivalent Operational Semantics

Imagine a *tuple variable* ranging over all tuples of the relation. For each tuple:

- Check if it satisfies the **WHERE** clause.
- Print the values of terms in **SELECT**, if so.

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	Paramount	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	Paramount	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	Paramount	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	Paramount	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	1.95

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	Paramount	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).

13

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.

14

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

15

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…

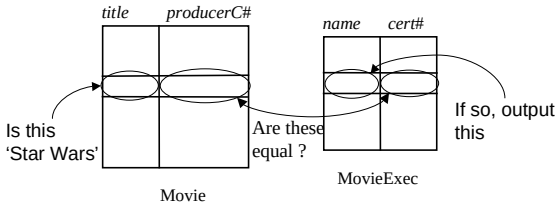
SELECT *name*
FROM *Movie*, *MovieExec*
WHERE *title* = 'Star Wars' **AND** *producerC# = cert#*

Exercise: Write the above query in relational algebra…

16

Example

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



17

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 for each relation in the **FROM**

- Imagine each tuple pairing up with each tuple from the other relation (*e.g.*, nested loops).
- If the current pair makes the **WHERE** true, then output the attributes of the **SELECT**

18

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.

19

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)

20

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*)

21

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*)

22

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.

23

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**

24

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*.

25

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 longest length…

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

26

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#*

27