Join/Subquery/View

Sang Shin
www.javapassion.com
“Learning is fun!”

Topics

- Join
- Table relationship
  - Primary key and foreign key
  - Types of relationship
  - Referential integrity
  - Automatic delete and update
- Union
- Subquery
- View

Join

What is Join?

- A SQL JOIN clause combines records from two or more tables to produce result set
  - A JOIN is a means for combining fields from two tables by using values common to each.
- Specified in WHERE clause
- The joined tables are typically related with foreign keys

Types of Join

- Cross join
- Inner join
- Outer join
- Self join

Join:
Cross Join

Cross Join

- Matches each row from one table to every row from another table
  - Cartesian Product
  - Very costly (cpu time)
  - May take a while to return large result set - Suppose tables A and B each has 1000 records, the cartesian product will be 1000,000
  - Rarely used in production environment
- Default
  - If you do not specify how to join rows from different tables, the database server assumes you want Cross Join

Cross Join Examples

```
/* Cross Join Option #1 */
SELECT 'Cross Join', e.ename, e.salary, d.dname
FROM employees AS e, departments AS d;

/* Cross Join Option #2 */
SELECT 'Cross Join', e.ename, e.salary, d.dname
FROM employees AS e CROSS JOIN departments AS d;
```

Test Tables

```
/* Let's assume we have two tables */

+-------------------+----------------+
| department_id | dname          |
+-------------------+----------------+
|                         1 | Engineering |
|                         2 | Sales          |
|                         3 | Marketing    |
|                         4 | HR              |
+-------------------+----------------+
+---------------+----------------+-------------------+--------------+
| employee_id | enam            | department_id | salary      |
+---------------+----------------+-------------------+---------------+
|                    1 | jack            |                         1 | 3000.00 |
|                    2 | mary           |                          2 | 2500.00 |
|                    3 | nichole        |                          1 | 4000.00 |
|                    4 | angie          |                          2 | 5000.00 |
|                    5 | jones          |                         3 | 5000.00 |
|                    6 | newperson |                   NULL | 5000.00 |
+---------------+----------------+-------------------+-----------+
```

Cross Join Result Set

```
+------------+-----------+---------+-------------+
| Cross Join | ename     | salary  | dname       |
+------------+-----------+---------+-------------+
| Cross Join | jack      | 3000.00 | Engineering |
| Cross Join | jack      | 3000.00 | Sales       |
| Cross Join | jack      | 3000.00 | Marketing   |
| Cross Join | jack      | 3000.00 | HR          |
| Cross Join | mary      | 2500.00 | Engineering |
| Cross Join | mary      | 2500.00 | Sales       |
| Cross Join | mary      | 2500.00 | Marketing   |
| Cross Join | mary      | 2500.00 | HR          |
| Cross Join | nichole   | 4000.00 | Engineering |
| Cross Join | nichole   | 4000.00 | Sales       |
| Cross Join | nichole   | 4000.00 | Marketing   |
| Cross Join | nichole   | 4000.00 | HR          |
| Cross Join | angie     | 5000.00 | Engineering |
| Cross Join | angie     | 5000.00 | Sales       |
| Cross Join | angie     | 5000.00 | Marketing   |
| Cross Join | angie     | 5000.00 | HR          |
| Cross Join | jones     | 5000.00 | Engineering |
| Cross Join | jones     | 5000.00 | Sales       |
| Cross Join | jones     | 5000.00 | Marketing   |
| Cross Join | jones     | 5000.00 | HR          |
| Cross Join | newperson | 5000.00 | Engineering |
| Cross Join | newperson | 5000.00 | Sales       |
| Cross Join | newperson | 5000.00 | Marketing   |
| Cross Join | newperson | 5000.00 | HR          |
+------------+-----------+---------+-------------+
24 rows in set (0.00 sec)
```

Join:
Inner Join

Inner Join

- Most common type of Join
- Require a match in each table
- Rows that do not match are excluded from the result set
- The most common type of Inner Join is “equi-join” where certain fields of the joined tables are equated to each other using equality (=) operator

Inner Join Examples

```
/* The following Inner Join statements are equivalent */

/* Inner Join Option #1 */
SELECT 'Inner Join', employees.ename, employees.salary, departments.dname
FROM employees, departments
WHERE employees.department_id=departments.department_id;

/* Inner Join Option #2 */
SELECT 'Inner Join', employees.ename, employees.salary, departments.dname
FROM employees
JOIN departments
WHERE employees.department_id=departments.department_id;

/* Inner Join Option #3 */
SELECT 'Inner Join', employees.ename, employees.salary, departments.dname
FROM employees
INNER JOIN departments
WHERE employees.department_id=departments.department_id;

/* Inner Join Option #4 */
SELECT 'Inner Join', employees.ename, employees.salary, departments.dname
FROM employees
INNER JOIN departments
ON employees.department_id=departments.department_id;
```

Inner Join Result Set

```
+------------+-----------+------------+-------------+
| Inner Join | ename     | salary     | dname       |
+------------+-----------+-----------+-------------+
| Inner Join | jack      | 3000.00 | Engineering |
| Inner Join | nichole   | 4000.00 | Engineering |
| Inner Join | mary      | 2500.00 | Sales       |
| Inner Join | angie     | 5000.00 | Sales       |
| Inner Join | jones     | 5000.00 | Marketing   |
+-----------+---------+---------+------------+
5 rows in set (0.00 sec)
```

Join:
Outer Join

Outer Join

- All records from one side of the Join are included in the result set regardless of whether they match records on the other side of the Join
- LEFT JOIN or RIGHT JOIN depending which side of the Join is “all included”
  - LEFT JOIN: All records (actually fields of the records) of the table on the left side of the Join will be included
  - RIGHT JOIN: All records (actually fields of the records) of the table on the right side of the Join will be included

OUTER LEFT JOIN Example

```
/* Outer Join could be either LEFT JOIN or RIGHT JOIN */

/* Outer Join #1 - LEFT JOIN */
/* All records (actually fields of the records) of the "employees" table
 * are included in the result set because the "employees" table is
 *  left side of the JOIN */
SELECT 'Outer Join - LEFT JOIN ', employees.ename, employees.salary,
departments.dname
FROM employees
LEFT JOIN departments
ON employees.department_id=departments.department_id;
```

OUTER LEFT JOIN Result Set

```
// Notice that all records (actually fields of the records) of employees
// table are included in the result set regardless of the match because
// employees table is the left side of the outer left join.
+----------------------------+---------------+-----------+----------------+
| Outer Join - LEFT JOIN  | ename         | salary    | dname          |
+----------------------------+---------------+-----------+----------------+
| Outer Join - LEFT JOIN  | jack          | 3000.00 | Engineering |
| Outer Join - LEFT JOIN  | mary          | 2500.00 | Sales          |
| Outer Join - LEFT JOIN  | nichole       | 4000.00 | Engineering |
| Outer Join - LEFT JOIN  | angie         | 5000.00 | Sales          |
| Outer Join - LEFT JOIN  | jones         | 5000.00 | Marketing     |
| Outer Join - LEFT JOIN  | newperson | 5000.00 | NULL           |
+----------------------------+---------------+------------+--------------+
6 rows in set (0.00 sec)
```

OUTER RIGHT JOIN Examples

```
/* Outer Join could be either LEFT JOIN or RIGHT JOIN */

/* Outer Join #2 - RIGHT JOIN */
/* All records (actually fields of the records) of the "departments" table
 * are included in the result set because the "departments" table is
 * right side of the JOIN */
SELECT 'Outer Join - RIGHT JOIN', employees.ename, employees.salary,
departments.dname
FROM employees
RIGHT JOIN departments
ON employees.department_id=departments.department_id;
```

OUTER RIGHT JOIN Result Set

```
// Notice that all records (actually fields of the records) of departments
// table are included in the result set regardless of the match because
// the departments table is the right side of the outer right join.
+---------------------------+--------------+-----------+---------------+
| Outer Join - RIGHT JOIN | ename        | salary    | dname         |
+---------------------------+--------------+-----------+---------------+
| Outer Join - RIGHT JOIN | jack          | 3000.00 | Engineering |
| Outer Join - RIGHT JOIN | nichole       | 4000.00 | Engineering |
| Outer Join - RIGHT JOIN | mary          | 2500.00 | Sales        |
| Outer Join - RIGHT JOIN | angie         | 5000.00  | Sales        |
| Outer Join - RIGHT JOIN | jones         | 5000.00  | Marketing   |
| Outer Join - RIGHT JOIN | NULL          |    NULL    | HR            |
+--------------------------+--------------+-----------+---------------+
6 rows in set (0.00 sec)
```

Table Relationship:

Primary key and Foreign key

Primary key and Foreign key

- A primary key is a field or combination of fields that uniquely identify a record (row) in a table
- A foreign key (sometimes called a referencing key) is a key used to link two tables together
- Typically you take the primary key field from one table and insert it into the other table where it becomes a foreign key (it remains a primary key in the original table).

Primary key and Foreign key Example

```
/* Create departments table */
CREATE TABLE departments (
    department_id int(11) NOT NULL AUTO_INCREMENT,
    dname varchar(255) NOT NULL,
    PRIMARY KEY  (department_id)
) ENGINE=InnoDB;


/* Create "employees" table with FOREIGN KEY */
 CREATE TABLE employees (
    employee_id int(11) NOT NULL AUTO_INCREMENT,
    ename varchar(255) NOT NULL,
    d_id int(11) NOT NULL,
    salary decimal(7,2) NOT NULL,
    PRIMARY KEY  (employee_id),
    FOREIGN KEY (d_id) REFERENCES departments (department_id)
) ENGINE=InnoDB;
```

Table Relationship:
Types of relationship

Types of Relationship

- One-to-one (1-1)
- One-to-many (1-n)
- Many-to-many (n-m)

One-to-One Relationship

- A person has only one primary address field
- “person” table has 1-1 relationship with “primary-address” table
- The “primary-address” table has a foreign key field referring to the primary key field of the “person” table

One-to-One Relationship Example

```
/* Create "person" table */
CREATE TABLE person (
    person_id INT NOT NULL AUTO_INCREMENT,
    pname varchar(255) NOT NULL,
    PRIMARY KEY  (person_id)
) ENGINE=InnoDB;


/* Create "primary_address" table with FOREIGN KEY */
 CREATE TABLE primary_address (
    primary_address_id INT NOT NULL,
    address varchar(255) NOT NULL,
    p_id INT NOT NULL,
    PRIMARY KEY  (primary_address_id),
    FOREIGN KEY (p_id) REFERENCES person (person_id)
) ENGINE=InnoDB;
```

One-to-One Relationship Example

```
+-----------+-------------+
| person_id | pname       |
+-----------+-------------+
|         1 | Sang Shin   |
|         2 | Casey Jones |
|         3 | Bull Fighter |
|         4 | Passion You |
+-----------+--------------+

+--------------------+--------------------+------+
| primary_address_id | address            | p_id |
+--------------------+--------------------+------+
|                 11 | 11 dreamland       |    1 |
|                 12 | 5 king road        |    2 |
|                 13 | 67 nichole st      |    3 |
|                 14 | 32 Washington st   |    4 |
+------------------+----------------------+------+
```

One-to-Many (1-n) Relationship

- A department has many employees and an employee belongs to only a single department
- “department” table has 1-n relationship with “employee” table
- The “employee” table has a foreign key field referring to the primary key field of the “department” table

One-to-Many Relationship Example

```
/* Create departments table */
CREATE TABLE departments (
    department_id int(11) NOT NULL AUTO_INCREMENT,
    dname varchar(255) NOT NULL,
    PRIMARY KEY  (department_id)
) ENGINE=InnoDB;


/* Create "employees" table with FOREIGN KEY */
 CREATE TABLE employees (
    employee_id int(11) NOT NULL AUTO_INCREMENT,
    ename varchar(255) NOT NULL,
    d_id int(11) NOT NULL,
    salary decimal(7,2) NOT NULL,
    PRIMARY KEY  (employee_id),
    FOREIGN KEY (d_id) REFERENCES departments (department_id)
) ENGINE=InnoDB;
```

One-to-Many Relationship Example

```
+---------------+-------------+
| department_id | dname       |
+---------------+-------------+
|             1 | Engineering |
|             2 | Sales       |
|             3 | Marketing   |
|             4 | HR          |
+---------------+-------------+

+-------------+---------+------+---------+
| employee_id | ename   | d_id | salary  |
+-------------+---------+------+---------+
|           1 | jack    |    1 | 3000.00 |
|           2 | mary    |    2 | 2500.00 |
|           3 | nichole |    1 | 4000.00 |
|           4 | angie   |    2 | 5000.00 |
|           5 | jones   |    3 | 5000.00 |
+-------------+---------+------+---------+
```

Many-to-Many (n-m) Relationship

- A student takes many courses and each course has many students
- “student” and “course” has m-n relationship with each other
- Need a join table (intersection table) called “student-course”
  - “student-course” table has foreign key fields to both “student” and “course” tables
  - “student-course” table's primary key is typically composite of the student's and course's primary keys
  - “student-course” table can contain other fields of its own such as “course registration date”

Many-to-Many Relationship Example

```
/* Create student table */
CREATE TABLE student (
    student_id INT NOT NULL AUTO_INCREMENT,
    sname varchar(255) NOT NULL,
    PRIMARY KEY  (student_id)
) ENGINE=InnoDB;


/* Create course table */
CREATE TABLE course (
    course_id INT NOT NULL AUTO_INCREMENT,
    cname varchar(255) NOT NULL,
    PRIMARY KEY  (course_id)
) ENGINE=InnoDB;
```

Many-to-Many Relationship Example

```
/* Create "student_course" join table with FOREIGN KEY to
 * both student and course tables. */
CREATE TABLE student_course (
    student_course_id INT NOT NULL AUTO_INCREMENT,
    s_id INT NOT NULL,
    c_id INT NOT NULL,
    PRIMARY KEY  (student_course_id),
    FOREIGN KEY (s_id) REFERENCES student (student_id),
    FOREIGN KEY (c_id) REFERENCES course (course_id)
) ENGINE=InnoDB;
```

Many-to-Many Relationship Example

```
+-----------+--------------------------+
| course_id | cname                    |
+-----------+--------------------------+
|        11 | Computer Science 101     |
|        22 | MySQL                    |
|        33 | Java programming         |
+-----------+--------------------------+
3 rows in set (0.00 sec)

+------------+---------+
| student_id | sname   |
+------------+---------+
|          1 | jack    |
|          2 | mary    |
|          3 | nichole |
|          4 | mike    |
+------------+---------+
4 rows in set (0.00 sec)
```

Many-to-Many Relationship Example

```
+-------------------------+------+------+
| student_course_id | s_id | c_id |
+-------------------------+------+------+
|                         1 |     1 |   11 |
|                         2 |     1 |   22 |
|                         3 |     3 |   22 |
|                         4 |     4 |   22 |
+--------------------------+------+------+
```

Table Relationship:
Referential Integrity

Referential Integrity

- FOREIGN KEY constraint specifies that the data in a foreign key must match the data in the primary key of the linked table
- The “d_id” foreign key field of the employees table must contain a valid department number
  - You cannot add a new employee which has a d_id value that is not existent in department table
- The departments table cannot be dropped as long as there is a employee whose foreign key refers to it

Referential Integrity Example

```
+-------------------+-----------------+
| department_id | dname         |
+-------------------+-----------------+
|                    1 | Engineering |
|                    2 | Sales           |
|                    3 | Marketing     |
|                    4 | HR               |
+-------------------+------------------+
+----------------+----------+------+-----------+
| employee_id | ename | d_id | salary  |
+----------------+----------+------+-----------+
|                1 | jack      |    1 | 3000.00 |
|                2 | mary      |    2 | 2500.00 |
|                3 | nichole  |    1 | 4000.00 |
|                4 | angie     |    2 | 5000.00 |
|                5 | jones     |    3 | 5000.00 |
+----------------+-----------+------+------------+
mysql> INSERT INTO employees(employee_id, ename, salary, d_id)
    -> VALUES (6, 'newperson', '5000.00', 10);
ERROR 1452 (23000): Cannot add or update a child row: a foreign key constraint fails
(`mydb`.`employees`, CONSTRAINT `employees_ibfk_1` FOREIGN KEY (`d_id`) REFERENCES
`departments` (`department_id`))
```

Table Relationship: Automatic Delete and Update

Automatic Delete and Update

- The ON DELETE CASCADE or ON UPDATE CASCADE clause to the FOREIGN KEY .. REFERENCES modifier enabled automatic deletion or update of the records

```
/* Create "employees" table with FOREIGN KEY */
CREATE TABLE employees (
    employee_id int(11) NOT NULL AUTO_INCREMENT,
    ename varchar(255) NOT NULL,
    d_id int(11) NOT NULL,
    salary decimal(7,2) NOT NULL,
    PRIMARY KEY  (employee_id),
    FOREIGN KEY (d_id) REFERENCES departments (department_id)
    ON DELETE CASCADE
    ON UPDATE CASCADE
) ENGINE=InnoDB;
```

Automatic Delete Example

```
mysql> DELETE FROM departments WHERE department_id = 2;
Query OK, 1 row affected (0.05 sec)

mysql> SELECT * FROM departments;
+---------------+-------------+
| department_id | dname       |
+---------------+-------------+
|             3 | Marketing   |
|             4 | HR          |
|            11 | Engineering |
+---------------+-------------+
3 rows in set (0.00 sec)

// Observe that the employee record whose foregin key is 2
// are autimatically deleted.
mysql> SELECT * FROM employees;
+-------------+---------+------+---------+
| employee_id | ename   | d_id | salary  |
+-------------+---------+------+---------+
|           1 | jack    |   11 | 3000.00 |
|           3 | nichole |   11 | 4000.00 |
|           5 | jones   |    3 | 5000.00 |
+-------------+---------+------+---------+
```

Union

Union

- UNION is used to combine the result from multiple SELECT statements into a single result set

```
/* Combine the output of multiple SELECT */
SELECT ename, salary FROM HighSalaryEmployees
UNION
SELECT ename, salary FROM LowSalaryEmployees;
```

Subquery

What is Subquery?

- A subquery is a SELECT statement within another statement except that its result set always returns a single column containing one or more values
- A subquery can be used anywhere an expression can be used
- A subquery must always appear within parentheses

Why Subquery?

- They allow queries that are structured so that it is possible to isolate each part of a statement.
- They provide alternative ways to perform operations that would otherwise require complex joins and unions.
- They are, in general, more readable than complex joins or unions.

Sunquery Example #1

```
SELECT ename, salary FROM employees
WHERE salary >
    (SELECT AVG(salary) FROM employees);

+-----------+---------+
| ename     | salary  |
+-----------+---------+
| nichole   | 4000.00 |
| angie     | 5000.00 |
| jones     | 5000.00 |
+-----------+---------+
```

Sunquery Example #2

```
SELECT ename, salary FROM employees
WHERE d_id =
    (SELECT department_id FROM departments
     WHERE dname = 'Sales');
```

```
+-------+---------+
| name  | salary  |
+-------+---------+
| mary  | 2500.00 |
| angie | 5000.00 |
+-------+---------+
2 rows in set (0.00 sec)
```

View

What is a View?

- A view is a virtual table which is composed of result set of a SELECT query.
- Because view is like the table which consists of row and column so you can retrieve and update data on it in the same way with table.
- When the tables which are the source data of a view changes; the data in the view change also

Why View?

- A complex query is called repeatedly, it would be beneficial to create a virtual table (view)

View Example

```
CREATE VIEW v_HighSalaryEmployees AS
  SELECT ename, salary FROM employees
  WHERE salary > 4000;

CREATE VIEW v_LowSalaryEmployees AS
  SELECT ename, salary FROM employees
  WHERE salary < 3000;

mysql> SELECT * from v_HighSalaryEmployees;
+-------+---------+
| ename  | salary  |
+-------+---------+
| angie | 5000.00 |
| jones | 5000.00 |
+-------+---------+
2 rows in set (0.00 sec)
```

Thank you!

Sang Shin
http://www.javapassion.com
“Learning is fun!”