

CSc 134

# Database Management and File Organization

## 6. SQL

Ying Jin

Computer Science Department

California state University, Sacramento

# SQL History

- ◆ SQL-86 (SQL 1)

- ◆ SQL-92 (SQL 2)

- ◆ SQL-99 (SQL 3)

- Core: supposed to be implemented by all RDBMS vendors
- Extension: optional modules such as data mining, spatial data, temporal data, data warehousing

- ◆ Later updates:

- 2003, 2006: Add XML features
- 2008: incorporated more object database features

# CREATE TABLE

- ◆ Specifies a new base relation by giving it a name, and specifying each of its attributes and their data types
- ◆ A constraint NOT NULL may be specified on an attribute

```
CREATE TABLE DEPARTMENT
(
    DNAME          VARCHAR(10) NOT NULL,
    DNUMBER        INTEGER      NOT NULL,
    MGRSSN         CHAR(9),
    MGRSTARTDATE   CHAR(9)
    ...);
```

# Attribute Data Types and Domains in SQL

## ◆ Numeric

- INTEGER or INT
- FLOAT or REAL
- DECIMAL(i,j), or DEC(i,j), or NUMERIC(i,j)
  - ◆ i: total number of decimal digits
  - ◆ j: number of digits after the decimal point

## ◆ Character-string

- fixed length
  - ◆ CHAR(n) or CHARACTER(n)
- varying length
  - ◆ VARCHAR(n)

# Attribute Data Types and Domains in SQL (Cont.)

## ◆ Boolean

- TRUE, FALSE

## ◆ Date

- DATE: year, month, day in the form YYYY-MM-DD
- TIME: hour, minute, second in the form HH:MM:SS

# CREATE TABLE (Cont.)

- ◆ Specify primary key
- ◆ Referential integrity constraints (foreign keys).
- ◆ Key attributes
  - PRIMARY KEY
  - UNIQUE phrases

```
CREATE TABLE DEPARTMENT
(  DNAME VARCHAR(10) NOT NULL,
   DNUMBER INTEGER NOT NULL CHECK
      (DNUMBER>0 AND DNUMBER <21),
   MGRSSN CHAR(9),
   MGRSTARTDATE DATE,
   PRIMARY KEY (DNUMBER),
   UNIQUE (DNAME),
   FOREIGN KEY (MGRSSN) REFERENCES EMPLOYEE(SSN) );
```

# REFERENTIAL INTEGRITY OPTIONS

◆ We can specify CASCADE, SET NULL or SET DEFAULT on referential integrity constraints

◆ CREATE TABLE EMPLOYEE

```
( ...  
  DNOINT NOT NULL DEFAULT 1,  
  ...  
  PRIMARY KEY (SSN),
```

```
  FOREIGN KEY (SUPERSSN) REFERENCES EMPLOYEE(SSN)  
  ON DELETE SET NULL  
  ON UPDATE CASCADE,
```

```
  FOREIGN KEY (DNO) REFERENCES DEPARTMENT(DNUMBER)  
  ON DELETE SET DEFAULT  
  ON UPDATE CASCADE );
```

# Giving Names to Constraints

```
CREATE TABLE EMPLOYEE
```

```
(  SSN CHAR(9),
```

```
...
```

```
CONSTRAINT EMPPK
```

```
PRIMARY KEY (SSN),
```

```
CONSTRAINT EMPDEPTFK
```

```
FOREIGN KEY(DNO) REFERENCES DEPARTMENT  
(DNUMBER) ON DELETE SET DEFAULT ON UPDATE  
CASCADE
```

```
...
```

```
)
```



# DROP TABLE

- ◆ Remove a relation (base table) and its definition
- ◆ The relation can no longer be used in queries, updates, or any other commands
- ◆ Example:  
DROP TABLE DEPENDENT RESTRICT;  
DROP TABLE DEPENDENT CASCADE;

# Drop Table (Cont.)

## ◆ Cascade

- All constraints (e.g. foreign key definitions in another relation) and views reference the table are dropped automatically from the schema.

## ◆ Restrict

- A table is dropped only if it is not referenced in any constraints.

# ALTER TABLE

## - Add column

- ◆ Add an attribute to one of the base relations
- ◆ New attribute=null automatically

◆ Example:

```
ALTER TABLE EMPLOYEE ADD JOB  
VARCHAR(12);
```

# ALTER TABLE

## - Drop column

◆ ALTER TABLE EMPLOYEE DROP ADDRESS CASCADE;

- All constraints and view that reference the column are dropped

◆ ALTER TABLE EMPLOYEE DROP ADDRESS RESTRICT

◆ ALTER TABLE DEPARTMENT ALTER MGRSSN DROP DEFAULT;

ALTER TABLE DEPARTMENT ALTER MGRSSN SET DEFAULT '122444444';

# ALTER TABLE

## - Drop constraints

◆ ALTER TABLE EMPLOYEE  
DROP CONSTRAINT EMPSUPERFK  
CASCADE;

# Queries

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

# EMPLOYEE

FNAME	MINIT	LNAME	<u>SSN</u>	BDATE	ADDRESS	SEX	SALARY	SUPERSSN	DNO
-------	-------	-------	------------	-------	---------	-----	--------	----------	-----

## DEPARTMENT

DNAME	<u>DNUMBER</u>	MGRSSN	MGRSTARTDATE
-------	----------------	--------	--------------

## DEPT\_LOCATIONS

<u>DNUMBER</u>	<u>DLOCATION</u>
----------------	------------------

## PROJECT

PNAME	<u>PNUMBER</u>	PLOCATION	DNUM
-------	----------------	-----------	------

## WORKS\_ON

<u>ESSN</u>	<u>PNO</u>	HOURS
-------------	------------	-------

## DEPENDENT

<u>ESSN</u>	<u>DEPENDENT_NAME</u>	SEX	BDATE	RELATIONSHIP
-------------	-----------------------	-----	-------	--------------

# Simple SQL Queries

Query 0: Retrieve the birthdate and address of the employee whose name is 'John B. Smith'.

**Q0:**

```
SELECT BDATE, ADDRESS
FROM   EMPLOYEE
WHERE  FNAME='John' AND MINIT='B'
      AND LNAME='Smith'
```

◆ SQL relation (table) is a *bag* of tuples; it *is not* a set of tuples.



# Simple SQL Queries (cont.)

- ◆ Query 1: Retrieve the name and address of all employees who work for the 'Research' department.

# Simple SQL Queries (cont.)

- ◆ Query 2: For every project located in 'Stafford', list the project number, the controlling department number, and the department manager's last name, and birthdate.

# Qualify attribute name

- ◆ Use the same name for two (or more) attributes as long as the attributes are in *different relations*
- ◆ *Qualify* the attribute name with the relation name by *prefixing* the relation name to the attribute name

Example:

- ◆ EMPLOYEE.LNAME, DEPARTMENT.DNAME

# ALIASES

- ◆ Query 8: For each employee, retrieve the employee's name, and the name of his or her immediate supervisor.

```
Q8: SELECT      E.FNAME, E.LNAME, S.FNAME,  
                S.LNAME  
FROM            EMPLOYEE E S  
WHERE           E.SUPERSSN=S.SSN
```

- ◆ We can think of E and S as two different copies of EMPLOYEE
  - E represents employees in role of *supervisees*
  - S represents employees in role of *supervisors*

# ALIASES (cont.)

- ◆ Aliasing can also be used in any SQL query for convenience
- ◆ Can also use the AS keyword to specify aliases

```
Q8: SELECT      E.FNAME, E.LNAME, S.FNAME,  
                S.LNAME  
      FROM      EMPLOYEE AS E,  
      EMPLOYEE AS S  
      WHERE     E.SUPERSSN=S.SSN
```

# UNSPECIFIED WHERE-clause

- ◆ *All tuples* of the relations in the FROM-clause are selected
- ◆ = WHERE TRUE
- ◆ Query 9: Retrieve the SSN values for all employees.

<b>Q9:</b>	<b>SELECT</b>	<b>SSN</b>
	<b>FROM</b>	<b>EMPLOYEE</b>

# UNSPECIFIED WHERE-clause (cont.)

◆ Example:

**Q10:            SELECT        SSN, DNAME  
                 FROM           EMPLOYEE, DEPARTMENT**

◆ *CARTESIAN PRODUCT* of employee and department is selected

# USE OF DISTINCT

◆ Q11:        **SELECT        SALARY**  
                 **FROM EMPLOYEE**

◆ Q11A:       **SELECT        **DISTINCT** SALARY**  
                 **FROM            EMPLOYEE**



# Set Operations

## ◆ UNION, EXCEPT, INTERSECT

- apply the operation have the same attributes
- attributes appear in the same order

## ◆ Result: sets of tuples

## ◆ UNION ALL, EXCEPT ALL, INTERSECT ALL: bags of tuples

# UNION Operation Example

- Make a list of all project numbers for projects that involve an employee whose last name is “Smith”, either as a worker or as a manager of the department that controls the project.

# ARITHMETIC OPERATIONS

◆ +, -, \*, /

◆ Query 27: Show the effect of giving all employees who work on the 'ProductX' project a 10% raise.

Q27:

```
SELECT      FNAME, LNAME, 1.1*SALARY AS INCREASED_SAL
FROM        EMPLOYEE, WORKS_ON, PROJECT
WHERE       SSN=ESSN AND PNO=PNUMBER AND
            PNAME='ProductX'
```

◆ Are the salaries different after execute the query?

# ORDER BY

- ◆ The **ORDER BY** clause is used to sort the tuples in a query result based on the values of some attribute(s)
- ◆ Query 28: Retrieve a list of employees and the projects each works in, ordered by the employee's department in a **descending order**, and within each department ordered alphabetically by employee last name, fname in an **ascending order**.

**Q28:**

```
SELECT      DNAME, LNAME, FNAME, PNAME
FROM        DEPARTMENT, EMPLOYEE,
            WORKS_ON, PROJECT
WHERE       DNUMBER=DNO AND SSN=ESSN AND
            PNO=PNUMBER
ORDER BY    DNAME DESC, LNAME ASC, FNAME ASC
```

# NESTING OF QUERIES

- ◆ Nested query
- ◆ Outer query
- ◆ Query 11: Retrieve the name and address of all employees who work for the 'Research' department.

```
SELECT      FNAME, LNAME, ADDRESS
FROM        EMPLOYEE
WHERE       DNO IN (SELECT      DNUMBER
                     FROM        DEPARTMENT
                     WHERE       DNAME='Research' )
```

# Comparison operators

- ◆ ANY ( or SOME)

- ◆ ALL

- ◆ List the names of employees whose salary is greater than the salary of all the employees in department 5.

# THE EXISTS FUNCTION

- ◆ Check whether the result of a correlated nested query is empty
- ◆ Query 12: Retrieve the name of each employee who has a dependent with the same first name as the employee.

# THE EXISTS FUNCTION (cont.)

- ◆ Retrieve the names of employees who have no dependents



# EXPLICIT SETS

- ◆ It is also possible to use an **explicit (enumerated) set of values** in the WHERE-clause rather than a nested query
- ◆ Query 13: Retrieve the social security numbers of all employees who work on project number 1, 2, or 3.

<b>Q13:</b>	<b>SELECT</b>	<b>DISTINCT ESSN</b>
	<b>FROM</b>	<b>WORKS_ON</b>
	<b>WHERE</b>	<b>PNO IN (1, 2, 3)</b>

# Renaming of Attributes

Rename an attribute that appears in the result of a query.

Q8A:Retrieve the last name of each employee and his or her supervisor, while renaming the resulting attribute names as Employee\_name and Supervisor\_name.

```
SELECT E.Iname AS employee_name, s.Iname AS  
superviosr_name  
FROM EMPLOYEE AS E, EMPLOYEE AS S  
WHERE E.super_ssn=S.ssn;
```

# Joined Tables

- ◆ Join operation in the FROM clause
- ◆ Separate the selection and join conditions in the where clause

```
SELECT      FNAME, LNAME, ADDRESS
FROM        EMPLOYEE, DEPARTMENT
WHERE       DNAME='Research' AND
           DNUMBER=DNO;
```

```
SELECT      FNAME, LNAME, ADDRESS
FROM        EMPLOYEE JOIN DEPARTMENT
           ON DNUMBER=DNO
WHERE       DNAME='Research';
```

# Outer join

- ◆ List all employee names and the departments they manage if they happen to manage a department; if they do not manage one, we can indicate it with a NULL value.

```
SELECT FNAME, LNAME, DNAME  
FROM EMPLOYEE LEFT OUTER JOIN  
DEPARTMENT ON SSN=MGR_SSN;
```

## Outer Join



FNAME	LNAME	DNAME
Lisa	Monroe	Administration
Rahim	Abdul	NULL
Lindsay	Fitzgerald	NULL
Louis	Duncan	NULL
Arnold	Chan	Research
Niko	Kurosawa	NULL
Claire	Prince	NULL
Scott	Cho	NULL
Mason	Cronkite	Marketing
Cindy	Rodriguez	NULL

## Inner Join

```
SELECT FNAME, LNAME, DNAME
FROM EMPLOYEE INNER JOIN
      DEPARTMENT ON SSN=MGRSSN;
```

FNAME	LNAME	DNAME
Lisa	Monroe	Administration
Arnold	Chan	Research
Mason	Cronkite	Marketing

# AGGREGATE FUNCTIONS

- ◆ Include **COUNT**, **SUM**, **MAX**, **MIN**, and **AVG**
- ◆ Query 15: Find the maximum salary, the minimum salary, and the average salary among all employees.

**Q15:**            **SELECT**            **MAX(SALARY),**  
   **MIN(SALARY), AVG(SALARY)**  
   **FROM**            **EMPLOYEE**

- ◆ **NOTE:** Some SQL implementations *may not allow more than one function* in the **SELECT**-clause

# AGGREGATE FUNCTIONS

(cont.)

- ◆ Query 16: Find the maximum salary, the minimum salary, and the average salary among employees who work for the 'Research' department.

# AGGREGATE FUNCTIONS

(cont.)

- ◆ Queries 17: Retrieve the total number of employees in the company

**Q17:       SELECT       COUNT (\*)  
              FROM       EMPLOYEE**

- ◆ Returns the number of rows in the result of the query



# AGGREGATE FUNCTIONS

(cont.)

- ◆ SELECT COUNT(DISTINCT SALARY)  
FROM EMPLOYEE;
- ◆ SELECT COUNT(SALARY)  
FROM EMPLOYEE;

# AGGREGATE FUNCTIONS

(cont.)

- ◆ (Q18) Retrieve the number of employees in the 'Research' department.

# AGGREGATE FUNCTIONS

(cont.)

Q5: retrieve the names of all employees who have two or more dependents.

# GROUPING

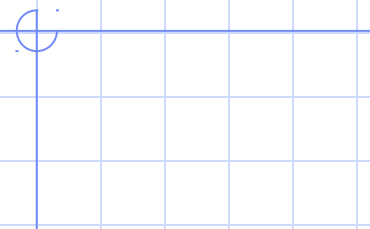
- ◆ apply the aggregate functions to **subgroups of tuples** in a relation
- ◆ Each subgroup of tuples consists of the set of tuples that have **the same value** for the **grouping attribute(s)**
- ◆ The function is applied to each subgroup independently
- ◆ SQL has a **GROUP BY**-clause for specifying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

# GROUPING

- ◆ Query 20: For each department, retrieve the department number, the number of employees in the department, and their average salary.

**Q20:**

```
SELECT      DNO, COUNT (*), AVG (SALARY)  
FROM        EMPLOYEE  
GROUP BY   DNO
```



FNAME	MINIT	LNAME	<u>SSN</u>	• • •	SALARY	SUPERSSN	DNO
John	B	Smith	123456789	• • •	30000	333445555	5
Franklin		Wong	333445555		40000	888665555	5
Ramesh	K	Narayan	666884444		38000	333445555	5
Joyce	A	English	453453453		25000	333445555	5
Alicia	J	Zelaya	999887777		25000	987654321	4
Jennifer	S	Wallace	987654321		43000	888665555	4
Ahmad	V	Jabbar	987987987		25000	987654321	4
James	E	Bong	888665555		55000	null	1

DNO	COUNT (*)	AVG (SALARY)
5	4	33250
4	3	31000
1	1	55000

Result of Q24.

# THE HAVING-CLAUSE

- ◆ Retrieve the values of these functions for only those groups *that satisfy certain conditions*
- ◆ The HAVING-clause
  - Specify a selection condition on groups (rather than on individual tuples)
- ◆ WHERE clause is executed before Having clause.

# THE HAVING-CLAUSE (cont.)

◆ Query 22: For each department which has *more than two employees*, retrieve the department number, the number of employees in the department, and their average salary.



# 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>]

# Specifying Updates in SQL

- ◆ There are three SQL commands to modify the database; INSERT, DELETE, and UPDATE

# INSERT

## ◆ Example 1:

```
INSERT INTO EMPLOYEE  
VALUES ('Richard','K','Marini', '653298653',  
       '30-DEC-52', '98 Oak Forest,Katy,TX', 'M',  
       37000,'987654321', 4 )
```

## ◆ Example 2:

```
INSERT INTO EMPLOYEE (FNAME, LNAME, SSN, DNO)  
VALUES ('Richard', 'Marini', '653298653', 4)
```

- ◆ Must include an attribute if the attribute is specified as NOT NULL and has no default value.
- ◆ Set to other attributes: DEFAULT, or NULL

# INSERT (cont.)

◆ CREATE TABLE DEPTS\_INFO  
(DEPT\_NAME VARCHAR(10),  
NO\_OF\_EMPS INTEGER,  
TOTAL\_SAL INTEGER);

◆ INSERT INTO DEPTS\_INFO (DEPT\_NAME,  
NO\_OF\_EMPS, TOTAL\_SAL)  
SELECT DNAME, COUNT (\*), SUM (SALARY)  
FROM DEPARTMENT, EMPLOYEE  
WHERE DNUMBER=DNO  
GROUP BY DNAME ;

# DELETE

◆ U4A:	DELETE FROM WHERE	EMPLOYEE LNAME='Brown'
U4B:	DELETE FROM WHERE	EMPLOYEE SSN='123456789'
U4C:	DELETE FROM WHERE (SELECT FROM DEPARTMENT WHERE	EMPLOYEE DNO IN DNUMBER DNAME='Research')
U4D:	DELETE FROM	EMPLOYEE

# UPDATE

◆ Example: Change the location and controlling department number of project number 10 to 'Bellaire' and 5, respectively.


**U5: UPDATE  
SET  
WHERE**

**PROJECT  
PLOCATION = 'Bellaire', DNUM = 5  
PNUMBER=10**

# UPDATE (cont.)

◆ Example: Give all employees in the 'Research' department a 10% raise in salary.

```
U6: UPDATE      EMPLOYEE
      SET        SALARY = SALARY *1.1
      WHERE      DNO IN (SELECT DNUMBER
                           FROM    DEPARTMENT
                           WHERE    DNAME='Research')
```



These slides are based on the textbook and the notes of:

R. Elmaseri and S. Navathe, *Fundamentals of Database Systems*, 6th Edition, Addison-Wesley. Chapter 4.