Database Integrity

Table of Contents

Data Integrity

Data integrity in the database is the correctness, consistency and completeness of data.

Integrity constraints are used to ensure accuracy and consistency of the data in a relational database. Data integrity is handled in a relational database through the concept of referential integrity.

When we are designing a database, there is lot of factors to be concentrated on. We need to make sure that all the required datas are distributed among right tables and there is no duplication/missing data. The space utilised for the appropriately for the database. Time taken for each query is minimal and so on.

Imagine we have a STUDENT table with Student details and the subjects that he has opted for. If we observe the table below, Joseph has opted for two subjects – Mathematics and Physics. That is fine. But what is wrong in below table? His address is repeated each time, which is not necessary and waste of space. This is called redundancy and is not allowed in a database.

STUDENT_ID	STUDENT_NAME	ADDRESS	SUBJECT
100	Joseph	Alaiedon Township	Mathematics
101	Allen	Fraser Township	Chemistry
100	Joseph	Alaiedon Township	Physics
102	Chris	Clinton Township	Mathematics
103	Patty	Troy	Physics

Similarly, if we have to insert one more record for Allen, then we have to enter all his details into the above table. But what is the guarantee that all his details are entered correctly? There could be a mistake and hence leading to mismatch in his details. But who will later say which entry is correct? No one! Hence the data in DB is wrong.

Same issue can happen when we update the data. If we update address one of the record, and leave other record for Joseph above, again a data mismatch.

And when we delete a data, say for Chris, who is having only one entry, whole of his information is lost!

Imagine there are two entities – Employee and Department, and they are not properly related by means of foreign key. What would be the result? We can enter as many department as we want to an employee for whom department may not exists at all in Department table! So mapping the tables appropriately is also a very important factor.

Data integrity ensures, all the above mentioned issues are not injected into the database while it is designed. It guarantees that database is perfect and complete.

Data integrity is enforced using the following integrity constraints:

Types of Integrity

There are five types of data integrity

Domain Integrity

This means that there should be a defined domain for all the columns in a database. Enforces valid entries for a given column by restricting the type, the format, or the range of values.

Here each columns of a table are verified so that correct data is entered into column. For example, numeric data is entered into a NUMBER column and not any character. In a DATE column, correct dates are entered and not any invalid values.

Imagine we have a table where date fields are stored as character and we have to copy this date field into a new table where this column is defined as DATE. What happens here is most of the data from the original table will not be loaded into new table, as there is mismatch in the data stored. i.e.; the original table will have dates in 22 March 2015 format which new table will not accept as date. In the foremost case, if there was domain integrity, original table would not have such dates and would have preserved the integrity of data in the original table itself.

Entity Integrity

This is related to the concept of primary keys. All tables should have their own primary keys which should uniquely identify a row and not be NULL.This integrity ensures that each record in the table is unique and has primary key which is not NULL. That means, there is no duplicate record or information of data in a table and each records are uniquely identified by non null attribute of the table.

In a STUDENT table, each student should be a different from other and there will not be duplicate records. Also, STUDENT_ID which is a primary key in the table has non-null values for each of the record.

STUDENT

STUDENT_ID	STUDENT_NAME	ADDRESS
100	Joseph	Alaiedon Township
101	Allen	Fraser Township
102	Chris	Clinton Township
103	Patty	Troy

Column Integrity

This integrity ensures that the values entered into a column are correct by means of business rules. Say, there is an age column and its value is negative which is not correct. This constraint refines from entering wrong age. Similar example of such constraint is salary cannot be negative; employee number will be in a given range etc. These are business rules/requirements that specify what kind of values could be entered into each column.

This integrity is different from domain constraint as here it checks for the validity of the data being entered- like correct age is being entered; Correct Employee Id is entered etc. In the domain constraint, it checks, whether correct set of data being is entered – like Date is entered into date column, Number is entered into number column etc.

User-Defined Integrity

Imagine, while entering a salary of an employee, we need to check if his salary is less than his manager. Though this is similar to column constraint, we cannot direct insist this constraint on the column as the system does not know who his manager is. We need to check for his manager's salary first, if it is more than his employer, then we will insert the data. For this we manually need to write code. This kind of constraints is called User-Defined Integrity.

Referential Integrity

This is related to the concept of foreign keys. A foreign key is a key of a relation that is referred in another relation. Rows cannot be deleted, which are used by other records.

As we discussed for Employee and Department tables, if they are not mapped correctly, there would be a data mismatch. It will allow us to enter a department for an employee which does not exist. It will allow us to delete a department for which employees are working. What would be the result? Employees without any department are not correct. Or updating any department number in the employee table will result in a department which does not exist at all. All these cases will lead to mismatch and invalid data in the database.

Hence to ensure above all cases are met, proper relationship has to be defined between the related tables by means of primary and foreign keys. i.e.; every foreign key in the table should be a primary key in the related table.

In our example, DEPARTMENT_ID should be a primary key in Department table and it should be a foreign key in the Employee table. This will stop entering/updating a department which does not exists. It will not allow us to delete any department from the Department table for which employees still exists in Employee table.

Database Constraints

Table of Contents

Constraints

Constraints are the rules enforced on the data columns of a 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 either on a column level or a table level. The column level constraints are applied only to one column, whereas the table level constraints are applied to the whole table.

Constraints are the conditions forced on the columns of the table to meet the data integrity. We have seen above what types of data integrities exists in the database. Now let see what constraints can be applied on tables so that data integrity is met.

NOT NULL

This constraint forces the column to have non-null value. We cannot enter/update any NULL value into such columns. It must have valid value all the time. For example, each student in STUDENT table should have class specified. No student can exist without class. Hence class column in the STUDENT table can be made NOT NULL.

CREATE TABLE STUDENT (STUDENT_ID NUMBER (10) **NOT NULL**
STUDENT_NAME VARCHAR2 (50) NOT NULL,
AGE NUMBER);

UNIQUE

This constraint ensures, the column will have unique value for each row. The column value will not repeat for any other rows in the table.

Passport number of individual person is unique. Hence passport column in the PERSON table is made UNIQUE. It avoids duplicate entry of passport number to other persons.

CREATE TABLE STUDENT (STUDENT_ID NUMBER (10) NOT NULL **UNIQUE**
STUDENT_NAME VARCHAR2 (50) NOT NULL,
AGE NUMBER);

OR

CREATE TABLE STUDENT (STUDENT_ID NUMBER (10) NOT NULL,
STUDENT_NAME VARCHAR2 (50) NOT NULL,
AGE NUMBER
CONSTRAINT uc_StdID UNIQUE (STUDENT_ID));

PRIMARY KEY

This constraint is another type of UNIQUE constraint. This constraint forces the column to have unique value and using which, we can uniquely determine each row.

As we have seen in STUDENT example, STUDENT_ID is the primary key in STUDENT tables.

CREATE TABLE STUDENT (STUDENT_ID NUMBER (10) NOT NULL **PRIMARY KEY**,
STUDENT_NAME VARCHAR2 (50) NOT NULL,
AGE NUMBER);

OR

CREATE TABLE STUDENT (STUDENT_ID NUMBER (10) NOT NULL,
STUDENT_NAME VARCHAR2 (50) NOT NULL,
AGE NUMBER
CONSTRAINT pk_StdID PRIMARY KEY (STUDENT_ID));

FOREIGN KEY

This constraint helps to map two or more tables in the database. It enforces parent-child relationship in the DB. Foreign key in the child table is the column which is a primary key in the parent table.

For example, each employee works for some department. Hence to map employee and department tables, we have to have DEPARMENT_ID of DEPARTMENT table in EMPLOYEE table too. DEPARTMENT_ID is the primary key in DEPARTMENT table (Parent table) and is foreign key in EMPLOYEE table (Child table).

```
CREATE TABLE EMPLOYEE (EMPLOYEE_ID VARCHAR2 (10) PRIMARY KEY,
EMP_NAME VARCHAR2 (50),
DOB DATE,
……
DEPT_ID NUMBER
CONSTRAINT fk_DeptId FOREIGN KEY (DEPT_ID) REFERENCES
DEPARTMENT (DEPARTMENT_ID));
```

CHECK

This constraint is used to check for specific conditions on the column. For example, if age has to be entered between 25 and 32, we can use CHECK Constraint. This will not allow to enter the age<25 and age>32.

```
CREATE TABLE STUDENT (STUDENT_ID NUMBER (10) NOT NULL,
STUDENT_NAME VARCHAR2 (50) NOT NULL,
AGE NUMBER CHECK (AGE >= 25 and AGE<= 32));
```

DEFAULT

This constraint specifies the default value to be entered when no value is entered to it. Suppose whenever we enter an entry in the STUDENT table, apart from Student details we also have to store the date when it is being entered. This entry would always be SYSDATE. Instead of entering it each time when we do an entry, if we set the default value of this column as SYSDATE, this column will be always inserted with SYSDATE. If we need to override this value with any other date, then we have to explicitly insert the new date.

```
CREATE TABLE STUDENT (STUDENT_ID NUMBER (10) NOT NULL,
STUDENT_NAME VARCHAR2 (50) NOT NULL,
AGE NUMBER,
….
CREATED_DATE DATE DEFAULT SYSDATE);
```

Creating Constraints and Keys in SQL

As we saw above constraints are used on the table to make sure the records and attributes entered are correct in that contest. It makes sure that there is no incorrect data being entered.

If there is any mismatch or wrong data being entered, then the transaction will be rejected. How do we create all these constraints in SQL? Either they are created when we create the table for the first time or it can be constructed after table is being created.

The general syntax for creating a constraint when a table is created is as shown below.

```
CREATE TABLE table_name (
                    Column1 DATATYPE (SIZE) CONSTRAINT_TYPE,
                    Column2 DATATYPE (SIZE) CONSTRAINT_ TYPE,
                    ….
                  ); –where CONSTRAINT_TYPE can be NOT NULL, PRIMARY KEY, CHECK, DEFAULT etc.
```

Few examples of creating constraints as above is given below:

```
CREATE TABLE STUDENT (
STD_ID NUMBER (10) NOT NULL PRIMARY KEY,
STD_NAME VARCHAR2 (255) NOT NULL,
ADDRESS VARCHAR2 (255),
DATE_OF_BIRTH DATE); — Example of NOT NULL and PRIMARY KEY Constraint

CREATE TABLE PERSON (
SSN_NUM NUMBER (10) NOT NULL UNIQUE,
PERSON_NAME VARCHAR2 (255) NOT NULL,
ADDRESS VARCHAR2 (255),
AGE NUMBER NOT NULL CHECK (AGE>=18),
LICENCE NUMBER,
DEPT_ID NUMBER DEFAULT 10); — Example of NOT NULL, UNIQUE, DEFAULT AND CHECK Constraint
```

For a given column, single or multiple constraints can be applied. Constraints can be created on the existing tables as below.

ALTER
TABLE *table_name* **ADD CONSTRAINT** *constraint_name* **CONSTRAINT_TYPE;**

ALTER TABLE *PERSON* **ADD CONSTRAINT** *checkAge* **CHECK (AGE >= 18);**

ALTER TABLE *STUDENT* **ADD CONSTRAINT** *stdPrimaryKey* **PRIMARY KEY** (STD_ID**);**

ALTER TABLE *STUDENT* **ADD CONSTRAINT** *stdUniqueKey* ***UNIQUE*** (STD_ID**);**

ALTER TABLE *STUDENT* **ADD CONSTRAINT** *stdUniqueKey* ***NOT NULL*** (STD_NAME**);**

Drop the Constraint

ALTER TABLE *table_name* **DROP CONSTRAINT** *constraint_name*;

ALTER TABLE *PERSON* **DROP CONSTRAINT** *checkAge*;

ALTER TABLE *STUDENT* **DROP CONSTRAINT** *stdPrimaryKey*;

ALTER TABLE *STUDENT* **DROP CONSTRAINT** *stdUniqueKey*;

ALTER TABLE *STUDENT* **DROP CONSTRAINT** *stdUniqueKey*;

Create Foreign Key Constraint

Creating a foreign key constraint is little different. In order to create a foreign key constraint, we need to have parent table created first. Then, when we create child table, we can have foreign key constraint mentioned as below.

CREATE TABLE *table_name* (

Column1 **DATATYPE (SIZE),**

Column2 **DATATYPE (SIZE)**

……

CONSTRAINT *constraint_name* **FOREIGN KEY (***column_name***)**
REFERENCES *parent_table* **(***parent_column_name***));**

Example:

CREATE TABLE *EMPLOYEE* (EMPLOYEE_ID VARCHAR2 (10) **PRIMARY KEY,**

EMP_NAME VARCHAR2 (50),

DOB DATE,

……

DEPT_ID NUMBER

CONSTRAINT *fk_DeptId* **FOREIGN KEY (DEPT_ID) REFERENCES** *DEPARTMENT* **(***DEPARTMENT_ID***)**);