

Chapter 3,4: SQL (Part 2)

INFSCI1022: Database Management Systems
Textbook: Database System Concepts - 6th Edition, 2010

Vladimir Zadorozhny, GIST, University of Pittsburgh

1

Basic Query Structure

- SQL is based on set and relational operations with certain modifications and enhancements
- A typical SQL query has the form:

```
select  $A_1, A_2, \dots, A_n$   
from  $r_1, r_2, \dots, r_m$   
where  $P$ 
```

- A_i represents an attribute
 - r_i represents a relation
 - P is a predicate.
- The result of an SQL query is a relation.

2

The select Clause

- The **select** clause list the attributes desired in the result of a query
 - corresponds to the projection operation of the relational algebra
- Find the names of all branches in the *loan* relation:
select branch_name
from loan
- NOTE: SQL names are case insensitive (i.e., you may use upper- or lower-case letters.)

loan

<i>loan_number</i>	<i>branch_name</i>	amount
L-170	Downtown	3000
L-230	Redwood	4000
L-260	Perryridge	1700

3

The select Clause (Cont.)

- SQL allows duplicates in relations as well as in query results.
- To force the elimination of duplicates, insert the keyword **distinct** after select.
- Find the names of all branches in the *loan* relations, and remove duplicates

select distinct branch_name
from loan

- The keyword **all** specifies that duplicates not be removed.

select all branch_name
from loan

loan

<i>loan_number</i>	<i>branch_name</i>	amount
L-170	Downtown	3000
L-230	Redwood	4000
L-260	Perryridge	1700
L-300	Perryridge	2000

4

The select Clause (Cont.)

- An asterisk in the select clause denotes “all attributes”

```
select *  
from loan
```

- The **select** clause can contain arithmetic expressions involving the operation, +, -, *, and /, and operating on constants or attributes of tuples.
- The query:

```
select loan_number, branch_name, amount * 100  
from loan
```

would return a relation that is the same as the *loan* relation, except that the value of the attribute *amount* is multiplied by 100.

loan

<i>loan_number</i>	<i>branch_name</i>	<i>amount</i>
L-170	Downtown	3000
L-230	Redwood	4000
L-260	Perryridge	1700

5

The where Clause

- The **where** clause specifies conditions that the result must satisfy
- To find all loan number for loans made at the Perryridge branch with loan amounts greater than \$1200.

```
select loan_number  
from loan  
where branch_name = 'Perryridge' and amount > 1200
```

- Comparison results can be combined using the logical connectives **and**, **or**, and **not**.

loan

<i>loan_number</i>	<i>branch_name</i>	<i>amount</i>
L-170	Downtown	3000
L-230	Redwood	4000
L-260	Perryridge	1700
L-300	Perryridge	2000

6

The where Clause (Cont.)

- SQL includes a **between** comparison operator
- Example: Find the loan number of those loans with loan amounts between \$2000 and \$3000 (that is, \geq \$2000 and \leq \$3000)

```
select loan_number
from loan
where amount between 2000 and 100000
```

loan

<i>loan_number</i>	<i>branch_name</i>	<i>amount</i>
L-170	Downtown	3000
L-230	Redwood	4000
L-260	Perryridge	1700
L-300	Perryridge	2000

7

Textbook: Database System Concepts – 6th Edition, 2010

V.Zadorozhny

The from Clause, Cartesian Product, and Natural Join

- The **from** clause lists the relations involved in the query
- Find the all pair-wise tuple combinations of *borrower* and *loan* (**Cartesian Product** operation)

```
select *
from borrower, loan
```

- Find the name, loan number, branch name and loan amount of all customers (**Natural Join** operation).

```
select customer_name, borrower.loan_number, branch_name, amount
from borrower, loan
where borrower.loan_number = loan.loan_number and
branch_name = 'Perryridge'
```

borrower

<i>customer_name</i>	<i>loan_number</i>
Smith	L-230
Jones	L-170

loan

<i>loan_number</i>	<i>branch_name</i>	<i>amount</i>
L-170	Downtown	3000
L-230	Redwood	4000

8

Textbook: Database System Concepts – 6th Edition, 2010

V.Zadorozhny

Another Example of Natural Join

- Find the name, loan number and loan amount of all customers having a loan at the Perryridge branch.

```
select customer_name, borrower.loan_number, amount
from borrower, loan
where borrower.loan_number = loan.loan_number and
      branch_name = 'Perryridge'
```

borrower

<i>customer_name</i>	<i>loan_number</i>
Smith	L-230
Jones	L-170

loan

<i>loan_number</i>	<i>branch_name</i>	<i>amount</i>
L-170	Downtown	3000
L-230	Redwood	4000

9

The Rename Operation

- The SQL allows renaming relations and attributes using the **as** clause:
old-name as new-name
- Find the name, loan number and loan amount of all customers; rename the column name *loan_number* as *loan_id*.

```
select customer_name, borrower.loan_number as loan_id, amount
from borrower, loan
where borrower.loan_number = loan.loan_number
```

borrower

<i>customer_name</i>	<i>loan_number</i>
Smith	L-230
Jones	L-170

loan

<i>loan_number</i>	<i>branch_name</i>	<i>amount</i>
L-170	Downtown	3000
L-230	Redwood	4000

10

Tuple Variables

- Tuple variables are defined in the **from** clause via the use of the **as** clause.
- Find the customer names and their loan numbers for all customers having a loan at some branch.

```

select customer_name, T.loan_number, S.amount
from borrower as T, loan as S
where T.loan_number = S.loan_number
    
```

borrower

<i>customer_name</i>	<i>loan_number</i>
Smith	L-230
Jones	L-170

loan

<i>loan_number</i>	<i>branch_name</i>	<i>amount</i>
L-170	Downtown	3000
L-230	Redwood	4000

11

String Operations

customer

<i>customer_name</i>	<i>customer_street</i>	<i>customer_city</i>
Adams	Spring Str	Pittsfield
Hayes	Main Ave	Harrison

- SQL includes a string-matching operator for comparisons on character strings. The operator “like” uses patterns that are described using two special characters:
 - percent (%). The % character matches any substring.
 - underscore (_). The _ character matches any character.
- Find the names of all customers whose street includes the substring “Main”.

```

select customer_name
from customer
where customer_street like '%Main%'
    
```

- Match the name “Main%”


```

like 'Main\%' escape '\'
```
- SQL supports a variety of string operations such as
 - concatenation (using “||”)
 - converting from upper to lower case (and vice versa)
 - finding string length, extracting substrings, etc.

12

Ordering the Display of Tuples

- List in alphabetic order the names of all customers having a loan in Perryridge branch
select distinct *customer_name*
from *borrower*
order by *customer_name*
- We may specify **desc** for descending order or **asc** for ascending order, for each attribute; ascending order is the default.
 - Example: **order by** *customer_name* **desc**

borrower

<i>customer_name</i>	<i>loan_number</i>
Smith	L-230
Jones	L-170
Smith	L-200

13

Set Operations

- Find all customers who have a loan, an account, or both:
(select *customer_name* **from** *depositor*)
union
(select *customer_name* **from** *borrower*)
- Find all customers who have both a loan and an account.
(select *customer_name* **from** *depositor*)
intersect
(select *customer_name* **from** *borrower*)
- Find all customers who have an account but no loan.
(select *customer_name* **from** *depositor*)
except
(select *customer_name* **from** *borrower*)

depositor

<i>customer_name</i>	<i>account_number</i>
Adams	A-100
Jones	A-200
Smith	A-300

borrower

<i>customer_name</i>	<i>loan_number</i>
Smith	L-230
Jones	L-170
Smith	L-200

14

Aggregate Functions

- These functions operate on the values of a column of a relation, and return a value

avg: average value

min: minimum value

max: maximum value

sum: sum of values

count: number of values

15

Aggregate Functions (Cont.)

- Find the average account balance at the Perryridge branch.

```
select avg (balance)
from account
where branch_name = 'Perryridge'
```

account:

<i>account_number</i>	<i>branch_name</i>	<i>balance</i>
A-101	Downtown	500
A-215	Brighton	700
A-102	Perryridge	400
A-201	Brighton	900
A-222	Perryridge	700
A-217	Brighton	500

16

Aggregate Functions (Cont.)

- Find the number of tuples in the *customer* relation.

```
select count (*)  
  from customer
```

customer

<i>customer_name</i>	<i>customer_street</i>	<i>customer_city</i>
Adams	Spring Str	Pittsfield
Hayes	Main Ave	Harrison

- Find the number of depositors in the bank.

```
select count (distinct customer_name)  
  from depositor
```

depositor

<i>customer_name</i>	<i>account_number</i>
Smith	A-100
Jones	A-200
Smith	A-300

17

Aggregate Functions – Group By

- Find the average balance for each branch.

```
select branch_name, avg (balance)  
  from account  
  group by branch_name
```

Note: Attributes in **select** clause outside of aggregate functions must appear in **group by** list

account:

<i>account_number</i>	<i>branch_name</i>	<i>balance</i>
A-101	Downtown	500
A-215	Brighton	700
A-102	Perryridge	400
A-201	Brighton	900
A-222	Perryridge	700
A-217	Brighton	500

18

Aggregate Functions – Having Clause

- Find the names of all branches where the average account balance is more than \$600.

```
select branch_name, avg (balance)
from account
group by branch_name
having avg (balance) > 600
```

Note: predicates in the **having** clause are applied after the formation of groups whereas predicates in the **where** clause are applied before forming groups

account:

account_number	branch_name	balance
A-101	Downtown	500
A-215	Brighton	700
A-102	Perryridge	400
A-201	Brighton	900
A-222	Perryridge	700
A-217	Brighton	500

```
select branch_name, avg (balance)
from account
where branch_name <> 'Brighton'
group by branch_name
having avg (balance) > 600
```

19

Null Values

- It is possible for tuples to have a null value, denoted by *null*, for some of their attributes
- null* signifies an unknown value or that a value does not exist.
- The predicate **is null** can be used to check for null values.

- Example: Find all loan number which appear in the *loan* relation with null values for *amount*.

```
select loan_number
from loan
where amount is null
```

- The result of any arithmetic expression involving *null* is *null*
 - Example: $5 + \text{null}$ returns null
- However, aggregate functions simply ignore nulls
 - More on next slide

20

Null Values and Three Valued Logic

- Any comparison with *null* returns *unknown*
 - Example: $5 < null$ or $null <> null$ or $null = null$
- Three-valued logic using the truth value *unknown*:
 - OR: $(unknown \text{ or } true) = true$, $(unknown \text{ or } false) = unknown$
 $(unknown \text{ or } unknown) = unknown$
 - AND: $(true \text{ and } unknown) = unknown$, $(false \text{ and } unknown) = false$,
 $(unknown \text{ and } unknown) = unknown$
 - NOT: $(\text{not } unknown) = unknown$
 - “*P* is unknown” evaluates to true if predicate *P* evaluates to *unknown*
- Result of **where** clause predicate is treated as *false* if it evaluates to *unknown*

21

Null Values and Aggregates

- Total all loan amounts

```
select sum (amount)
from loan
```

 - Above statement ignores null amounts
 - Result is *null* if there is no non-null amount
- All aggregate operations except **count(*)** ignore tuples with null values on the aggregated attributes.

22

Nested Subqueries

- SQL provides a mechanism for the nesting of subqueries.
- A subquery is a **select-from-where** expression that is nested within another query.
- A common use of subqueries is to perform tests for set membership, set comparisons, and set cardinality.

23

Example Query

- Find all customers who have both an account and a loan at the bank.

```
select distinct customer_name
from borrower
where customer_name in (select customer_name
from depositor)
```

- Find all customers who have a loan at the bank but do not have an account at the bank

```
select distinct customer_name
from borrower
where customer_name not in (select customer_name
from depositor)
```

depositor

<i>customer_name</i>	<i>account number</i>
Adams	A-100
Jones	A-200
Smith	A-300

borrower

<i>customer_name</i>	<i>loan number</i>
Smith	L-230
Jones	L-170
Smith	L-200

24

Set Comparison

- Find all branches that have greater assets than some branch located in Brooklyn.

```

select branch_name
   from branch
  where assets > some
        (select assets
         from branch
         where branch_city = 'Brooklyn')
    
```

branch

<i>branch_name</i>	<i>branch_city</i>	<i>assets</i>
Downtown	Brooklyn	550000
Mianus	Horseneck	750000
Perryridge	Horseneck	400000
Round Hill	Horseneck	350000
Brighton	Brooklyn	900000
Perryridge	Brooklyn	700000
Redwood	Palo Alto	750000

25

Definition of Some Clause

$(5 < \text{some } \begin{matrix} 0 \\ 5 \\ 6 \end{matrix}) = \text{true}$ (read: 5 < some tuple in the relation)

$(5 < \text{some } \begin{matrix} 0 \\ 5 \end{matrix}) = \text{false}$

$(5 = \text{some } \begin{matrix} 0 \\ 5 \end{matrix}) = \text{true}$

$(5 \neq \text{some } \begin{matrix} 0 \\ 5 \end{matrix}) = \text{true}$ (since $0 \neq 5$)

26

Example Query

- Find the names of all branches that have greater assets than all branches located in Brooklyn.

```

select branch_name
  from branch
  where assets > all
        (select assets
         from branch
         where branch_city = 'Brooklyn')
    
```

branch

<i>branch_name</i>	<i>branch_city</i>	<i>assets</i>
Downtown	Brooklyn	550000
Mianus	Horseneck	750000
Perryridge	Horseneck	400000
Round Hill	Horseneck	350000
Brighton	Brooklyn	900000
Perryridge	Brooklyn	700000
Redwood	Palo Alto	750000

27

Definition of all Clause

$(5 < \text{all } \begin{matrix} 0 \\ 5 \\ 6 \end{matrix}) = \text{false}$

$(5 < \text{all } \begin{matrix} 6 \\ 10 \end{matrix}) = \text{true}$

$(5 = \text{all } \begin{matrix} 4 \\ 5 \end{matrix}) = \text{false}$

$(5 \neq \text{all } \begin{matrix} 4 \\ 6 \end{matrix}) = \text{true (since } 5 \neq 4 \text{ and } 5 \neq 6)$

28

Grouping, Aggregating and Subqueries

- Find the branches with the largest average account balance

```
select branch_name
from account
group by branch_name
having avg(balance) >= all (select avg(balance)
                           from account
                           group by branch_name )
```

account

<i>account_number</i>	<i>branch_name</i>	balance
A-101	Downtown	500
A-215	Downtown	700
A-102	Perryridge	1400
A-305	Downtown	300
A-201	Brighton	1900
A-222	Perryridge	1700
A-217	Brighton	800

29

Derived Relations

- SQL allows a subquery expression to be used in the **from** clause
- Find the average account balance of those branches where the average account balance is greater than \$1200.

```
select branch_name, avg_balance
from (select branch_name, avg (balance)
      from account
      group by branch_name )
as branch_avg ( branch_name, avg_balance )
where avg_balance > 1200
```

Note that we do not need to use the **having** clause, since we compute the temporary (view) relation *branch_avg* in the **from** clause, and the attributes of *branch_avg* can be used directly in the **where** clause.

account:

<i>account_number</i>	<i>branch_name</i>	balance
A-101	Downtown	500
A-215	Downtown	700
A-102	Perryridge	1400
A-305	Downtown	300
A-201	Brighton	1900
A-222	Perryridge	1700
A-217	Brighton	800

30

Views

- Consider a person who needs to know a customer's loan number but has no need to see the loan amount. This person should see a relation described, in SQL, by

```
(select customer_name, loan_number
  from borrower, loan
  where borrower.loan_number = loan.loan_number)
```

- A **view** provides a mechanism to hide certain data from the view of certain users.

```
create view v as(select customer_name, loan_number
  from borrower, loan
  where borrower.loan_number = loan.loan_number)
```

borrower

<i>customer_name</i>	<i>loan_number</i>
Smith	L-230
Jones	L-170

loan

<i>loan_number</i>	<i>branch_name</i>	<i>amount</i>
L-170	Downtown	3000
L-230	Redwood	4000

31

View Definition

- A view is defined using the **create view** statement which has the form

```
create view v as < query expression >
```

where <query expression> is any legal SQL expression. The view name is represented by *v*.

- Once a view is defined, the view name can be used to refer to the virtual relation that the view generates.
- View definition is not the same as creating a new relation by evaluating the query expression
 - Rather, a view definition causes the saving of an expression; the expression is substituted into queries using the view.

32

Example Queries

- A view consisting of branches and their customers

```

create view all_customer as
  (select branch_name, customer_name
   from depositor, account
   where depositor.account_number =
         account.account_number)
union
  (select branch_name, customer_name
   from borrower, loan
   where borrower.loan_number = loan.loan_number)
  
```

- Find all customers of the Perryridge branch

```

select customer_name
from all_customer
where branch_name = 'Perryridge'
  
```

33

More on Data Manipulation Language

- Delete all accounts at every branch located in the city 'Brooklyn'.

```

delete from account
where branch_name in (select branch_name
                      from branch
                      where branch_city = 'Brooklyn')
  
```

account

<i>account_number</i>	<i>branch_name</i>	<i>balance</i>
A-101	Downtown	500
A-215	Downtown	700
A-102	Perryridge	1400
A-305	Downtown	350
A-201	Brighton	1900
A-222	Perryridge	1700
A-217	Brighton	800

branch

<i>branch_name</i>	<i>branch_city</i>	<i>assets</i>
Downtown	Brooklyn	550000
Mianus	Horseneck	750000
Perryridge	Horseneck	400000
Round Hill	Horseneck	350000
Brighton	Brooklyn	900000
Perryridge	Brooklyn	700000
Redwood	Palo Alto	750000

Example Query

- Delete the record of all accounts with balances below the average at the bank.

```
delete from account
where balance < (select avg (balance )
from account )
```

- Problem: as we delete tuples from deposit, the average balance changes
- Solution used in SQL:
 1. First, compute **avg** balance and find all tuples to delete
 2. Next, delete all tuples found above (without recomputing **avg** or retesting the tuples)

account

<i>account_number</i>	<i>branch_name</i>	<i>balance</i>
A-101	Downtown	100
A-215	Downtown	110
A-102	Perryridge	120

35

More on Data Manipulation Language

- Provide as a gift for all loan customers of the Perryridge branch, a \$200 savings account. Let the loan number serve as the account number for the new savings account

```
insert into account
select loan_number, branch_name, 200
from loan
where branch_name = 'Perryridge'
```

```
insert into depositor
select customer_name, loan_number
from loan, borrower
where branch_name = 'Perryridge'
and loan.account_number = borrower.account_number
```

36