BIRZEIT UNIVERSITY

Faculty of Engineering and Tecnology

Computer Science Department

SQL

Chapter 5

SQL – Introduction

- Standard DML/DDL for relational DB's
  - DML = "Data Manipulation Language" (queries, updates)
  - DDL = "Data Definition Language" (create tables, indexes, …)
- Also includes:
  - view definition
  - security
  - integrity constraints
  - transactions
- History:
  - System R project at IBM: "SEQUEL"
  - later, becomes standard: Structured Query Language

Banking Example

branch (branch-name, branch-city, assets)

customer (customer-name, customer-street, customer-other)

account (account-number, branch-name, balance)

loan (loan-number, branch-name, amount)

depositor (customer-name, account-number)

borrower (customer-name, loan-number)

A Simple SELECT-FROM-WHERE Query

BIRZEIT UNIVERSITY

bname	lno	amt
Downtown	L-170	3000
Redwood	L-230	4000
Perry	L-260	1700
Redwood	L-450	3000

```
SELECT  bname
FROM    loan
WHERE   amt > 1000
```

Similar to:

$$\pi_{\text{bname}} (\sigma_{\text{amt}>1000} (\text{loan}))$$

But not quite….

bname
Downtown
Redwood
Perry
Redwood

Duplicates are retained, i.e. result not a set

Why preserve duplicates?

- eliminating them is costly
- often, users don't care
- can also write:

```
SELECT DISTINCT bname
FROM    loan
WHERE   amt> 1000
```

Another query

```
SELECT cname, balance
FROM   depositor, account
WHERE  depositor.acct_no = account.acct_no
```

depositor (customer-name, account-number)
account (account-number, branch-name, balance)

Similar to :

$$\pi_{\text{cname, balance}} (\text{depositor} \bowtie \text{account})$$

cname	balance
Johnson	500
Smith	400
Turner	350
Smith	300
Jones	240
Smith	300

Note: you can also write

```
SELECT cname, balance
FROM   depositor  AS d, account AS a
WHERE  d.acct_no = a.acct_no
```

In general

SELECT A1, A2, …, An

FROM r1, r2, …, rm

[WHERE P]

- WHERE clause optional (missing WHERE clause means WHERE is true)

Conceptual Algorithm:

1. FROM clause: cartesian product (X)
   - t1 ← r1 x r2 x … x rm
2. WHERE clause: selection (s)
   - $t2 \leftarrow \sigma_P (t1)$
3. SELECT clause: projection (p)
   - $result \leftarrow \pi_{A1, A2, \ldots, An} (t2)$

Note: will never be implemented with product (X) !

The SELECT clause

equivalent to projection, despite name

can use "*" to get all attributes

e.g.,

```
SELECT *
FROM   loan
```

can write SELECT DISTINCT to eliminate duplicates

can write SELECT ALL to preserve duplicate (default)

can include arithmetic expressions

e.g., SELECT bname, acct_no, balance*1.05
FROM account

The WHERE clause

- equivalent to selection, despite name…

- WHERE predicate can be:
  - Simple:

    attribute relop attribute or constant

    (relop: <,>,=,£, ³, ¹)
  - Complex: using AND, OR, NOT, BETWEEN
- e.g.

```
SELECT lno
FROM   loan
WHERE  amt BETWEEN
                9000 AND  10000
```

```
SELECT lno
FROM   loan
WHERE  amt >= 9000 AND  amt <= 10000
```

Formal Semantics of SQL

- RA can only express SELECT DISTINCT queries
- to express SQL, must extend to a bag algebra ,

  a bag (aka: multiset) like sets, but can have duplicates

e.g. { 4, 5, 4, 6}

e.g.

balances =

cname	balance
Johnson	500
Smith	400
Turner	350
Smith	300
Jones	240
Smith	300

The FROM clause

- Equivalent to cartesian product (X)

(or ⋈ depending on the WHERE clause)

- binds tuples in relations to variable names
- e.g.: FROM borrower, loan
  - computes: borrower x loan
  - identifies borrower, loan columns (attrs) in the results

e.g. allowing one to write:

WHERE borrower.lno = loan.lno

```
FROM   borrower  b,  loan  l
WHERE  b.lno = l.lno
```

- Simplifies the expression
- Needed for self-joins

More SQL: Range variables

1. Using AS in FROM clause

   introduces tuple variables

   e.g.:

```
SELET DISTINCT T.bname
FROM  branch AS T, branch AS S
WHERE T.assets > S.assets
```

2. Using AS in SELECT clause

   renames columns in result (ρ)

   e.g.:

```
SELET bname, acct_no, balance*1.05 AS newbal
FROM  account
```

bname	acct_no	newbal
Downtown	A-170	450
Redwood	A-230	400

SQL - String Operations

- SQL includes a string-matching operator
- percent (**%**): zero or more characters.
- underscore (**_**): exactly one character.
- E.g. Find the names of all customers whose street includes the substring "Main".

```
SELECT customer-name
FROM   customer
WHERE cstreet LIKE '%Main%'
```

- E.g. Find the names of all customers whose names begin with A and end with b and consists at least three characters.

```
SELECT customer-name
FROM   customer
WHERE customer-name  LIKE 'A_%B'
```

More SQL: Set operations

Example Queries:

(SELECT cname FROM depositor)

?

(SELECT cname FROM borrower)

? = UNION

- returns names of customers with saving accts, loans, or both

? = INTERSECT

- returns names of customers with saving accts AND loans

? = EXCEPT

- returns names of customers with saving accts but NOT loans

Order by

Example: List in alphabetical order, the names of all customers with loans at Kenmore branch:

```
SELECT   DISTINCT cname
FROM     borrower b, loan l
WHERE    b.lno = l.lno  AND  bname = "Kenmore"
ORDER  BY  cname
```

Result:

cname
Adams
Byers
Smith
.....

can also write:

ORDER BY cname DESC or
ORDER BY cname ASC (default)

like SELECT DISTINCT, very expensive...

Aggregate Operators

- Aggregate Operators:

AVG (col): average of values in col
MIN (col) : minimum value in col
MAX (col): maximum value in col
SUM (col): sum of values in col
COUNT (col): number of values in col

Examples:

1. Find the average acct balance @ Perry:

account (acct_no, bname, bal)

```
SELECT AVG (bal)
FROM   account
WHERE  bname = "Perry"
```

2. Find the number of tuples in customer:

```
SELECT COUNT(*)
FROM  customer
```

3. Find the number of depositors

```
SELECT COUNT( DISTINCT cname)
FROM   depositor
```

COUNT, SUM, AVG have a DISTINCT version

Aggregates and Group By

- Usually, aggregates used with “Group By”
- E.g.

```
SELECT    bname, COUNT (DISTINCT cname)
FROM      depositor d, account  a
WHERE     d.acct_no = a.acct_no
GROUP BY bname
```

depositor (customer-name, account-number)
account (account-number, branch-name, balance)

Result:

bname	COUNT
Downtown	2
Mianus	5
Perry	1
Brighton	5
Kenmore	7

Aggregates and Group By

Intuition behind "Group By"

```
SELECT    bname, COUNT (DISTINCT cname)
FROM      depositor d, account  a
WHERE     d.acct_no = a.acct_no
GROUP BY  bname
```

Step 1: "Group " results of join

bname	a.acct_no	balance	cname	d.acct_no
Downtown	A-101	500	Johnson	A-101
Mianus	A-215	700	Smith	A-215
Perry	A-102	400	Hayes	A-102
Brighton	A-202	900	Johnson	A-202
Brighton	A-217	800	Jones	A-217
Kenmore	A-305	700	Smith	A-305
Kenmore	A-232	600	Lindsay	A-232

Step 2: aggregate on groups and project on result

bname	COUNT
Downtown	1
Mianus	1
Perry	1
Brighton	2
Kenmore	2

Group By

- Another example:

branch(bname, bcity, assets)

Result ??

bname	total
Redwood	2.1M
Pownal	0.3M
N. Town	3.7M
????	2M
?????	10.1M

```
SELECT    bname, SUM(assets) as total
FROM      branch
GROUP BY  bcity
```

Above query is NOT allowed

Non-aggregated values in SELECT clause (e.g., bname) must also appear in GROUP BY clause

```
SELECT    A1, A2, ..., Ak, Agg1(), ...., Aggi()
FROM      .........
WHERE     ...........
GROUP BY A1, A2, ..., Ak, Ak+1, ..., An
```

HAVING

WHERE :: FROM as HAVING :: GROUP BY

- HAVING P: selects rows from result of GROUP BY
- Optional (missing HAVING means TRUE)

Example: Find names of branches and the average account balance for those branches having average account balance > $1200

```
SELECT  bname, AVG(balance) AS avg
FROM    account
GROUP BY bname
HAVING  avg > 1200
```

NULLs

can be a value for any attribute

e.g. :

branch2=

bname	bcity	assets
Downtown	Boston	9M
Perry	Horse	1.7M
Mianus	Horse	.4M
Kenmore	Boston	NULL

What does this mean?

- We don't know Kenmore's assets?
- Kenmore has no assets?
- .....

Effect on Queries:

```
SELECT * FROM branch2
WHERE assets = NULL
```

bname	bcity	assets

```
SELECT * FROM branch2
WHERE assets  IS NULL
```

bname	bcity	assets
Kenmore	Boston	NULL

NULLs

- Arithmetic with nulls:
  - n op null = null

    op : + , - , *, /

SELECT .....
FROM .....
WHERE boolexpr IS UNKNOWN

- Booleans with nulls: One can write:

3-valued logic (true, false, unknown)

What expressions evaluate to UNKNOWN?

1. Comparisons with NULL (e.g. assets = NULL)
2. FALSE OR UNKNOWN (but: TRUE OR UNKNOWN = TRUE)
3. TRUE AND UNKNOWN
4. UNKNOWN AND/OR UNKNOWN

NULLs

Given:

branch2=

bname	bcity	assets
Downtown	Boston	9M
Perry	Horse	1.7M
Mianus	Horse	.4M
Kenmore	Boston	NULL

Aggregate operations:

```
SELECT SUM(assets)
FROM   branch2
```

returns

```
SUM
--------
11.1M
```

NULL is ignored
Same for AVG, MIN, MAX

But.... COUNT(assets) retunrs 4!

Let branch3 an empty relation
Then: SELECT SUM(assets)
FROM branch3 returns NULL

but COUNT(<empty rel>) = 0

Review - Summary

account (acct_no, bname, balance)
branch(bname, bcity, assets)

```
SELECT    bcity, sum(balance) AS totalbalance
FROM      branch b, account a
WHERE     b.bname=a.bname  AND assets >= 1M
GROUP BY  bcity
HAVING    totalbalance > 700
ORDER BY  bcity DESC
```

Steps 1,2 : FROM, WHERE

b.bname	bcity	assets	a.bname	acct_no	balance
Downtown	Bkln	9M	Downtown	A-101	500
Redwood	Palo Alto	2.1M	Redwood	A-215	700
Perry	Horse	1.7M	Perry	A-102	400
RH	Horse	8M	RH	A-202	350
Brighton	Bkln	7.1M	Brighton	A-305	900
Brighton	Bkln	7.1M	Brighton	A-217	750

Summary thus far

Steps 3,4 : GROUP BY, SELECT

bcity	totalbalance
Bkln	2150
Palo Alto	700
Horse	750

Steps 5 : HAVING

bcity	totalbalance
Bkln	2150
Horse	750

Steps 6 : ORDER BY

bcity	totalbalance
Horse	750
Bkln	2150

Summary thus far

Clause	Evaluation Order	Semantics (RA)
SELECT[DISTINCT]	4	π
FROM	1	X
WHERE	2	σ
INTO	7	$\leftarrow$
GROUP BY	3	Can't express
HAVING	5	σ
ORDER BY	6	Can't express

Find sailors who've reserved at least one boat

```
SELECT  S.sid
FROM  Sailors S, Reserves R
WHERE  S.sid=R.sid
```

- **What is the effect of replacing *S.sid* by *S.sname* in the SELECT clause?**
  - Would adding DISTINCT to this variant of the query make a difference?

Find sid's of sailors who've reserved a red or a green boat

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

```
SELECT R.sid
   FROM Boats B,Reserves R
WHERE R.bid=B.bid AND
(B.color='red' OR B.color='green')
```

Vs.

```
SELECT  R.sid
   FROM Boats B, Reserves R
WHERE R.bid=B.bid AND B.color='red'
UNION
SELECT R.sid
   FROM Boats B, Reserves R
WHERE R.bid=B.bid AND B.color='green'
```

Find sid's of sailors who've reserved a red but did not reserve a green boat

```
SELECT S.sid
FROM Sailors S, Boats B,
     Reserves R
WHERE S.sid=R.sid
      AND R.bid=B.bid
      AND B.color='red'
EXCEPT
SELECT S.sid
FROM Sailors S, Boats B,
     Reserves R
WHERE S.sid=R.sid
     AND R.bid=B.bid
     AND B.color='green'
```

Find the sailors who are older than any sailor named Ahmad ordered in descending

```
Select *
From Sailor s
Where s.age > (     select s2.age
                    from sailor s2
                    where s2.sname LIKE 'Ahmad%')
ORDER BY sname DESC, age ASC
```

- *Find the average age of sailors with a rating of 10.*
-

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

Find the name and age of the oldest sailor

- SELECT S.sname, MAX (S.age)
  FROM Sailors S
- SELECT S.sname, S.age
  FROM Sailors S
  WHERE S.age = (SELECT MAX (S2.age)
  FROM Sailors S2)
- If the SELECT clause uses aggregate operators, then it must use only aggregate operators unless the query contains a GROUP BY clause

Find the names of sailors who are older than the oldest sailor with a rating of 10.

```
SELECT S.sname
FROM Sailors S
WHERE S.age > ( SELECT MAX ( S2.age ) FROM Sailors S2
                WHERE S2.rating = 10 )
```

Find the age of the youngest sailor for each rating level

- SELECT s.rating, MIN(s.age)
- FROM sailors s
- GROUP BY s.rating

Find the age of the youngest sailor who is eligible to vote (i.e., is at least 18 years old) for each rating level with at least two such sailors.

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

How this query evaluated

1. Cross-product for tables in from-list
2. Apply WHERE
3. Eliminate unwanted columns not mentioned in select-list
4. Sort the table according to GROUP BY clause
5. Apply HAVING
6. Apply aggregate operators

Find number of reservations for each red boat

```
SELECT    B.bid, COUNT (*) AS reservationcount
FROM      Boats B, Reserves R
WHERE     R.bid = B.bid AND B.color = 'red'
GROUP BY  B.bid
```

```
SELECT    [illegible] AS reservationcount
FROM      [illegible] R
WHERE     [illegible]
GROUP BY  [illegible]
HAVING    [illegible]
```

Compute increments for the ratings of sailors who have sailed two different boats on the same day

```
SELECT  S.sname, S.rating+1 AS rating
FROM    Sailors S, Reserves R1, Reserves R2
WHERE   S.sid = R1.sid AND S.sid = R2.sid
        AND R1.day = R2.day AND R1.bid <> R2.bid
```

Nested Query

- A nested query is a query that has another query embedded within it
- the embedded query is called a subquery.
- The embedded query can of course be a nested query itself
- thus queries that have very deeply nested structures are possible.
- A subquery typically appears within the **WHERE** clause , **FROM** clause or the **HAVING** clause

Find the names of sailors who have reserved boat 103

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

IN & NOT IN

- IN : operator allows us to test whether a value is in a given set of elements
- NOT IN : operator allows us to test whether a value is not in a given set of elements
- Subquery is evaluated for each row in the outer query.

Find the names of sailors who have reserved a red boat.

```
SELECT  S.sname
FROM    Sailors S
WHERE   S.sid IN ( SELECT  R.sid
                   FROM    Reserves R
                   WHERE   R.bid IN (SELECT  B.bid
                                     FROM    Boats B
                                     WHERE   B.color = 'red'
```

What???

```
SELECT  S.sname
FROM    Sailors S
WHERE   S.sid NOT IN ( SELECT  R.sid
                       FROM    Reserves R
                       WHERE   R.bid IN ( SELECT  B.bid
                                          FROM    Boats B
                                          WHERE   B.color = 'red' )
```

Correlated Nested Queries

- the inner subquery could depend on the row currently being examined in the outer query
- *Find the names of sailors who have reserved boat 103*

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

EXIST & NOT EXIST

- EXIST: test whether the set **is non** empty
- NOT EXIST: test whether the set **is empty**

Find the names of sailors who have reserved all boats

```
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 ))
```

Set-Comparison:

- ANY and ALL
- operators {<, <=, =, <>, >=, >} ANY, ALL
- *Find sailors whose rating is better than some sailor called Horatio*

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

- The name of sailor who is older than the oldest sailor with rating is 10.

```
SELECT S.sname
FROM   Sailors S
WHERE  S.age > ALL ( SELECT S2.age
                     FROM   Sailors S2
                     WHERE  S2.rating = 10 )
```

What is the output???

- The rating with minimum average age.

```
SELECT  Temp.rating, Temp.avgage
FROM    ( SELECT    S.rating, AVG  (S.age) AS  avgage,
          FROM      Sailors S
          GROUP  BY S.rating) AS  Temp
WHERE   Temp.avgage = ( SELECT  MIN  (Temp.avgage) FROM  Temp)
```