**BDA NOTES**

**UNIT 2: SQL**

CHAPTER 1.- DML: QUERIES AND DATA MANIPULATION

2.4.-UNIVERSAL QUANTIFICATION

SQL’92 and most DBMS nowadays do not provide the universal quantification (FORALL). We must transform the query to solve it with an EXISTS: for all x that accomplish a condition, is equal to, it doesn’t exist anyone that doesn’t accomplish the condition.

2.5.-QUANTIFIED COMPARISON PREDICATES

The syntax is:

row\_constructor { ALL | ANY | SOME } (table\_expression)

The comparison predicate which is quantified with **ALL** is evaluated to true if it is **true for all the row**s in the table expression (if the table is empty it is also evaluated to true). The comparison predicate which is quantified with **ANY** or **SOME** is evaluated to true if it is **true for some** of the rows in the table expression (if the table is empty it is evaluated to false).

2.6.-GROUPING

A group is a set of rows with the same value for the subset of columns used for grouping (used in the GROUP BY). The **aggregated functions** in the grouped queries do not work like the rest of queries. Here, they return **a value for each group** which is formed.

The **HAVING** clause can only appear in grouped queries, and it is an extra condition similar to **WHERE**, but applied to the groups:

1. **WHERE** condition (used for the rows).
2. Grouping and calculus of aggregated functions.
3. **HAVING** condition (used for the groups).

In the **HAVING** clause can only appear references to columns used in the group by, aggregated functions, or subqueries. In the **SELECT** clause can only appear references to columns used in the group by, aggregated functions, or literals.

The **where** clause is applied **before** grouping. In the grouped queries, it is possible to use **nested aggregated functions**.

2.7.-SET OPERATIONS

There are other ways to **combine several tables** in the same query. All of them, along with the ways we have already seen, are what we have called “**table\_expression**”:

* Include several tables in the **FROM** clause.
* Use of **subqueries** in the conditions in the where or having clause.
* **Set table combinations**: use the set operators to combine the tables.
* **Table joins**: combine two tables by using different variants of the JOIN operator in Relational Algebra.

Set combinations correspond to the UNION, DIFFERENCE and INTERSECTION in the relational algebra:

* **UNION**.
* **EXCEPT**.
* **INTERSECT**.

They make possible to combine tables with **compatible** schemas.

The syntax of UNION is:

table\_expression UNION [ALL] table\_expression

It performs a **union** of the rows of the tables expressed by the two “table\_expression”. Duplicates will be allowed if the option ALL is set.

The syntax of INTERSECT is:

table\_expression INTERSECT table\_expression

It performs an **intersection** of the rows of the tables expressed by the two “table\_expression”.

The syntax of DIFFERENCE is:

table\_expression1 EXCEPT table\_expression2

It returns the tuples in table\_expression1 which do not appear in table\_expresion2.

2.8.-JOINS

There are three types of Joins (concatenation in the relational algebra):

* Cross join.
* Inner join.
* Outer join.

The **cross** **join** syntax is:

SELECT \* FROM table\_reference1, table\_reference2

The **inner** **join** has three different form of syntax:

1. table1 [**INNER**] **JOIN** table2 **ON** conditional expression.
2. table1 [**INNER**] **JOIN** table2 **USING** (c1, c2,… cn). It is a regular JOIN but using the tuples where c1,c2,… cn are the same in both tables.
3. table1 **NATURAL** [**INNER**] **JOIN** table2. It is a regular JOIN but using the **common** attributes of both tables.

The outer join combines **all** the rows from one of the tables (even if there is no correspondence for some row in the other table). The syntax is:

table\_expression [**NATURAL**] {**LEFT** | **RIGHT** | **FULL**} [**OUTER**] **JOIN** table\_expression [**ON** condition| **USING** (column1, column2,…, columnn) ]

3.-DATABASE UPDATES

DML (Data Manipulation Language):

* **SELECT**: allows the declaration of queries to retrieve the information from the database.
* **INSERT**: performs the insertion of one or more rows in a table.
* **DELETE**: allows the user to delete one or more rows from a table.
* **UPDATE**: modifies the values of one or more columns and/or one or more rows in a table.

The syntax for INSERT is:

**INSERT INTO** table [(column1, column2,…, columnn)] { **DEFAULT VALUES** | **VALUES** (atom1, atom2,… atomn) | table\_expression}

If we do **not** include the **list** of columns, we will have to specify **all the attributes** of the table. If we include the option “**default values**”, we will insert a single row with all the default values which were defined in the definition of the table. In the option (**atom\_commalist**), the atoms are given by **scalar** expressions. In the option **table\_expression**, we insert the rows which result from the execution of the expression ( a **SELECT** ).

The syntax for DELETE is:

**DELETE FROM** table [ **WHERE** conditional\_expression]

If we include the WHERE clause, then it will only delete the rows which make the condition true. In other case, all the tuples will be deleted.

The syntax for UPDATE is:

**UPDATE** table **SET** asignnment1, asignnment 2,…, asignnment n [ **WHERE** conditional\_expression ]

Where the assignments are of the form: column = {DEFAULT | NULL | scalar\_expression}

4.-COMMANDS FOR HANDLING TRANSACTIONS

A **transaction** is a logical unit of work consisting of one or more SQL statements that is guaranteed to be atomic with respect to recovery.

The **transaction initiation** is **implicit**. A transaction begins with the first SQL statement in a session, or when the previous transaction ends.

Transaction completion:

* **COMMIT**: the transaction ends **successfully**, making the database changes permanent.
* **ROLLBACK**: the transaction **aborts**, backing out any changes made by the transactions.

UNIT 3: DATABASE MANAGEMENT SYSTEMS (DBMS)

1.-THE ANSI/SPARC ARCHITECTURE

1.1.-SCHEMAS

1.2.-DBMS FUNDAMENTALS

1.3.-DATA INDEPENDENCE

2.-TRANSACTIONS, INTEGRITY AND CONCURRENCY

2.1.-TRANSACTIONS

2.2.-SEMANTIC INTEGRITY

2.3.-CONCURRENT ACCESS CONTROL

3.-RECOVERY AND SECURITY

3.1.-DB RECOVERY

3.2.-SECURITY