BDA NOTES

UNIT 1: RELATIONAL DATABASES

CHAPTER 1.- FUNDAMENTALS

1.-INFORMATION SYSTEM

An **information system (IS)** is a collection of elements, which are **orderly related** to each other **following some rules**, that provide the entity they serve with the necessary information for the completion of its goals.

Basic functions of an IS:

- Data gathering.
- Data processing.
- Data storage.
- Data elaboration and presentation.

2.-DATABASE AND DBMS

Database (DB) is a collection of structured data.

Database management system (DBMS) is a software tool (collection of programs) that enables users to create, manipulate and maintain a database.

A data modelling system or a "**data model**" is the way in which reality is represented in the context of databases. A DBMS assumes one data model and builds everything upon it.

A **Relational DBMS (RDBMS)** is a DBMS which is based on the relational model.

3.-DATABASE CHARACTERISTICS

Characteristics:

- **Integrating** all the organization's information.
- Data **persistence**.
- **Shared accessibility** to several users (or applications).
- **Unified** data description, independent of the applications.
- **Independence** between the **applications** and the physical representation.
- Description of **partial views** of the data for different users.
- Mechanisms to ensure data **integrity** and **security**.

DBs pursue a **general goal**, that is, the global integration of the system's information in order to avoid **redundancies**, with no loss of the different database **perspectives** by users.

Additionally, the **software tools** (DBMSs), specifically designed to apply these techniques, must ensure data **independence**, **integrity** and **security**.

There are three different levels in a DB:

1. Physical.
2. Logical.
3. External.

CHAPTER 2.- THE RELATIONAL DATA MODEL

1.-INTRODUCTION

The reason behind the success of the relational data model is just simplicity, a database is a “set of tables”.

2.-INTRODUCTION TO RELATIONA DATABASES

2.1.-INFORMAL VIEW OF A RELATIONAL DATABASE

The information is organized in tables, with columns and rows:

- Entities are represented as **tables**.
- Objects (entity instances) correspond to table **rows** or **tuples**.
- Object’s features are represented by **attributes**. These attributes correspond to the **columns** of the tables and are also known as **fields**.
- Attributes in the same column must have the same **datatype** or **domain**.

There are attributes which **identify** the tuples of a relation.

There are attributes which **associate** two relations.

2.2.-RELATIONAL DATABASE GOALS

The ultimate goal of a database is that users and applications can:

1. **Store** and **modify** the information of interest:
    a. **Insertion**.
    b. **Deletion**.
    c. **Update**.
2. **Access** and retrieve that information:
    a. **Query**.

A **relational query** is a retrieval operation to a database which returns part of the information of the database, possibly combined and/or aggregated, in the form of a **single table**.

Relational databases can be queried by different query languages:

- Relational algebra (operational, based on set and relational operators)
- Relational calculus (declarative, based on logic)
- SQL: a standard computer language which integrates most of the two previous approaches and looks like natural language.

The set operators are:

- UNION: ∪ : The union of two relations R and S defines a relation that contains all the tuples of R or S or both R and S, duplicate tuples being eliminated.
- INTESECTION: ∩ : R ∩ S defines a relation consisting of the set of all tuples that are in both R and S.
- DIFFERENCE: — : R — S defines a relation consisting of the tuples that are in relation R, but not in S.
- PRODUCT: x : R x S defines a relation that is the concatenation of every tuple of relation R with every tuple of relation S.

The relational operators are:

- SELECTION: WHERE … : selects the tuples that satisfy the specified condition (predicate).
- PROYECTION: […]: extracts the specified attributes (columns) and eliminates duplicates.
- JOIN: ⍰… : defines a relation that contains tuples satisfying some condition from the cartesian product.
- RENAME: (old, new) : changes the name of a column.
- Logical operators, conditions and expressions (AND, OR, NOT, etc.).

Queries using SQL:

- **FROM**: Specifies the table/s to be used.
- **WHERE**: Filters the rows subject to some condition.
- **GROUP BY**: Forms groups of rows with the same column value.
- **HAVING**: Filter the groups subject to some condition.
- **SELECT**: Specifies which columns are to appear in the output.
- **ORDER BY**: Specifies the order of the output.

SELECT[ALL | DISTINCT] *{expression$_1$, expression$_2$,…expression$_n$}* | *

FROM table

[WHERE *condition*]

[GROUP BY *condition*] [HAVING *condition*]

[ORDER BY *{column$_1$, column$_2$, … column$_m$}*]

3.-THE RELATIONAL DATA MODEL

Common terminology	RDM
table	relation
row / record	tuple
column / field	attribute
data type	domain

They are not exactly equivalent.

3.1.-DATA TYPES

Data types depend on the Relational DataBase Manager System (DBMS).

3.2.-TUPLE AND RELATION

A **tuple schema t** is a set of pairs of the form:

t = {(A1, D1), (A2, D2),…, (An, Dn)}

Where:

- {A1, A2,…, An} (n>0) is the set of **attribute names** in the schema, necessarily different.
- {D1, D2,…, Dn} are the **domains** associated with the above-mentioned attributes.

A **tuple t** of tuple schema t = {(A1, D1), (A2, D2),…, (An, Dn)} Is a set of pairs of the form:

t = {(A1, v1), (A2, v2),…, (An, vn)}

A **relation** is a set of tuples of the same schema.

A **relation schema** is the schema of the tuples composing the relation.

R (A1:D1, A2: D2,…, An: Dn} defines a relation R of schema { (A1, D1), (A2, D2),…, (An, Dn) }.

** Typical question *

Properties of a relation

- **Degree of a relation**: number of attributes of its schema.
- **Cardinality of a relation**: number of tuples that compose the relation.
- **Compatibility**: two relations R y S are compatible if their schemas are identical.

**

The representation of a relation is a **table**:

- **Tuples** are represented as rows.
- **Attributes** give name to the column headers.

The Table is only a **Matrix Representation** of a **relation**. Traits which distinguish a **relation** (derived from the definition of relation as a **set of sets**):

- There can't be **repeated tuples** in a relation (a relation is a set).
- There isn't a top-down **order** in the **tuples** (a relation is a set).
- There isn't a left-to-right order in the **attributes** of a relation (a tuple is a set).

The set of relation definitions which represent an information system is called **relational (logical) schema**. The content (set of tuples) of the relations of the relational schema is the **database**.

3.3.-NULL VALUE

A **domain** is something more than a datatype, it is a set of elements which always includes the NULL value. We may use the operator "**isnull**" to check the value.

The **null value** represents that there is **no known value**, so it is neither true nor false because it is undefined. We need a **tri-valued logic**:

- True.
- False.
- Undefined.

A α B (where a is a comparison operator) could be evaluated as undefined if at least one A or B is null, otherwise it is evaluated to the certainty value of the comparison A α B.

3.4.-CONSTRAINTS

In order to have a valid representation of reality:

- Definition of **domains**.
- **Uniqueness** constraints.
- **Not null** constraints.
- Definition of **primary keys**.
- Definition of **foreign keys**.
- **General** integrity constraints.

They are specified together with the database schema. The responsible for ensuring them is the DBMS.

The definition of a not null constraint over a set of attributes K={A1, A2,… A3} of a relation R expresses the following property: "There cannot be a tuple in R having the null value in any attribute of K".

$$\text{NNV: } \{A_1, \ldots, A_p\}$$

The definition of a uniqueness constraint over a set of attributes K={A1, A2,… A3} of a relation R expresses the following property: "There cannot be two tuples in R having the same value in all the attributes of K".

$$\text{UNI: } \{A_1, \ldots, A_p\}$$

Given a set of attributes K={A1, A2,… A3} which has been defined as primary key for R, we say that R satisfies the **primary key** constraint if the following properties hold:

1. R satisfies a **not null** constraint over PK.
2. R satisfies the **uniqueness** constraints over PK.

Note that PK must be **minimal**: there cannot be any proper subset that could also be primary key for R.

$$\text{PK: } \{A_1, \ldots, A_p\}$$

The use of **foreign keys** is the mechanism provided by the relational model to **express associations** between the objects in a database schema. This mechanism is defined such that these associations, if performed, would be carried out adequately. With this goal, we can add to the schema of a relation **S**, a **set of attributes** which **refer** to a set of attributes of a relation **R**. This set of attributes K = {A1,…, Ap} is called **foreign key** in relation S which refers to relation R.

Given a foreign key FK **in S which refers to R**, this is defined as:

1. A set of **attributes** K = {A1, A2,… A3} in the schema of S.
2. A **bijection** f: K-> J such as:
   a) J is a set of attributes in R.
   b) J has a **uniqueness** constraint.
   c) $\forall A_i \in K \rightarrow A_i$ and f(Ai) have the same domain.
3. A **type** of referential integrity:
   a) Weak
   b) Partial
   c) Full or complete

FK: {A_1,…, A_p} -> R

If FK = {A1} (**contains only one attribute**) the three types of referential integrity match. **S satisfies the referential integrity constraint** if all tuple in S met one of these:

- A1 is NULL.
- There is one tuple (and only one) in R with the same value in the f(A1) attribute than A1 in S.

If K has more than one attribute S satisfies the referential integrity if the following property is met:

- **Weak R.I.**: "If in a tuple of S **all** the values for the attributes in K have a **non-null** value, then there **must exis**t a tuple in R taking the **same values** for the attributes in J as the attributes in K".
- **Partial R.I.**: "If in a tuple of S **one or more** attributes in K have a **non-null** value, then there **must exist** a tuple in R taking the same values for the attributes in J as the values **in the non-null attributes** in K."
- **Complete R.**I.: "In any tuple of S **all** the attributes in K have a **null** value, **or none** of the attributes in K has a null value. In the **latter** case, there **must exist** a tuple in R taking the **same values** for the attributes in J as the attributes in K."

Given two relations R y S such that **S has a foreign key** K which refers to the attributes J in R, the only **operations** which **may violate their referential integrity** are:

- Operations over S:
  - Insert a tuple in S.
  - Modify some attribute in K in a tuple of S.
- Operations over R:
  - Delete a tuple in R.
  - Modify some attribute in J in a tuple of R.

If any of those operations attempts to break the referential integrity, the **DBMS aborts the operations** (by-default behaviour). But there are other options that can be applied by the DBMS if the foreign key has been previously defined in that way:

- Setting values to null.

or

- Applying the operation in cascade.

The referential integrity defined by a foreign key is always preserved but can be done in different ways depending on the foreign key definition:

- **Reject** the operation (default option).
- Perform the operation but **set** some values to **null** to restore integrity.
- Perform the operation but **propagate** the action in **cascade** to restore integrity.

Options to ensure the referential integrity:

- DELETE:
    - **Restrictive** deletion (default option in SQL).
    - On delete **cascade**.
    - On delete **set** to **nulls**.
- UPDATE:
    - **Restrictive** update (default option in SQL).
    - On update **cascade**.
    - On update **set** to **nulls**.

General integrity constraints are those constraints which cannot be expressed by the predefined constraints seen before. They can be:

- **Static integrity constraints**:
    - **Affecting one table**: attribute or table constraints (usually represented with "CHECK")
    - **Affecting several tables**: can be expressed with "CREATE ASSERTION ..." or with triggers.
- **Transition integrity constraints**: triggers.

A database is **valid** (**it is in a consistent state**), if all the defined integrity constraints are satisfied. The DBMS ensures that every update in the database generates a new extension which satisfies all the constraints.

4.-CONSTRAINTS AND TRANSACTIONS

A transaction is a sequence of [manipulation or query] operations which constitutes a logical execution unit. We can put a batch of single operations into a transaction (by using appropriate commands). Constraints can be disabled during a transaction:

- Some constraints are evaluated after every single atomic operation (immediate evaluation).
- Some constraints are evaluated after the transaction is completed (deferred evaluation).

The database designer or manager are responsible for determining the mode (immediate or deferred) of each constraint in the system.

CHAPTER 3.- INTERPRETATION OF A RELATIONAL DATABASE

1.-INTRODUCTION: REPRESENTATION OF REALITY

For each object in reality about which we want to have information we define a **relation** whose **attributes** denote the most significant properties of these objects in such a way that each **tuple** which is present in this relation must be interpreted as a particular **instance** of an object. In order to represent **associations** between objects we use explicit references through **attributes** which identify each object. Associations between objects where the cardinality is many-to-many require an extra **relation**.

2.-INTERPRETING DATABASE SHEMAS

Tables: **objects** are represented by tables for which none of the attributes of their PK refer to other tables (they have existence on their own). This is not true for hierarchical relationships (specialization). The rest of tables represent **relationships** (non-entity tables).

Attributes: represent properties of objects (if they are not FK). If they are FK, they represent relationships between objects.

Relationship types:

1. The FK is in a table representing an object:
    a. FK doesn't have UNI constraint: 1:M relationship (one-to-many).
    b. FK has UNI constraint: 1:1 relationship (one-to-one), or 0:1.
2. The FK is in a non-entity (object) table. The PK is composed of two FK: M:M relationship (many-to-many).

Constraints:

- **Non null value**:
    - If a FK from S to R has a NNV constraint, then **every** object in S is associated with **one** object in R.
    - If a FK from S to R does not have a NNV constraint, then every object in S is not necessarily associated with any object in R.
- **Uniqueness:**
    - If a FK from S to R has a uniqueness constraint, then **every** object in R can **at most** be associated with **one** object in S.

UNIT 2: SQL

CHAPTER 1: DML: QUERIES AND DATA MANIPULATION

1.-INTRODUCTION TO SQL

SQL (Structured Query Language) is a standard language for defining and manipulating a relational database.

DDL (Data Definition Language): creation and modification of relational DB schemas.

DML (Data Manipulation Language): queries and database updates:

- **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 control Language dynamically changes the database properties

2.-QUERIES

2.1.-SIMPLE QUERIES USING ONE TABLE

DISTINCT eliminates repeated tuples.

* lists all the information in a table.

LIKE is followed by a pattern which will be used with strings:

- The percent sign (%) represents zero, one, or multiple characters.
- The underscore (_) represents a single character.

BETWEEN number AND number, represents the values between those numbers, included these ones.

IS NULL returns if a value is null.

A x B (where x is a comparison operator) is evaluated as undefined if at least one A or B is null; otherwise it is evaluated to the certainty value of the comparison A x B.

These are the aggregated functions:

{ avg | max | min | sum | count } ([all | distinct]expression) | count(*)

Distinct is used to remove the duplicate values before the aggregated function calculate the result. The NULL values are removed before the aggregated function calculate the result. If the number of selected rows is 0, COUNT returns 0, and the rest of the functions return the NULL value.

AS is useful to create alias. ORDER BY is used to sort tuples.

2.2.-SIMPLE QUERIES WITH SEVERAL TABLES

If the information required by the query is in several tables, the query will include those tables in the **FROM** clause.

A query over several tables corresponds to the **Cartesian product**:

- If we do not express several conditions to connect them, the number of rows will be very high.
- If there are **foreign keys** defined, it is usual that some conditions are formed by an **equality between the foreign key** and the corresponding attributes in the table **to which it refers**.
- With **n** tables, we will typically have (at least) **n-1 connections**.

2.3.-SUBQUERIES

A subquery is a query between parenthesis included inside another query. Subqueries can appear in the search conditions, either in the “where” or in the “having” clauses, as arguments of some predicates:

- **Comparison predicates**: =, <>, >, <, >=, <=.
- **IN**: checks that a value belongs to the collection (table) returned by the subquery
- **EXISTS**: it is equivalent to the existential quantifier. It checks if a subquery returns some row.

The syntax is:

row_constructor **comparison predicate** row_constructor

“row_constructor” is either a sequence of constants or a subquery.

When row constructor returns **more than one column**, the lexicographic order will be used in the comparison of each operator. For simplicity, we will only see queries with **one column** in the subquery.

Subqueries can be a parameter of a comparison if (and only if):

- Return only **a single row**, and.
- The number of **columns match** (in number and type) with the other side of the comparison predicate.

If the **result of the subquery is empty**, the row is converted into a row with **NULL** values in all its columns and the result of the comparison will be **undefined**.

Syntax of the IN predicate:

row_constructor [NOT] **IN** (table_expression)

When using **NOT IN**, the expression will be **true** if the first expression is **different to all** the values of the subquery. If there is a **NULL** value, the result will be **undefined**.

Syntax of the EXISTS predicate:

EXISTS (table_expression)

The exists predicate is evaluated to **true** if the expression SELECT **returns at least one** row. In general, **IN and EXISTS are interchangeable** and, when there is no negation, they can be eliminated.