Introduction to SQL

Tags	
# Chapter	3
Date	@October 1, 2023
Person	 Tahsin Islam

Overview of The SQL

1. SQL Classification

SQL can be categorised into 2 major classes:

i. Data-definition Language (DDL)

- Provides commands for
  - Create table
  - Drop table
  - Modify table

ii. Data-manipulation Language (DML)

- Provides ability to query database
  - Update
  - Insert
  - Delete

2. Data Definition Language

It allows the specification of information about relations including:

- The schema for each relation.
- The type of values associated with each attribute.
- The Integrity constraints (such as primary constraint and others)
- The set of indices to be maintained for each relation.
- Security and authorization information for each relation.
- The physical storage structure of each relation on disk.

3. Domain Types in SQL

- char(n)
- varchar(n)
- int
- smallint
- numeric(p,d)
- real, double precision
- float(n)

Oracle Implementation

- varchar2(n)
- number(p,s)
- Date

4. Create Table Construct

```
CREATE TABLE table_name
(
attribute1 datatype [ NULL | NOT NULL ],
attribute2 datatype [ NULL | NOT NULL ],
...,
[CONSTRAINT constraint_name] PRIMARY KEY (primary_attribute1, ...),
[CONSTRAINT constraint_name] FOREIGN KEY (foreign_attribute1, ...)
REFERENCES reference_table_name [ON DELETE CASCADE | SET NULL | SET DEFAULT]
[ON UPDATE CASCADE | SET NULL | SET DEFAULT]
);
```

- **r** → name of relation
- **Ai** → attribute
- **Di** → Data type
- Integrity constraints:
  - primary key (A1, …, An)
  - foreign key (Am, …, An) references r

- not null

Example:

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

  constraint pk_instructor primary key (ID),
  constraint fk_instructor_dept foreign key (dept_name) references department
);
```

5. Dropping Table

1. To delete the schema, we use: (structure and information)

   `DROP TABLE table_name ;`

2. To delete the table with constraints:

   `DROP TABLE table_name CASCADE CONSTRAINTS;`

   Example:

   `DROP TABLE instructor ;`

6. Alter Table

`ALTER TABLE table_name ADD attribute_name datatype;`

Example:

`alter table instructor add address varchar2( 40 ) ;`

Basic Structure of SQL Queries

1. Single Relation

- Three clauses of Basic SQL Structure:
  - SELECT,

- FROM
- WHERE

```
select id,name, salary
from instructors
where dept_name='CSE' or dept_name='EEE';
```

- Select all can be expressed by the `*` sign as follows

```
SELECT * FROM STUDENTS;
```

- Keyword **DISTINCT** is used to eliminate duplicate records (not duplicate column)

```
SELECT DISTINCT DEPT, NAME FROM STUDENTS;
```

2. Multiple Relations

Queries on a Cartesian Product. Remember it is all possible combinations between 2 relations.

```
create table dept(
  dept_name varchar2(10),
  location varchar2(10),
  year_est number(4,0),
  yearly_budget number(10,2),
  constraint pk_dept primary key (dept)
);
create table students(
  name varchar2(20),
  prog varchar2(20),
  dob date,
  cgpa number,
  dept varchar(20),
  constraint fk_students foreign key (dept) references dept(dept)
);


SELECT * FROM dept, students;
```

The Rename Operation

- Renaming in SQL is very useful in a number of cases:
  - attribute name,
  - expression,
  - relation name
  - entire query
  - compare tuples in the same relation (Self-reference)

Renaming Attribute:

```
select name as instructor_name,
course_id
from instructor, teaches
where instructor.ID = teaches.ID
```

Renaming Expression

```
select ID, name, (salary/12) MSalary
from emp;
```

Renaming Relation

Renaming Relation is called **Correlation Name** in the SQL standard, but it is also commonly referred to as a **Table Alias**

```
select name as instructor_name,
course_id
from instructor I, teaches T
where I.ID = T.ID
```

Renaming Query

```
select A.N as Name, A.CID as Course
from
(select name as N, CourseID as CID
from instructor I, teaches T
where I.ID = T.ID) A;
```

Rename for Self-reference

```
select E1.ID, E1.Name, E1.BID
from Emp E1, Emp E2
where E1.BID = E2.ID;
```

String Operations

- Two Special characters:
  1. percent(%) → matches **any substring**
  2. underscore (_) → matches **any character**
- equal (=) → **exact** match.
  - Single quote (‘ ‘) is used for string matching
  - backslash (\ ) as escape character
- Pattern matching examples:
  - ’Intro%’ matches any string beginning with "Intro".
  - ’%Comp%’ matches any string containing “Comp” as a substring.
  - ’_ _ _’ matches any string of exactly three characters.
  - ’%m%’ matches any string containing letter m anywhere.
- SQL supports a variety of string operations such as:
  - concatenation (using || sign)
  - Many built-ins are normally used (i.e. UPPER, LOWER, SUBSTR..)

Set Operation

- Used to extract data from two relations having no references
- Useful to merge data from different sources where conditions vary in each case.
- Alternate way is using **AND, OR, NOT** operators

Example:

```
(select course_id from section where sem = 'Fall' and year = 2017)
union
```

```
(select course_id from section where sem = 'Spring' and year = 2018)
```

```
(select course_id from section where sem = 'Fall' and year = 2017)
intersect
(select course_id from section where sem = 'Spring' and year = 2018)
```

Null Values

A special problem in relational operations, including arithmetic operations, comparison operations, and set operations.

Solution: use `nvl()` built-in

Example:

```
select Name, (Salary + nvl(bonus,0)) Total
from emp;
```

Now null values will be treated as 0

Aggregate Functions

Take a collection of values and return a single value

- Average: avg (must be numbers)
- Minimum: min
- Maximum: max
- Total: sum (must be numbers)
- Count: count

Examples:

```
select avg(salary)
from instructor
where dept_name = 'Comp. Sci.';
```

```
select count(distinct ID)
from teaches
```

```
where semester = 'spring' and year = 2018;
```

```
select count(*) from course
```

Aggregation with Grouping

Tuples with the same value on all attributes in the **group by** clause are placed in one group

```
select department, avg(salary) as avg_salary
from instructor
group by department;
```

Basic Principle

All selected attributes must be present either in

1. aggregate functions
2. group by clause

Having Clause

- `where` clause can only be applied on the values already stored
- `having` clause the values are not stored rather will be computed on-the-fly

> `HAVING` is used to filter groups after they have been formed

```
select dname, max(budget), coount (*) totalstudent
from depts, students
where depts.dname = students.dept
group by dept
having count (*) >= 200;
```

> predicates in the having clause are applied after the formation of groups
> predicates in the where clause are applied before forming groups

Nested Sub-query

> A subquery is a select-from-where expression that is nested within another query

It can happen in 2 places:

1. In `WHERE` clause (nested subquery)
2. In `FROM` clause (inline view)

Subquery in WHERE clause

```
select id, name, salary
from emp
where salary > (select avg(salary) from emp);
```

- Here subquery in the where part is executed first and the value is locked.

Set Membership

`IN` and `NOT IN` are normally used in the where clause as a mechanism of set-membership

```
select distinct course_id
from section
where semester = 'Fall' and year = 2017 and
course_id in (select course_id
from section
where semester = 'Spring' and year = 2018);
```

Subquery in FORM Clause

```
select ROWNUM as RANK, lastn_name salary
from
(
  select last_name, salary from employees order by salary desc
```

```
)
where ROWNUM ,= 5;
```

DML Statements

There are 3 DML Statements

1. `INSERT`: new record entry
2. `UPDATE`: existing record modification
3. `DELETE`: Existing record remove

1. Insert

General Insert

- Add a new tuple to course:

```
Method-1:
INSERT INTO COURSE VALUES('CS-437', 'Database Systems', 'Comp. Sci.', 4);

Method-2:
INSERT INTO COURSE (course_id, title, dept_name, credits)
VALUES('CS-437', 'Database Systems', 'Comp. Sci.', 4);
```

- Add a new tuple to student with tot_creds set to null

```
INSERT INTO STUDENT
VALUES ('3003', 'Green', 'Finance', null);
```

Using Query

- Make each student in the Music department who has earned more than 144 credit hours an instructor in the Music department with a salary of $18, 000 (fixed values can be passed as well).

```
INSERT INTO INSTRUCTOR
SELECT ID, name, dept_name, 18000
```

```
FROM STUDENT
WHERE dept_name = 'Music' and total_cred > 144;
```

2. Update Statement

Simple Update

- Give a 5% salary raise to those instructors who earn less than 70000

```
update instructor set salary = salary * 1.05 where salary < 70000;
```

- Give a 5% salary raise to instructors whose salary is less than average

```
update instructor set salary = salary * 1.05
where salary < (select avg(salary) from instructor);
```

Compound Update

- Increase salaries of instructors whose salary is over $100, 000 by 3%, and all others by a 5%

```
update instructor set salary = salary * 1.03 where salary > 100000;
update instructor set salary = salary * 1.05 where salary <= 100000;
```

- Using case

```
update instructor set salary = case
when salary <= 100000 then salary * 1.05
else salary * 1.03 end;
```

3. Delete Statement

- Delete all instructors:

  `Delete [from] instructors;`

- Delete all tuples in the instructor relation for those instructors associated with a department located in the Watson building

```
delete from instructor
where dept_name in (select dept_name from department where building = 'Watson');
```

- Delete all instructors whose salary is less than the average salary of instructors

```
delete from instructor where salary < (select avg(salary) from instructor);
```