JDBC

- Satya Kaveti

Small Codes

Programming Simplified

A SmlCodes.Com Small presentation

In Association with Idleposts.com

For more tutorials & Articles visit SmlCodes.com

Small Codes Programming Simplified

So called TITLE

If you discover any errors on our website or in this tutorial, please notify us at support@smlcodes.com or smlcodes@gmail.com

First published on Aug 2016, Published by **SmlCodes.com**

Author Credits

Name : **Satya Kaveti**

Email : satyakaveti@gmail.com

Website : smlcodes.com, satyajohnny.blogspot.com

Digital Partners

Table of Content

1. SQL Basics

I.Introduction to SQL

SQL is a standard language for accessing and manipulating databases

RDBMS

- RDBMS stands for Relational Database Management System.
- RDBMS is the basis for SQL, and for all modern database systems such as MS SQL Server, IBM DB2, Oracle, MySQL, and Microsoft Access.
- The data in RDBMS is stored in database objects called tables.
- A table is a collection of related data entries and it consists of columns and rows

1. Database Tables: A database most often contains one or more tables. Each table is identified by a name (e.g. "Customers" or "Orders"). Tables contain records (rows) with data

2. SQL Statements Most of the actions you need to perform on a database are done with SQL statements. SQL keywords are **NOT case sensitive**: **select** is the same as **SELECT**

1.1 SQL Operations

Some of the Most Important SQL Commands

1. **CREATE DATABASE** - creates a new database

```
CREATE DATABASE `jdbc`
```

2. **DROP DATABASE** - modifies a database

```
DROP DATABASE `jdbc`
```

3. **CREATE TABLE** - creates a new table

```
CREATE TABLE `student` (
        `sno` INT NOT NULL,
        `name` VARCHAR(50) NULL,
        `address` VARCHAR(50) NULL,
PRIMARY KEY (`sno`)
);
```

4. **ALTER TABLE** - modifies a table

```
ALTER TABLE `student` ADD COLUMN `city` INT(11) NOT ;
```

5. **DROP TABLE** - deletes a table

```
DROP TABLE `student;
```

6. **CREATE INDEX** - creates an index (search key)

```
ALTER TABLE `student`   ADD UNIQUE INDEX `city` (`city`);
```

7. **DROP INDEX** - deletes an index

```
ALTER TABLE `student`    DELETE UNIQUE INDEX `city` (`city`);
```

8. **INSERT INTO** - inserts new data into a database

```
INSERT INTO `student` (`name`, `address`) VALUES ('Ravi', 'HYD');
```

9. **SELECT** - extracts data from a database
10. **UPDATE** - updates data in a database
11. **DELETE** - deletes data from a database

Select Operations

```
SELECT * FROM `student`;
```

sno	name	address
101	Satya	HYD
102	Ravi	HYD
103	Surya	VIJ
104	Rakesh	CHENNAI
105	Madhu	GUN
106	Krishna	HYD
107	Satya	VIJ

```
SELECT s.sno, s.name FROM `student` s;
```

student (2×4)

sno	name
101	Satya
102	Ravi
103	Surya
104	Rakesh

```
SELECT * FROM `student` s WHERE s.sno =104;
```

student (3×1)

sno	name	address
104	Rakesh	CHENNAI

```
SELECT * FROM `student` s WHERE s.name="Satya" AND s.address="HYD";
```

sno	name	address
101	Satya	HYD

```
SELECT * FROM `student` s WHERE s.name="Satya" OR s.address="HYD";
```

sno	name	address
101	Satya	HYD
102	Ravi	HYD
106	Krishna	HYD
107	Satya	VIJ

```
SELECT * FROM `student` s WHERE s.name like '%a';
```

- `like '%a'` ➔ **Ending with 'a'**
- `like 'a%'` ➔ **Starting with 'a'**
- `like '%a%'` ➔ **Contains with 'a'**
- `like '%'` ➔ **Contains any**

```
SELECT * FROM `student` s WHERE s.address IN ('HYD', 'VIJ');
```

IN operator allows you to apply multiple where conditions

sno	name	address
101	Satya	HYD
102	Ravi	HYD
103	Surya	VIJ
106	Krishna	HYD
107	Satya	VIJ

```
SELECT * FROM `student` s WHERE s.sno BETWEEN 103 AND 106
```

(Mostly used in date comparision)

sno	name	address
103	Surya	VIJ
104	Rakesh	CHENNAI
105	Madhu	GUN
106	Krishna	HYD

```
SELECT DISTINCT s.address FROM `student` s;
```

address
HYD
VIJ
CHENNAI
GUN

```
SELECT s.name AS 'Student_Name', s.address AS 'CITY' FROM `student` s
```

Student_Name	CITY
Satya	HYD
Ravi	HYD

```
SELECT * FROM `student` s ORDER BY s.address asc;     (default)
```

sno	name	address
104	Rakesh	CHENNAI
105	Madhu	GUN
101	Satya	HYD
102	Ravi	HYD
106	Krishna	HYD
103	Surya	VIJ
107	Satya	VIJ

```
SELECT * FROM `student` s ORDER BY s.address desc;
```

sno	name	address
103	Surya	VIJ
107	Satya	VIJ
101	Satya	HYD
102	Ravi	HYD
106	Krishna	HYD
105	Madhu	GUN
104	Rakesh	CHENNAI

```
UPDATE student` SET `name`='Vishnu' WHERE `sno`=107 AND `name`='Satya' AND
`address`='VIJ' LIMIT 1;
```

```
DELETE FROM `mydb`.`student` WHERE  `sno`=107 AND `name`='Vishnu' AND
`address`='VIJ' LIMIT 1;
```

1.2 JOINS

An SQL JOIN clause is used to combine rows from two or more tables, based on a common field between them.

Rules

1. At least on common column must be present beween two tables
2. PRIMARY_KEY of one table as FORIEN_KEY of another table is recommended.

PRIMARY KEY Constraint

- The PRIMARY KEY constraint uniquely identifies each record in a database table.
- Primary keys must contain UNIQUE values.
- A primary key column cannot contain NULL values.
- Table can have **only ONE primary key.**

FOREIGN KEY Constraint

- A **FOREIGN KEY in one table points to a PRIMARY KEY in another table**.

CASCADE will propagate the change when the parent changes. (If you delete a row, rows in constrained tables that reference that row will also be deleted, etc.)

SET NULL sets the column value to NULL when a parent row goes away.

RESTRICT causes the attempted DELETE of a parent row to fail.

```
CREATE TABLE `customer` (
`cid` INT NOT NULL,
`name` VARCHAR(50) NULL,
`addr` VARCHAR(50) NULL,
PRIMARY KEY (`cid`)
) ;
```

```sql
CREATE TABLE `order` (
  `ord_id` INT NOT NULL,
  `ord_item` VARCHAR(50) NULL,
  `ord_date` VARCHAR(50) NULL,
  `cid` INT NULL,
  PRIMARY KEY (`ord_id`),
  CONSTRAINT `FK__customer` FOREIGN KEY (`cid`) REFERENCES `customer`
  (`cid`) ON UPDATE CASCADE ON DELETE CASCADE
)
COLLATE='utf8_general_ci'
ENGINE=InnoDB;
SELECT `DEFAULT_COLLATION_NAME` FROM `information_schema`.`SCHEMATA`
WHERE `SCHEMA_NAME`='mydb';
```

Customer Table

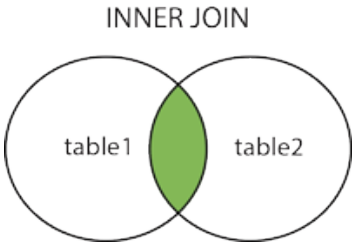
cid	name	addr
101	Satya	HYD
102	Ravi	VIJ
103	RAKESH	CHENNEI
104	Surya	BANG

Order Table

ord_id	ord_item	ord_date	cid
2005	COMPUTER	23-FEB-16	102
2007	CAR	20-JAN-16	102
2001	TV	20-JAN-16	103
2003	LAPTOP	20-MAR 16	103

We have 4 types of Joins

1. **INNER JOIN**:
   Returns all rows when there is at least one match in BOTH tables

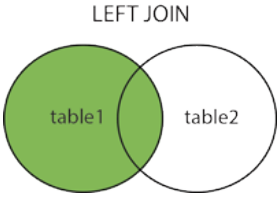



```sql
SELECT * FROM customer c INNER JOIN `order` o ON c.cid=o.cid
```

cid	name	addr	ord_id	ord_item	ord_date	cid
103	RAKESH	CHENNEI	2001	TV	20-JAN-16	103
103	RAKESH	CHENNEI	2003	LAPTOP	20-MAR 16	103
102	Ravi	VIJ	2005	COMPUTER	23-FEB-16	102
102	Ravi	VIJ	2007	CAR	20-JAN-16	102

2. LEFT JOIN:

Return **all rows from the left table, and the matched rows from the right table**

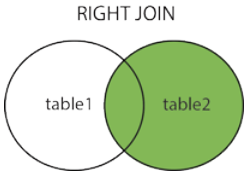


```
SELECT * FROM customer c LEFT JOIN `order` o ON c.cid=o.cid
```

	name	addr	ord_id	ord_item	ord_date	cid
101	Satya	HYD	(NULL)	(NULL)	(NULL)	(NULL)
102	Ravi	VIJ	2005	COMPUTER	23-FEB-16	102
102	Ravi	VIJ	2007	CAR	20-JAN-16	102
103	RAKESH	CHENNEI	2001	TV	20-JAN-16	103
103	RAKESH	CHENNEI	2003	LAPTOP	20-MAR 16	103
104	Surya	BANG	(NULL)	(NULL)	(NULL)	(NULL)

3. RIGHT JOIN:

Return all rows from the right table, and the matched rows from the left table

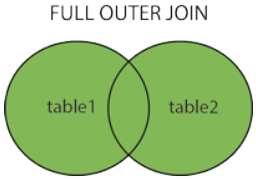



```
SELECT * FROM customer c RIGHT JOIN `order` o ON c.cid=o.cid
```

cid	name	addr	ord_id	ord_item	ord_date	cid
102	Ravi	VIJ	2005	COMPUTER	23-FEB-16	102
102	Ravi	VIJ	2007	CAR	20-JAN-16	102
103	RAKESH	CHENNEI	2001	TV	20-JAN-16	103
103	RAKESH	CHENNEI	2003	LAPTOP	20-MAR 16	103

4. FULL JOIN:

Return **all rows when there is a match in ONE of the tables**




```
SELECT * FROM customer c FULL OUTER JOIN `order` o ON c.cid=o.cid
```

In MySQL we don't have Full Outer Join, for this we have to use Unions

UNION Operator

SQL UNION operator **combines the result of two or more SELECT statements**

```
SELECT * FROM customer c WHERE c.cid>102
UNION
SELECT * FROM customer c
```

cid	name	addr
103	RAKESH	CHENNEI
104	Surya	BANG
101	Satya	HYD
102	Ravi	VIJ

- The UNION operator selects only distinct values by default.
- To allow duplicate values, to get duplicates also use **UNION ALL**
- Column name must be equal in TWO tables

SELECT INTO

SELECT INTO statement **copies data from one table** and inserts it into a **new table**.

```
SELECT * INTO old_customer FROM customer;            (mysql not supporting)
```

INSERT INTO SELECT

It selects data from one table and inserts it into an **existing table**

Constraints

- NOT NULL - Indicates that a column cannot store NULL value
- UNIQUE - Ensures that each row for a column must have a unique value
- PRIMARY KEY - A combination of a NOT NULL and UNIQUE
- FOREIGN KEY - Ensure the referential integrity of the data in one table to match values in another table
- CHECK - Ensures that the value in a column meets a specific condition
- DEFAULT - Specifies a default value for a column

Views

View is a **virtual table based on the result-set of an SQL statement**.

- A view contains rows and columns, just like a real table.
- The fields in a view are fields from one or more real tables in the database

```
CREATE OR REPLACE VIEW view_name AS
SELECT column_name(s)
FROM table_name
WHERE condition
```

```
CREATE OR REPLACE VIEW [Current Product List] AS
SELECT ProductID,ProductName,Category
FROM Products
WHERE Discontinued=No
```

We can perform INSERT, DELETE, UPDATE Opetations as normal as Table operations

Date Functions

The following table lists the most important built-in date functions in MySQL:

Function	Description
NOW()	**Returns the current date and time**
CURDATE()	**Returns the current date**
CURTIME()	**Returns the current time**
DATE()	**Extracts the date part of a date or date/time expression**
EXTRACT()	**Returns a single part of a date/time**
DATE_ADD()	**Adds a specified time interval to a date**
DATE_SUB()	**Subtracts a specified time interval from a date**
DATEDIFF()	**Returns the number of days between two dates**
DATE_FORMAT()	**Displays date/time data in different formats**

1.3 Sql Functions

Number functions

- **AVG()** - Returns the average value
- **COUNT()** - Returns the number of rows
- **FIRST()** - Returns the first value
- **LAST()** - Returns the last value
- **MAX()** - Returns the largest value
- **MIN()** - Returns the smallest value
- **SUM()** - Returns the sum

String functions

- **UCASE()** - Converts a field to upper case
- **LCASE()** - Converts a field to lower case
- **MID()** - Extract characters from a text field
- **LEN()** - Returns the length of a text field
- **ROUND()** - Rounds a numeric field to the number of decimals specified
- **NOW()** - Returns the current system date and time
- **FORMAT()** - Formats how a field is to be displayed

2. PL/SQL

PL/SQL stands for (Procedural Language/Structure Query Language). It is extension of SQL

2.1 Bascis

1. **Datatpye**

Below are supported Datatypes in PL/SQL which are supported in SQL.

	Data type	Syntax
1	Integer	INTEGER
2	Smallint	SMALLINT
3	Numeric	NUMERIC(P,S)
4	Real	REAL
5	Decimal	DECIMAL(P,S)
6	Float	FLOAT(P)
7	Character	CHAR(X)
8	Character varying	VARCHAR2(X)
9	Date	Date
10	Time	TIME

2. **Variables**

Variable Declaration

```
Variable_name datatype;
a number;
```

Variable Initialization

```
Variable_name datatype:=value;
a number=10;
```

3. Input Statement: to provide input value at run time by using operator (&).

4. Output Statement:

```
dbms_output.put_line ('Message' || variable);

dbms_output.put_line ('Sum: ' || c);
```

Basic Syntax

```
begin
        dbms_output.put_line("Hello word");
end;
```

Very simple, in the place of { }, we use **begin … end;**

2.2 Procedure

A pl-sql Procedure does not return any value. Procedure has two sections;

- **Declaration of the procedure:** Declaration of procedure always start with a keyword create ends with last **variable parameters**.
- **Body of the procedure:** Body of procedure starts with a keyword called **is or as** and ends with end statement.

Example

```
create or replace procedure p1(a in number, b out number, c out number, d out number)
is
        begin;
           select ename, sal, deptno, into b, c, d from emp where empno=a;
        end;
```

2.3 Function

A PL/SQL Function is a self control block which is used to perform some specific task. **function must always return a value**, but a procedure may or may not return a value.

```
create or replace function add(a number, b number)
return number
is
number;
        begin;
                a:=10;
                b:=20;
                c=a+b;
        return c;
        end;
```

Package is a collection of sub-programs that means function or procedure

```
        create package package_name AS
          procedure procedure_name(parameter);
        end package_name;
```

2.4 Cursors

A Cursors is a temporary work area created in the system memory when a SQL statement is executed. It is a temporary memory which is used to fetch more than one record at a time from existing table.

Implicit cursor cursor is perform by the system internally

Explicit cursor This type of cursor is performed by the user manually or programatically

2.5 Steps to perform cursor

Steps	Syntax
Declare the cursor	**open cursor_name;**
Open the cursor	**open cursor_name;**
Fetch the record from the cursor	**fetch cursor_name into variables;**
Close the cursor	**close cursor_name;**

Declare the cursor

```
declare
a emp %rowtype;
cursor c is select * from emp where depno=&deptno;
begin
open c;
loop fetch c into a;
if c % found then
dbms_output.put_line(a.empno || ' ' a.ename || ' ' || a.sal);
else
exit;
end if;
end loop;
close c;
end;
```

Trigger

Trigger is a pl/sql block structure which is fired when a DML statements like Insert, Delete, Update is executed on a database table. A trigger is triggered automatically when an associated DML statement is executed

Purpose of Triggers

Triggers can be written for the following purposes:

- Generating some derived column values automatically
- Enforcing referential integrity
- Event logging and storing information on table access Auditing
- Synchronous replication of tables
- Imposing security authorizations
- To avoid invalid transactions
- To generate the resulting data automatically.

Part of Trigger

A database trigger has 5 parts.

- Trigger timing
- Trigger event or statement
- Trigger level
- Trigger restriction
- Trigger body

```
CREATE [OR REPLACE ] TRIGGER trigger_name
 {BEFORE | AFTER | INSTEAD OF }
 {INSERT [OR] | UPDATE [OR] | DELETE}
 [OF col_name]
 ON table_name
 [REFERENCING OLD AS o NEW AS n]
 [FOR EACH ROW]
 WHEN (condition)
 BEGIN
   --- sql statements
 END;
```

Write a trigger to stop delete operation on emp_table

```
create or replace trigger mytrigger
before
delete
on emp
begin
raise_application_error(-20000, 'sorry we can not delete any record from this table');
end;
```

References

- http://www.w3schools.com/sql/
- http://www.sitesbay.com/plsql/