Chapter

SQL DML

Week 3 Readings (SQL):

4.1, 4.2, 4.3, 4.4

SQL: Structured Query Language.

- SQL is the "standard" query language for relational databases.
- All relational DBMSs implement a standard (or a subset) of SQL.
- SQL can be used interactively (on-line) from a terminal or can be executed in programs.
- SQL is a complete language.
  - DDL (Data Definition Language)
  - DML (Data Manipulation Language)
  - DCL (Data Control Language)

SQL: Structured Query Language.

SQL can be used for

- Creating and managing databases
- Retrieving data
- Modifying and deleting existing data
- Inserting new data

SQL - history

- The early SQL standards was published by ANSI and IBM in middle 1980s.
- SQL was universally accepted shortly.
- The SQL89 and SQL92 standards were published by ISO/IEC.
- The SQL:1999 standard incorporated the object model. This standard defined the object-relational database model. It was a milestone in SQL development.
- Newer SQL standards were SQL:2003, SQL:2006, SQL:2008, and SQL:2011.

Retrieval Queries in SQL

- SQL has one basic statement for retrieving information from a database; the SELECT statement
- This is *not the same as* the SELECT (σ) of relational algebra
- Important distinction between SQL and relational algebra (formal language); SQL allows a table (relation) to have two or more tuples that are identical in all their attribute values
- Hence, an SQL relation is a bag of tuples; it *is not* a set of tuples
- SQL relations can be constrained to be sets by specifying PRIMARY KEY or UNIQUE attributes, or by using the DISTINCT clause (will be discussed later).

Retrieval Queries in SQL

- Basic form of the SQL SELECT statement (*SELECT-FROM-WHERE) block*

SELECT <attribute list>
FROM <table list>
WHERE <condition>

  - <condition> is a conditional (Boolean) expression that identifies the tuples to be retrieved by the query

  - There are more clauses than just these (SELECT FROM WHERE) – will be discussed later

```
chaturvr=> select * from s;
 sno | sname | status |  city
-----+-------+--------+--------
 S1  | SMITH |     20 | LONDON
 S2  | JONES |     10 | PARIS
 S3  | BLAKE |     30 | PARIS
 S4  | CLARK |     20 | LONDON
 S5  | ADAMS |     30 | ATHENS
 S6  | HENRY |     12 | GUELPH
(6 rows)

chaturvr=> select * from P;
 pno | pname | color | weight |  city
-----+-------+-------+--------+--------
 P1  | NUT   | RED   |  12.00 | LONDON
 P2  | BOLT  | GREEN |  17.00 | PARIS
 P3  | SCREW | BLUE  |  17.00 | ROME
 P4  | SCREW | RED   |  14.00 | LONDON
 P5  | CAN   | BLUE  |  12.00 | PARIS
 P6  | COG   | RED   |  19.00 | LONDON
(6 rows)

chaturvr=> select * from sp;
 sno | pno | qty
-----+-----+-----
 S1  | P1  | 200
 S2  | P3  | 400
 S2  | P5  | 100
 S3  | P3  | 200
 S3  | P4  | 500
 S4  | P6  | 300
 S5  | P2  | 200
 S5  | P5  | 500
 S5  | P6  | 200
 S5  | P1  | 100
 S5  | P3  | 200
 S5  | P4  | 800
(12 rows)
```

Retrieval Queries in SQL

Table S

SNo	SName	Status	City
S1	Smith	20	London
S2	Jones	10	Paris
S3	Blake	30	Paris
S4	Clark	20	London
S5	Adams	30	Athens

Write a query to list names of all suppliers who live in London.

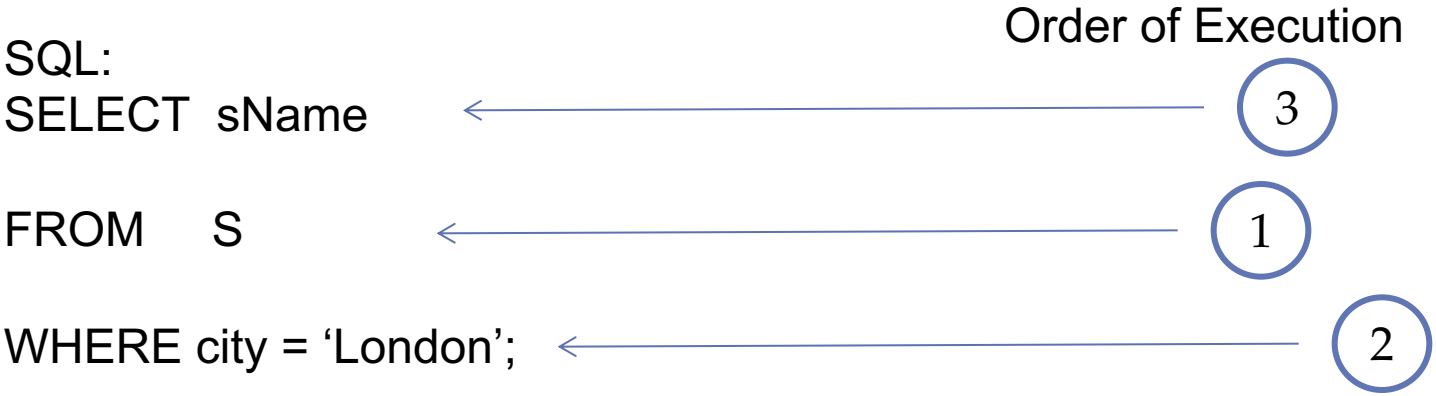
RA $\pi_{sname}(\sigma_{city='London'}(S))$

SQL:

```
SELECT  sName
FROM    S
WHERE city = 'London';
```

Order of execution in SELECT query

Write a query to list names of all suppliers who live in London.



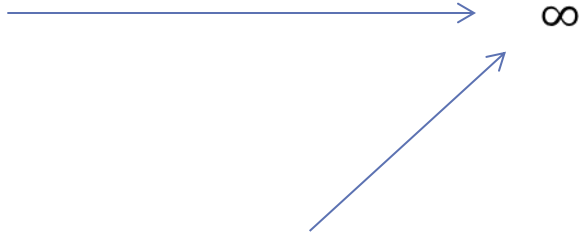
FROM Clause - examples

```
SELECT *                        --- all attributes
FROM S, P;                        ---  Cartesian product

SELECT *
FROM S temp1, S temp2;          --- self-join
                                    --- temp1 and temp2 are
                                    --- alias for S

SELECT *                      --- FROM can have a query itself
FROM  (SELECT
          *
          FROM P);
```

FROM clause - examples

List all suppliers and the parts they supply.

```
SELECT *
FROM S NATURAL JOIN SP;

SELECT *
FROM    S    JOIN    SP    USING    (SNO);
```



FROM clause - examples

SELECT *
FROM S JOIN SP ON S.SNO=SP.SNO;

→ *theta j*

SELECT *
FROM S LEFT JOIN SP ON S.SNO=SP.SNO;

→ *Left outer join*

WHERE

A WHERE clause condition can be constructed from

- column name, constants
- subqueries
- parentheses
- arithmetic operators (+,−, , /)
- comparison operators (=,>=,>,<>,<,<=)
- boolean operators (AND, OR, NOT)

WHERE clause - examples

List all suppliers

RA??

```
SELECT *
FROM S;
```

List all suppliers and the parts they supply.

```
SELECT *
FROM S, SP
WHERE S.SNO = SP.SNO;
```

WHERE clause - ex

List all suppliers and the parts they supply, i
supplied at a time is less than 350.

```
SELECT *
FROM S, SP
WHERE S.SNO = SP.SNO AND
        QTY < 350;
```

```
chaturvr=> select * from s;
 sno | sname | status |  city
-----+-------+--------+--------
 S1  | SMITH |     20 | LONDON
 S2  | JONES |     10 | PARIS
 S3  | BLAKE |     30 | PARIS
 S4  | CLARK |     20 | LONDON
 S5  | ADAMS |     30 | ATHENS
 S6  | HENRY |     12 | GUELPH
(6 rows)

chaturvr=> select * from P;
 pno | pname | color | weight |
-----+-------+-------+--------+---
 P1  | NUT   | RED   |  12.00 | LO
 P2  | BOLT  | GREEN |  17.00 | PA
 P3  | SCREW | BLUE  |  17.00 | RO
 P4  | SCREW | RED   |  14.00 | LO
 P5  | CAN   | BLUE  |  12.00 | PA
 P6  | COG   | RED   |  19.00 | LO
(6 rows)

chaturvr=> select * from sp;
 sno | pno | qty
-----+-----+-----
 S1  | P1  | 200
 S2  | P3  | 400
 S2  | P5  | 100
 S3  | P3  | 200
 S3  | P4  | 500
 S4  | P6  | 300
 S5  | P2  | 200
 S5  | P5  | 500
 S5  | P6  | 200
 S5  | P1  | 100
 S5  | P3  | 200
 S5  | P4  | 800
(12 rows)
```

WHERE clause - examples

List all suppliers and the parts they supply, if their name starts with an A.

LIKE and Wildcards: % and _

```
SELECT *
FROM S, SP
WHERE S.SNO = SP.SNO AND
        SNAME LIKE 'A%';
```

Range using BETWEEN

List all suppliers and parts they supply, if the qty supplied is between 100 and 200 (both inclusive).

```
SELECT *
FROM S, SP
WHERE S.SNO = SP.SNO AND
        QTY BETWEEN 100 AND 200;
```

WHERE clause- examples

List all suppliers and the parts they supply, if the qty supplied is either 100 or 200.

IN – set membership

```
SELECT *
FROM S, SP
WHERE S.SNO = SP.SNO AND
        QTY IN (100, 200)
```

Subqueries in WHERE – coming up later

WHERE - example

- Get all pairs of supplier numbers such that the two suppliers concerned are located in the same city.

```
chaturvr=> select * from s;
 sno | sname | status |  city
-----+-------+--------+--------
 S1  | SMITH |     20 | LONDON
 S2  | JONES |     10 | PARIS
 S3  | BLAKE |     30 | PARIS
 S4  | CLARK |     20 | LONDON
 S5  | ADAMS |     30 | ATHENS
 S6  | HENRY |     12 | GUELPH
(6 rows)

chaturvr=> select * from P;
 pno | pname | color | weight |
-----+-------+-------+--------+--
 P1  | NUT   | RED   |  12.00 | L
 P2  | BOLT  | GREEN |  17.00 | P
 P3  | SCREW | BLUE  |  17.00 | P
 P4  | SCREW | RED   |  14.00 | L
 P5  | CAN   | BLUE  |  12.00 | P
 P6  | COG   | RED   |  19.00 | L
(6 rows)

chaturvr=> select * from sp;
 sno | pno | qty
-----+-----+-----
 S1  | P1  | 200
 S2  | P3  | 400
 S2  | P5  | 100
 S3  | P3  | 200
 S3  | P4  | 500
```

SELECT (attributes)

List names of all suppliers in the DB.

```
SELECT SNAME
FROM S;
```

List names of all cities in which a supplier lives.

```
SELECT CITY
FROM S;
```

Use DISTINCT – otherwise duplicates will show (bag vs set)

```
SELECT DISTINCT CITY
FROM S;
```

NULL

- Not useful to use = and <> with NULL

- IS NULL
- IS NOT NULL

```
SELECT *
FROM S
WHERE SNAME IS NULL;
```

ORDER BY

- The **ORDER BY** clause is used to sort the tuples in a query result based on the values of some attribute(s)

- The default order is in ascending order of values. We can specify the keyword DESC if we want a descending order; the keyword ASC can be used to explicitly specify ascending order, even though it is the default.

- ORDER BY is the last clause executed just before SELECT

```
SELECT SNAME
FROM S
WHERE SNO IN (SELECT SNO FROM SP)
ORDER BY SNAME;
```

Subqueries

- Subqueries can be written and used in the FROM clause and (more commonly) in the WHERE clause

- Syntax:
  - Subqueries must be parenthesized
  - If we need to use the results of this subquery in some outer query – we must name it

```
[chaturvr=> select * from s;
 sno | sname | status |  city
-----+-------+--------+--------
 S1  | SMITH |     20 | LONDON
 S2  | JONES |     10 | PARIS
 S3  | BLAKE |     30 | PARIS
 S4  | CLARK |     20 | LONDON
 S5  | ADAMS |     30 | ATHENS
 S6  | HENRY |     12 | GUELPH
(6 rows)

[chaturvr=> select * from P;
 pno | pname | color | weight |  city
-----+-------+-------+--------+--------
 P1  | NUT   | RED   |  12.00 | LONDON
 P2  | BOLT  | GREEN |  17.00 | PARIS
 P3  | SCREW | BLUE  |  17.00 | ROME
 P4  | SCREW | RED   |  14.00 | LONDON
 P5  | CAN   | BLUE  |  12.00 | PARIS
 P6  | COG   | RED   |  19.00 | LONDON
(6 rows)

[chaturvr=> select * from sp;
 sno | pno | qty
-----+-----+-----
 S1  | P1  | 200
 S2  | P3  | 400
 S2  | P5  | 100
 S3  | P3  | 200
 S3  | P4  | 500
 S4  | P6  | 300
 S5  | P2  | 200
 S5  | P5  | 500
 S5  | P6  | 200
 S5  | P1  | 100
 S5  | P3  | 200
 S5  | P4  | 800
(12 rows)
```

List names of all suppliers who supply a part.

Subqueries -

- List names of all suppliers who supply

The query below throws and error

```
SELECT SNAME
FROM S
WHERE SNO = (SELECT SNO FROM SP);
```

FIX?

```
SELECT SNAME
FROM S
WHERE SNO IN (SELECT SNO FROM S
```

-

```
 sno | sname | status |  city
-----+-------+--------+------
 S1  | SMITH |     20 | LONDO
 S2  | JONES |     10 | PARIS
 S3  | BLAKE |     30 | PARIS
 S4  | CLARK |     20 | LONDO
 S5  | ADAMS |     30 | ATHEN
 S6  | HENRY |     12 | GUELP
(6 rows)

chaturvr=> select * from P;
 pno | pname | color | weight
-----+-------+-------+-------
 P1  | NUT   | RED   |  12.00
 P2  | BOLT  | GREEN |  17.00
 P3  | SCREW | BLUE  |  17.00
 P4  | SCREW | RED   |  14.00
 P5  | CAN   | BLUE  |  12.00
 P6  | COG   | RED   |  19.00
(6 rows)

chaturvr=> select * from sp;
 sno | pno | qty
-----+-----+-----
 S1  | P1  | 200
 S2  | P3  | 400
 S2  | P5  | 100
 S3  | P3  | 200
 S3  | P4  | 500
 S4  | P6  | 300
 S5  | P2  | 200
 S5  | P5  | 500
 S5  | P6  | 200
 S5  | P1  | 100
 S5  | P3  | 200
 S5  | P4  | 800
(12 rows)
```

Subqueries - IN

- Get supplier names for suppliers who supply part P3.

Subqueries - example

List names of suppliers who have a status less than that of supplier ‘S1’.

```
SELECT SNAME
FROM S
WHERE STATUS < (SELECT STATUS
                FROM S
                WHERE SNO = ‘S1’);
```

The inner subquery must result in a single value, when operators such as = or < are used

Subqueries - example

What happens?

```
SELECT SNAME
FROM S
WHERE STATUS < (SELECT STATUS
                FROM S);
```

Subqueries

- For example, the query below throws an error

```
SELECT SNAME
FROM S
WHERE STATUS <  (SELECT STATUS
                 FROM S);
```

- Fix?
- Use
  - ANY
  - ALL
  - IN
  - EXISTS

Subqueries - ANY

- For example, the query below throws an error because its inner query returns more than 1 value of status!

```
SELECT SNAME
FROM S
WHERE STATUS <  (SELECT STATUS
                 FROM S);
```

- Fix? Use ANY?

Original table S

sno	sname	status	ci
S2	JONES	10	PARI
S3	BLAKE	30	PARI
S4	CLARK	20	LOND
S5	ADAMS	30	ATHE
S1	SMITH	20	LOND

```
ritu=> select sname from s
ritu-> where status <ANY (select status from s);
 sname
-------
 JONES
 CLARK
 SMITH
(3 rows)
```

Subqueries - ALL

- Find names of suppliers who have a status greater than all status values in table STATUS (i.e. max value)

Original table S

sno	sname	status	city
S2	JONES	10	PARIS
S3	BLAKE	30	PARIS
S4	CLARK	20	LONDON
S5	ADAMS	30	ATHENS
S1	SMITH	20	LONDON

```
ritu=> select sname from s where status >=ALL (select status from s);
 sname
-------
 BLAKE
 ADAMS
(2 rows)
```

Subqueries – ANY and ALL

- Condition c > ANY R is true if c is greater than at least one value in relation R (any 1 or more). Note that R must be a 1-column relation

- Condition c > ALL R is true if c is greater than every value in relation R. Note that R must be a 1-column relation

Subqueries - IN

- Get supplier names for suppliers who supply at least one red part.

```
SELECT sname
FROM S
WHERE sno IN (SELECT sno
              FROM SP
              WHERE pno IN (SELECT pno
                            FROM P
                            WHERE color = 'RED'));
```

Friday's class

Subqueries - IN

- NOT IN
- Get supplier names for suppliers who do not supply part P2.

SET OPERATIONS

- SQL has directly incorporated some set operations
- There is a union operation (**UNION)**, and in *some versions* of SQL there are set difference (**EXCEPT)** and intersection (**INTERSECT)** operations
- The resulting relations of these set operations are sets of tuples; *duplicate tuples are eliminated from the result*
- The set operations apply only to *union compatible relations* ; the two relations must have the same attributes and the attributes must appear in the same order

Set operators: Example

- Get supplier names for suppliers who do not supply part P2.

Worksheet

AGGREGATE FUNCT

- Include COUNT, SUM, MAX, MIN, and
- Apply to a column and return a singl
- **Example:**

```
ritu=> select * from
 sno | pno | qty
-----+-----+-----
 S1  | P1  | 200
 S2  | P3  | 400
 S2  | P5  | 100
 S3  | P3  | 200
 S3  | P4  | 500
 S4  | P6  | 300
 S5  | P2  | 200
 S5  | P5  | 500
 S5  | P6  | 200
 S5  | P1  | 100
 S5  | P3  | 200
 S5  | P4  | 800
(12 rows)
```

SELECT COUNT(SNO), MAX(QTY), SUM(QTY)
FROM SP;

```
 count | max | sum
-------+-----+------
    12 | 800 | 3700
```

AGGREGATE FUNCT

```
ritu=> select * from
 sno | pno | qty
-----+-----+-----
 S1  | P1  | 200
 S2  | P3  | 400
 S2  | P5  | 100
 S3  | P3  | 200
 S3  | P4  | 500
 S4  | P6  | 300
 S5  | P2  | 200
 S5  | P5  | 500
 S5  | P6  | 200
 S5  | P1  | 100
 S5  | P3  | 200
 S5  | P4  | 800
(12 rows)
```

- **Example:**

SELECT COUNT(DISTINCT SNO), COUNT(*)
FROM SP;

```
 count | count
-------+-------
     5 |    12
(1 row)
```

COUNT(*) gives the number of tuples in the result of FROM (and WHERE if any).

GROUPING

- In many cases, we want to apply the aggregate functions *to subgroups of tuples in a relation*
- Each subgroup of tuples consists of the set of tuples that have *the same value* for the *grouping attribute(s)*
- The function is applied to each subgroup independently
- SQL has a **GROUP BY**-clause for specifying the grouping attributes, which *must also appear in the SELECT-clause*

GROUPING

```
ritu=> select * from sp;
 sno | pno | qty
-----+-----+-----
 S1  | P1  | 200
 S2  | P3  | 400
 S2  | P5  | 100
 S3  | P3  | 200
 S3  | P4  | 500
 S4  | P6  | 300
 S5  | P2  | 200
 S5  | P5  | 500
 S5  | P6  | 200
 S5  | P1  | 100
 S5  | P3  | 200
 S5  | P4  | 800
(12 rows)
```

How many parts does each supplier supply?

S1 supplies 1 part	P1
S2 supplies 2 parts	P3
	P5
S3 supplies 2 parts	P3
	P4
S4 supplies 1 part	P6
S5 supplies 6 parts	P1
	P2
	P3
	P4
	P5
	P6

GROUP BY

```
ritu=> select * from sp;
 sno | pno | qty
-----+-----+-----
 S1  | P1  | 200
 S2  | P3  | 400
 S2  | P5  | 100
 S3  | P3  | 200
 S3  | P4  | 500
 S4  | P6  | 300
 S5  | P2  | 200
 S5  | P5  | 500
 S5  | P6  | 200
 S5  | P1  | 100
 S5  | P3  | 200
 S5  | P4  | 800
(12 rows)
```

How many parts does each supplier supply?

```
SELECT      SNO, COUNT(PNO)
FROM        SP
GROUP BY SNO;
```

```
 sno | count
-----+-------
 S3  |     2
 S1  |     1
 S4  |     1
 S5  |     6
 S2  |     2
(5 rows)
```

GROUP BY

```
SELECT        SNO, COUNT(PNO)
FROM  SP
GROUP BY SNO;
```

- The result of the SELECT-FROM-WHERE query is grouped according to the values of the attributes listed in GROUP BY
- any aggregation is applied only within each group and gives a single value per group

GROUP BY - restrictions

If an aggregate function is used, then each element of the SELECT list must be either:

- an aggregate function (MAX, MIN, COUNT, AVG, SUM) or

- an attribute on the GROUP BY list.

GROUP BY - restrictions

For example, the following queries are invalid.

```
SELECT    SNO, MAX(Qty)
FROM      SP;
```

Valid:

```
SELECT    SNO, MAX(Qty)
FROM      SP
GROUP     BY SNO;
```

```
SELECT    SNO
FROM      SP
WHERE     Qty = MAX(QTY)
```

Valid:

```
SELECT   SNO
FROM     SP
WHERE  Qty = (SELECT MAX(QTY)
              FROM SP);
```

GROUP BY - example

Write a query to list supplier numbers and the total qty they supply, in ascending order of the total.

```
SELECT SNO, SUM(QTY) SQ
FROM SP
GROUP BY SNO
ORDER BY SQ;
```

```
SNO|  SQ
-----+------
 S1  |  200
 S4  |  300
 S2  |  500
 S3  |  700
 S5  | 2000
(5 rows)
```

THE HAVING-CLAUSE

- Sometimes we want to retrieve the values of these functions for only those *groups that satisfy certain conditions*

- The HAVING-clause is used for specifying a selection condition on groups (rather than on individual tuples)

HAVING - restrictions

- The same requirement as for SELECT clauses with aggregation

HAVING may refer to attributes only if they are either:

- aggregated, or
- an attribute on the GROUP BY list.

HAVING - Example

Write a query to list supplier numbers and the total qty they supply, if the total is more than 1000.

```
ritu=> SELECT SNO, SUM(QTY) SQ
        FROM SP
        GROUP BY SNO
        HAVING SUM(QTY) >1000;
 sno |  sq
-----+------
 S5  | 2000
(1 row)
```

HAVING - Example

Write a query to

```
SELECT    pno
FROM      SP
GROUP     BY pno
HAVING    COUNT(*)>1;
```

Summary of SQL Queries

- A query in SQL can consist of up to six clauses, but only the first two, SELECT and FROM, are mandatory. The clauses are specified in the following order:

SELECT **<attribute list>**
FROM **<table list>**
[WHERE **<condition>]**
[GROUP BY **<grouping attribute(s)>]**
[HAVING **<group condition>]**
[ORDER BY **<attribute list>]**

Next Class

- Correlated Subqueries