CptS 451- Introduction to Database Systems

SQL as a Query Language - part1

(DMS ch-5)

Instructor: Sakire Arslan Ay

SQL = Structured Query Language

Standard language for querying and manipulating relational data

- Query capabilities of SQL are similar to those in *relational algebra*
- Many standards: SQL92, SQL2, SQL3, SQL99
- SQL language has several aspects:
  - ✓ Data Definition Language (DDL)
    - → CREATE TABLE, ALTER TABLE, DROP TABLE
  - Query Language
    - → SELECT
  - Data Manipulation Language(DML)
    - → INSERT, DELETE, UPDATE
  - Triggers and Advanced Integrity Constraints

What is special about SQL?

- You describe ***what*** you want
- The job of the DBMS is to figure out ***how*** to compute what you want efficiently.

Topics

- SQL as a Query Language
  - Select queries
  - Set operations: UNION, …
  - Aggregation, Group by

The basic form of a SQL query is *select-from-where*

SELECT desired attributes

FROM one or more tables

WHERE condition on the rows of the tables

Project out everything not in the final answer

Every table you want to join, together

All the join and selection conditions

SQL as a Query Language

SELECT A1, A2, …, An
FROM R1, R2, …, Rm
WHERE conditions;

- **Example**: Emp(ssn, ename, dno, sal),
  Dept(dno, dname, mgr),
  Proj(proj_id, ptitle,startdate,enddate,numEmp),
  ProjEmp(proj_id,ssn,begindate)

Query 1: “Find employees’ names who work in department 132.”

```
SELECT   ename
FROM     Emp
WHERE    dno=132;
```

Query 2: “Find the manager of the ‘Marketing’ department.”

```
SELECT   mgr
FROM     Dept
WHERE    dname = ‘Marketing’;
```

SQL vs Relational Algebra

```
SELECT A1, A2, …, An
FROM    R1, R2, …, Rm
WHERE conditions;
```

- Equivalent relational algebra expression:

  $\Pi_{A1,\ldots,An} (\sigma_{cond} (R1 \times R2 \times \ldots Rm))$

- Difference:
  - Relational algebra uses set semantics
  - Most SQL operators uses bag semantics
    - However, SQL set operators use set semantics
    - Set operators are applied on query results

“Select” Clause

- Specify attributes to project onto (different from the “selection” operator in the relational algebra)

- Use star * to denote all attributes:

```
SELECT   *
FROM     Emp
WHERE    ename ='Jack' AND sal>50K;
```

Emp(ssn,ename,dno,salary)

ssn	Ename	Dno	sal
111-111-1111	Jack	111	81K
~~222-111-2222~~	~~Alice~~	~~111~~	~~70K~~
~~333-111-3333~~	~~Lisa~~	~~222~~	~~32K~~
~~444-111-4444~~	~~Tom~~	~~333~~	~~56K~~
~~555-111-5555~~	~~Mary~~	~~333~~	~~65K~~
~~666-111-6666~~	~~Jack~~	~~444~~	~~45K~~

Here is a way to think about how the query might be implemented

1. Imagine a *tuple variable* ranging over each tuple of the relation mentioned in FROM.
2. Check if the “current” tuple satisfies the WHERE clause.
3. If so, output the attributes/expressions of the SELECT clause using the components of this tuple.

A	B	C
A1	B1	C1
A2	B2	C2
A3	B3	C3
A4	B4	C4
A5	B5	C5
A6	B6	C6
A7	B7	C7

```
SELECT   A, B
FROM     R
WHERE  A =‘A3’;
```

A	B
A3	B3

“Select” Clause

- Single Relation vs. Multi Relation Queries
- Single relation:

```
SELECT   *
FROM     Emp
WHERE    ename ='Jack' AND sal>50K;
```

- Multiple relations:
  - Can use relation prefix (especially when we need to disambiguate attribute names)

```
SELECT   *
FROM     Emp, Dept
WHERE    Emp.dno = Dept.dno;
```

Emp(ssn,ename,dno,salary)

ssn	Ename	dno	sal
111-111-1111	Jack	111	81K
222-111-2222	Alice	111	70K
333-111-3333	Lisa	222	32K
444-111-4444	Tom	333	56K
555-111-5555	Mary	333	65K

Dept(dno,dname, mgr)

dno	dname	mgr
111	HR	Alice
222	R&D	Lisa
333	Production	Mary

“Select” Clause

```
SELECT   *
FROM     Emp, Dept
WHERE  Emp.dno = Dept.dno;
```

Emp(ssn,ename,dno,salary)

ssn	Ename	dno	sal
111-111-1111	Jack	111	81K
222-111-2222	Alice	111	70K
333-111-3333	Lisa	222	32K
444-111-4444	Tom	333	56K
555-111-5555	Mary	333	65K

Dept(dno,dname, mgr)

dno	dname	mgr
111	HR	Alice
222	R&D	Lisa
333	Production	Mary

Emp.ename	Emp.dno	Emp.sal	Dept.dno	Dept.dname	Dept.mgr
Jack	111	81K	111	HR	Alice
Alice	111	70K	111	HR	Alice
Lisa	222	32K	222	R&D	Lisa
Tom	333	56K	333	Production	Mary
Mary	333	65K	333	Production	Mary

Eliminate Duplicates

- “SELECT” does **not** automatically eliminate duplicates.

```
SELECT   dno
FROM     Emp;
```

  - If there are more than 1 employee in the department 333, then ‘333’ will appear more than once in the result.
- Use keyword **distinct** to explicitly remove duplicates

Emp(ssn,ename,dno,salary)

ssn	Ename	dno	sal
111-111-1111	Jack	111	81K
222-111-2222	Alice	111	70K
333-111-3333	Lisa	222	32K
444-111-4444	Tom	333	56K
555-111-5555	Mary	333	65K

```
SELECT  distinct dno
FROM    Emp;
```

dno
111
222
333

“Select” Clause (cont.)

- You can rename the attributes in the result, using “as <new name>”

```
SELECT   ename, mgr as manager
FROM    Emp, Dept
WHERE  Emp.dno = Dept.dno AND
             manager=‘Alice’;
```

Emp(ssn,ename,dno,salary)

ssn	Ename	dno	sal
111-111-1111	Jack	111	81K
222-111-2222	Alice	111	70K
333-111-3333	Lisa	222	32K
444-111-4444	Tom	333	56K
555-111-5555	Mary	333	65K

- You can create a new column and give it a constant value, in the SELECT clause

```
SELECT ename, dno, ‘temporary’  as status
FROM    Emp
WHERE  dno = 111;
```

ename	dno	status
Jack	111	temporary
Alice	111	temporary

“Select” Clause (cont.)

- You can use math in the SELECT clause

Case-insensitive, except inside quoted strings

```
SELECT   eNaMe, sal*1.05 as newSalary
FROM    Emp
WHERE  ename=‘O’’Fallon’;
```

Two single quotes inside a string
= one apostrophe

ename	newSalary
O’Fallon	85.05K

“FROM” clause

- Specify relations
- Renaming relations:
  - Use “**as**” to define “variables,” to disambiguate multiple references to the same relation
  - Example: “who has higher salary than their manager”

```
SELECT    E1.ename
FROM      Emp as E1, Dept D, Emp as E2
WHERE     E1.dno = D.dno AND
          D.mgr = E2.ename AND
          E1.sal > E2.sal;
```

E1: Emp(ssn,ename,dno,salary)

ssn	Ename	dno	sal
111-111-1111	Jack	111	81K
222-111-2222	Alice	111	70K
333-111-3333	Lisa	222	32K
444-111-4444	Tom	333	56K
555-111-5555	Mary	333	65K

Dept(dno,dname, mgr)

dno	dname	mgr
111	HR	Alice
222	R&D	Lisa
333	Production	Mary

E1: Emp(ssn,ename,dno,salary)

ssn	Ename	dno	sal
111-111-1111	Jack	111	81K
222-111-2222	Alice	111	70K
333-111-3333	Lisa	222	32K
444-111-4444	Tom	333	56K
555-111-5555	Mary	333	65K

“WHERE” clause

- Specify conditions
- Optional
- Complex conditions:
  - AND, OR, NOT, …
  - “Employees who work for Lisa and have a salary < 70K”

```
SELECT   ename
FROM     Emp, Dept
WHERE    Emp.dno=Dept.dno AND
          mgr = ‘Lisa’ AND
         sal < 70K;
```

“WHERE” clause (cont.)

- String patterns:
  - LIKE keyword uses a regular expression to contain the pattern that the values are matched against
  - “s LIKE p”: string **s** matches pattern **p**
  - Pattern may include:
    - % (percent): zero, one, or multiple occurrences of any character
      - `dname LIKE ‘TOM %’`
        - » ‘TOM KERRY’, ‘TOM JOHNSON’, ‘TOM ‘ …
    - _ (underbar): one-character wildcard
      - `dname LIKE ‘a_c’`
        - » ‘abc’ ‘adc’ ‘azc’ ‘a9c’ …

Conditions in a “WHERE” clause

The following may appear in the WHERE condition

- constants of any supported type
- attribute names of the relation(s) used in the FROM.
- comparison operators: =, <>, <, >, <=, >=
- arithmetic operations: price*2
- operations on strings (e.g., CONCAT for concatenation).
- lexicographic order on strings (lastname<‘Norman’).
- pattern matching: s LIKE p , s NOT LIKE p
- special operations for comparing dates and times.

- and combinations of the above using AND, OR, NOT, and parentheses

- Use relation prefix to disambiguate attribute names

```
SELECT  ename,  dname, dept.dno
FROM     Emp,  Dept
WHERE  Emp.dno = Dept.dno;
```

Conditions in a “WHERE” clause

- What if an attribute value is unknown, or the attribute is inapplicable (i.e. is NULL)?
  - Example:

Emp(ssn,ename,dno,salary)

ssn	Ename	dno	sal
111-111-1111	Jack	111	81K
222-111-2222	Alice	111	70K
333-111-3333	Lisa	222	NULL
444-111-4444	Tom	333	NULL
555-111-5555	Mary	333	65K

```
SELECT ename, sal
FROM   Emp
WHERE  sal<=50K OR sal>50K;
```

ename	sal
Jack	81K
Alice	70K
Mary	65K

Why???

Conditions involving NULL evaluate to *unknown*, rather than *true* or *false*

Example condition	Evaluates to
‘Tom’ = ‘Tom’	**true**
2 > 6	**false**
‘Tom’ = NULL	unknown
2 < NULL	unknown
true AND unknown	unknown
true OR unknown	**true**
false AND unknown	**false**
false OR unknown	unknown
unknown OR unknown	unknown

True-> 1

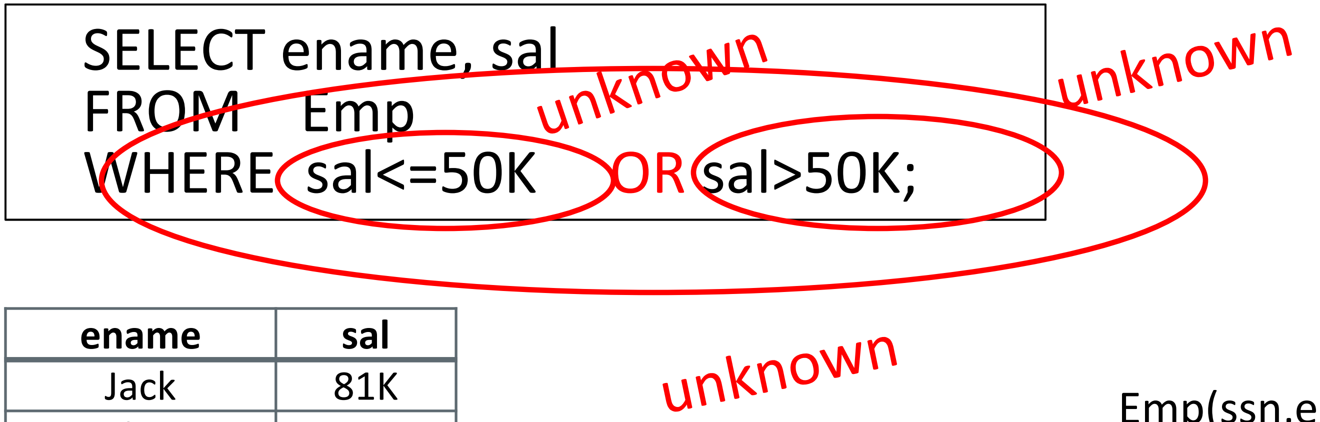
False-> 0

Unknown->1/2

A tuple only goes in the answer if its truth value for the WHERE clause is true.

Conditions in a “WHERE” clause

- What if an attribute value is unknown, or the attribute is inapplicable?
  - Example:



```
SELECT ename, sal
FROM    Emp
WHERE  sal<=50K  OR  sal>50K;
```

ename	sal
Jack	81K
Alice	70K
Mary	65K

Emp(ssn,ename,dno,salary)

ssn	Ename	dno	sal
111-111-1111	Jack	111	81K
222-111-2222	Alice	111	70K
333-111-3333	Lisa	222	NULL
444-111-4444	Tom	333	NULL
555-111-5555	Mary	333	65K

Dealing with NULL Values

Can test for NULL explicitly:

- IS NULL
- IS NOT NULL

```
SELECT ename, sal
FROM   Emp
WHERE  sal<=50000 OR sal>50000 OR sal is NULL;
```

The answer includes all employees!

Ordering Output Tuples

```
SELECT *
FROM    Emp
WHERE  sal<=50000
ORDER BY dno, sal desc, ename;
```

- First, order the tuples by dno (department).
- Within each department, order salaries from highest to lowest.
- For salary ties, use alphabetical order on the name.

Ordering Output Tuples

```
SELECT *
FROM    Emp
WHERE  sal<=50000
ORDER BY dno, sal desc, ename;
```

By default, ORDER BY orders in ascending order. Use keyword “desc” for descending order.

- What if there are NULL values?

Emp(ssn,ename,dno,salary)

ssn	ename	dno	sal
111-111-1111	Jack	NULL	81K
222-111-2222	Alice	111	70K
333-111-3333	Lisa	222	32K
444-111-4444	Tom	333	56K
555-111-5555	Mary	NULL	65K

ssn	ename	dno	sal
111-111-1111	Jack	NULL	81K
555-111-5555	Mary	NULL	65K
222-111-2222	Alice	111	70K
333-111-3333	Lisa	222	32K
444-111-4444	Tom	333	56K

SQL:2003 standard

```
SELECT *
FROM    Emp
WHERE  sal<=50000 OR sal IS NULL
ORDER BY dno NULLS FIRST, sal desc, ename;
```

Set Operations

- **Use the set semantics**
  - duplicates are eliminated in the result.
- **Example:**
- Union: ∪ “Find employees who work either for the ‘Purchasing’ or the ‘HR’ department.”

```
(SELECT  ename FROM Emp, Dept
 WHERE Emp.dno=Dept.dno  AND dname=‘Purchasing’)
UNION
(SELECT  ename FROM Emp, Dept
 WHERE Emp.dno=Dept.dno  AND dname=‘HR’)
```

The schema of the SELECT results should be same

- Intersect: ∩ “Find employees who work both for the ‘Purchasing’ or the ‘HR’ departments.”

```
(SELECT  ename FROM Emp, Dept
 WHERE Emp.dno=Dept.dno  AND dname=‘Purchasing’)
INTERSECT
(SELECT  ename FROM Emp, Dept
 WHERE Emp.dno=Dept.dno  AND dname=‘HR’)
```

Set Operations

- Except: –“Find employees who work for the ‘Accounting’ department but not for the ‘Purchasing’ department.”

```
(SELECT  ename FROM Emp, Dept
 WHERE Emp.dno=Dept.dno  AND dname=‘Accounting’)
EXCEPT
(SELECT  ename FROM Emp, Dept
 WHERE Emp.dno=Dept.dno  AND dname=‘Purchasing’)
```

Set Operations - Conserving Duplicates

- The UNION, INTERSECT, and EXCEPT operators use the set semantics, not bag semantics.
- To keep duplicates, use "ALL" after the operators:
  - UNION ALL, INTERSECT ALL, EXCEPT ALL
  - Example:

```
(SELECT  ssn, name, "student" as standing FROM Student)
UNION ALL
(SELECT  ssn, name, "TA" as standing FROM TeachingAssistant)
```

Student (ssn, name)

ssn	name
111	Tom
222	Jack
444	Mary

TA (ssno, name)

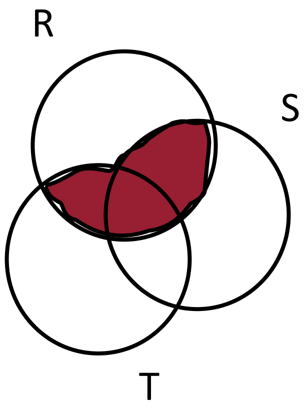
ssn	name
111	Tom
222	Jack
555	Alice

Result

ssn	name	standing
111	Tom	student
222	Jack	student
444	Mary	student
111	Tom	TA
222	Jack	TA
555	Alice	TA

Set Operations - Example

- Relations: R(A), S(A), T(A)
- Query: “R ∩ (S ∪ T)”



```
SELECT   R.A FROM R
 intersect
(    (SELECT   S.A FROM S)
      union
      (SELECT   T.A FROM T)
);
```

Solution-1

```
(SELECT   R.A FROM R, S  WHERE   R.A=S.A)
union
(SELECT   R.A FROM R, T WHERE   R.A=T.A);
```

Solution-2

```
SELECT   R.A
FROM      R, S, T
WHERE   R.A=S.A OR R.A=T.A;
```

Wrong!

- The SQL result becomes empty when T is empty

Aggregations

- MIN, MAX, SUM, COUNT, AVG
  - input: collection of numbers/strings (depending on operation)
  - output: relation with a single attribute with a single row

- Example: “What is the minimum, maximum, average salary of employees in the ‘Marketing’ department”

```
SELECT MIN(sal), MAX(sal), AVG(sal)
FROM   Emp, Dept
WHERE  Emp.dno = Dept.dno  and  Dept.dname = ’Marketing’;
```

Aggregations (cont.)

- Except "count," all aggregations apply to a single attribute
- "Count" can be used on "*"

```
SELECT  Count(*) FROM Emp;
```

```
SELECT  Count(ename) FROM Emp;
```

Emp(ssn,ename,dno,salary)

ssn	Ename	dno	sal
111-111-1111	Jack	111	81K
222-111-2222	Alice	111	70K
333-111-3333	Lisa	222	32K
444-111-4444	Tom	333	56K
555-111-5555	Mary	333	65K

Result : 5

Duplication in Aggregations

- “What is the number of different dno’s in the Emp table”

```
    SELECT  count(dno)
    FROM Emp;
```

Wrong! Since there can be duplicates

- Right Query:

```
    SELECT  count(DISTINCT dno)
    FROM Emp;
```

Emp(ssn,ename,dno,salary)

ssn	Ename	dno	sal
111-111-1111	Jack	111	81K
222-111-2222	Alice	111	70K
333-111-3333	Lisa	222	32K
444-111-4444	Tom	333	56K
555-111-5555	Mary	333	65K

Result : 3

GROUP BY Clause

- GROUP BY is used to apply aggregate function to a group of sets of tuples.
  - The aggregate function is applied to each group separately.
- **Example:** “For each department, list its total number of employees and total salary”

```
SELECT Dept.dno, SUM(sal), COUNT(ssn)
FROM   Emp, Dept
WHERE  Emp.dno = Dept.dno
GROUP BY Dept.dno;
```

Emp(ssn,ename,dno,salary)

ssn	Ename	dno	sal
111-111-1111	Jack	111	81K
222-111-2222	Alice	111	70K
333-111-3333	Lisa	222	32K
444-111-4444	Tom	333	56K
555-111-5555	Mary	333	65K

Dept(dno,dname, mgr)

dno	dname	mgr
111	HR	Alice
222	R&D	Lisa
333	Production	Mary

Result

dname	Sum(sal)	Count(ename)
HR	151K	2
R&D	32K	1
Production	121K	2

GROUP BY Clause (cont.)

- Standard SQL: “SELECT” attributes must appear in Group-by attributes.
- The following queries cannot group the tuples.

```
SELECT dname, Emp.dno, SUM(sal), COUNT(ssn)
FROM   Emp, Dept
WHERE  Emp.dno = Dept.dno
GROUP BY Emp.dno;
```

```
SELECT dname, SUM(sal), COUNT(ssn)
FROM   Emp, Dept
WHERE  Emp.dno = Dept.dno
GROUP BY Emp.dno;
```

GROUP BY Clause (cont.)

- Do the following queries return the same result?

```
SELECT dno
FROM Emp
GROUP BY dno;
```

```
SELECT distinct dno
FROM Emp;
```

HAVING Clause

- HAVING clause used along with GROUP BY clause to select some groups.
  - We can't define conditions on aggregate results in the WHERE clause
  - Syntax: HAVING aggregate_function(column_name) operator value
- Predicate in having clause applied after the formation of groups.
- **Example:** "List the department name, total salary, and number of employees for all departments with more than 2 employees."

```
SELECT dname, SUM(sal), COUNT(ssn)
FROM   Emp, Dept
WHERE  Emp.dno = Dept.dno
GROUP BY dname
HAVING COUNT(ssn)>2;
```

Emp(ssn,ename,dno,salary)

ssn	Ename	dno	sal
111-111-1111	Jack	111	81K
222-111-2222	Alice	111	70K
333-111-3333	Lisa	222	32K
444-111-4444	Tom	333	56K
555-111-5555	Mary	333	65K

Dept(dno,dname, mgr)

dno	dname	mgr
111	HR	Alice
222	R&D	Lisa
333	Production	Mary

Result

dname	Sum(sal)	Count(ename)
HR	151K	2
Production	121K	2

A General SQL Select Query

- “For each employee that works in two or more departments, print the total salary of his/her managers.”

```
SELECT ssn, ename, count(*)
FROM  Emp, Dept
WHERE Emp.dno=Dept.dno
GROUP BY ssn, ename
HAVING count(*) > 1
ORDER BY ssn,ename;
```

Find employees that works in two or more departments

```
SELECT E1.ssn,E1.ename,  sum(E2.sal)
FROM  Emp  as E1, Dept, Emp  as E2
WHERE  E1.dno = Dept.dno  AND E2.ename = Dept.mgr
GROUP BY E1.ssn, E1.ename
HAVING count(*) > 1
ORDER BY E1.ssn, E1.ename;
```

For those employees, find their managers and calculate the sum of the managers’ salaries.

A General SQL Select Query

- For each employee that works in two or more departments, print the total salary of his/her managers.

```
SELECT E1.ssn, E1.ename,  SUM(E2.sal)                        5
FROM  Emp  E1, Dept, Emp  E2                                 1
WHERE  E1.dno = Dept.dno  AND E2.ename = Dept.mgr            2
GROUP BY E1.ssn,E1.ename                                     3
HAVING count(distinct(Dept.dno)) > 1                         4
ORDER BY E1.ssn,E1.ename;                                    6
```

A General SQL Query

- For each employee that works in two or more departments, print the total salary of his/her managers. Assume each dept has one manager.

```
SELECT E1.ssn, E1.ename,  SUM(E2.sal)                      5
FROM  Emp  E1, Dept, Emp  E2                                1
WHERE  E1.dno = Dept.dno  AND E2.ename = Dept.mgr           2
GROUP BY E1.ssn,E1.ename                                    3
HAVING count(distinct(Dept.dno)) > 1                        4
ORDER BY E1.ssn,E1.ename;                                   6
```

<u>Execution steps:</u>

Step 1: tuples are formed (Cartesian product)
Step 2: tuples satisfying the conditions are chosen
Step 3: groups are formed
Step 4: groups are eliminated using “Having”
Step 5: the aggregates are computed for the select line, flattening the groups
Step 6: the output tuples are ordered and printed out.