Introduction to SQL

CS3042 - Database Systems

Dr Gayashan Amarasinghe

Department of Computer Science and Engineering
University of Moratuwa

Overview

- Brief History
- SQL as a language
- Data Types in SQL
- CREATE TABLE construct and Integrity Constraints
- DROP and ALTER TABLE constructs
- SELECT clause
- WHERE clause
- FROM clause
- Joins
- Natural join
- Rename operation
- Ordering the results
- String operations
- Aggregate functions
- Nested Subqueries

Database System Concepts 6th Edition by Abraham Silberschatz, Henry F. Korth, and S. Sudarshan

History

- IBM Sequel language developed as part of System R project at the IBM San Jose Research Laboratory
- Renamed Structured Query Language (SQL)
- ANSI and ISO standard SQL:
  - SQL-86, SQL-89, SQL-92
  - SQL:1999, SQL:2003, SQL:2008
- Commercial systems offer most, if not all, SQL-92 features, plus varying feature sets from later standards and special proprietary features.
- Not all examples here may work on your particular system.

Structured Query Language

SQL is a...

- **Very high level** language
  - Works well because it is optimized well.
- **Data Definition Language**

  CREATE TABLE

  DROP TABLE

- **Data Manipulation Language**

  SELECT

  INSERT

  DELETE

  UPDATE

Data types in SQL

- **char(n).** Fixed length character string, with user-specified length *n*.
- **varchar(n).** Variable length character strings, with user-specified maximum length *n*.
- **int.** Integer (a finite subset of the integers that is machine dependent).
- **smallint.** Small integer (a machine-dependent subset of the integer domain type).
- **numeric(p,d).** Fixed point number, with user-specified precision of *p* digits, with *d* digits to the right of decimal point.
- **real, double precision.** Floating point and double-precision floating point numbers, with machine-dependent precision.
- **float(n).** Floating point number, with user-specified precision of at least *n* digits.

CREATE TABLE construct

- An sql relation is defined using **CREATE TABLE** construct

CREATE TABLE r(A_1 D_1, A_2 D_2, ..., A_n D_n,

(integrity-constraint$_1$),

...,

(integrity-constraint$_k$))

r - name of the relation
A_i - Attribute name in the schema of the relation
D_i - Data type of values in the domain of attribute A_i

Eg:

CREATE TABLE *instructor* (

ID **char**(5),
name **varchar**(20) **not null,**
dept_name **varchar**(20),
salary **numeric**(8,2))

Integrity constraints in CREATE TABLE

- **not null**
- **primary key** (A_1, A_2,..., A_n)
- **foreign key** (A_m,..., A_n) **references** r

  Eg: Declare ID as the primary key for the instructor table.

```
create table instructor (
        ID char(5),
        name varchar(20) not null,
        dept_name varchar(20),
        salary numeric(8,2),
        primary key (ID),
        foreign key (dept_name)
                references department(dept_name))
```

- **primary key** declaration on an attribute automatically ensures **not null**

DROP and ALTER TABLE constructs

- **DROP TABLE** student
  - Deletes the table and its content
- **DELETE FROM** student
  - Deletes all the content of the table, but retains the table
- **ALTER TABLE**
  - **ALTER TABLE** r **ADD** A D
    - where *A* is the name of the attribute to be added to relation *r* and *D* is the domain/data type of *A*.
    - All tuples in the relation are assigned *null* as the value for the new attribute.
  - **ALTER TABLE** r **DROP** A
    - where A is the name of an attribute in relation r.
    - Many databases do not support this functionality.

Basic Query Structure

- SQL is also a Data Manipulation language
- A typical SQL query has the form

  SELECT $A_1,A_2,\ldots,A_n$
  FROM $r_1,r_2,\ldots,r_m$
  WHERE P

  - A_i represents an attribute
  - r_j represents a relation
  - P is a predicate

- The result of a query is another relation

SELECT clause

- The **SELECT** clause list the attributes desired in the result of a query
  - corresponds to the projection operation of the relational algebra
- Eg: Find the names of all the departments with instructors

```
SELECT dept_name
FROM instructor
```

SQL names are case insensitive.
name ≡ Name ≡ NAME

- To force the elimination of duplicates, insert the keyword **DISTINCT** after **SELECT.**
- Eg:

```
SELECT DISTINCT dept_name
FROM instructor
```

SELECT clause

- An asterisk in the SELECT clause denotes “all attributes”

  Eg:

  SELECT * **FROM** *instructor*

- The **SELECT** clause can contain arithmetic expressions involving the operation, +, –, , and /, and operating on constants or attributes of tuples.

  Eg:

  SELECT *ID, name, salary/12*
  FROM *instructor*

  would return a relation that is the same as the *instructor* relation, except that the value of the attribute *salary* is divided by 12.

WHERE clause

- **WHERE** clause specifies conditions that the result must satisfy
  - Corresponds to the selection predicate of the relational algebra.
- To find all instructors in Comp. Sci. dept with salary > 80000

  SELECT *name*
  FROM *instructor*
  WHERE *dept_name* = ‘Comp. Sci.' **AND** *salary* > 80000

- Comparison results can be combined using the logical connectives **and, or,** and **not.**
- Comparisons can be applied to results of arithmetic expressions.

FROM clause

- **FROM** clause lists the relations involved in the query
  - Corresponds to the Cartesian product operation of the relational algebra.
- Find the Cartesian product *instructor X teaches*

  SELECT *
  FROM *instructor, teaches*

  - generates every possible instructor – teaches pair, with all attributes from both relations
- Cartesian product is not very useful directly, but useful combined with where-clause condition (selection operation in relational algebra)

Joins

- For all instructors who have taught some course, find their names and the course ID of the courses they taught.

SELECT *name, course_id*
FROM *instructor, teaches*
WHERE *instructor.ID = teaches.ID*

Cartesian product of instructor relation and teaches relation
instructor x teaches

- Find the course ID, semester, year and title of each course offered by the Comp. Sci. department

SELECT *section.course_id, semester, year, title*
FROM *section, course*
WHERE *section.course_id = course.course_id*
AND *dept_name =* 'Comp. Sci.'

Writing some queries

Instructor(ID, name, dept_name, salary)

ID	*name*	*dept_name*	*salary*
10101	Srinivasan	Comp. Sci.	65000
12121	Wu	Finance	90000
15151	Mozart	Music	40000
22222	Einstein	Physics	95000
32343	El Said	History	60000

Teaches(ID, course_id, sec_id, semester, year

ID	*course_id*	*sec_id*	*semester*	*year*
10101	CS-101	1	Fall	2009
10101	CS-315	1	Spring	2010
10101	CS-347	1	Fall	2009
12121	FIN-201	1	Spring	2010
15151	MU-199	1	Spring	2010
22222	PHY-101	1	Fall	2009

- Write the SQL query and a sample output.

```
select name
  from instructor
  where dept_name = 'Music'
> Mozart
```

1. Find all the names of instructors in the Music Dept.
2. Find all the names of instructors who make more than 50000
3. Find all the names of instructors who are in the Music dept. and make more than 60000
4. Find all the course id's of courses taught in the Fall semester in year 2009.
5. Find all the names of instructors and the corresponding course id that they teach.
6. Find all the course id's taught by "Mozart" in Fall 2011.

Natural Join

- **NATURAL JOIN** matches tuples with the same values for all common attributes, and retains only one copy of each common column.

SELECT *

ID	*name*	*dept_name*	*salary*	*course_id*	*sec_id*	*semester*	*year*
10101	Srinivasan	Comp. Sci.	65000	CS-101	1	Fall	2009
10101	Srinivasan	Comp. Sci.	65000	CS-315	1	Spring	2010
10101	Srinivasan	Comp. Sci.	65000	CS-347	1	Fall	2009
12121	Wu	Finance	90000	FIN-201	1	Spring	2010
15151	Mozart	Music	40000	MU-199	1	Spring	2010
22222	Einstein	Physics	95000	PHY-101	1	Fall	2009
32343	El Said	History	60000	HIS-351	1	Spring	2010
45565	Katz	Comp. Sci.	75000	CS-101	1	Spring	2010
45565	Katz	Comp. Sci.	75000	CS-319	1	Spring	2010
76766	Crick	Biology	72000	BIO-101	1	Summer	2009
76766	Crick	Biology	72000	BIO-301	1	Summer	2010

Natural Join example

- List the names of instructors along with the course ID of the courses that they taught.
  - Without NATURAL JOIN

```
SELECT name, course_id
FROM instructor, teaches
WHERE instructor.id = teaches.id
```

  - With NATURAL JOIN

```
SELECT name, course_id
FROM instructor NATURAL JOIN teaches
```

Be cautious with Natural Join

- Dangers in NATURAL JOIN
  - beware of unrelated attributes with same name which get equated incorrectly
- List the names of instructors along with the the titles of courses that they teach
  - Incorrect version (makes course.dept_name = instructor.dept_name)

    SELECT *name*, *title*
    FROM *instructor* **NATURAL JOIN** *teaches* **NATURAL JOIN** *course*

    - Correct version

      SELECT *name*, *title*
      FROM *instructor* **NATURAL JOIN** *teaches*, *course*
      WHERE *teaches.course_id* = *course.course_id*

    - Another correct version

      SELECT *name*, *title*
      FROM (*instructor* **NATURAL JOIN** *teaches*) **JOIN** *course* **USING**(*course_id*)

Rename operation

- SQL allows renaming relations and attributes using the **AS** clause

  SELECT ID, name, salary/12 **AS** monthly_salary
  FROM instructor

- Find the names of all instructors who have a higher salary than some instructor in ‘Comp. Sci’.

  SELECT DISTINCT T. name
  FROM instructor **AS** T, instructor **AS** S
  WHERE T.salary > S.salary **AND** S.dept_name = ‘Comp. Sci.’

- Keyword **AS** is optional and can be ommitted.

  SELECT ID, name, salary/12 monthly_salary
  FROM instructor

Ordering the results

- List in alphabetic order the names of all instructors

  SELECT DISTINCT *name*
  FROM *instructor*
  ORDER BY *name*

Ascending order is the default.

- We may specify **DESC** for descending order or **ASC** for ascending order, for each attribute.

  Example: **ORDER BY** *name* **DESC**

- Can sort on multiple attributes

  Example: **ORDER BY** *dept_name, name*

String operations

- **LIKE** operator can be used for string matching

```
SELECT name
FROM instructor
WHERE name LIKE “%dar%”
```

%	matches any substring
_	matches any character
\	escape character for % or _

- Patterns are case sensitive
- SQL supports a variety of string operations such as
  - concatenation (using “||”)
  - converting from upper to lowercase (and vice versa)
  - finding string length, extracting substrings, etc.

WHERE clause predicates

- SQL includes a **BETWEEN** comparison operator

  Example: Find the names of all instructors with salary between $90,000 and $100,000 (that is, $90,000 and $100,000)

  SELECT *name*
  FROM *instructor*
  WHERE *salary* **BETWEEN** 90000 **AND** 100000

- Tuple comparison

  SELECT *name*, *course_id*
  FROM *instructor*, *teaches*
  WHERE (*instructor.ID*, *dept_name*) = (*teaches.ID*, ‘Biology’)

Aggregate functions

- These functions operate on the multiset of values of a column of a relation, and return a value
  - **AVG:** average value
  - **MIN:** minimum value
  - **MAX:** maximum value
  - **SUM:** sum of values
  - **COUNT:** number of values

- Find the average salary of instructors in the Computer Science department

```
SELECT AVG(salary)
FROM instructor
WHERE dept_name = 'Comp. Sci.';
```

Aggregate functions

- Find the total number of instructors who teach a course in the Spring 2010 semester

  SELECT COUNT (**DISTINCT** *ID*)
  FROM *teaches*
  WHERE *semester* = ’Spring’ **AND** *year* = 2010

- Find the number of tuples in the *course* relation

  SELECT COUNT (*)
  FROM *course*;

Aggregate functions - GROUP BY clause

- Find the average salary of instructors in each department

```
SELECT dept_name, AVG(salary)
FROM instructor
GROUP BY dept_name
```

- Attributes in **SELECT** clause outside of aggregate functions must appear in **GROUP BY** list

```
/* erroneous query */
SELECT dept_name, ID, AVG(salary)
FROM instructor
GROUP BY dept_name
```

Aggregate functions - HAVING clause

- Find the names and average salaries of all departments whose average salary is greater than 42000

SELECT *dept_name*, **AVG**(*salary*)
FROM *instructor*
GROUP BY *dept_name*
HAVING AVG(*salary*) > 42000

Predicates in the **HAVING** clause are applied after the formation of groups whereas predicates in the **WHERE** clause are applied before forming groups.

Aggregate functions - NULL values

- All aggregate operations **except COUNT(*)** ignore tuples with null values on the aggregated attributes
- What if collection has only null values?
  - count returns 0
  - all other aggregates return null

Eg: Total salary of all the instructors

SELECT SUM(salary)
FROM instructor

- this statement ignores null values
- result becomes null, if there are only null values in salary column.

Nested subqueries

- SQL provides a mechanism for the nesting of subqueries.
- A **subquery** is a **select-from-where** expression that is nested within another query.
- A common use of subqueries is to perform tests for set membership, set comparisons, and set cardinality.

Eg: Find courses offered in Fall 2009 and in Spring 2010

```
SELECT DISTINCT course_id
FROM section
WHERE semester = 'Fall' AND year= 2009 AND
        course_id IN (SELECT course_id
                      FROM section
                      WHERE semester = 'Spring'
                            AND year= 2010)
```

Nested subqueries

Find courses offered in Fall 2009 but not in Spring 2010

```
SELECT DISTINCT course_id
FROM section
WHERE semester = 'Fall' AND year= 2009 AND
        course_id NOT IN (SELECT course_id
                          FROM section
                          WHERE semester = 'Spring'
                                AND year= 2010)
```

Thank you!

Practice task

Employee(emp_no, emp_name, emp_city, …….)

Assignment(proj_no, emp_no, hours,…….)

Project(proj_name, budget, proj_no, proj_start_date, proj_end_date, proj_location,……..)

Express each of the following queries in SQL statements:

1. List the name(s) and budget(s) of projects started before 1st May 2008.
2. List the name(s) of projects with a budget value above Rs. 1,000,000.
3. Find the name(s) of employees who are from city “Moratuwa”.
4. Find the name(s) of employees who are from city “Moratuwa” and work on projects located in “Moratuwa”.
5. Find the name(s) of employees who work on projects valued above Rs. 1,000,000.