"शीलं परं भूषणम्"

M. K. S. S. S.
CUMMINS COLLEGE OF ENGINEERING FOR WOMEN, PUNE.

STUDENT'S ROLL NO. : 3330

Assignment 3

1. Foreign Key Constraint + Syntax
2. Different types of joins + Syntax
3. Index
4. Simple Index
5. Compound Index
6. Show profiles command
7. Database views
8. Materialized Views
9. Updatable Views.

1. Foreign Keys Constraint + Syntax –

The foreign key is a constraint that is used to prevent actions that would destroy links between tables. A foreign key is a field (or a collection of fields) in one table that refers to the primary key in another table. Foreign keys are used to maintain data integrity.

eg –

```
ALTER table Books
ADD column Member_id INT
ADD Foreign key(Member_id) REFERENCES Members
                                (Member_id);
```

Joins – Joins are clauses that are used to combine rows from 2/more tables based on a related column between them.

1. Natural Joins – In a natural join, all common columns are displayed only once and only the related

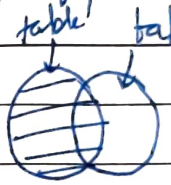
need to
rows are displayed. The common columns have same name in the tables that are having the join

SELECT * FROM Books Natural Join Issues;

Inner Join – Returns the records that have matching values in both the tables

SELECT * FROM Books INNER JOIN Issues
ON Books.Book_ID = Issues.BookId;

Left Join – Returns all the records from the left table and the matched records from the right table.



SELECT * FROM Books LEFT JOIN Issues
ON Books.Book_ID = Issues.BookId;

Right Join – Returns all the records from the right table & the matched records from the left table. The rows for which there is no matching row on left side, the result set will contain null. Books,

SELECT Books.*, Book_id, Books.Bookname, Issues.Member_Id, Issues.Membername FROM Books RIGHT JOIN ISSUES
~~Full Join~~ ON Books.Book_id = Issues.BookId;

Full Join – Full Join creates the result set by combining the result of both left & right join. The result set

will contain all the rows from both the tables. The rows for which there is no matching, the result set will contain NULL values.

```
SELECT Members.Member_id, Members.Membername,
Issues.bookId, Issues.Bookname FROM Members
FULL Join Issues ON Members.Member_id = Issues.
Member_id;
```

Cross Join - SQL Cross join produces a result set which is the number of rows in the first table multiplied by the no. of rows in the second table if no where clause is used, along with Cross join. This kind of result is called as Cartesian Product. If where clause is used then cross join functions like inner join.

```
SELECT * FROM Books CROSS JOIN Issues;
```

Index - Indexes are used to retrieve data from the database more quickly than others. The users cannot see the indexes, they are just used to speed up the searches/querries. However insert and update querries take longer as index modification overhead is introduced.

~~simple index~~ -
The CREATE INDEX statement is used to create indexes in tables. Duplicate values are allowed in indexes.

```
CREATE INDEX indexName
ON table_name (column1, column2, . . . );
```

If only 1 column is specified, the index is called as a simple index. If more than one column is specified then it is called as a compound index.

Show Profiles Command – The SHOW PROFILES statement displays profiling information that indicates resource usage for statements executed during the course of the current index. To control profiling use the profiling session variable, which has a default value of 0. Enable profiling by setting profiling to 1/ON.

SHOW PROFILE or the

SET profiling = 1;

SHOW PROFILES displays a list of the most recent statements sent to the server. The size of the list is controlled by the profiling_history_size session variable, which has a default value of 15. Max. value is 100. Setting the value to 0, has the practical effect of disabling profiling. All statements are profiled except SHOW PROFILE and SHOW PROFILES, so neither of those statements appears in the profile list.

Views – A view is a database object that has no values. Its contents are based on the base table. View is a virtual table. It is operated similarly to the base table but it does not contain any data of its own. Views are built on top of other tables or views. If any changes occur in the underlying table, the same changes are reflected in the View also.

```
CREATE VIEW Issued (Membername, BookId, BookName)
AS
SELECT MemberName, Book_id, Book_name
FROM Issues;
```

Materialized views – When the results of an expression are stored in a database system, they are called as materialized views.
SQL does not provide any standard way of defining materialized view however some database management systems provide custom extensions to use materialized views. The process of keeping materialized views updated is called as view maintainence.
Materialized view is useful when the view is accessed frequently as it saves the computation time as the results are stored in the database beforehand.
Materialized view is also useful when the relation on which the view is defined is very large & the resulting relation of the view is very small.

Updatable Views – Views can be used to update the underlying base tables. To create an updatable view, the SELECT statement that defines the view must not contain any of the following –

1) Aggregate functions such as MIN, MAX, SUM, AVG & COUNT
2) DISTINCT
3) GROUP BY clause
4) HAVING clause
5) UNION clause

6. UNION ALL clause
7) LEFT join / outer join
8) subquerry in the select clause or in the WHERE clause that refers to the table appeared in the FROM clause
9) Reference to non updatable view in FROM clause
10) multiple references to any column of the base table.

The view statement must contain a primary key.