Normalization of databases

Database normalization is a technique of organizing the data in the database. Normalization is a systematic approach of decomposing tables to eliminate data redundancy and undesirable characteristics such as insertion, update and delete anomalies. It is a multi-step process that puts data in to tabular form by removing duplicated data from the relational table.

Normalization is used mainly for 2 purpose.

- Eliminating redundant data
- Ensuring data dependencies makes sense. ie:- data is stored logically

Problems without normalization

Without normalization, it becomes difficult to handle and update the database, without facing data loss. Insertion, update and delete anomalies are very frequent is databases are not normalized.

Example :-

S_Id	**S_Name**	**S_Address**	**Subjects_opted**
401	Adam	Colombo-4	Bio
402	Alex	Kandy	Maths
403	Steuart	Ja-Ela	Maths
404	Adam	Colombo-4	Chemistry

Updation Anomaly – To update the address of a student who occurs twice or more than twice in a table, we will have to update S_Address column in all the rows, else data will be inconsistent.

Insertion Anomaly – Suppose we have a student (S_Id), name and address of a student but if student has not opted for any subjects yet then we have to insert null, which leads to an insertion anomaly.

Deletion Anomaly – If (S_id) 401 has opted for one subject only and temporarily he drops it, when we delete that row, entire student record will get deleted.

Normalisation Rules

1st Normal Form – No two rows of data must contain repeating group of information. Ie. Each set of column must have a unique value, such that multiple columns cannot be used to fetch the same row. Each row should have a primary key that distinguishes it uniquely.

Primary Key – The primary key is usually a single column, but sometimes more than one column can be combined to create a single primary key.

Example – Student Table (Before 1st Normal form)

Student	**Age**	**Subject**
Adam	20	Bio, Maths
Alex	21	Maths
Steuart	19	Maths

In 1st normal form, any row must not have a column in which more than one value is saved. However in 1st normal form, data redundancy will increase as there will be many columns with the same data in multiple rows, but each row as a whole will be unique.

Student Table (After 1st Normal form)

Student	Age	Subject
Adam	20	Bio
Adam	20	Maths
Alex	21	Maths
Steuart	19	Maths

2nd normal form

In the 2nd normal form there should not be any partial dependency of any columns on primary key. A table that has concatenated primary key, each column in the table that is not part of the primary key must depend upon the entire concatenated key for its existence.

Student Table (Before 2nd Normal form)

Student	Age	Subject
Adam	20	Bio
Adam	20	Maths
Alex	21	Maths
Steuart	19	Maths

Student Table (After 2nd Normal form)

Student	Age
Adam	20
Alex	21
Steuart	19

Student	Subject
Adam	Bio
Adam	Maths
Alex	Maths
Steuart	Maths

While the candidate key is (Student, Subject), Age of student only depends on Student column.

3rd normal form

Every non-prime attribute of table must be dependent on primary key. The transitive functional dependency must be removed from the table.

Example – **Student_Detail table (before 3rd normal form)**

Student_Id	Student_Name	DOB	Street	City	State	Zip

In this table Student_Id is the primary key, but street, city and State depends on Zip. The dependency between Zip and other fields is called transitive dependency. Hence, to apply 3rd normal form, we need to remove Street, City and State to a new table with Zip as a primary key.

Student_Detail table (after 3rd normal form)

Student_Id	Student_Name	DOB

Address table

Zip	Street	City	State

The advantage of removing transitive dependency is as follows.

- Amount of data duplication is removed
- Data integrity achieved

<u>Entity Relationship Diagram</u>

An entity diagram is a visual representation of how data is related to each other.

Entity – An entity is a person, place or object which is represented as rectangles.

Student
Teacher
Projects

Attribute – Attributes are properties of an entity. Each attribute will have a type

Eg:-Student_Name : VARCHAR (50)

Student_DOB : DATETIME

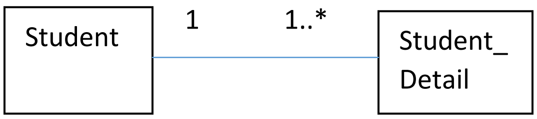
Teacher_Qualification : VARCHAR (100)

Student_Age :NUMERIC

Relationship

There are 4 types of relationships that could exist between entities. Many to many relationships should be resolved using an intermediate entity.

- One-to-one (Eg:-Husband and Wife)
- One-to-many (Eg:-Student and Student_Detail)
- Many-to-one
- Many-to-many (Eg:-Student and Address)



Structured Query Language (SQL)

SQL is a programming language used for storing and managing data in RDBMS. All RDBMS (SQL Server, Oracle, MySQL, MS Access) use SQL as the standard database language. SQL is used to perform all types of operations in a database. The following SQL commands are usually used.

Data Definition Language (DDL)

All DDL commands are auto committed.

Command	Description
CREATE	To create new table or database
ALTER	For alteration
TRUNCATE	Delete data from table
DROP	Drop a table
RENAME	To rename a table

Data Manipulation Language (DML)

DML commands are not auto committed. They can be rolled back.

Command	Description
INSERT	To create new table or database
UPDATE	For alteration
DELETE	Delete data from table
MERGE	Drop a table

Data Query Language (DQL)

Command	Description
SELECT	Retrieve records from one or more tables

Data Control Language (DCL)

Command	Description
GRANT	Grant permission of right
REVOKE	Take back permission

Transactional Control Language (TCL)

Command	Description
COMMIT	To permanently save
ROLLBACK	To undo change
SAVEPOINT	To save temporarily

CREATE statement

```
CREATE DATABASE Tuition;

CREATE TABLE Student (Student_IdVARCHAR(10), Student_Name VARCHAR(50), DOB DATETIME );
```

ALTER statement

```
ALTER TABLE Student add (Address VARCHAR(100));

ALTER TABLE Student add (Address VARCHAR(100), default 'Colombo-5');

ALTER TABLE Student rename Address to Location;

ALTER TABLE Student drop Location;
```

DROP statement

```
DROP TABLE Student

DROP DATABASE Tuition
```

RENAME statement

```
Rename TABLE Student to Student_Record
```

INSERT statement

```
INSERT into Student values ('100', 'Silva', '1-JAN-1990')
```

UPDATE statement

```
UPDATE Student set Student_Name='Kumar' where Student_Id='100'
```

DELETE statement

```
DELETE from Student where Student_Id='100'
```

GRANT and REVOKE statement

```
GRANT create table to UserName

REVOKE create table from UserName
```

SQL queries

```
SELECT *

FROM Student

WHERE Student_NameLIKE  'a%'

ORDER BY asc

GROUP BY Student_AGE

HAVING Student_AGE>20
```

SQL Functions

1. **AVG**
   SELECT AVG(Student_Marks) FROM Marks

2. **COUNT**
   SELECT COUNT(Student_Name) FROM Student

3. **DISTINCT**
   SELECT COUNT(DISTINCT Salary) FROM Employee

4. **FIRST**
   SELECT FIRST(Student_Name) FROM Student ORDERBY Age

5. **LAST**
   SELECT LAST(Student_Name) FROM Student ORDERBY Age

6. **MAX**
   SELECT MAX(Student_Marks) FROM Marks

7. **MIN**
   SELECT MIN(Student_Marks) FROM Marks

8. **SUM**
   SELECT SUM(Salary) FROM Employee

9. **ROUND**
   SELECT ROUND(Salary) FROM Employee

10. **MID**
    SELECT MID(Employee_Name, 1, 10) FROM Employee

11. **LCASE**
    SELECT LCASE(Employee_Name)

12. **UCASE**
    SELECT UCASE(Employee_Name)

AND and OR Operators

AND

SELECT * FROM Employee WHERE Employee_Salary>25,000 AND Employee_Age<25

OR

SELECT * FROM Employee WHERE Employee_Salary>25,000 OR Employee_Sage<25

Database Keys

1. **Primary Key**
   Primary Key helps to uniquely identify a row. Primary key could either be a single attribute or a group of attributes (composite keys)
   Eg1-Student_ID for Student table.
   Eg2-Student_ID and Subject_ID for Marks table

2. **Foreign Key**
   A key which is primary key when referred in another table is made as a foreign key. Foreign Key is used to relate 2 tables.
   Eg:-Emp_Id is a foreign key for Emp_Salary table

UNION and INTERSECT

UNION

SELECT * FROM A UNION ALL SELECT * FROM B will obtain all records from both tables, excluding common records.

UNION ALL

SELECT * FROM A UNION ALL SELECT * FROM B will obtain all records from both tables, including common records.

INTERSECT

SELECT * FROM A UNION ALL SELECT * FROM B will obtain all records from both tables, which are common records.

MINUS

SELECT * FROM A MINUS SELECT * FROM B will remove all common records in both tables as well unique records in table B

Alias

Alias will rename the query results as something different from the table name.

SELECT Emp_Name FROM Employee AS NAME