

Lecture 3 Part I

SQL: Defining & Basic Querying

SEQUEL Structured English QUery Language

D. D. Chamberlin, et al., “SEQUEL 2 A Unified Approach to Data Definition, Manipulation, and Control,” IBM J. Res. Develop., Nov. 1976, pp. 560-575.

SQL Structured Query Language

SQL - Parts of the Language

- Data Definition Language (DDL)
 - create table
 - create index
- Data Manipulation Language (DML)
 - select (retrieve)
 - update
 - insert
 - delete

DDL - Creating Tables - Employee

```
create table EMP
( EMPNO          integer      not null,
  NAME           varchar(40)  not null,
  DNO            integer,
  JOB            varchar(20),
  MGR            integer,
  SAL            decimal(10,2),
  COMMISSION    decimal (5,2),
  primary key (EMPNO),
  foreign key (DNO)      references DEPT,
  foreign key (MGR)     references EMP )
```

```
create DOMAIN job_type varchar(20)
```

SQL

DDL continued

```
create table DEPT  
( DNO          integer      not null,  
  DNAME        varchar(20)  not null,  
  LOC          varchar(20),  
  primary key (DNO) )
```

Add a new column for the number of employees, called NEMPS, of integer type, to the table DEPT.

```
alter table DEPT  
  add NEMPS integer
```

Drop the EMP table.

```
drop table EMP
```

DDL - Indexes

Create an index for location in DEPT.

```
create index XDEPTLOC on DEPT(LOC)
```

Create an index for EMPNO, the primary key, in EMP table.

```
create unique index EMP on EMP(EMPNO)
```

Drop the department location index.

```
drop index XDEPTLOC
```

SQL - Data Manipulation Language (DML)

- select (retrieve)
- update
- insert
- delete

Select - Basic Form

Cartesian product followed by select and project.

```
select    project-list  
from      Cartesian-product-list  
where     select-condition(s)
```

Abstract example: Given tables R(A,B) and S(B,C)

```
select    R.A, R.B, S.C  
from      R, S  
where     R.A > 10
```

BUT - Duplicates NOT eliminated

Select as a JOIN

Cartesian product followed by

select (“join” & “select” conditions) and project.

```
select    project-list
from     Cartesian-product-list
where    join-condition
and      select-condition
```

Abstract example: Given tables R(A,B) and S(B,C)

```
select   R.A, R.B, S.C
from     R, S
where    R.B = S.B    /* join condition */
and      R.A > 10    /* “select” condition */
```

Using EMP and DEPT

From Relational Algebra to SQL

- List the names, employee numbers, department numbers and locations for all clerks.

```
select  NAME, EMPNO, E.DNO, LOC
from    EMP E, DEPT D
where   E.DNO = D.DNO    /* join condition */
and     JOB = 'Clerk'    /* "select" condition */
```

Note use of alias in from clause.

- Duplicates in project - must use explicit **distinct**
- List the different department numbers in the EMP table (eliminate duplicates).

```
select distinct DNO  
from EMP
```

- Specify sort order

List employee number, name, and salary of employees in department 50.

```
select EMPNO, NAME, SAL  
from EMP  
where DNO = 50  
order by EMPNO
```

- **Union**

List the numbers of those departments which have an employee named ‘Smith’ or are located in ‘Columbus’.

```
select DNO
from EMP
where ENAME = ‘Smith’
union
select DNO
from DEPT
where LOC = ‘Columbus’
```

Duplicates ARE eliminated by default.

union all - leaves duplicates

Functions and Groups

- Find the average salary of clerks.

```
select  avg(SAL)
from    EMP
where   JOB = 'Clerk'
```

- How many different jobs are held by employees in department 50?

```
select  count(distinct JOB)
from    EMP
where   DNO = 50
```

Functions and Groups, continued

- List the departments (DNO) and the average salary of each.

```
select    DNO, avg(SAL)
from      EMP E, DEPT D
where     E.DNO = D.DNO
group by DNO
```

Groups (contd.)

- List the departments (DNO, DNAME) in which the average employee salary < \$25,000.

```
select DNO, DNAME
from EMP E, DEPT D
where E.DNO = D.DNO
group by DNO, DNAME
having avg(SAL) < 25000
```

- List departments that employ more than 10 clerks.

```
select DNO
from EMP
where JOB = 'Clerk'
group by DNO
having count(*) > 10
```

Nested Select: No analog in Relational Algebra

- List names of employees in departments 25, 47 and 53.

```
select NAME
from EMP
where DNO in (25, 47, 53)
```

- List names of employees who work in departments in Ann Arbor.

```
select NAME
from EMP
where DNO in (select DNO
              from DEPT
              where LOC = 'Ann Arbor' )
```

Big Summary

- For all departments in Columbus with average salary > \$25,000, list the department's number, name, and average salary ordered by average salary in descending order.

```
select DNO, DNAME, avg(SAL)
from EMP, DEPT
where EMP.DNO = DEPT.DNO
and LOC = 'Columbus'
group by DNO, DNAME
having avg(SAL) > 25000
order by 3 desc
```

Lecture 3 Part II

Advanced SQL Operators

Exists and Not Exists

- List names of employees who work in departments in Ann Arbor.

```
select NAME
from EMP
where exists
    ( select *
      from DEPT
      where EMP.DNO = DEPT.DNO
      and LOC = 'Ann Arbor' )
```

Exists and Not Exists, continued

- List names of employees who do not work in departments in Ann Arbor.

```
select NAME
from EMP
where not exists
      ( select *
        from DEPT
        where EMP.DNO = DEPT.DNO
          and LOC = 'Ann Arbor' )
```

Substrings in Queries

- List those departments whose names start with the letter C.

```
select DNAME  
from DEPT  
where DNAME like 'C%'
```

Arithmetic in Select Clause.

Renaming Attributes.

- Show the result of giving everyone in departments in Columbus a 10% pay raise.

```
select EMPNO, NAME, 1.1 * SAL as NEWSAL
from EMP, DEPT
where EMP.DNO = DEPT.DNO
and LOC = 'Columbus'
```

Other Data Manipulation Language (DML) Commands

- **Insert** employee named 'Jones' with employee number 535 in department 51.

Other attributes are null.

```
insert  
[into] EMP(EMPNO, NAME, DNO)  
values      (535, 'Jones', 51)
```

[optional]

- **Add** to the CANDIDATES relation all employees whose commission is greater than half their salary.

CANDIDATES(EMPNO, NAME, DNO, SAL)

```
insert  
[into] CANDIDATES  
select EMPNO, NAME, DNO, SAL  
from EMP  
where COMM > 0.5 * SAL
```

- **Delete** from EMP the employee with employee number 561.
delete EMP
where EMPNO = 561

- **Delete** from the DEPT table the departments having no employees.

```
delete DEPT  
where ( select count(*)  
from EMP  
where DNO = DEPT.DNO) = 0
```

A note on nested queries

```
delete DEPT
where (select count(*)
       from EMP
       where DNO = DEPT.DNO) = 0
```

```
delete DEPT
where DNO not in (select distinct (DNO)
                 from EMP )
```

- **2nd way is more efficient than 1st**

- **Update** the EMP table by giving a 10% raise to all those whose employee number appears in the CANDIDATES table.

```
update EMP
set SAL = SAL * 1.1
where EMPNO in
      (select EMPNO
       from CANDIDATES)
```

Null Values

All of the following conditions are always **false**.

 null > 25 null < 25 null = 25 null <> 25
 null >= 25 null <= 25 null = null null <> null

However we can use the following:

```
select  NAME  
from    EMP  
where   SAL < 35000  
or     SAL is null
```

Views

- In SQL
 - A named, derived table (a virtual table).
 - Derived from base tables and/or other views.
- In Three-schema Architecture (pp. 27-29)
 - External View
 - A collection of several tables, some views, other base tables.

- Create a view called D50 containing the employee number, name and job of those employees in department 50.

```
create view D50  
as select EMPNO, NAME, JOB  
from EMP  
where DNO = 50
```

Views: Changing Attribute Names

- Create a view called PROGS consisting of the EMPNO, name and salary of all programmers. Include the locations of their departments.

```
create view    PROG(EMPNO, NAME,  
                SALARY, HOMEBASE)  
as select    EMPNO, NAME, SAL, LOC  
from        EMP, DEPT  
where       EMP.DNO = DEPT.DNO  
and        EMP.JOB = 'Programmer'
```

Using and Dropping a View

- Using the PROGS view, find the average salary of programmers in Columbus.

```
select avg(SAL)  
from PROGS  
where HOMEBASE = 'Columbus'
```

- Drop the view PROGS.
drop view PROGS