The Database Language SQL (cont.)

1

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**.

2

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'

3

Forcing Set/Bag Semantics

- Default for select-from-where is bag; default for union, intersection, and difference is set.
- 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**

4

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*

5

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

6

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**

7

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*

8

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

9

Restriction on SELECT Lists With 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?

10

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.

11

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(year) < 1930

12

Database Modifications

Modification = insert + delete + update
e.g. to insert a tuple we use the statement
INSERT INTO <relation> **VALUES** (<list of values>)

- In the above statement the values are listed in the same order with which the attributes were declared
- If we want to ignore this order then we list the attributes as arguments of the relation

Example:
Consider *StarsIn*(*movieTitle*, *movieYear*, *starName*)
Insert the fact that Sydney Green stars in The Maltese Falcon

INSERT INTO *StarsIn*(*movieTitle, starName, movieYear)*
VALUES('The Maltese Falcon', 'Sydney Green', 1942);

13

Insertion of the Result of a Query

Syntax: **INSERT INTO** <relation> (<subquery>).

Example:
Consider
Studio(*name, address, presC#*)
Movie(*title*, *year*, *length*, *inColor*, *studio-Name*, *prodC#*)
Add to relation *Studio* all the studios that are mentioned in relation *Movie*

INSERT INTO *Studio*(*name)*
SELECT DISTINCT *studio-Name*
FROM *Movie*
WHERE *studio-Name* **NOT IN** (**SELECT** *name*
FROM *Studio)*

14

Deletion

Syntax: **DELETE FROM** <relation> **WHERE** <condition>
Semantic: Deletes all tuples satisfying the condition from the named relation

Example:
Consider *StarsIn*(*movieTitle*, *movieYear*, *starName*)
Sydney Green was not a star in The Maltese Falcon…

DELETE FROM *StarsIn*
WHERE *movieTitle* = 'The Maltese Falcon' **AND**
movieYear = 1942 **AND** *starName* = 'Sydney Green'

As another example, to make the *StarsIn* relation empty execute

DELETE FROM *StarsIn*

15

Updates

Syntax: **UPDATE** <relation>
SET <new-value assignments>
WHERE <condition>

Example:
Consider
MovieExec(*name*, *address*, *cert#*, *netWorth*)
Studio(*name, address, presC#*)
Prefix the name of every movie executive with the title 'Pres. ' if the executive is the president of a studio.

UPDATE *MovieExec*
SET *name* = 'Pres. ' || *name*
WHERE *cert#* **IN** (**SELECT** *presC#* **FROM** *Studio)*

16

Defining a Database Schema

Syntax: **CREATE TABLE** <name> (<list of elements>)

- Principal elements are attributes and their types, but key declarations and constraints also appear.
- Similar **CREATE X** commands for other schema elements **X**: **VIEWS**, **INDEX**, **ASSERTION**, **TRIGGER**.
- "**DROP X** *name*" deletes the created element of kind **X** with that name.

Example: *Studio*(*name*, *address*, *presC#*)

CREATE TABLE *Studio* (
name **CHAR**(20),
address **VARCHAR**(20),
presC# **REAL**)

DROP TABLE *Studio*

17

Types

- **INT** or **INTEGER**.
- **BOOLEAN**
- **REAL** or **FLOAT**.
- **CHAR**(*n*) = fixed length character string, padded with "pad characters."
- **VARCHAR**(*n*) = variable-length strings up to *n* characters.
- **DATE** & **TIME**
- **Plus many many more, check the manuals**

18

Declaring Keys

Use **PRIMARY KEY** or **UNIQUE**.

- But only one primary key, many **UNIQUE**s allowed.
- SQL permits implementations to create an *index* (data structure to speed access given a key value)
- SQL does not allow nulls in primary key, but allows them in "unique" columns (which may have two or more nulls, but not repeated non-null values).

19

Declaring Keys

Two places to declare:

- After an attribute's type, if the attribute is a key by itself.
- As a separate element.
  - Essential if key is >1 attribute.

20

Example

CREATE TABLE *Studio* (
name **CHAR**(20),
address **VARCHAR**(20),
presC# **REAL**,
PRIMARY KEY(*name*))

21

Example

Find the differences…

CREATE TABLE *Studio-version1* (
name **CHAR**(20),
address **VARCHAR**(20) **UNIQUE**,
presC# **REAL UNIQUE**,
PRIMARY KEY(*name*))

CREATE TABLE *Studio-version2* (
name **CHAR**(20),
address **VARCHAR**(20),
presC# **REAL**,
UNIQUE (*address*, *presC#*),
PRIMARY KEY(*name*))

22

Other Properties You Can Give to Attributes

- **NOT NULL** = every tuple must have a real value for this attribute.
- **DEFAULT** value = a value to use whenever no other value of this attribute is known.

Example:

CREATE TABLE *Studio* (
name **CHAR**(30) **PRIMARY KEY**,
addr **CHAR**(50) **DEFAULT** '123 Sesame St',
presC# **REAL NOT NULL**)

23

Example

Consider executing the following statement

INSERT INTO *Studio*(*name, presC#*)
VALUES('Fox', 90943)

The result is the following tuple:

name	addr	*presC#*
Fox	123 Sesame St.	90943

Primary key is by default not NULL.
This insertion is legal. It is OK to list a subset of the attributes and values for only this subset. But if we had forgot to specify a value for *presC#* then the insertion could not be made.

24

Changing Columns

Add an attribute to relation *R* with
ALTER TABLE *R* **ADD** <column declaration>

Example:
CREATE TABLE *MovieStar* (
name **CHAR**(30),
address **VARCHAR**(255),
gender **CHAR(1)**,
birthdate **DATE,**
PRIMARY KEY(*name*))

ALTER TABLE *MovieStar* **ADD** *phone* **CHAR**(16) **DEFAULT '**unlisted**'**

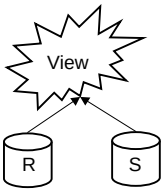
Columns may also be dropped…

Example"
ALTER TABLE *MovieStars* **DROP** *birthdate*

25

Views

An expression that describes a table without creating it.



View definition form is:
CREATE VIEW <name> **AS** <query>

26

Example

The view *ParamountMovie* is the titles and years of the movies made by Paramount. We can define this view as follows:

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

CREATE VIEW *ParamountMovie* **AS**
SELECT *title*, *year*
FROM *Movie*
WHERE *studioName* = 'Paramount'

Querying Views: Treat the view as if it were a materialized relation

Example:
SELECT *title*
FROM *ParamountMovie*
WHERE *year* = 1979

27

Interpreting Queries Involving Views

CREATE VIEW *ParamountMovie* **AS**
SELECT *title*, *year*
FROM *Movie*
WHERE *studioName* = 'Paramount'

$\pi_{title, year}$
$\sigma_{studio\text{-}Name=\text{"Paramount"}}$
Movie

28

Interpreting Queries Involving Views (cont.)

SELECT *title*
FROM *ParamountMovie*
WHERE *year* = 1979

π_{title}
$\sigma_{year = 1979}$
ParamountMovie

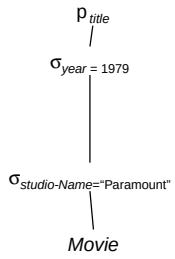
29

Interpreting Queries Involving Views (cont.)

π_{title}
$\sigma_{year = 1979}$
$\pi_{title, year}$
$\sigma_{studio\text{-}Name=\text{"Paramount"}}$
Movie

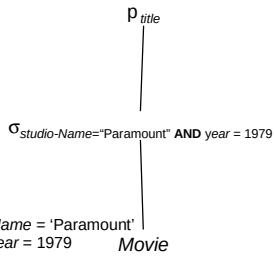
30

Interpreting Queries Involving Views (cont.)



31

Interpreting Queries Involving Views (cont.)



SELECT *title*
FROM *Movie*
WHERE *studioName* = 'Paramount'
AND *year* = 1979

32