SQL: Data Manipulation

Part One

Database Management - CIS 386 01 FA17

By: Dr. Aos Mulahuwaish

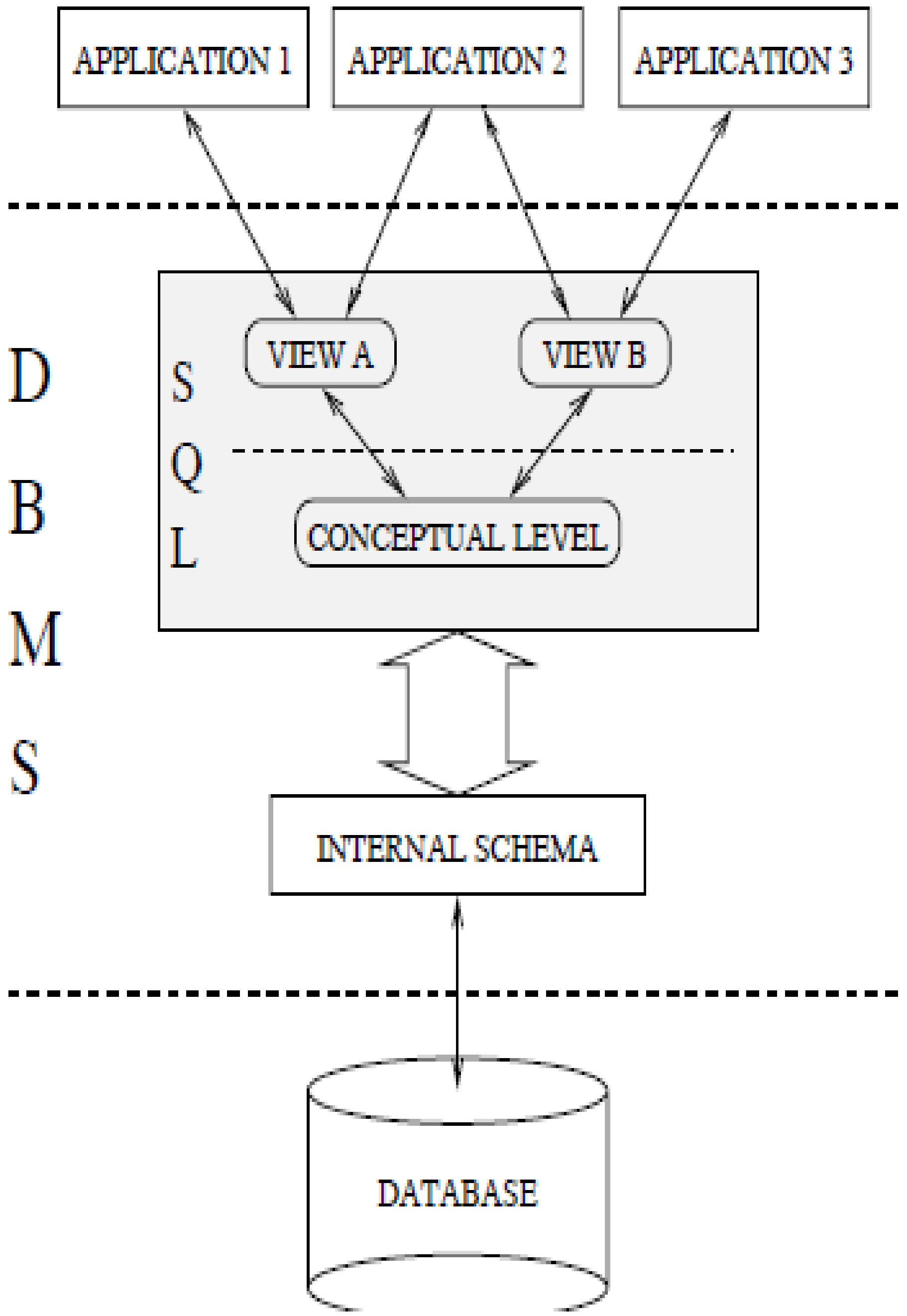
Outline

- The SQL Standard
- SQL DML
  - Basic Queries
  - Data Modification
  - Complex Queries
    - Set and Multiset Operations
    - Unknown values
    - Ordering results
    - Aggregation function
    - Outer joins

Structured Query Language

- Structured Query Language (SQL) is made up of two sub-languages:
    - SQL Data Manipulation Language (DML)
        - **SELECT** statements perform queries
        - **INSERT, UPDATE, DELETE** statements modify the instance of a table
- SQL Data Definition Language (DDL)
    - **CREATE, DROP** statements modify the database schema
    - **GRANT, REVOKE** statements enforce the security model

The SQL Standard



SQL DML: Queries

- **Find the last names and hire dates of employees who make more than $100000**

```
select LastName, HireDate
from Employee
where Salary > 100000
```

Multisets

- Relational model: relations are sets
- SQL standard: tables are multisets
    - Duplicate tuples may be stored
    - SQL queries may result in duplicates even if none of the input tables themselves contain duplicates
    - select distinct is used to eliminate duplicates from the result of a query

```
select distinct LastName, HireDate
from Employee
where Salary > 100000
```

SQL Query Involving Several Relations

- **For each project for which department E21 is responsible, find the name of the employee in charge of that project**

```
select P.ProjNo, E.LastName
from Employee  E, Project  P
where P.RespEmp = E.EmpNo
     and P.DeptNo = ‘ E21’
```

The SQL Basic Query Block

select *attribute-expression-list*

from *relation-list*

[where *condition***]**

Note:

The result of such a query is a relation which has one attribute for each element of the query's attribute-expression-list

The SQL "Where" Clause

- Conditions may include
  - arithmetic operators +,-,*,/
  - comparisons =,<>, <, <=, >, >=
  - logical connectives **and, or,** and **not**

- **List the last names of employees who make more than their manager**

```
select E.LastName
from Employee E, Department D, Employee Emgr
where E.WorkDept = D.DeptNo
   and D.MgrNo = Emgr.EmpNo
   and E.Salary > Emgr.Salary
```

NULL Values

- The value NULL can be assigned to an attribute to indicate unknown or missing data
- NULLs are a necessary evil – lots of NULLs in a database instance suggests poor schema design
- NULLs can be prohibited for certain attributes by schema constraints, **eg., NOT NULL, PRIMARY KEY**
- Predicates and expressions that involve attributes that may be NULL may evaluate to NULL

  - *x + y* evaluates to NULL if either *x* or *y* is NULL
  - *x > Y* evaluates to NULL if either *x* or *y* is NULL
  - How to test for NULL? Use is **NULL** or is **not NULL**
- Note:

SQL uses three valued logic: TRUE, FALSE, NULL

Logical Expressions in SQL

AND	TRUE	FALSE	NULL
TRUE	TRUE	FALSE	NULL
FALSE	FALSE	FALSE	FALSE
NULL	NULL	FALSE	NULL

OR	TRUE	FALSE	NULL
TRUE	TRUE	TRUE	TRUE
FALSE	TRUE	FALSE	NULL
NULL	TRUE	NULL	NULL

NOT	TRUE	FALSE	NULL
	FALSE	TRUE	NULL

Ordering Results

- No particular ordering on the rows of a table can be assumed when queries are written (This is important!)
- No particular ordering of rows of an intermediate results in the query can be assumed either
- However, it is possible to order the final result of query, using the **order by** clause

```
select distinct e.empno, emstdate, firstnme, lastname
from  emoloyee e, emp_act a
where e.empno = a.empno and a.projno = ‘PL2100’
order by emstdate
```

Aggregation Functions in SQL

- ***count(*):*** number of tuples in the group
- ***count(E):*** number of tuples for which E (an expression that may involve non-grouping attributes) is non-NULL
- ***count(distinct E):*** number of distinct non-NULL E values
- ***sum(E):*** sum of non-NULL E values
- ***sum(distinct E):*** sum of distinct non-NULL E values
- ***avg(E):*** average of non-NULL E values
- ***avg(distinct E):*** average of distinct non-NULL E values
- ***min(E):*** minimum of non-NULL E values
- ***max(E):*** maximum of non-NULL E values

Outer Joins

- **List the manager of each department. Include in the result departments that have no manger**

```
select deptno, deptname, lastname
from  department d left outer join employee e
    on d.mgrno = e.empno
where deptno like ‘D%’
```

Note:

SQL supports left, right, and full outer joins

Set Operations

- **SQL defines UNION, INTERSECT and EXCEPT operations (EXCEPT is set difference)**

```
select empno
from employee
except
select mgrno
from department
```

- **These Operations result in sets**
  - Q1 **UNION** Q2 includes any tuple that is found (at least once) in both Q1 or in Q2
  - Q1 **INTERSECT** Q2 includes any tuple that is found (at least once) in both Q1 and Q2
  - Q1 **EXCEPT** Q2 includes any tuple that is found (at least once) in both Q1 and is not found Q2

Multiset Operations

- SQL provides a multiset version of each of the set operations:
- **UNION ALL, INTERSECT ALL, EXCEPT ALL**
- Suppose Q_1 includes n_1 copies of some tuple *t,* and Q_2 includes n_2 copies of the same tuple *t*.
- Q_1 **UNION ALL** Q_2 will include n_1+n_2 copies of *t*
- Q_1 **INTERSECT ALL** Q_2 will include min (n_1, n_2) copies of *t*
- Q_1 **EXCEPT ALL** Q_2 will include max $(n_1 – n_2, 0)$ copies of *t*

SQL DML: Insertion & Deletion

- **Insert a single tuple into the Employee relation**

```
insert into Employee
values ('000350', 'Sheldon', 'Q', 'Jetstream', 'A00', 01/10/2000, 25000.00);
```

- **Delete all employee from the Employee table**

```
delete from Employee;
```

- **Delete all employees in department A00 from the Employee table**

```
delete from Employee
where WorkDept = 'A00';
```

SQL DML: Update

- **Increase the salary of every employee by five percent**

```
update Employee
set Salary = Salary * 1.05;
```

- **Move all employees in department E21 into department E01**

```
update Employee
set WorkDept = ‘E01’
where WorkDept = ‘E21’;
```