RELATIONAL MODEL

A relational model gives a simple way of representing data in form of a two-dimensional table called relation. Relational model was invented by Edger Codd. The columns or fields in the table identify the attribute such as surname, lastname, age, gender, Admission number etc.

A tuple or row contains all the data of a single instance of the table such as a person named James Bond. In the relational model, every tuple must have a unique identification or key based on the data. In the table below, the Admission number is the key that uniquely identifies each tuple in the relation. Keys are often used to join data from two or more relations based on matching identification. Another concept used in a relational model is the foreign keys, which are primary keys in one relation that are kept in another relation to allow for the joining of data.

Admission number	Surname	Last name	Age	Gender
1201	James	Bond	15	M
1202	Onyeka	Onwenu	14	F
1203	Adeneye	Dayo	16	M
1204	Jude	Okoye	14	M
1205	Joseph	Anita	15	F
1206	Williams	Chika	14	F

Definition of terms

1. **Field**: is a named column in the relation (table). There are five fields in the student relation above – Admission number, Surname, Last name, Age, Gender.
2. **Tuple**: is a record (i.e. a row of data in a relation). All tuples in a relation must be distinct.
3. **Domain**: is defined as the set of all unique values permitted for an attribute. For example, the domain of the field **Gender** as seen in the table above is M and F. Also the domain of the attribute **Surname** consist the combination of all possible letters in the alphabet. A domain is referred to in a relation schema by the domain name and has a set of values.
4. The **Degree** (also called ***arity***) of a relation is the number of fields in it. The relation above is of degree 5.
5. The **cardinality** of a relation is the number of tuples in it. For example, the relation above contains 6 tuples and hence the cardinality is 6.
6. A **relational database** is a collection of relations with distinct names that can be linked together.
7. **Relation schema** describes the column heads of the table. The schema specifies the relation's name, the name of the field (or attribute or column) and the domain of each field. For example the schema for the Student relation above is given below:

Student (*Admission number*: integer, *Surname*: **string**, *last name*: **string**, *Age*: **integer**, *Gender*: **string**).

Relational model keys

1. Candidate key: Candidate keys are defined as the minimal set of fields which can uniquely identify each record in the table. It is an attribute or a set of attributes that can act as a primary key for a table to uniquely identify each record in that table.
2. Primary key: is any candidate key that is most appropriate to be the main reference key for the table. It is used to establish relationship with other tables. It must never be null and must be unique.
3. Foreign key: is generally a primary key of one table that appears as a field in another where the first table has a relationship with the second. In other words, if we have a table A with a primary key X that is linked to a table B where X is a field in B, then X is a foreign key in B.
4. Secondary key or alternative key: a table may have one or more choices for the primary key. Collectively these are known as candidate keys as shown above. One is selected as the primary key. Those not selected are known as secondary keys or alternative keys.
5. Composite key is combination of two or more columns in a table that can be used to uniquely identify each row in a table

CREATING AND MODIFYING RELATION USING SQL

MS-Access, a relational DBMS typically provides a Graphical User Interface (GUI) that allows the creation of a database and tables that store information about different subject. This can be achieved using a query language called SQL (Structured Query Language). SQL can do more than query a database; it can define the structure of the database, modify data in the database and specify security constraints. A subset of the SQL that is used to define the structure of the database and the table is called data definition language (DDL). In SQL, a relation is a table.

Creating a database using SQL

The syntax is as follows:

CREATE DATABASE *databasename;*

The "CREATE DATABASE" is the keyword and "*databasename*" is the name of the database. For example assuming we want to create a database for students information, then the create statement would be as follows:

CREATE DATABASE StudentDB;

Note: Use the Commands:

SHOW DATABASES; - to see the list of databases in the DBMS
DROP DATABASE database_name; - to delete a database from a DBMS

Creating relation (Table) using SQL

The syntax is as follows:

```
CREATE TABLE table_name (
    Column1 datatype,
    Column 2 datatype,
    Column3 datatype,
    . . . . .
    columnN datatype,
    PRIMARY KEY (one or more columns)
    );
```

The "CREATE TABLE" is the keyword that tells the D system what to do table_name is the name of the table, "PRIMARY KEY" is a keyword that indicate which column listed above is to be used as the primary key, column1…ColumnN indicates the fields (or column as in a table).

For example, if you want to create a table that stores student information (i.e. student table), the statements below will achieve that.

```
CREATE TABLE StudentTbl (
        AdmissionNumber CHAR (10),
Surname CHAR (20),
Lastname CHAR(20),
Age INT (2),
Gender CHAR (1),
PRIMARY KEY (AdmissionNumber)
 );
```

Note: To delete a table from a database, use the command:
DROP TABLE *table_name*; (i.e. DROP TABLE StudentTbl).

MODIFYING RELATIONS USING SQL

Inserting tuple into a relation

In order to insert tuples (records) into a table, the INSERT command is used. Here is the syntax:

```
INSERT INTO Table_name (column1, column2, ….., columnN)
VALUES (value1, value2,…….., valueN);
OR
INSERT INTO table_name (value1, value2, ….., valueN);
```

For example to insert the first tuple as indicated in the table above, use the following statements:

```
INSERT INTO StudentTbl (AdmissionNumber, Surname, Lastname, Age, Gender)
VALUES (1201, 'James', 'Bond', 15, 'M');
OR
INSERT INTO students (1201, 'James', 'Bond', 15, 'M');
```

Deleting tuple from a relation

To delete a tuple (record) from the table, use the DELETE Command. Here is the syntax:

```
DELETE FROM tablename
WHERE [condition];
```

For example, to delete a tuple with surname = James from the relation above, use the following statement:

```
DELETE FROM StudentTbl
WHERE surname = 'James' ;
```

The database system deletes the first record with AdmissionNumber = 1201.
Note: The command '**DELETE FROM students**;' - will delete all tuples (records)in the table.

Updating the content of a relation

The UPDATE query is used to modify the existing records in a table. The syntax is as follows:

```
UPDATE table_name
SET column1 = value1, column2 = value2, ……, column = valueN
WHERE [condition];
```

For example, if we want to change the gender of James Bond in the table above (i.e. from M to F) , use the following statement:

```
UPDATE StudentTbl
SET Gender = 'F'
WHERE surname = 'James';
```

The DELETE, INSERT and UPDATE statements above are part of the SQL subset called the Data Manipulation Language (DML).

Querying Database

You query the database when you are requesting for certain information from the database. You can include condition that such items you want should meet. The query helps us retrieve data from the database.

In its simplest form, a query consists of two parts:

- A SELECT list, where the columns to be retrieved are specified
- A FROM clause, where the table or tables to be accessed are specified

Example

The following SQL statement selects the entire rows of data from the StudentTbl:

*SELECT * FROM* StudentTbl

It is also possible to specify certain columns to be retrieved from the StudentTbl.

Example

SELECT FirsName, LastName, Age FROM StudentTbl

We can structure the query to limit the number of columns of data to be retrieved from the StudentTable.

Example

SELECT FirstName, LastName, Date
FROM StudentTbl
WHERE Age > 20

Integrity Constraint over Relations

Constraints

Constraints are the rules enforced on data columns in table. These are used to limit the type of data that can go into a table. This ensures the accuracy and reliability of the data in the database. Constraints could be column level or table level. Column level constraints are applied only to one column, whereas table level constraints are applied to the whole table. The following are commonly used constraints in SQL:

1. **Entity Constraints**

Entity constraints can be enforced when you create a table and assign certain column of the table to be the primary key or and when you assign certain column of that table not to accept NULL values. The PRIMARY key constraint tells the database engine that none of the rows should have a duplicate value. The NOT NULL constraint ensures that a column cannot have a NULL value

Examples

```
CREATE TABLE StudentTbl (
        StudentId INT Primary Key,
                FirstName Varchar(15) NOT NULL,
                LasttName Varchar(15) NOT NULL,
                Gender Char(2),
                DOB DateTime
                );
```

You can also use the UNIQUES constraint to ensure that all values in a column are different; that is, no duplicate entries should exist in the table.

```
CREATE Table StudentData (
        StudentId INT Primary Key,
                FullName Varchar(50) unique NOT NULL,
                Age INT,
                RegDate DateTime Default GETDATE()
                );
```

The *GetDate* function inserts the current date into the table.

2. Domain Integrity

This constraint can be enforced by specifying that certain columns in the table should be of a particular data type or that it should meet certain conditions. Example below employs two types of constraints: *CHECK* and *UNIQUE* constraints. The CHECK constraint ensures that all values in a column satisfy certain conditions (i.e. any record to be inserted must have age value greater than or equal to 10).

Example

```
CREATE Table StudentData (
        StudentId INT Primary Key,
        FullName Varchar(50) unique NOT NULL,
        Age INT Check(Age > = 10),
        RegDate DateTime Default GETDATE()
        );
```

The DEFAULT constraint provides a default value for the date column when no value is specified.

3. Referential Integrity

Referential Integrity constraint can be enforced using two or more tables. The first table is called the parent table while any other table or tables will be regarded as the child table(s). The child table(s) will be used to reference the parent table.

Example

Create the first table called the parent table:

CREATE TABLE Student*(*

StudentId INT Primary Key,

FirstName Varchar(15) NOT NULL,

LasttName Varchar(15) NOT NULL,

Gender Char(6),

DOB DateTime

);

Create the second table called the child table:

CREATE TABLE Subject*(*

SubjectId INT Primary Key,

StudentId INT Foreign Key References Student_tbl(StudentId),

SubjectName varchar(15) Unique,

TeacherName varchar(50) Not Null,

Duration DateTime

);

StudentId in the ***Subject*** table is a *foreign key* that "references" ***StudentId*** in the ***Student*** table called the *parent* table. The idea behind this is that a subject cannot exist for a student when that student is not in the student (parent) table. For example, in order for a student to have a subject in the ***Subject*** table, such student must exist in the ***Student*** table.