Ceng 302
Database Management Systems

SQL: Structured Query Language

Prof. Dr. Adnan YAZICI

Department of Computer Engineering,

Middle East Technical University

(Fall 2021)

DML - Overview

- select-from-where
- set operations
- nested subqueries
- built-in functions
- insert, delete, update
- joins and outer joins
- group by and having clauses
- ordering of rows
- embedded DML
- views
- indexing
- triggers
- integrity constraints
- db authorizations
- recursive queries

Relational Query Languages

- ***Query languages***: Allow manipulation and retrieval of data from a database.
- Relational model supports simple, powerful QLs:
  - Strong formal foundation based on formal logic.
  - Allows for much optimization.
- Query Languages **!=** programming languages!
  - QLs are not expected to be "Turing complete" or "computationally universal. "
  - QLs are not intended to be used for complex calculations.
  - QLs support easy, efficient access to large data sets.

Interactive DML - select-from-where

SELECT $A_1, A_2, \ldots A_n$

FROM $R_1 , R_2 , \ldots R_m$

WHERE P

- the **SELECT** clause specifies the columns of the result
- the **FROM** clause specifies the tables to be scanned in the query
- the **WHERE** clause specifies the condition on the columns of the tables in the **FROM** clause
- equivalent algebra statement:

$$\pi_{A_1, A_2, \ldots A_n} (\sigma_P \; (R_1 \text{ x } R_2 \text{ x } \ldots \text{ x } R_m))$$

Basic SQL Query

```
SELECT      [DISTINCT]  target-list
FROM        relation-list
WHERE       qualification
```

- *relation-list* A list of relation names (possibly with a *range-variable* after each relation name).
- *target-list* A list of attributes of relations included in *relation-list*
- *qualification* Comparisons (**Attr** *op* **const** or **Attr1** *op* **Attr2**, where *op* is one of $(<, >, =, \leq, \geq, \neq)$ combined using AND, OR and NOT.
- DISTINCT is an optional keyword indicating that the answer should not contain duplicates. Default is that duplicates are **not** eliminated!

Conceptual Evaluation Strategy

- Semantics of an SQL query defined in terms of the following conceptual evaluation strategy:
  - Compute the cross-product of *relation-list.*
  - Discard resulting tuples if they fail *qualifications.*
  - Delete attributes that are not in *target-list.*
  - If DISTINCT is specified, eliminate duplicate rows.
- This strategy is probably the least efficient way to compute a query! An optimizer will find more efficient strategies to compute the same answers.

Example Instances

- We use these instances of the Sailors, Boats and Reserves relations in our examples.

Reserves

sid	bid	day
22	101	10/10/96
58	103	11/12/96

Sailors

sid	sname	rating	age
22	dustin	7	45.0
31	lubber	8	55.5
58	rusty	10	35.0

Boats

bid	bname	color
101	Intertake	blue
102	Intertake	red
103	Clipper	green
104	Marine	red

Example of Conceptual Evaluation

Query: Retrieve the sailors names who reserved the boat id 103.

```
SELECT  S.sname
FROM    Sailors S, Reserves R
WHERE   S.sid=R.sid AND R.bid=103
```

(sid)	sname	rating	age	(sid)	bid	day
22	dustin	7	45.0	22	101	10/10/96
22	dustin	7	45.0	58	103	11/12/96
31	lubber	8	55.5	22	101	10/10/96
31	lubber	8	55.5	58	103	11/12/96
58	rusty	10	35.0	22	101	10/10/96
58	rusty	10	35.0	58	103	11/12/96

A Note on Range Variables

- Needed only if the same relation appears twice in the FROM clause. The previous query can also be written as:

```
SELECT  sname
FROM    Sailors, Reserves
WHERE   Sailors.sid=Reserves.sid
          AND bid=103
```

OR

```
SELECT  S.sname
FROM    Sailors S, Reserves R
WHERE   S.sid=R.sid AND bid=103
```

Using range variables is a good style!

Expressions and Strings

Query: *'Find triples (of ages of sailors and two fields defined by expressions) for sailors whose names begin and end with B and contain at least three characters.'*

```
SELECT  S.age, age1 = S.age-5, 2*S.age AS age2
FROM  Sailors S
WHERE  S.sname LIKE 'B_%B'
```

This query illustrates the use of arithmetic expressions and string pattern matching:

- AS and = are two ways to name fields in result.
- LIKE is used for string matching. `_' stands for any one character and `%' stands for 0 or more arbitrary characters.

Nested Queries

Q: "Find names of sailors who've reserved boat #103"

```
SELECT  S.sname
FROM    Sailors S
WHERE   S.sid IN  (SELECT  R.sid
                   FROM    Reserves R
                   WHERE   R.bid=103)
```

- IN operator performs a direct match between the columns specified before the IN keyword and a subquery result.
- The IN clause scan all records fetched from the given subquery column.
- For this query, for each **Sailors** tuple, it checks the qualification by computing the nested subquery, if at least one tuple is in the result of nested query, then select that **Sailors** tuple.
- To find "*sailors who've not reserved #103*," use NOT IN.

Nested Queries with Correlation

***Q:** "Find names of sailors who've reserved boat #103"*

```
SELECT  S.sname
FROM    Sailors S
WHERE   EXISTS  (SELECT  *
                 FROM    Reserves R
                 WHERE   R.bid=103 AND S.sid=R.sid)
```

- EXISTS is another set comparison operator. EXISTS is used to check whether the result of a correlated nested subquery is empty or not. So, it checks the subquery result and returns an either TRUE or FALSE value.
- EXISTS operator returns TRUE if the subquery returns single or multiple records. Otherwise, it gives a FALSE result when no records are returned.
- If UNIQUE is used, and * is replaced by *R.bid*, "*finds sailors with at most one reservation for boat #103*." (UNIQUE checks for duplicate tuples; * denotes all attributes.)

Examples of Division A/B

sno	pno
s1	p1
s1	p2
s1	p3
s1	p4
s2	p1
s2	p2
s3	p2
s4	p2
s4	p4

A

pno
p2

B1

pno
p2
p4

B2

pno
p1
p2
p4

B3

sno
s1
s2
s3
s4

A/B1

sno
s1
s4

A/B2

sno
s1

A/B3

Division in SQL

Q: *“Find sailors who’ve reserved **all** boats.”*

(1)

```
SELECT  S.sname
FROM    Sailors S
WHERE  NOT EXISTS
          ((SELECT  B.bid
            FROM     Boats B)
          EXCEPT
            (SELECT  R.bid
             FROM     Reserves R
             WHERE  R.sid=S.sid))
```

- Let’s do it without EXCEPT:

(2)

```
SELECT  S.sname
FROM    Sailors S
WHERE  NOT EXISTS  (SELECT  B.bid
                    FROM    Boats B
                    WHERE  NOT EXISTS  (SELECT  R.bid
                                        FROM    Reserves R
                                        WHERE  R.bid=B.bid
                                           AND  R.sid=S.sid))
```

Sailors S such that …

there is no boat B

without any Reserves tuple showing S reserved B

OR “Select each sailor such that there does not exist any boat that the sailor does not reserve it.”

More on Set-Comparison Operators

- We've already seen IN, EXISTS and UNIQUE. Can also use NOT IN, NOT EXISTS and NOT UNIQUE.
- Also available: *op* ANY, *op* ALL,

$$op \text{ IN } \{>,<,=,\geq,\leq,\neq\}$$

Q: *"Find sailors whose rating is greater than that of some sailor called Horatio"*

```
SELECT  *
FROM    Sailors S
WHERE  S.rating > ANY  (SELECT  S2.rating
                         FROM    Sailors S2
                         WHERE  S2.sname='Horatio')
```

Aggregate Operators

- **Significant extension of relational algebra.**

```
COUNT (*)
COUNT ( [DISTINCT] A)
SUM ( [DISTINCT] A)
AVG ( [DISTINCT] A)
MAX (A)
MIN (A)
```

single column

```
SELECT  COUNT (*)
FROM  Sailors S
```

```
SELECT  S.sname
FROM  Sailors S
WHERE  S.rating = (SELECT  MAX(S2.rating)
                   FROM  Sailors S2)
```

```
SELECT  AVG (S.age)
FROM  Sailors S
WHERE  S.rating=10
```

```
SELECT  COUNT (DISTINCT S.rating)
FROM  Sailors S
WHERE S.sname='Bob'
```

```
SELECT  AVG (DISTINCT S.age)
FROM  Sailors S
WHERE  S.rating=10
```

AIRPORT

airportcode	name	city	state

FLT-SCHEDULE

flt#	airline	dtime	from-airportcode	atime	to-airportcode	miles	price

FLT-WEEKDAY

flt#	weekday

FLT-INSTANCE

flt#	date	plane#	#avail-seats

AIRPLANE

plane#	plane-type	total-#seats

CUSTOMER

cust#	first	middle	last	phone#	street	city	state	zip

RESERVATION

flt#	date	cust#	seat#	check-in-status	ticket#

Interactive DML - from clause

Q: “Find FLT#, WEEKDAY, and FROM-AIRPORTCODE in FLT-WEEKDAY and FLT-SCHEDULE”

```
SELECT FLT-SCHEDULE.FLT#, WEEKDAY, FROM-AIRPORTCODE
FROM FLT-WEEKDAY FW, FLT-SCHEDULE FS
WHERE FW.FLT# = FS.FLT#;
```

- dot-notation disambiguates FLT# in FLT-WEEKDAY and FLT-SCHEDULE
- this is a **natural join**:

$$\pi_{\text{FLT-SCHEDULE.FLT\#, WEEKDAY, FROM-AIRPORTCODE}} (\text{FLT-SCHEDULE} \bowtie \text{FLT-WEEKDAY})$$

Interactive DML - ordering of rows

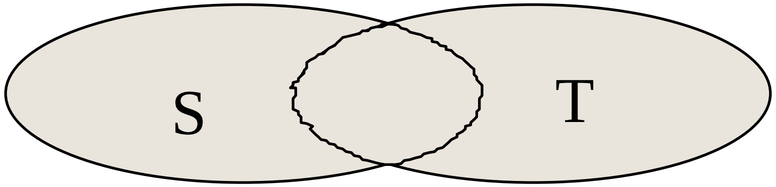
- the **order by** clause orders the rows in a query result in ascending (**asc**) or descending (**desc**) order

Q: *“Find FLT#, airline, and price from FLT-SCHEDULE for flights out of Atlanta ordered by ascending airline and descending price:”*

```
SELECT FLT#, AIRLINE, PRICE
FROM FLT-SCHEDULE
WHERE FROM-AIRPORTCODE=“ATL”
ORDER BY AIRLINE ASC, PRICE DESC;
```

Interactive DML - set operations

S ∪ T



S union T

***Q:** "Find FLT# for flights on Tuesdays in FLT-WEEKDAY or FLT# with more than 100 seats in FLT-INSTANCE "*

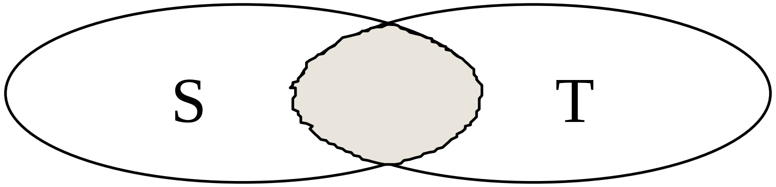
```
SELECT FLT#
FROM FLT-WEEKDAY
WHERE WEEKDAY = "TU"
UNION
SELECT FLT#
FROM FLT-INSTANCE
WHERE #AVAIL-SEATS > 100;
```

- UNION: Can be used to compute the union of any two *union-compatible* sets of tuples (which are themselves the result of SQL queries).

- **UNION ALL** preserves duplicates

Interactive DML - set operation

S ∩ T



S intersect T

Q: "Find FLT# for flights on Tuesdays in FLT-WEEKDAY with more than 100 seats in FLT-INSTANCE"

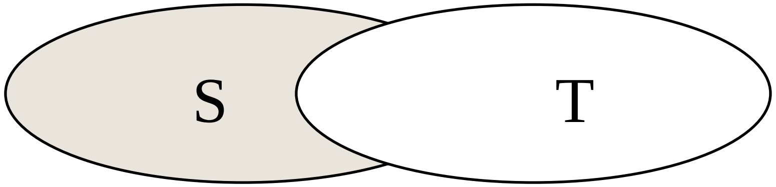
```
SELECT FLT#
FROM FLT-WEEKDAY
WHERE WEEKDAY = "TU"
INTERSECT
SELECT FLT#
FROM FLT-INSTANCE
WHERE #AVAIL-SEATS > 100;
```

- INTERSECT: Can be used to compute the intersection of any two *union-compatible* sets of tuples.
- Included in the SQL/92 standard, but some systems don't support it.

- **INTERSECT ALL** preserves duplicates

Interactive DML - set operation

S - T



S minus T

- “Find FLT# for flights on Tuesdays in FLT-WEEKDAY except FLT# with more than 100 seats in FLT-INSTANCE”

```
SELECT FLT#
FROM FLT-WEEKDAY
WHERE WEEKDAY = “TU”
EXCEPT
SELECT FLT#
FROM FLT-INSTANCE
WHERE #AVAIL-SEATS > 100;
```

- **EXCEPT ALL** preserves duplicates

Interactive DML - nested subqueries

- Set membership: **IN, NOT IN**

Q: *"Find airlines from FLT-SCHEDULE where FLT# is not in the set of FLT#'s for flights on Tuesdays from FLT-WEEKDAY"*

```
SELECT DISTINCT AIRLINE
FROM FLT-SCHEDULE
WHERE FLT# NOT IN
     (SELECT FLT#
      FROM FLT-WEEKDAY
      WHERE WEEKDAY = "TU");
```

Q: "Find FLT#'s for flights on Tuesdays or Thursdays from FLT-WEEKDAY"

```
SELECT DISTINCT FLT#
FROM FLT-WEEKDAY
WHERE WEEKDAY IN ("TU", "TH");
```

Interactive DML - nested subqueries

Q: *"Find FLT# for flights from Atlanta to Chicago with a price so low that there does not exist any cheaper flights from Birmingham to Chicago"*

```
SELECT  S.FLT#
FROM FLT-SCHEDULE S
WHERE S.FROM-AIRPORTCODE="ATL"  AND
            S.TO-AIRPORTCODE="CHI" AND
NOT EXISTS
       (SELECT T.FLT#
        FROM FLT-SCHEDULE T
        WHERE    T.FROM-AIRPORTCODE="BIR" AND
                 T.TO-AIRPORTCODE="CHI" AND
                 T.PRICE < S.PRICE);
```

Interactive DML - nested subqueries

Q: “*Find FLT# for flights from Atlanta to Chicago with a price that is lower than all flights from Birmingham to Chicago*”

```
SELECT  FLT#
FROM FLT-SCHEDULE
WHERE FROM-AIRPORTCODE=“ATL”
            AND TO-AIRPORTCODE=“CHI”
            AND PRICE <  ALL  (SELECT PRICE
                                  FROM FLT-SCHEDULE
                                  WHERE FROM-AIRPORTCODE=“BIR”
                                            AND TO-AIRPORTCODE=“CHI”);
```

Interactive DML - joins

- **cross join**: Cartesian product
- **[inner] join**: only keeps rows that satisfy the join condition
- **left outer join**: keeps all rows from left table; fills in nulls as needed
- **right outer join**: keeps all rows from right table; fills in nulls as needed
- **full outer join**: keeps all rows from both tables; fills in nulls as needed
- **natural** or **on-condition** must be specified for all inner and outer joins
- **natural**: equi-join on columns with same name; one column preserved

Interactive DML - joins

Q: *"Find all two-leg, one-day trips out of Atlanta; show also a leg-one even if there is no connecting leg-two the same day"*

```
SELECT X.FLT# LEG-ONE, Y.FLT# LEG-TWO
FROM  ((FLT-SCHEDULE NATURAL JOIN FLT-INSTANCE) X
        LEFT OUTER JOIN
        (FLT-SCHEDULE NATURAL JOIN FLT-INSTANCE) Y
        ON  (X.TO-AIRPORTCODE = Y.FROM-AIRPORTCODE  AND
        X.DATE = Y.DATE AND X.ATIME < Y.DTIME))
WHERE X.FROM-AIRPORTCODE="ATL";
```

Queries With GROUP BY and HAVING

```
SELECT       [DISTINCT]  target-list
FROM         relation-list
WHERE        qualification
GROUP BY     grouping-list
HAVING       group-qualification
```

- The *target-list* contains (i) attribute names (ii) terms with aggregate operations (e.g., MIN (*S.age*)).
  - The attribute list (i) must be a subset of *grouping-list*. Intuitively, each answer tuple corresponds to a *group*, and these attributes must have a single value per group.
  - A *group* is a set of tuples that have the same value for all attributes in *grouping-list*.

Queries With GROUP BY and HAVING

Conceptual Evaluation:

- The cross-product of *relation-list* is computed, tuples that fail *qualification* are discarded, `*unnecessary*' fields are deleted (projected out), and the remaining tuples are partitioned into groups by the value of attributes in *grouping-list*.
- The *group-qualification* (having clause) is then applied to eliminate some groups.
- In effect, an attribute in *group-qualification* that is not an argument of an aggregate op also appears in *grouping-list*.

Q: *"Find the age of the youngest sailor with age ≥ 18, for each rating with at least 2 such sailors"*

```
SELECT  S.rating,  MIN (S.age)
FROM  Sailors S
WHERE  S.age >= 18
GROUP BY  S.rating
HAVING  COUNT (*) > 1
```

sid	sname	rating	age
22	dustin	7	45.0
31	lubber	8	55.5
71	zorba	10	16.0
64	horatio	7	35.0
29	brutus	1	33.0
58	rusty	10	35.0

- Only S.rating and S.age are mentioned in the SELECT, GROUP BY or HAVING clauses; other attributes `*unnecessary*'.
- 2nd column of result is unnamed.

  (Use AS to name it.)

rating	age
1	33.0
7	45.0
7	35.0
8	55.5
10	35.0

rating	
7	35.0

Answer relation

Q: Find age of the youngest sailor with age ≥ 18, for each rating with at least 2 such sailors.

rating	age
7	45.0
1	33.0
8	55.5
8	25.5
10	35.0
7	35.0
10	16.0
9	35.0
3	25.5
3	63.5
3	25.5

rating	age
1	33.0
3	25.5
3	63.5
3	25.5
7	45.0
7	35.0
8	55.5
8	25.5
9	35.0
10	35.0

rating	minage
3	25.5
7	35.0
8	25.5

Interactive DML - built-in functions

Q: *“Find the average ticket price by airline for scheduled flights out of Atlanta for airlines with more than 5 scheduled flights out of Atlanta from FLT-SCHEDULE”*

```
SELECT AIRLINE, AVG(PRICE)
FROM FLT-SCHEDULE
WHERE FROM-AIRPORTCODE = “ATL”
GROUP BY AIRLINE
HAVING COUNT (FLT#) >= 5;
```

Interactive DML - insert, **delete, update**

```
INSERT INTO FLT-SCHEDULE
VALUES ("DL212", "DELTA", 11-15-00, "ATL",
              13-05-00, "CHI", 650, 00351.00);

INSERT INTO FLT-SCHEDULE(FLT#,AIRLINE)
VALUES ("DL212", "DELTA");   /*default nulls added*/
```

Q: "*Insert into FLT-INSTANCE all flights scheduled for Thursday, 9/10/98*"

```
INSERT INTO FLT-INSTANCE(FLT#, DATE)
   (SELECT S.FLT#, 1998-09-10
    FROM FLT-SCHEDULE S, FLT-WEEKDAY D
    WHERE S.FLT#=D.FLT#
    AND D.WEEKDAY="TH");
```

Interactive DML - **insert,** delete, **update**

Q: *“Cancel all flight instances for Delta on 9/10/98”*

```
DELETE FROM FLT-INSTANCE
WHERE DATE=1998-09-10
AND FLT# IN
    (SELECT FLT#
    FROM FLT-SCHEDULE
    WHERE AIRLINE=“DELTA”);
```

Interactive DML - **insert, delete,** update

Q: *“Update all reservations for customers on DL212 on 9/10/98 to reservations on AA121 on 9/10/98”*

```
UPDATE RESERVATION
SET FLT#=“AA121”
WHERE DATE=1998-09-10
        AND FLT#=“DL212”;
```