2.1 The Relational Model

CollateNotes

The Relational model today is the primary data model for commercial data processing applications It has attained its primary position because of its Simplicity, which eases the job of the programmer Compared to earlier data models such as the network model or the hierarchical model.

The Relational model represents the database as a collection of relations. A relation is nothing but a table of values.

Every row in the table represents a collection of related data values.

These rows in the table denote a real world entity or relationship

The table name and column names are helpful to interpret the meaning of values in each row.

The data are represented as a set of relations

In the relational model, data are stored as tables However, the physical storage of the data is independent of the way the data are logically organised.

↳ Advantages:

→ SIMPLICITY — A relational data model is Simpler than the hierarchical and network model.

→ STRUCTURAL INDEPENDENCE — The relational database only concerned with database model

and not with the structure which can improve the Performance of the model.

→ EASY TO USE — The relational model is easy as tables consisting of rows and columns is easier to implement and understand

→ SCALABLE — Regarding a number of records, or rows and the number of fields, a database should be enlarged to enhance its usability

→ DATA INDEPENDENCE — The structure of a database can be changed without having to change any application

→ QUERY CAPABILITY — It makes possible for a high level query language like SQL to avoid Complex database navigation

↳ Disadvantages:

- Complex relational database systems may lead to isolated databases where the information cannot be shared from one system to another
- Few relational databases have limits on field lengths which cannot be exceeded
- Relational databases can sometimes become complex as the amount of data grows, and the relations between pieces of data become more complicated.

↳ Relational Model Concepts

(1) Attribute : Each column in a table

(2) Tables : Relations are saved in the table format along with entities. It has rows and

Columns as properties Rows represent records and Columns represent attributes.

(3) Tuple : Single row of a table ie single record

(4) Relation Schema : Represents the name of the relation with its attributes.

(5) Degree : Total number of attributes in the relation

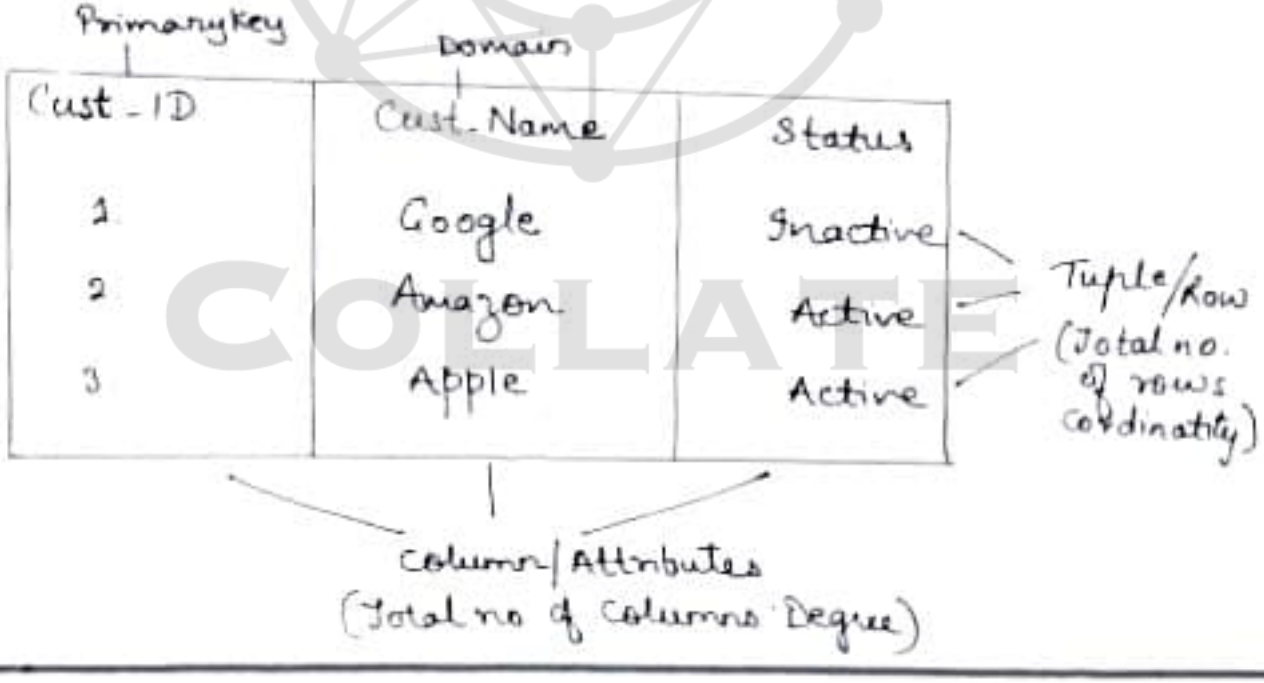
(6) Cardinality : Total number of rows in the table

(7) Relation Instance : Finite set of tuples with no duplicate tuples

(8) Column : Set of values for a specific attribute

(9) Relation Key : Every row has one, two or multiple attributes called the relation key

(10) Attribute Domain : Every attribute has some pre-defined value and scope which is known as attribute domain.



As a rule of thumb, the catalog is placed on a separate disk dence so that it can be managed and tuned independently from other application data.

DBAs should use the catalog to actively manage their database environment. It is a good practice to monitor the database objects and delete the obsolete ones. If a tablespace exists that is no longer used, it is consuming valuable resources that can be freed up when the database object is dropped.

It is possible to directly query the catalog tables using SQL. many DBMs products provide catalog views to simplify writing such quenes.

Example: Oracle provides views for DBA usage and dynamic performance information.

2.3 Types

Collate Notes

Depending on the data several relational database models can be set up.

In each of the model types, it is important to differentiate between a child table (dependent table) and a parent table (Primary table).

The types are given as :-

(1) One-to-many model : The most common relational model where a single record in the parent table relates to

2.2 The Catalog

Collate Notes

The Catalog of any relational model documents the database objects and the system settings being used.

It is important to understand what is in the System Catalog, as well as how to access and manipulate the information it contains.

A Catalog is a "mini database" that Stores special data called Metadata

Metadata can be defined as the data about the data

This metadata contains information on the Schemas, mapping between Schemas, information needed by specific DBMS modules etc

A relational DBMS requires a Catalog but it may be called Something different depending upon the DBMS being used

Example: In oracle it is called the Data Dictionary
In SQL, it is referred to as the System Catalog.
In My SQL, it is referred to as the Information Schema.

The Physical location and setup of the Catalog has an impact on the overall performance of the relational model.

multiple records in the child table. This is called a one-to-many relationship

Example: An application that tracks accounts receivables invoices. One table for invoice data and another for the Customer data. In this case, one Customer can place many orders. So Customers is the parent table. Invoices is the child table and the common field is the Customer table's primary key.

(2) One-to-One Model :- If your data requires that one record in the parent table be related to only one record in the child table, it is a one-to-one model.

The most common use of one-to-one relations is to create separate entity classes to enhance security

Example: In a hospital, each patient's data is a single entity class, but it make sense to create separate tables for the patient's basic information and medical history. The two tables then become related based on a common patient ID key field.

(3) Many-to-many Model : In some cases, data may have many records in one table to many records, in another table. In this case, there is no direct way to establish a common field between the two tables.

2.4 Keys

Key plays an important role in relational database. It is used for identifying unique rows from the table. It also establishes relationship among tables.

The different types of keys used in the relational model are given as:

↳ Super Key

A Super Key is a set of one or more attributes (columns) which can uniquely identify a row in a table

Eg :- EMPLOYEE

EMP. SSN	EMP. NO	EMP. NAME
1246785	105	Steve
999512	106	Ajeet
8854631	107	Chaitanja
762415	109	Robert

The above table has the following Super Keys

{ EMP. SSN }
{ Emp. NO }
{ Emp. SSN , Emp. NO }
{ Emp. SSN , Emp. Name }
{ Emp SN , Emp. NO, Emp. Name }
{ Emp. NO, Emp. Name }

Therefore, the key should be having more than one attribute.

Composite key: {Cust. ID, Prod. code}

↳ Foreign Key

Foreign keys are the columns of a table that points to the primary key of another table. It acts as a cross-reference between tables.

Example:

COURSE

Course. ID	Stu. ID
C01	101
C02	102
C03	101

STUDENT

Stu. ID	Stu. Name	Stu. Age
101	Anya	21
102	Brian	23
103	Jon	22

The Stu. ID column is COURSE table is a foreign key as it points to the primary key of the STUDENT table

↳ Compound Key

A Composite key containing of two or more fields that uniquely describe a row in a table. The difference between compound and candidate is that all the fields in the compound key are foreign keys is the candidate key one or more of the fields may be foreign keys.

↳ Natural Key

A Composite primary key composed of attributes already existing in the real world (Eg: First Name, last Name, etc)

↳ Candidate Key

A Super Key with no redundant attribute is known as Candidate Key.

Candidate Keys are selected from the set of Super Keys and it is also called as minimal Super Key.

Example : The following are Candidate Keys chosen from the Super Keys of Employee table

{Emp. SSN}

{Emp. NO}

↳ Primary Key

A Primary Key is a minimal set of attributes (columns) in a table that uniquely identifies tuples (rows) in that table.

Example :

STUDENT

Stu-ID	Stu. Name	Stu. Age
101	Steve	23
102	John	24
103	Robert	23

Attribute Stu. ID alone is a primary key as each student has a unique ID that can identify the Student record.

A primary key must not have duplicate or null values.

↳ Alternate key

Among all the Candidate Keys, only one key gets selected as the primary key, the remaining keys are known as Alternate or Secondary Keys.

Example: The Candidate Keys of the employee table are {Emp.SSN} and {Emp.No}

DBA can choose any one of them as primary key Lets Say Emp.ID is chosen as primary key Since, we selected Emp.ID as primary key. the remaining Key Emp.No would become the alternate key.

↳ Composite Key

A Key that Consists of more than one attribute to uniquely identify rows in a table is called as a Composite key.

Example

SALES

Cust.ID	Order.ID	Prod.code	Prod.count
C01	0001	P007	23
C02	0123	P007	19
C02	0123	P230	82

None of these columns alone can play a role of key in the table.

Cust.ID alone Cannot be used as Same Customer Can place multiple orders

Order.ID Cannot be primary key as Same order canno Contain order of multiple products

Prod.code Cannot be primary key as more than one Customer can place order for Same product.

Prod.Count alone Can't be primary key as two Orders can be placed for Same product Count

2.6 Fundamental operations

(Collate Notes)

The Select, project and rename operations are called Unary operations because they operate on one relation.

The other three: union, Rename and set difference operate on pairs of relations and are therefore called Binary operations.

↳ SELECT (σ)

The SELECT operation is used for selecting a Subset of the tuples according to a given selection condition

Sigma (σ) symbol denotes it

Syntax:

$\sigma_{P(r)}$ / σ condition/Predicate (Relation/Tablename)

Here, σ : predicate

r : relation (name of the table)

P : propositional logic

Example:

- $\sigma_{topic = "Database"}$ (Tutorials)

  output: selects tuples from tutorials table where topic = "Database"

- $\sigma_{Sales > 5000}$ (customers)

  output: selects tuples from Customers table where Sales is greater than 5000

Collate Notes

2.5 Relational Algebra

Relational Algebra is a procedural query language that works on the relational model. The purpose of a query language is to retrieve data from database or perform various operations such as insert, update, delete on the data. Relational Algebra collects instances of relations as input and gives occurences of relations as output.

It uses various operations to perform this action. The output of these operations is a new relation, which might be formed from one or more input relations.

↳ Types of operations in Relational algebra

→ Fundamental operations

- select (σ)
- Project (π)
- union (∪)
- set difference (−)
- Cartesian product (x)
- Rename (ρ)

→ Additional operations

- Natural Join (⋈)
- left, right, full outer join (⟕ ⟖)
- Intersection (∩)
- Division (÷)

↳ UNION (U)

Union operator is Symbolized by U symbol. Union operation includes all tuples that are in tables. A or in B

It also eliminates duplicate tuples

for Union operation to be valid, following conditions must hold ;-

(1) Both tables must have Same number of attributes.
(2) Attribute domains need to be Compatible
(3) Duplicate tuples should be automatically

Syntax :

table_name 1 U table_name 2

Example :

Table A

Column 1	Column 2
1	1
1	2

Table B

Column 1	Column 2
1	1
1	3

A U B gives

Table A U B

Column 1	Column 2
1	1
1	2
1	3

↳ SET DIFFERENCE (-)

Set difference is denoted by "−" symbol.

The result of set difference on tables R1 and R2, is

↳ PROJECTION (π)

Project operator is denoted by pi (π) symbol. Projection is used to select desired columns from a table.

The projection eliminates all attributes of the input relation but those mentioned in the Projection list.

Syntax :

$$\pi \text{ Column\_Name1, column\_name2, ... Column\_NameN } (\text{table\_name})$$

Example :

Customers

Cust. ID	Cust Name	Status
1	Google	Active
2	Amazon	Active
3	Apple	Inactive

Here, Projection of Cust. Name and status will give :

$$\pi \text{ Cust\_Name, Status (customers)}$$

Output :

Cust_Name	Status
Google	Active
Amazon	Active
Apple	Inactive

a relation which includes all tuples in R1 but not in R2

for set difference operation to be valid, following conditions should be followed

(1) Attribute name of R1 has to match with attribute name in R2

(2) The two operand relations R1 and R2 should either be Compatible or Union Compatible

(3) It should be defined relation consisting of the tuples that are in relation R1 but not in R2.

Syntax:

table_name1 − table_name2

Example:

TABLE A

Column 1	Column 2
1	1
1	2

TABLE B

Column 1	Column 2
1	1
1	3

A−B gives Table A − B

Colum 1	Colum 2
1	2

↳ CARTESIAN PRODUCT (X)

This operation is helpful in merging columns from two relations.

Cartesian product is denoted by X Symbol

Lets Say two relations R1 and R2 then the

Cartesian product of the two (R1 x R2) would combine each tuple of R1 with each tuple of R2

Syntax:

table_name1 x table_name2

Example:

R1

ColA	Col-B
AA	100
BB	200
CC	300

R2

Col-X	Col-Y
xx	99
YY	11
ZZ	101

Table R1 x R2

Col-A	Col-B	Col-X	Col-Y
AA	100	xx	99
AA	100	YY	11
AA	100	ZZ	101
BB	200	xx	99
BB	200	YY	11
BB	200	ZZ	101
CC	300	xx	99
CC	300	YY	11
CC	300	ZZ	101

Note: The number of rows in the output will always be the cross product of number of rows in each table

Here, the output has 3 x 3 = 9 rows

↳ INTERSECTION (∩)

Intersection operator is denoted by ∩ Symbol. It is used to select common rows (tuples) from two tables (relations).

Only those rows that are present in both the tables will appear in the result set.

Let's say we have two relations R_1 and R_2, both have same columns and to select all those tuples (rows) present in both the relations, then we can apply intersection operation.

Syntax:

table_name1 ∩ table_name2

Example:

COURSE

Course_ID	Student_Name	Stu_ID
C101	Aditya	S901
C104	Aditya	S901
C106	Steve	S911
C109	Paul	S921

STUDENT

Stud_ID	student_Name	Stu_Age
S901	Aditya	19
S911	steve	18
S921	Paul	18

Query: Π Student_Name (COURSE) ∩ Π Student_Name (STUDENT)

Output:

Student_Name
Aditya
steve
Paul

↳ NATURAL JOIN (⋈)

Natural join can only be performed if there is a common attribute (column) between the relations. The name and type of the attribute must be same

Syntax:

table name 1 ⋈ table name 2

Example: Table C

Num	Square
2	4
3	9

Table D

Num	Cube
2	8
3	27

Query: C ⋈ D

Table C ⋈ D

Num	Square	Cube
2	4	8
3	9	27

↳ LEFT OUTER JOIN (⟕)

In the left outer join, operation allows keeping all tuple in the left relations. However, if there is no matching tuple found in right relation, then the attributes of right relation in the join result are filled with null values.

Syntax:

table 1 ⟕ table 2

↳ RENAME (ρ)

Rename (ρ) operation can be used to rename a relation or an attribute of a relation.

Syntax:

ρ (new_relation_name, old_relation_name)

Example:

CUSTOMER

Cust_ID	Cust_Name	Cust_City
C1010	Steve	Agra
C1011	Carl	Noida
C1015	Ajay	Delhi

Query: ρ (Cust_Names, Π (Cust_Name) (CUSTOMER))

output: Cust_Names

Steve
Carl
Ajay

2.7 Additional Operations

CollateNotes

The fundamental operations of the relational algebra are sufficient to express any relational algebra query. However, if we restrict ourselves to just the fundamental operations, certain common queries are lengthy to express

Therefore, we define additional operations that do not add any power to the algebra but simplify common queries.

↳ FULL OUTER JOIN (⟗)

In a full outer join, all tuples from both relations are includes in the result, irrespective of the matching condition.

Syntax:

table_name1 ⟗ table_name2

Num	Cube	Square
2	8	4
3	27	9
4	-	16
5	125	-

↳ DIVISION (÷)

The Divison operation is denoted by ÷ Symbol. It is Suited to queries that include the phrase 'for-all'

Division operator on two relations R_1 and R_2 can be applied if

(1) Attributes of R_2 is proper subset of attributes of R_1

(2) The relation return by division operator will have attributes (All attributes of R_1 - All attributes of R_2)

(3) The relation returned by division operator will return those tuple from relation R_1 which are associated to every R_2's tuple

Example:

Table A

Num	Square
2	4
3	9
4	16

Table B

Num	Cube
2	8
3	27
5	125

Query: A ⋈ B

Table A ⋈ B

Num	Square	Cube
2	4	4
3	9	27
4	16	-

↳ RIGHT OUTER JOIN (A ⟖ B)

In the right outer join, operation allows keeping all tuple in the right relation. However if there is no matching tuple found in left relation, then the attributes of the left relation in the join result are filled with the null values.

Syntax:

table_name 1 ⟖ table_name 2

Example:

Table A ⟖ B

Num	Cube	Square
2	8	4
3	27	9
5	125	-

Syntax:

table_name 1 ÷ table_name 2

Example:

STUDENT

RollNo	Name	Address	phone	Age
1	Ajay	Delhi	9143214	18
2	Carl	Noida	4623145	16
3	Steve	Delhi	9514621	19
4	Paul	Agra	9887134	20

ALL_SPORTS

SPORTS
Badminton
Cricket

STUDENT_SPORTS

Roll_No	Sports
1	Badminton
2	Cricket
2	Badminton
4	Badminton

Query: STUDENT_SPORTS ÷ ALL_SPORTS

Output: Roll_No

2

2.8 SQL Fundamentals

SQL stands for Structured Query language It is a type of programming language that's designed specifically for querying updating and retrieving databases.

In SQL we express our requests to the database in the form of "queries"

For example, we might send a query to a database to return a specific subset of data, like a particular table, or to update a particular value in the database.

The SQL language has several parts:

- Interactive data-manipulation language (DML)
  The SQL DML includes a query language based on both the relational algebra and the tuple relational Calculus.
- Integrity
  The SQL DDL includes commands for specifying integrity constraints that the data stored in the database must satisfy
- View Definition
  The SQL DDL includes commands for defining views
- Transaction Control
  SQL includes commands for specifying the beginning and ending of transactions

- Embedded SQL and Dynamic SQL:

Embedded and dynamic SQL define how SQL statements can be embedded within general purpose programming languages such as C, C++, Java, Pascal etc

- Authorisation

The SQL DDL includes commands for specifying access rights to relations and views.

Some of the popular relational database example are MySQL, Oracle, MariaDB, PostgreSQL

SQL is used To perform CRUD (Create, Read, update and Delete) operations on relational databases.

Query: A query is a set of instruction given to the database management System, which tells RDBMS what information to retrieve.

The SQL commands are mainly categorized into four categories are:

(1) Data Definition language (DDL)
(2) Data Query language (DQL)
(3) Data Manipulation language (DML)
(4) Data Control language (DCL)

Syntax:

```
CREATE TABLE table_name
(
  column1 data_type (size)
  column2 data_type (size)

);
```

Example:

```
CREATE TABLE students
(
  ROLL NO int(3),
  Name varchar (20),
  Subject varchar (20)
);
```

(2) DROP

DROP command is used to delete a whole database or a table

The DROP statement destroys the objects like an existing database, table, view or index

Syntax

```
DROP DATABASE database_name,
DROP TABLE table_name;
```

Example:

```
DROP TABLE students
```

(3) TRUNCATE

TRUNCATE statement is used to mark the extents of a table for deallocation (empty for reuse) It quickly remover all data from a table, typically bypassing a number of integrity enforcing mechanisms

Syntax:

```
TRUNCATE TABLE table_name;
```

Example:

```
TRUNCATE TABLE students
```

>> DROP Vs TRUNCATE

(i) Truncate is faster and ideal for deleting data from a temporary table.

(ii) Truncate preserve the structure of the table for future use, unlike drop table where the table is deleted with its full structure.

(iii) Table or database using DROP statement Cannot be rolled back

(4) ALTER

Alter table is used to add, delete or modify columns in the existing table.

It is also used to add and drop various constraints on the existing table

- ALTER TABLE - ADD

Add is used to add columns into the existing table.

Syntax :

```
ALTER TABLE table_name
    ADD (column_name1 datatype,
         column_name2 datatype,
         );
```

Example :

```
ALTER TABLE students ADD ( Age int(3),
Course varchar(40));
```

- ALTER TABLE - DROP

DROP COLUMN is used to drop column in a table ie deleting unwanted columns.

Syntax :

```
ALTER TABLE table_name DROP
COLUMN column_name;
```

Example :

```
ALTER TABLE Students DROP COLUMN Course;
```

- ALTER TABLE MODIFY

It is used to modify the existing columns in a table

:

Multiple columns can also be modified at once

Syntax :

```
ALTER TABLE table_name ALTER COLUMN
column_name column_type
```

2.9 DDL

(Collate Notes)

DDL or data definition Language consists of the SQL commands that can be used to define the database Schema

It simply deals with descriptions of the database schema and is used to create and modify the structure of database objects in the database

DDL Commands are the following:

(1) CREATE

There are two CREATE statements in SQL:

→ CREATE DATABASE

A database is defined as a structured set of data.

The CREATE DATABASE statement is used to create a new database in SQL

Syntax: CREATE DATABASE database name.

Example: CREATE DATABASE my database

→ CREATE TABLE

We need a table to store the data which is done by using CREATE TABLE Command The table comprises of rows and columns. So while creating tables, we have to provide name, data type and size of the column of the data.

⊙ INSERT

The INSERT INTO statement of SQL is used to insert a new row in a table

There are two ways of using INSERT INTO statement for inserting rows.

(1) Only Values : specify only the value of data to be inserted without the column names.

Syntax :

```
INSERT INTO table_name (value 1, value 2, ... );
```

Example :

```
INSERT INTO student VALUES ('5', HARSH',
'MATHS', '15');
```

(2) Column names and values : specify both the columns which we want to fill and their corresponding values.

Syntax :

```
INSERT INTO table_name (column 1, column 2, ... )
VALUES (value 1, value 2, ... );
```

Example :

```
INSERT INTO student (Roll_NO, Name) VALUES
('6', 'Raj')
```

⊙ UPDATE

The update Statement in SQL is used to update the data of an existing table in database. we can update single columns as well as

Example:

```
ALTER TABLE Students MODIFY COURSE
varchar (20),
```

OR

```
ALTER TABLE Students ALTER COLUMN COURSE
varchar (20);
```

- ALTER TABLE - RENAME

We use this statement to rename the table

Syntax:

```
ALTER TABLE table_name RENAME COLUMN
OLD_name to newname;
ALTER TABLE table_name RENAME TO
new_table_name;
```

Example:

```
ALTER TABLE Students RENAME COLUMN Name
To first_name;
ALTER TABLE Students RENAME TO student;
```

2.10 DML

The SQL Commands that deal with the manipulation of data present in the database belongs to DML or Data Manipulation language.

Different DML Commands are given below:-

(2) Deleting Multiple records

Example:

```
DELETE FROM Student, WHERE Age = 18;
```

(3) Delete all records

Example:

```
DELETE FROM Student;
```

OR

```
DELETE * FROM student;
```

2.11 DQL

Collate Notes

The purpose of DQL command is to get some Schema relation based on the query passed to it.

DQL command is given as:

⊙ SELECT

The SELECT statement is used to retrieve or fetch data from a database.

Entire table or some specific data can be fetched according to the clause.

The data returned in stored in a result table also called result set.

Syntax:

```
SELECT Column1, Column2, FROM table name
```

Examples:

(1) Fetch entire table

```
SELECT * from table.name;
SELECT * from student
```

multiple columns using this statement.

Syntax:

```
UPDATE table_name SET Column1 = value1,
Column2 = value2, ..... WHERE Condition;
```

→ Updating Single column

Example:

```
UPDATE Student SET Name = 'PRATIK'
Where Age = 20;
```

→ Updating multiple columns

Example:

```
UPDATE Student SET Name = 'PRATIK'
Subject = 'English' WHERE ROLLNO = 5;
```

→ Omitting WHERE

Example:

```
UPDATE Student SET Name = 'Pratik'
```

⊙ DELETE

The DELETE statement in SQL is used to delete existing records from a table. Single or multiple records can be deleted depending upon the Condition specified

Syntax:

```
DELETE FROM table_name WHERE Some_Conditions;
```

(1) Deleting Single record

Example:

```
DELETE FROM student WHERE Name = 'Raj'
```

A Cursor holds the rows returned by an SQL Statement

The Set of rows the Cursor holds is referred to as the active set.

There are two types of Cursors
→ Implict Cursors
→ Explicit Cursors

(1) Implict Cursors

Implicit Cursors are automatically created whenever an SQL statement is executed Programmers cannot control the implicit Cursors and the information in it.

For INSERT operations the Cursor holds the data that needs to be inserted

For UPDATE and DELETE operations, the Cursor identifies the rows that would be affected

The description of the most used attributes is given as :

→ % FOUND : Returns TRUE if an INSERT, UPDATE or DELETE statement affected one or more rows or a SELECT INTO statement returned one or more rows otherwise, it returns FALSE

→ % NOTFOUND : Logical opposite of % FOUND Returns TRUE if INSERT, UPDATE or DELETE Statement affected no rows, or a SELECT

(2) Fetch specified fields

SELECT Name, Age FROM Student;

2.12 DCL

(CollateNotes)

DCL or Data Control language includes commands such as GRANT and REVOKE which deal with the rights, permissions and other controls of the database system.

DCL commands are given as

⊙ GRANT

The GRANT Statements is used to provide any user access privileges or other privileges for the database

Syntax :

→ GRANT CREATE SESSION TO Username;

→ GRANT CREATE TABLE TO Username;

⊙ REVOKE

The REVOKE Statement is used to take back permissions from any user.

Syntax :

REVOKE CREATE TABLE FROM Username;

Example :

GRANT CREATE TABLE TO Raymond;

REVOKE CREATE TABLE FROM Raymond;

2.13 PL/SQL

(Collate Notes)

The PL/SQL programming language was developed by oracle corporation in late 1980s as Procedural extension language for SQL and the oracle relational database

↳ Features of PL/SQL

⇒ It is completely portable high performance transaction processing language
⇒ It provides a built in interpreted and OS independent programming environment
⇒ PL/SQL is tightly integrated with SQL
⇒ It offers extensive error checking
⇒ It supports structured programming through functions and procedures
⇒ It supports object oriented programming
⇒ PL/SQL supports the development of web applications and Server pages
⇒ Direct call can also be made from external Programming language calls to database
⇒ PL/SQL's general syntax is based on that of ADA and pascal programming language.

↳ Advantages

1) PL/SQL Supports both static and dynamic SQL. Static SQL supports DML operations and transaction control from PL/SQL block In

Dynamic SQL. SQL allows embedding DDL Statements in PL/SQL blocks

(2) PL/SQL allows sending an entire block of Statements to the database at one time which reduces network traffic and provides high performance for the applications

(3) It gives high productivity to programmers as it can query, transfer and update data in a database.

(4) It saves time on design and debugging due to exception handling, encapsulation, data hiding and object oriented data types

(5) PL/SQL provides high level security

(6) PL/SQL provides access to predefined SQL packages

(7) PL/SQL provides support for object oriented programming

(8) It provides support for developing web applications and Server pages

Syntax

```
DECLARE
    < declarations section >

BEGIN
    < executable command(s) >
```

```
EXCEPTION
    <exception handling>
END ;
```

Example:

```
DECLARE
  message varchar(20) := 'Hello World!';
BEGIN
    dbms_output.put_line(message);
END;
/
```

NOTE: To run the code from SQL command line you may need to type 1 at the beginning of the first blank line after last line of the code

Output

```
Hello World
PL/SQL procedure successfully completed
```

2.14 Cursor

CollateNotes

A Cursor is a pointer to context area in PL/SQL. The context area is a memory area that contains all the information needed for processing an SQL statement.

PL/SQL controls are the context area through a Cursor

Example:

```
DECLARE
        c_id customers.id %type
        c_name customers.NO.ami%. type;
        CURSOR c_customers is
          SELECT ID-Name. FROM customers;
BEGIN
        OPEN c.customers
        LOOP
        FETCH c.customers into c_id, c_name;
              EXIT. WHEN c_customers %. not-found;
              dbms. output. put_line (c_id ||
              || c_name ||' ');
        END LOOP;
        CLOSE c.customers;
        END;
```

output :

1. Ramesh
2. Hardik
3. Konal
4. chaitali

PL/SQL procedure successfully completed.

2.15 Stored Procedures

CollateNotes

A stored procedures in PL/SQL is a series of declarative SQL statements which can be stored in the database catalog. If you have an SQL query that is being written over and

INTO Statement returned no rows otherwise it returns FALSE.

→ %ISOPEN : Always returns FALSE for Complicit Cursors as the SQL Cursor automatically closes after execution.

→ %ROWCOUNT : Returns the number of rows affected by an INSERT, UPDATE or DELETE Statement or returned by a SELECT INTO Statement.

Example :

CUSTOMERS

ID	Name	Age	salary
1	Ramesh	23	2000
2	Hardik	25	2500
3	Komal	29	4600
4	Chaitali	35	7500

```
DECLARE
    total_rows number(2);
BEGIN
    update customers
    SET salary = salary + 500;
    IF SQL%. not found THEN
      dbms_output.put_line('no customers
                            Selected');
    ELSIF SQL%found THEN
      total_rows := SQL%rowcount;
      dbms_output.put_line(total_rows ||
        ' customers Selected');
```

```
    END IF;
END;
```

Output 4 Customers selected

PL/SQL procedure Successfully Completed.

(2.) EXPLICIT CURSORS

Explicit Cursors are programmer defined Cursors for gaining more control over the Context area.

An explicit cursor should be defined in the declaration section of the PL/SQL block. It is created on a 'SELECT' statement which returns more than one row.

Syntax:

```
CURSOR Cusor_name IS select_statement;
```

Steps to include an explicit cursor:

(a) Declaring the Cursor with a name and the associated SELECT Statement for initiating the memory.

(b) opening the Cursor allocates the memory for the cursor and makes it ready for fetching the rows returned by the SQL statement into it.

(c) Fetching the Cursor involves accessing one rows at a time.

(d) Closing the cursor to release the allocated memory.

and over again, then save it as a stored procedure and just call it to execute it

You can also pass parameters to a stored procedure

↳ Advantages

→ Improves performance of application
→ Reduces traffic between the database and application as the queries are not being sent again and again
→ Adds to code reusability similar to how functions and methods work in other languages such as C/C++ and Java.

↳ Disadvantages

→ Stored procedures can cause a lot of memory usage
→ My SQL does not provide debugging functionality for stored procedures.

Syntax:

```
CREATE PROCEDURE Procedure_name
AS
SQL_Statement
GO;
```

Execution `EXEC Procedure_name;`

Example:

Customers

Cust ID	Cust. Name	city	Age
1	Alfreds	Berlin	18
2	Maria	Mexico	46
3	Ana	London	25

> CREATE PROCEDURE Select All Customers

As

SELECT * FROM Customers

Go;

Execution: EXEC Select All Customers;

> CREATE PROCEDURE Select All Customers @ Age number (2), @ city nvarchar (30)

As

SELECT * FROM Customers WHERE city = @city AND Age = @ Age

Go;

Execution: EXEC Select All customers @ city: 'London' @ Age = 25

2.16 Stored Functions

Collate Notes

A stored function in PL/SQL is same as a procedure except that it returns a value. A function can be used as a part of SQL expression to perform select / update / merge commands.

```
    END IF;
    RETURN z;
END;
BEGIN
    a := 23;
    b := 45;
    c := findMax(a,b)
    dbms_output.put_line('Maximum of (23,45):
                ' ||c);
    END
```

output: Maximum of (23,45): 45
PL/SQL procedure Successfully completed.

2.17 Triggers

CollateNotes

Triggers in PL/SQL are stored programs which are automatically executed or fired when some events occur.

Triggers can be defined on the table, view, schema or database with which the event is associated.

Advantages :-

(a) generating some derived column values automatically
(b) Event logging and storing information on table access
(c) Enforces referential integrity
(d) Auditing
(e) Synchronous replication of tables

```
DECLARE
    c number (2);
BEGIN
    c := total customers ();
    dbms_output.put_line ('Total number of
                Customer ' || c);
END;
```

Output :

Total number of customer : 4
PL/SQL procedure successfully completed

Example 2 : Declaring, defining and invoking a PL/SQL function that computes and returns maximum of two values.

```
DECLARE
    a number;
    b number;
    c number;
FUNCTION findmax (x IN number, y IN number)
  RETURN number
  IS
    z number;
  BEGIN
    IF x > y THEN
      z := x;
    ELSE
      z := y;
    END IF;
```

Syntax:

```
CREATE [OR REPLACE] FUNCTION function_name
[ (Parameter name [IN|OUT|IN OUT] type [, ...]) ]
RETURN return_datatype
    { IS/AS }
BEGIN
    < function body >
END [function_name];
```

Example 1:

Customers

ID	Name	Age	City	Amount
1	Maria	32	london	2500
2	Ana	25	Mexico	407
3	Ramesh	41	Delhi	1000
4	Komal	19	Agra	5000

Creating a Function

```
> CREATE OR REPLACE FUNCTION total Customers
    RETURN number
    IS
      total number(2) := 0;
    BEGIN
      SELECT count(*) into total
      FROM customers,
      RETURN total
    END
```

Calling the function

(f) Imposing Security authorizations
(g) Preventing invalid transactions

Syntax:

```
CREATE [OR REPLACE] TRIGGER trigger_name
{BEFORE | AFTER | INSTEAD OF }
{INSERT [OR] | UPDATE [OR] | DELETE }
[OF col_name]
 ON table_name
[REFERENCING OLD AS o NEW AS n]
[FOR EACH ROW]
 WHEN (condition)
 DECLARE
     Declaration-statements
  BEGIN
     Executable-statements
  EXCEPTION
       Exception-handling-statements
   END
```

Example:

Student

Roll_NO	Name	Age	Course
11	Anu	20	BSC
12	Ava	21	B.Com
13	Asha	19	BBA

> CREATE OR REPLACE TRIGGER checkAge
BEFORE
INSERT OR UPDATE ON student

```
FOR EACH ROW
BEGIN
  IF !new.Age>30 THEN
      raise_application_error(-20001,'Age should
        not be greater than 30');
  END IF;
END;
```

output: Trigger created

> INSERT INTO student values (16,'sana',32,'Bcom');

output: Age should not be greater than 30.

2.18 Database Integrity

CollateNotes

The term database integrity refers to the accuracy and consistency of data.

Maintaining data have integrity means making sure the data remains intact and unchanged throughout its entire life cycle. This includes the capture of the data, storage, updates, transfers, backups etc. Everytime data is processed there is a risk it could get corrupted.

↳ Risks to database integrity

⇒ User tries to enter a data outside acceptable range

⇒ User tries to update a primary key value when

there's already a foreign key in a related table pointing to that value.

⇒ A bug in application attempts to delete the wrong record

→ Network lost while transferring data between two databases

⇒ Hacker steals all user passwords from the database

⇒ Irregular backups of the database

→ A user tries to delete a record in a table, but another table is referring that record as a part of relationship

Many of these risks can be addressed from within the database itself through the use of data types and constraints against each column while others can be addressed through other features of the DBMS like regular backups, testing etc

Some of the risks require other factors to be present such as an offsite backup location, Proper training, Security polices etc.