

Table Joins And Indexes In SQL

Joins

A join is a query that combines rows from two or more tables. In a join-query, more than one table are listed in FROM clause. The function of combining data from multiple tables is called joining.

Consider the following query :

```
SELECT *  
FROM EMP1, DEPT ;
```

This query will give you the cartesian product i.e., all possible concatenations are formed of all rows of both the tables EMP1 and DEPT. That is when no particular rows (using WHERE clause) and **[www.cbsepythonedu.com]** columns (through SELECT list) are selected. Such an operation is also known as unrestricted Join. It returns $n1 \times n2$ rows where $n1$ is number of rows in first table and $n2$ is number of rows in second table. In MySQL we are able to join a multitude of tables together. In order to find the location of the employee named ANOOP, we would use the following **[www.cbsepythonedu.com]** query.

```
mysql> SELECT ENAME, LOC  
FROM EMPL, DEPT  
WHERE ENAME = 'ANOOP'  
AND EMPL.DEPTNO = DEPT.DEPTNO ; see this is join condition
```

The virtually joined table that we created, consists of one row that meets the specifications of the **[www.cbsepythonedu.com]** WHERE clause (ENAME = 'ANOOP'). Since we selected only ENAME and LOC fields, only those columns were returned. However, we can retrieve as many attributes from each **[www.cbsepythonedu.com]** table as we need.

Qualified Names

In all the WHERE conditions of join-queries given so far, the field names are given as :

< tablename > . < fieldname >

This type of field names are called qualified field names. Qualified field names are very useful in identifying a field if the two joining tables have fields with same name. The columns that have similar names in both the tables have been given qualified names i.e., table-name.column-name, to avoid ambiguity. **[www.cbsepythonedu.com]** If the tables do not have the columns with the same name, using table names as qualifiers in the SELECT statement is optional. The FROM clause lists all the table involved and WHERE clause specifies the join. Without a WHERE clause, each row of the first table will be joined with every row of the second table which results in a Cartesian Product.

Using Table Aliases

A table alias can be used anywhere in the SELECT statement. The cut down on the amount of typing required in your queries you can use aliases for table names in the SELECT and WHERE clauses. **[www.cbsepythonedu.com]** For example, if you want to use the abbreviation 'E' for the EMPL table in your query all you need to is tell MySQL the EMPL will be referenced by E in the FROM clause.

Equi Join

In an equi-join the values in the columns being joined are compared for equality. All the columns in the tables-being-joined are included in the results. In equi-join, all the columns from joining table (www.cbsepythonedu.com) appear in the output even if they are identical.

Non-Equi-Joins

A non-equi-join is a query that specifies some relationship other than equality between the columns.

Display employee details fro Analysts in the following format.

Ename Job SaL Grade

Solution.

```
mysql> SELECT ENAME,JOB,SAL, GRADE
        FROM EMPL E, salary Grade S
        WHERE SAL BETWEEN LOSAL AND HISAL
        AND JOB = 'ANALYST' ;
```

ename	job	sal	grade
SCOTT	ANALYST	3000.00	4
FAKIR	ANALYST	3000.00	4

Natural Join

By definition, the results of an equi-join contain two identical columns. One of the two identical columns can be eliminated by restating the query. This result is called a Natural Join (Equi-Join minus one of the two (www.cbsepythonedu.com) identical columns.)

```
SELECT    empl. *, dname, loc
FROM      empl, Dept
WHERE     empl. deptno = dept. dept no ;
```

Joining Tables using Join clauses of SQL SELECT

Till now you have learnt how to create joined-table using traditional SQL style. MySQL also supports special clauses JOIN (and NATURAL JOIN) clause as part of SQL SELECT statement.

The (www.cbsepythonedu.com) syntax for using JOIN clasue SQL SELECT :

```
SELECT *
FROM < table1>
[CROSS] [NATURAL] JOIN <table2>
[ON (<join-Condition>)| USING (<joinfields>)]
```

It works as follows :

* To create Cartesian product of two table (say empl and dept) write query as :

```
SELECT *
FROM empl JOIN Dept ;
```

If no join-condition is specified with a JOIN clause.
cartesian product is output.

Cross Join

You can also produce cartesian product of two tables using CROSS JOIN clause as follows.

```
SELECT *
FROM empl CROSS JOIN dept ;
```

The above product of two tables is also known as CROSS JOIN . The cross join (or cartesian product) is a very basic type of join that simply matches each row from one table to every row from another table. Cross join by default, does not apply any filtering condition on the joined data. [www.cbsepythonedu.com] However, if you want to filter, you can use WHERE clause for SELECT query.

* To create equi-join on some common field (say deptno) of two (empl and dept here), write query [www.cbsepythonedu.com] as :

```
SELECT * FROM empl e
JOIN dept d ON (e.deptno = d.deptno) ;
```

You may write above query without table-aliases i.e., as

```
SELECT * FROM empl JOIN dept
ON (empl. deptno = dept. deptno) ;
```

Natural Join

MySQL provides a NATURAL JOIN clause for you that does the same for you that is, one of [www.cbsepythonedu.com] the duplicate columns get removed from the output.

All you need to do is two write :

```
SELECT * FROM < table1 >
NATURAL JOIN < table2 >;
```

That is, above query will now become :

```
SELECT *
FROM empl NATURAL JOIN dept ;
```

with NATURAL JOIN clause, you need not specify join condition

Filtering Output

IF you want to further filter the records from a join-output, you can write the condition with a WHERE clause. For instance, following query will first create a natural join from tables empl and [www.cbsepythonedu.com] dept and then extract only those records where sal is more than 2000

;

```
mysql> SELECT * FROM empl e
        NATURAL JOIN dept d
        WHERE sal > 2000 ;
```

Two joining tables whose
Natural join is being produced.

Left Right Join

When you join tables based on some condition, you may find that only some, not all rows from either table match with rows of other table. When you display an equi-join or natural-join, it shows only the matched rows. What if you want to know which all rows from a table did not match with **[www.cbsepythonedu.com]** other. In such a case, MySQL left or right JOIN can be very helpful and it refers to the order in which the tables are put together and the results are displayed.

Left join

You can use LEFT JOIN clause of SELECT to produce left join i.e., as :

```
SELECT <select - list >  
FROM <table1> LEFT JOIN <table2>  
ON <joining-condition>;
```

When using LEFT JOIN all rows from the first table will be returned whether there are matches in the second table or not. For unmatched rows of first table, NULL is shown in columns of second **[www.cbsepythonedu.com]** table.

Right Join

RIGHT JOIN works just like the LEFT JOIN but with table order reversed. All rows from the second table are going to be returned whether or not there are matches in the first table. Right join is produced in the same manner as that of left join, just you need to replace the keyword LEFT and **[www.cbsepythonedu.com]** RIGHT. You can use RIGHT JOIN clause of SELECT to produce right join i.e., per following syntax :

```
SELECT <select-list>  
FROM <table1> RIGHT JOIN <table2>  
ON <joining-condition>;
```

Indexes in Database

Database have really eased out so much for us storing bulk of **[www.cbsepythonedu.com]** data records, updating them, manipulating them, querying them. Database create and store indexes to store the location of records. You can decide the indexing based on the chosen index field. For example, you can index same table on multiple different fields. Database will create and store that many different index table as per the order of those index fields. To understand this, consider the following example :

Suppose you have a table namely student in your database with three fields in it Roll no., Name and Percentage marks.

* If you create index on field Roll no., then the database will create an index table that will have the **[www.cbsepythonedu.com]** locations of the data records of table student in the order of index field Roll no.

* If you create index on field Name, then the database will create an index table that will have the locations of the data records of table Student in the order of index field Name.

* If you create index on field Percentage, then the database will create an index table that will have the locations of the data records of table Student in the order of index field Percentage

Creating Indexes in MySQL

MySQL also provides commands to create indexes in a database. You can create indexes in MySQL using following commands. MySQL provides two ways to create indexes.

- (i) Create index at the time of table creation i.e., along with CREATE TABLE command.
- (ii) Create **[www.cbsepythonedu.com]** index on an already existing table.

(i) Creating Index at Time of table creation (i.e., with CREATE TABLE)

To create index on a field at the time of table creation, you can specify the index fields with INDEX clause of CREATE TABLE command, whose syntax is given below :

```
CREATE TABLE table_name
(
    column1 datatype [NULL | NOT NULL]
    column2 datatype [NULL | NOT NULL]
    ....
    column_n datatype [NULL | NOT NULL]
    ....
    INDEX index_name (index_col1 [(length)] [ASC | DESC]),
                    index_col2 [(length)] [ASC | DESC]
    ....
    index_col_n [(length)] [ASC | DESC] )
);
```

Please note that a CREATE TABLE statement with INDEX clause creates both the table as well as the index **[www.cbsepythonedu.com]** at the same time. Consider the following example :

```
CREATE TABLE PLAYERS
(
    PLAYERNO INTEGER NOT NULL,
    NAME CHAR (15) NOT NULL,
    DOB DATE,
    SEX CHAR (1) NOT NULL,
    ADDRESS VARCHAR (100) NOT NULL,
    PHONE CHAR (15),
    TEAMO CHAR (4) NOT NULL,
    PRIMARY KEY (PLAYERNO),
    INDEX Player_indx (TEAMNO)
```

(ii) Create index on an already existing table

Another way of creating index is to create it for already existing tables using following command :

```
CREATE INDEX <indexname> ON <tablename> (<column> [ASC | DESC],
                                         <column> [ASC | DESC]...);
```

to enforce unique values, add the UNIQUE keyword :

```
CREATE UNIQUE INDEX <indexname> ON (<column>[ASC | DESC],  
                                     <column> [ASC | DESC]....);
```

For [www.cbsepythonedu.com] example,

```
CREATE UNIQUE INDEX Players_idx on Players (TeamNo)
```

The above statement will create a Unique index on the basis of field TeamNo on an existing table [www.cbsepythonedu.com] namely Players.

Advantages and Disadvantages of Indexes

Although indexes improve the search time and increase a database's performance, yet it is a fact [www.cbsepythonedu.com] that each index is stored separately and takes up storage space on disk.

The advantage of index are as follows :

- (i) With indexes, queries give much better performance.
- (ii) Data retrieval is much faster [www.cbsepythonedu.com] with indexes.
- (iii) Indexes are very useful for sorting purposes.
- (iv) Unique indexes guarantee uniquely identifiable records in the database.

The disadvantages of indexes are as follows :

- (i) With indexes, the performance on inserts, updates and deletes decreases. As every time insert/update/delete operation happens, the index is to be updated accordingly. And with larger number of records, it takes up lot of time
- (ii) Indexes consume storage space and this (space consumption) increases with the number of fields used and the [www.cbsepythonedu.com] length of the fields.