LTI
Let's Solve

Celebrating 20 Years

Database

A Larsen & Toubro Group Company

Introduction to DBMS

Introduction to DBMS

- What is Database & DBMS?
- The need for a database
- The File-Based Systems
- Features of DBMS
- Usage of Database

Database Management System (DBMS)

- Database can be defined as the storage of inter related data that has been organized in such a fashion that the process of retrieving data is effective and efficient

- DBMS contains information about a particular enterprise
  - Collection of interrelated data
  - Set of programs to access the data
  - An environment that is both *convenient* and *efficient* to use

- Database Applications:
  - Banking: all transactions
  - Airlines: reservations, schedules
  - Universities: registration, grades
  - Sales: customers, products, purchases
  - Online retailers: order tracking, customized recommendations

Purpose of Database Systems

- In the early days, database applications were built directly on top of file systems
- Drawbacks of using file systems to store data:
  - Data redundancy and inconsistency
  - Difficulty in accessing data
  - Data isolation — multiple files and formats
  - Integrity problems

Purpose of Database Systems (Cont.)

- Drawbacks of using file systems (cont.)
  - Atomicity of updates
    - Failures may leave database in an inconsistent state with partial updates carried out
    - Example: Transfer of funds from one account to another should either complete or not happen at all
  - Concurrent access by multiple users
    - Concurrent accessed needed for performance
    - Uncontrolled concurrent accesses can lead to inconsistencies
      - Example: Two people reading a balance and updating it at the same time
  - Security problems
    - Hard to provide user access to some, but not all, data
- Database systems offer solutions to all the above problems

DBMS Architecture

DBMS Architecture

- Three-level architecture of DBMS
- The functions of Database Systems
- Overall system architecture

Levels of Architecture

- **Physical level:**

- Physical level describes the physical storage structure of data in database
- It is also known as Internal Level
- This level is very close to physical storage of data
- At lowest level, it is stored in the form of bits with the physical addresses on the secondary storage device
- At highest level, it can be viewed in the form of files
- The internal schema defines the various stored data types. It uses a physical data model

Levels of Architecture

- **Logical/Conceptual level:**

- Conceptual level describes the structure of the whole database for a group of users
- It is also called as the data model
- Conceptual schema is a representation of the entire content of the database
- These schema contains all the information to build relevant external records
- It hides the internal details of physical storage

Levels of Architecture

- **View/External level:** application programs hide details of data types. Views can also hide information (such as an employee's salary) for security purposes

- External level is related to the data which is viewed by individual end users
- This level includes a no. of user views or external schemas
- This level is closest to the user
- External view describes the segment of the database that is required for a particular user group and hides the rest of the database from that user group
- At lowest level, it is stored in the form of bits with the physical addresses on the secondary storage device
- At highest level, it can be viewed in the form of files

Introduction to Data Modelling

Introduction to Data Modeling

- Explain the structure of Data
- Explain the process of data access in various data-models
- Explain the steps involved in the database designing pattern
- Design a Conceptual database using ER model

Data Models

- According to Hoberman (2009),

  "A data model is a way of finding the tools for both business and IT professionals, which uses a set of symbols and text to precisely explain a subset of real information to improve communication within the organization and thereby lead to a more flexible and stable application environment"

  A data model is an idea which describes how the data can be represented and accessed from software system after its complete implementation

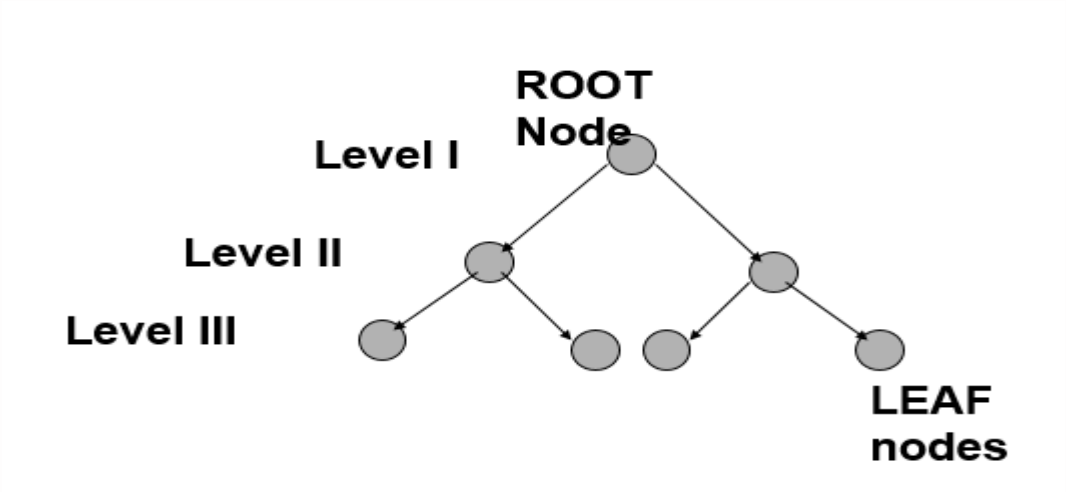
- It is a simple abstraction of complex real world data gathering environment
- It defines data elements and relationships among various data elements for a specified system
- The main purpose of data model is to give an idea that how final system or software will look like after development is completed

Data Models

- Types
  - Hierarchical DBMS
  - Network DBMS
  - Relational DBMS
  - Object Relational DBMS
  - Object Oriented DBMS

Hierarchical Data Model

- Definition
  - A hierarchical data model is a model that organizes data in a hierarchical tree structure
- Description
  - A hierarchical tree structure is made up of nodes and branches
  - The dependent nodes are at lower levels in the tree



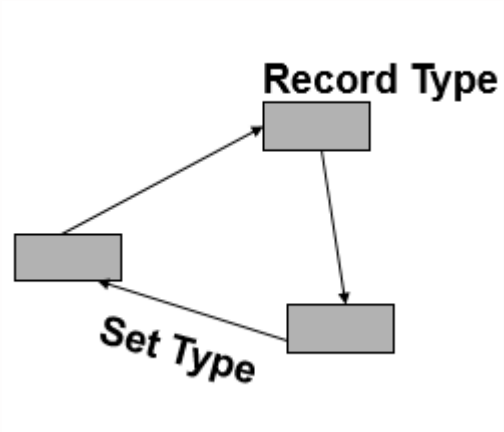
Network Data Model

Definition

- The network data model interconnects the entities of an enterprise into a network

Description

- A block represents an entity or record type. Each record type is composed of zero, one, or more attributes



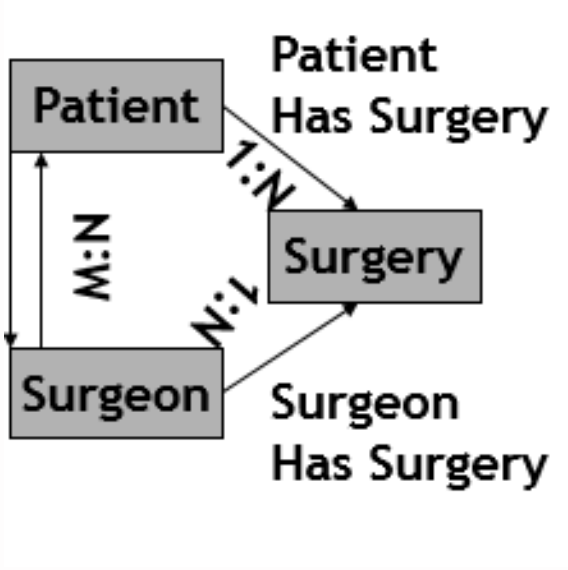
Network Data Model

1:N Relationship

- An owner record type owns zero, one, or many occurrences of a member record type

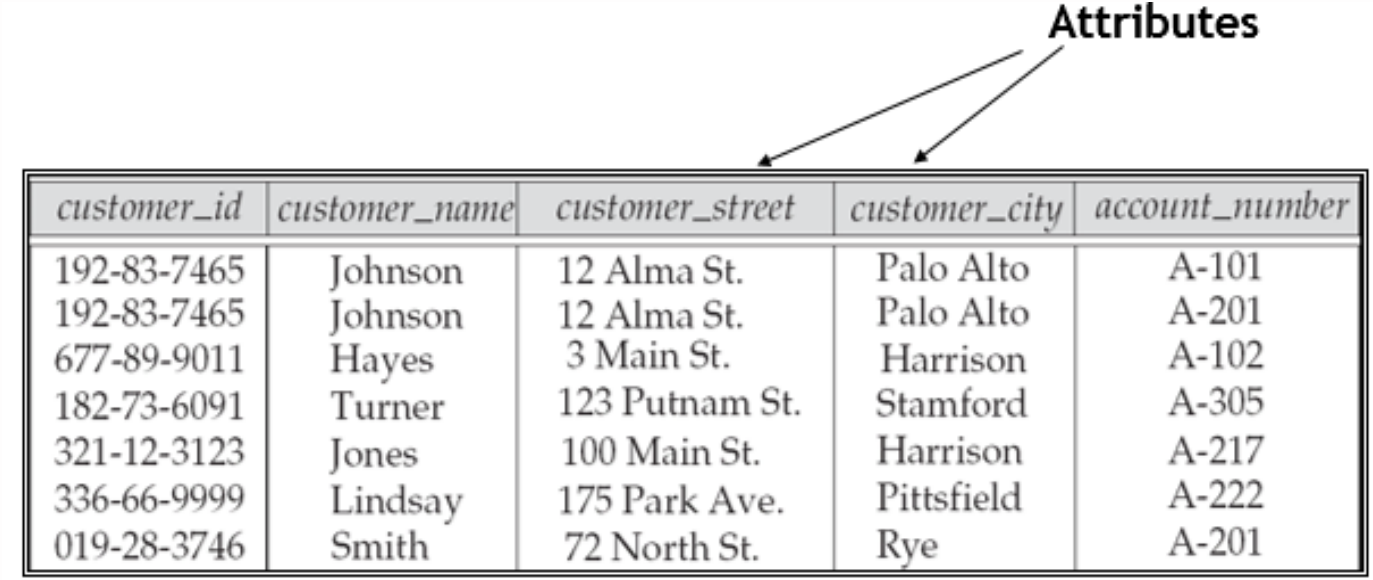
M:N Relationship

- A many-to-many relationship can be implemented by creating two one-to-many relationships
- Two record types are connected with a third entity type called connector record type
- In this case member record type has two owner record type



Relational Model

- Example of tabular data in the relational model



customer_id	*customer_name*	*customer_street*	*customer_city*	*account_number*
192-83-7465	Johnson	12 Alma St.	Palo Alto	A-101
192-83-7465	Johnson	12 Alma St.	Palo Alto	A-201
677-89-9011	Hayes	3 Main St.	Harrison	A-102
182-73-6091	Turner	123 Putnam St.	Stamford	A-305
321-12-3123	Jones	100 Main St.	Harrison	A-217
336-66-9999	Lindsay	175 Park Ave.	Pittsfield	A-222
019-28-3746	Smith	72 North St.	Rye	A-201

Database Design

The process of designing the general structure of the database:

- Logical Design – Deciding on the database schema. Database design requires that we find a “good” collection of relation schemas
  - Business decision – What attributes should we record in the database?
  - Computer Science decision – What relation schemas should we have and how should the attributes be distributed among the various relation schemas?
- Physical Design – Deciding on the physical layout of the database

Other Data Models(Self Study)

- Object-oriented data model
- Object-relational data model

Relational Model

- Structure of Relational Databases
- Fundamental Relational-Algebra-Operations
- Additional Relational-Algebra-Operations
- Extended Relational-Algebra-Operations
- Null Values
- Modification of the Database

Example of a Relation

account_number	*branch_name*	*balance*
A-101	Downtown	500
A-102	Perryridge	400
A-201	Brighton	900
A-215	Mianus	700
A-217	Brighton	750
A-222	Redwood	700
A-305	Round Hill	350

Relational Algebra

- Procedural language
- Six basic operators
  - select: σ
  - project: Π
  - union: $\cup$
  - set difference: –
  - Cartesian product: x
  - rename: ρ
- The operators take one or two relations as inputs and produce a new relation as a result

Select Operation(σ) – Example

- Syntax: $\sigma_p(r)$

Where, σ represents the Select Predicate, r is the name of relation(table name in which you want to look for data), and p is the prepositional logic, where we specify the conditions that must be satisfied by the data

e.g:

$\sigma_{age > 17}$ (Student)

This will fetch the tuples(rows) from table Student, for which age will be greater than 17

σage > 17 and gender = 'Male' (Student)

This will return tuples(rows) from table Student with information of male students, of age more than 17.(Consider the Student table has an attribute Gender too)

Project Operation(∏) – Example

- Project operation is used to project only a certain set of attributes of a relation

  Syntax: ∏A1, A2...(r)

  where A1, A2 etc are attribute names(column names)

  For example,

  ∏Name, Age(Student)

  Above statement will show us only the Name and Age columns for all the rows of data in Student table

Union Operation(∪) – Example

- This operation is used to fetch data from two relations(tables) or temporary relation(result of another operation)
- For this operation to work, the relations(tables) specified should have same number of attributes(columns) and same attribute domain. Also the duplicate tuples are automatically eliminated from the result

Syntax: A ∪ B

where A and B are relations.

For example, if we have two tables RegularClass and ExtraClass, both have a column student to save name of student, then,

∏Student(RegularClass) ∪ ∏Student(ExtraClass)

Above operation will give us name of Students who are attending both regular classes and extra classes, eliminating repetition

Intersection Operation(∩) – Example

- Defines a relation consisting of a set of all tuple that are in both A and B

Syntax: A ∩ B

where A and B are relations

For example, if we want to find name of students who attend the regular class and the extra class as well, then, we can use the below operation:

∏Student(RegularClass) ∩ ∏Student(ExtraClass)

Set Difference Operation – Example

- This operation is used to find data present in one relation and not present in the second relation. This operation is also applicable on two relations, just like Union operation

Syntax: A - B

where A and B are relations

For example, if we want to find name of students who attend the regular class but not the extra class, then, we can use the below operation:

∏Student(RegularClass) - ∏Student(ExtraClass)

Cartesian-Product Operation – Example

- Cross/Cartesian product is used to join two relations. For every row of Relation1, each row of Relation2 is concatenated. If Relation A has m tuples and and Relation B has n tuples, cross product of A and B will have m X n tuples

Syntax:

A X B

$\sigma_{\text{column 2} = \text{'1'}}(A \times B)$

Keys Concept in DBMS

KEYS concept in RDBMS

- Super Key
- Primary Key
- Candidate Key
- Alternate Key
- Foreign Key
- Composite Key

Super Keys

Super Key is defined as a set of attributes within a table that can uniquely identify each record within a table. Super Key is a superset of Candidate key

Primary Keys

A column or group of columns in a table which helps us to uniquely identifies every row in that table is called a primary key

Candidate Keys

A super key with no redundant attribute is known as candidate key

Composite Keys

A key that consists of more than one attribute to uniquely identify rows (also known as records & tuples) in a table is called composite key

Alternate keys

All the keys which are not primary key are called an alternate key. It is a candidate key which is currently not the primary key

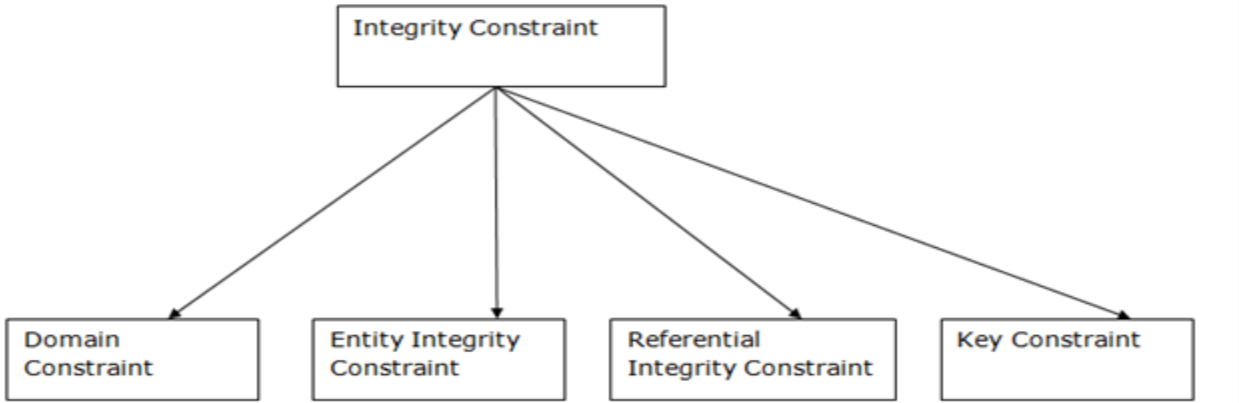
Foreign Keys

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

Constraints in DBMS

Integrity Constraint

- Integrity constraints are a set of rules. It is used to maintain the quality of information
- Integrity constraints ensure that the data insertion, updating, and other processes have to be performed in such a way that data integrity is not affected
- Thus, integrity constraint is used to guard against accidental damage to the database



Domain Integrity

- Domain constraints can be defined as the definition of a valid set of values for an attribute
- The data type of domain includes string, character, integer, time, date, currency, etc. The value of the attribute must be available in the corresponding domain

Entity integrity

- The entity integrity constraint states that primary key value can't be null
- This is because the primary key value is used to identify individual rows in relation and if the primary key has a null value, then we can't identify those rows
- A table can contain a null value other than the primary key field

Referential Integrity

- A referential integrity constraint is specified between two tables
- In the Referential integrity constraints, if a foreign key in Table 1 refers to the Primary Key of Table 2, then every value of the Foreign Key in Table 1 must be null or be available in Table 2

Entity integrity

- Keys are the entity set that is used to identify an entity within its entity set uniquely
- An entity set can have multiple keys, but out of which one key will be the primary key. A primary key can contain a unique and null value in the relational table

Introduction to Normalization

Introduction to Normalization

- Explain the role of Normalization in database design
- Explain the steps in Normalization
- Types of Normal Forms – 1NF, 2NF, 3NF and Boyce Codd Normal Form(BCNF)

Normalization

- Normalization is the process of organizing the data in the database
- Normalization is used to minimize the redundancy from a relation or set of relations. It is also used to eliminate the undesirable characteristics like Insertion, Update and Deletion Anomalies
- Normalization divides the larger table into the smaller table and links them using relationship
- The normal form is used to reduce redundancy from the database table

Anomalies

Relations that have redundant data may have problems called **anomalies**, which are classified as :

- Insertion anomalies
- Deletion anomalies
- Modification anomalies

STUDENT

STUD_NO	STUD_NAME	STUD_PHONE	STUD_STATE	STUD_COUNTRY	STUD_AGE
1	RAM	9716271721	Haryana	India	20
2	RAM	9898291281	Punjab	India	19
3	SUJIT	7898291981	Rajsthan	India	18
4	SURESH		Punjab	India	21

Table 1

STUDENT_COURSE

STUD_NO	COURSE_NO	COURSE_NAME
1	C1	DBMS
2	C2	Computer Networks
1	C2	Computer Networks

Table 2

Insertion Anomalies

If a tuple is inserted in referencing relation and referencing attribute value is not present in referenced attribute, it will not allow inserting in referencing relation. For Example, If we try to insert a record in STUDENT_COURSE with STUD_NO =7, it will not allow

STUDENT

STUD_NO	STUD_NAME	STUD_PHONE	STUD_STATE	STUD_COUNTRY	STUD_AGE
1	RAM	9716271721	Haryana	India	20
2	RAM	9898291281	Punjab	India	19
3	SUJIT	7898291981	Rajsthan	India	18
4	SURESH		Punjab	India	21

Table 1

STUDENT_COURSE

STUD_NO	COURSE_NO	COURSE_NAME
1	C1	DBMS
2	C2	Computer Networks
1	C2	Computer Networks

Table 2

Updation & Deletion Anomalies

- If a tuple is deleted or updated from referenced relation and referenced attribute value is used by referencing attribute in referencing relation, it will not allow deleting the tuple from referenced relation. For Example, If we try to delete a record from STUDENT with STUD_NO =1, it will not allow. To avoid this, following can be used in query:

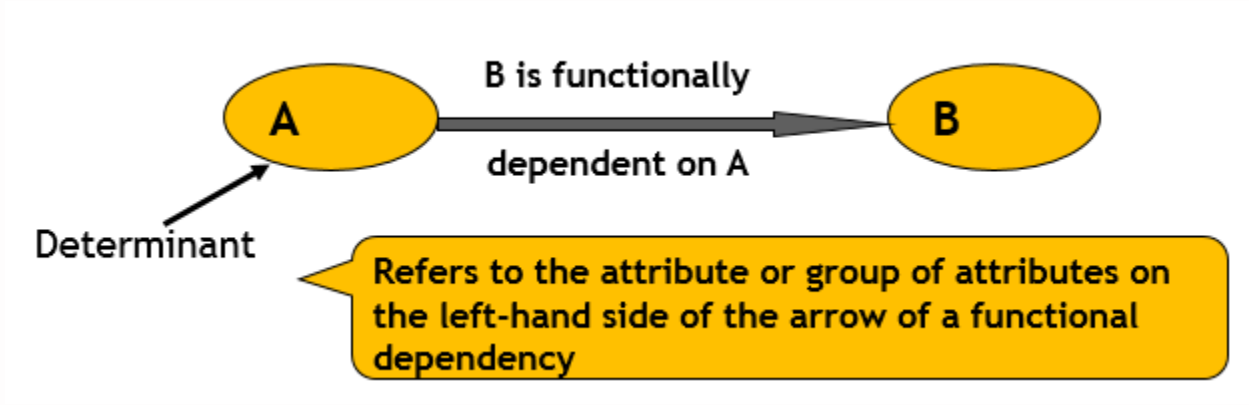
- ON DELETE/UPDATE SET NULL: If a tuple is deleted or updated from referenced relation and referenced attribute value is used by referencing attribute in referencing relation, it will delete/update the tuple from referenced relation and set the value of referenced attribute to NULL

- ON DELETE/UPDATE CASCADE: If a tuple is deleted or updated from referenced relation and referenced attribute value is used by referencing attribute in referencing relation, it will delete/update the tuple from referenced relation and referencing relation as well

Functional Dependencies

Functional dependency describes the relationship between attributes in a relation

For example, if A and B are attributes of relation R, and B is functionally dependent on A (denoted A B), if each value of A is associated with exactly one value of B. (A and B may each consist of one or more attributes)



Functional Dependencies (2)

Trival functional dependency:

- A → B has trivial functional dependency if B is a subset of A
- The following dependencies are also trivial like: A → A, B → B
- **Example:**
- Consider a table with two columns Employee_Id and Employee_Name
- {Employee_id, Employee_Name} → Employee_Id is a trivial functional dependency as
- Employee_Id is a subset of {Employee_Id, Employee_Name}
- Also, Employee_Id → Employee_Id and Employee_Name → Employee_Name are trivial dependencies too

Functional Dependencies (2)

Non-trival functional dependency:

- A → B has a non-trivial functional dependency if B is not a subset of A
- When A intersection B is NULL, then A → B is called as complete non-trivial
- **Example:**
- ID → Name,
- Name → DOB

Functional Dependencies (2)

Transitive dependency:

- A transitive is a type of functional dependency which happens when t is indirectly formed by two functional dependencies

Example:

Company	CEO	Age
Microsoft	Satya Nadella	51
Google	Sundar Pichai	46
Alibaba	Jack Ma	54

{Company} -> {CEO} (if we know the compay, we know its CEO's name)

{CEO } -> {Age} If we know the CEO, we know the Age

Therefore according to the rule of rule of transitive dependency:

{ Company} -> {Age} should hold, that makes sense because if we know the company name, we can know his age

Functional Dependencies (5)

Inference Rules

A set of all functional dependencies that are implied by a given set of functional dependencies X is called closure of X, written X+. A set of inference rule is needed to compute X+ from X

Armstrong's axioms

1. Reflexivity:	If B is a subset of A, them A → B
2. Augmentation:	If A → B, then A, C → B
3. Transitivity:	If A → B and B → C, then A → C
4. Self-determination:	A → A
5. Decomposition:	If A → B,C then A → B and A → C
6. Union:	If A → B and A → C, then A → B,C
7. Composition:	If A → B and C → D, then A,C → B,

More..

Types of Normal Forms

Normal Form	Description
1NF	A relation is in 1NF if it contains an atomic value
2NF	A relation will be in 2NF if it is in 1NF and all non-key attributes are fully functional dependent on the primary key
3NF	A relation will be in 3NF if it is in 2NF and no transition dependency exists
4NF/BCNF	A relation will be in 4NF if it is in Boyce Codd normal form and has no multi-valued dependency

First Normal Form (1NF)

- A relation will be 1NF if it contains an atomic value
- It states that an attribute of a table cannot hold multiple values. It must hold only single-valued attribute
- First normal form disallows the multi-valued attribute, composite attribute, and their combinations

First Normal Form (1NF)

- **Example:** Relation EMPLOYEE is not in 1NF because of multi-valued attribute EMP_PHONE

EMPLOYEE table:

EMP_ID	EMP_NAME	EMP_PHONE	EMP_STATE
14	John	7272826385, 9064738238	UP
20	Harry	8574783832	Bihar
12	Sam	7390372389, 8589830302	Punjab

First Normal Form (1NF)

The decomposition of the EMPLOYEE table into 1NF has been shown below:

EMP_ID	EMP_NAME	EMP_PHONE	EMP_STATE
14	John	7272826385	UP
14	John	9064738238	UP
20	Harry	8574783832	Bihar
12	Sam	7390372389	Punjab
12	Sam	8589830302	Punjab

Second Normal Form (2NF)

Second normal form (2NF) is a relation that is in first normal form and every non-primary-key attribute is fully functionally dependent on the primary key

The normalization of 1NF relations to 2NF involves the removal of **partial dependencies**. If a partial dependency exists, we remove the function dependent attributes from the relation by placing them in a new relation along with a copy of their determinant

Second Normal Form (2NF)

For example:

Consider the a table in which there are three below columns:

STUD_NO	COURSE_NO	COURSE_FEE
1	C1	1000
2	C2	1500
1	C4	2000
4	C3	1000
4	C1	1000
2	C5	2000

COURSE_NO -> COURSE_FEE , i.e., COURSE_FEE is dependent on COURSE_NO, which is a proper subset of the candidate key. Non-prime attribute COURSE_FEE is dependent on a proper subset of the candidate key, which is a partial dependency and so this relation is not in 2NF

Third Normal Form (3NF)

Transitive dependency

A condition where A, B, and C are attributes of a relation such that if A → B and B → C, then C is transitively dependent on A via B (provided that A is not functionally dependent on B or C)

Third normal form (3NF)

A relation that is in first and second normal form, and in which no non-primary-key attribute is transitively dependent on the primary key

The normalization of 2NF relations to 3NF involves the removal of transitive dependencies by placing the attribute(s) in a new relation along with a copy of the determinant

3NF relation

Example: Suppose a company wants to store the complete address of each employee, they create a table named employee details that looks like this:

emp_id	emp_name	emp_zip	emp_state	emp_city	emp_district
1001	John	282005	UP	Agra	Dayal Bagh
1002	Ajeet	222008	TN	Chennai	M-City
1006	Lora	282007	TN	Chennai	Urrapakkam
1101	Lilly	292008	UK	Pauri	Bhagwan
1201	Steve	222999	MP	Gwalior	Ratan

Here, emp_state, emp_city & emp_district dependent on emp_zip. And, emp_zip is dependent on emp_id that makes non-prime attributes (emp_state, emp_city & emp_district) transitively dependent on super key (emp_id). This violates the rule of 3NF

3NF relation

employee table:

emp_id	emp_name	emp_zip

employee_zip table:

emp_zip	emp_state	emp_city	emp_district

Structured Query Language (SQL)

SQL - Basic Operations

- Work with the SQL Data Definition Language (DDL)
- Work with the SQL Data Manipulation Language (DML)
- Write Queries using SQL select statements
- Work with SQL Operators
- Work with SQL Functions

Data Definition Language (DDL)

- Data Definition Language (DDL) is a standard for commands that define the different structures in a database
- DDL Statements are
- CREATE :Use to create objects like CREATE TABLE, CREATE FUNCTION, CREATE SYNONYM, CREATE VIEW. Etc.
- ALTER :Use to Alter Objects like ALTER TABLE, ALTER USER, ALTER TABLESPACE, ALTER DATABASE. Etc.
- DROP :Use to Drop Objects like DROP TABLE, DROP USER, DROP TABLESPACE, DROP FUNCTION. Etc.
- REPLACE :Use to Rename table names
- TRUNCATE :Use to truncate (delete all rows) a table

Data Definition Language (DDL)

- create table emp (empno number(5) primary key,
  name varchar2(20),
  sal number(10,2),
  job varchar2(20),
  mgr number(5),
  Hiredate date,
  comm number(10,2));

- Now Suppose you have emp table now you want to create a TAX table with the following structure and also insert rows of those employees whose salary is above 5000

```
create table tax (empno number(5), tax number(10,2));

insert into tax select empno,(sal-5000)*0.40
        from emp where sal > 5000;
```

Data Definition Language (DDL)

- Instead of executing the above two statements the same result can be achieved by giving a single CREATE TABLE AS statement

```
create table tax as select empno,(sal-5000)*0.4
  as tax from emp where sal>5000
```

- Alter

- Use the ALTER TABLE statement to alter the structure of a table

Examples:

- To add new columns addr, city, pin, ph, fax to employee table you can give the following statement

```
alter table emp add (addr varchar2(20), city varchar2(20),
   pin varchar2(10),ph varchar2(20));
```

Data Definition Language (DDL)

- To drop columns

For example to drop PIN, CITY columns from emp table

```
alter table emp drop column (pin, city);
```

- Remember you cannot drop the column if the table is having only one column
- If the column you want to drop is having primary key constraint on it then you have to give cascade constraint clause

```
alter table emp2 drop column (empno) cascade constraints;
```

Data Definition Language (DDL)

- Using ALTER to add, modify or remove constraints:
- INTEGRITY CONSTRAINTS
- Integrity Constraints are used to prevent entry of invalid information into tables. There are five Integrity Constraints Available in Oracle. They are :
- Not Null
- Primary Key
- Foreign Key
- Check
- Unique

Data Definition Language (DDL)

- Not Null:

By default all columns in a table can contain null values. If you want to ensure that a column must always have a value, i.e. it should not be left blank, then define a NOT NULL constraint on it

```
ALTER TABLE table_name MODIFY(column_names NOT NULL)
```

- Primary Key:

Each table can have one primary key, which uniquely identifies each row in a table and ensures that no duplicate rows exist

```
alter table emp add constraint emppk primary key (empno);
```

Data Definition Language (DDL)

- FOREIGN KEY:

A foreign key column can refer to primary key or unique key column of other tables. This Primary key and Foreign key relationship is also known as PARENT-CHILD relationship i.e. the table which has Primary Key is known as PARENT table and the table which has Foreign key is known as CHILD table

```
alter table attendance add constraint empno_fk
  foreign key (empno) references emp(empno);
```

Data Definition Language (DDL)

- FOREIGN KEY:
- You cannot delete a parent record if any existing child record is there. If you have to first delete the child record before deleting the parent record
- If you define the FOREIGN KEY with ON DELETE CASCADE option then you can delete the parent record and if any child record exist it will be automatically deleted
- To define a foreign key constraint with ON DELETE CASCADE option give the following command
- ALTER TABLE attendance ADD CONSTRAINT empno_fk
  FOREIGN KEY (empno) REFERENCES emp(empno)
  ON DELETE CASCADE;

Data Definition Language (DDL)

- FOREIGN KEY:
- You can also set the value for foreign key to null whenever the parent record is deleted
- To define a foreign key constraint with ON DELETE SET NULL option give the following command
- ALTER TABLE attendance ADD CONSTRAINT empno_fk
  FOREIGN KEY (empno) REFERENCES emp(empno)
  ON DELETE SET NULL;
- You also cannot drop the parent table without first dropping the FOREIGN KEY constraint from attendance table. However if you give CASCADE CONSTRAINTS option in DROP TABLE statement then Oracle will automatically drop the references and then drops the table

Data Definition Language (DDL)

- CHECK Constraints:
- Use the check constraint to validate values entered into a column

```
alter table table_name constraint constraint_name
        check (condition(s));
```

- UNIQUE KEY

```
alter table table_name add constraint constraint_name unique (column_name);
```

- DEFAULT:

```
ALTER TABLE table_name ALTER COLUMN column_name DEFAULT value;

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

Data Definition Language (DDL)

- Viewing Constraints:
- To see information about constraints, you can query the following data dictionary tables
- select * from user_constraints;
  select * from user_cons_columns;
- ENABLING AND DISABLING CONSTRIANTS
- ALTER TABLE <TABLE_NAME> ENABLE/DISABLE
  CONSTRAINT <CONSTRAINT_NAME>
- Dropping constraints
- alter table table_name drop constraint constraint_name;

Data Definition Language (DDL)

- **Rename**
- Use the RENAME statement to rename a table, view, sequence, or private synonym for a table, view, or sequence
- Oracle automatically transfers integrity constraints, indexes, and grants on the old object to the new object
- Oracle invalidates all objects that depend on the renamed object, such as views, synonyms, and stored procedures and functions that refer to a renamed table

Example

- To rename table emp2 to employee2 you can give the following command

rename emp2 to employee2

Data Definition Language (DDL)

- Drop
- Use the drop statement to drop tables, functions, procedures, packages, views, synonym, sequences, tablespaces etc
- Example
- The following command drops table emp2
- drop table emp2;
- If emp2 table is having primary key constraint, to which other tables refer to, then you have to first drop referential integrity constraint and then drop the table. Or if you want to drop table by dropping the referential constraints then give the following command
- drop table emp2 cascade constraints;

Data Definition Language (DDL)

- Truncate

- Use the Truncate statement to delete all the rows from table permanently . It is same as “DELETE FROM <table_name>” except
  - Truncate does not generate any rollback data hence, it cannot be roll backed
  - If any delete triggers are defined on the table. Then the triggers are not fired
  - It deallocates free extents from the table. So that the free space can be use by other tables

Example: **truncate table emp;**

- If you do not want free space and keep it with the table. Then specify the REUSE storage clause like this

truncate table emp reuse storage;

Data Manipulation Language (DML)

- Data manipulation language (DML) statements query and manipulate data in existing schema objects
- The following are the DML statements available in Oracle
- INSERT :Use to Add Rows to existing table
- UPDATE :Use to Edit Existing Rows in tables
- DELETE :Use to Delete Rows from tables
- MERGE :Use to Update or Insert Rows depending on condition

Data Manipulation Language (DML)

- Insert
- Use the Insert Statement to Add records to existing Tables

Examples.

To add a new row to an emp table

```
insert into emp values (101,'Sami','G.Manager',
        '8-aug-1998',2000);
```

- If you want to add a new row by supplying values for some columns not all the columns then you have to mention the name of the columns in insert statements

```
Insert into emp (empno,ename,sal) values (102,'Ashi',5000);
```

Data Manipulation Language (DML)

- Update
- Update statement is used to update rows in existing tables which is in your own schema or if you have update privilege on them
- For example to raise the salary by Rs.500 of employee number 104. You can give the following statement

```
update emp set sal=sal+500 where empno = 104;

update emp set name='Mohd Sami',
 sal=sal+(sal*10/100) where empno=102;
```

Now we want to raise the salary of all employees by 5%

```
update emp set sal=sal+(sal*5/100);
```

Data Manipulation Language (DML)

- Delete
- Use the DELETE statement to delete the rows from existing tables which are in your schema or if you have DELETE privilege on them

For example to delete the employee whose empno is 102

```
delete from emp where empno=102;
```

- Suppose we want to delete all employees whose salary is above 2000. Then give the following DELETE statement

```
delete from emp where salary > 2000;
```

- To delete all rows from emp table

```
delete from emp;
```

Data Manipulation Language (DML)

- Merge

- Use the MERGE statement to select rows from one table for update or insertion into another table. The decision whether to update or insert into the target table is based on a condition in the ON clause. It is a new feature of Oracle Ver. 9i. It is also known as UPSERT i.e. combination of UPDATE and INSERT.

- For example suppose we are having sales and sales_history table with the following structure

SALES

Prod	Month	Amount
SONY	JAN	2200
SONY	FEB	3000
SONY	MAR	2500
SONY	APR	3200
SONY	MAY	3100
SONY	JUN	5000

SALES HISTORY

Prod	Month	Amount
SONY	JAN	2000
SONY	MAR	2500
SONY	APR	3000
AKAI	JAN	3200

Data Manipulation Language (DML)

- Now we want to update sales_history table from sales table i.e. those rows which are already present in sales_history, their amount should be updated and those rows which are not present in sales_history table should be inserted

- merge into sales_history sh
  using sales s
  on (s.prod=sh.prod and s.month=sh.month)
  when matched then update set sh.amount=s.amount
  when not matched then insert values (prod,month,amount);

SALES_HISTORY

Prod	Month	Amount
SONY	JAN	2200
SONY	FEB	3000
SONY	MAR	2500
SONY	APR	3200
AKAI	JAN	3200
SONY	MAY	3100
SONY	JUN	5000

Data Control Language (DCL)

- Data Control Language Statements are used to grant privileges on tables, views, sequences, synonyms, procedures to other users or roles
- The DCL statements are
- GRANT :Use to grant privileges to other users or roles
  REVOKE :Use to take back privileges granted to other users and roles
- Privileges are of two types :
  - System Privileges
  - Object privileges
- System Privileges are normally granted by a DBA to users. Examples of system privileges are CREATE SESSION, CREATE TABLE, CREATE USER etc
- Object privileges means privileges on objects such as tables, views, synonyms, procedure. These are granted by owner of the object

Data Control Language (DCL)

- Grant
- Grant is use to grant privileges on tables, view, procedure to other users or roles
- Examples
- Suppose you own emp table. Now you want to grant select,update,insert privilege on this table to other user “SAMI”
- grant select, update, insert on emp to sami;
- Suppose you want to grant all privileges on emp table to sami. Then
- grant all on emp to sami;

Data Control Language (DCL)

- Suppose you want to grant select privilege on emp to all other users of the database. Then
- grant select on emp to public;
- Suppose you want to grant update and insert privilege on only certain columns not on all the columns then include the column names in grant statement. For example you want to grant update privilege on ename column only and insert privilege on empno and ename columns only. Then give the following statement
- grant update (ename),insert (empno, ename) on emp to sami;
- To grant select statement on emp table to sami and to make sami be able further pass on this privilege you have to give WITH GRANT OPTION clause in GRANT statement like this
- grant select on emp to sami with grant option;

Data Control Language (DCL)

- REVOKE
- Use to revoke privileges already granted to other users
- For example to revoke select, update, insert privilege you have granted to Sami then give the following statement
- revoke select, update, insert on emp from sami;
- To revoke select statement on emp granted to public give the following command
- revoke select on emp from public;
- To revoke update privilege on ename column and insert privilege on empno and ename columns give the following revoke statement
- revoke update, insert on emp from sami;

Transaction Control Language (TCL)

- Transaction control statements manage changes made by DML statements
- What is a Transaction?
- A transaction is a set of SQL statements which Oracle treats as a Single Unit. i.e. all the statements should execute successfully or none of the statements should execute
- To control transactions Oracle does not made permanent any DML statements unless you commit it. If you don't commit the transaction and power goes off or system crashes then the transaction is roll backed
- TCL Statements available in Oracle are
- COMMIT : Make changes done in transaction permanent
  ROLLBACK : Rollbacks the state of database to the last commit point
  SAVEPOINT : Use to specify a point in transaction to which later you can rollback

Transaction Control Language (TCL)

- To make the changes done in a transaction permanent issue the COMMIT statement
- The syntax of COMMIT Statement is
- COMMIT [WORK] [COMMENT ‘your comment’];
- WORK is optional
- COMMENT is also optional, specify this if you want to identify this transaction in data dictionary DBA_2PC_PENDING
- Example
- insert into emp (empno,ename,sal) values (101,’Abid’,2300);
- commit;

Transaction Control Language (TCL)

- ROLLBACK
- To rollback the changes done in a transaction give rollback statement. Rollback restore the state of the database to the last commit point
- Example :
- delete from emp;
- rollback; /* undo the changes */

Transaction Control Language (TCL)

- SAVEPOINT
- Specify a point in a transaction to which later you can roll back
- Example
- insert into emp (empno,ename,sal) values (109,'Sami',3000);
  savepoint a;
  insert into dept values (10,'Sales','Hyd');
  savepoint b;
  insert into salgrade values ('III',9000,12000);

Data Query Language (DQL)

- **DQL commands** are basically SELECT statements
- SELECT statements let you query the database to find information in one or more tables, and return the query as a result set
- A result set is an array structure; or more precisely, a result set is a two-dimensional array
- For example to retrieve all rows from emp table
- SQL> select empno, ename, sal from emp;
- Or (if you want to see all the columns values
- You can also give * which means all columns)
- SQL> select * from emp;

Data Query Language (DQL)

- If you want to see only employee names and their salaries then you can type the following statement
- **SQL> select name, sal from emp;**
- **Filtering Information using Where Conditions**

You can filter information using where conditions like suppose you want to see only those employees whose salary is above 5000 then you can type the following query with where condition

SQL>**select * from emp where sal > 5000;**

Data Query Language (DQL)

- Logical Conditions

- A logical condition combines the results of two component conditions to produce a single result based on them or to invert the result of a single condition. Table below lists logical conditions
  - **NOT:** Returns TRUE if the following condition is FALSE. Returns FALSE if it is TRUE. If it is UNKNOWN, it remains UNKNOWN

```
SELECT * FROM emp WHERE NOT (sal IS NULL);

SELECT * FROM emp WHERE NOT (salary BETWEEN  1000 AND 2000);
```

Data Query Language (DQL)

- Logical Conditions
  - **AND:** Returns TRUE if both component conditions are TRUE. Returns FALSE if either is FALSE. Otherwise returns UNKNOWN

```
SELECT * FROM employees WHERE ename =‘John' AND sal=3000;
```

  - **OR:** Returns TRUE if either component condition is TRUE. Returns FALSE if both are FALSE. Otherwise returns UNKNOWN

```
SELECT * FROM emp WHERE ename = ' John ' OR sal >= 1000;
```

  - **IN:** "Equal to any member of" test

```
SELECT * FROM emp   WHERE deptno IN (SELECT deptno FROM dept WHERE city=’HYD’)
```

  - **NOT IN:** Equivalent to "!=ALL". Evaluates to FALSE if any member of the set is NULL

```
SELECT * FROM emp  WHERE ename NOT IN  ('SCOTT', 'SMITH');
```

Data Query Language (DQL)

- NULL Conditions
- Tests for nulls. This is the only condition that you should use to test for nulls
- IS [NOT] NULL:

```
SELECT ename  FROM emp  WHERE deptno  IS NULL;


SELECT * FROM emp WHERE ename IS NOT NULL;
```

Data Query Language (DQL)

- LIKE Conditions
  - The LIKE conditions specify a test involving pattern matching
- to see all employees whose name starts with S char. Then you can use LIKE condition as follows
- SQL> select * from emp where ename like ‘S%’ ;
- Similarly you want to see all employees whose name ends with “d”
- SQL>select * from emp where ename like ‘%d’;
- You want to see all employees whose name starts with ‘A’ and ends with ‘d’ like ‘Abid’, ’Alfred’, ’Arnold’
- SQL>select * from emp where ename like ‘A%d’;

Data Query Language (DQL)

- LIKE Conditions
- You want to see those employees whose name contains character ‘a’ anywhere in the string
- SQL> select * from emp where ename like ‘%a%’;
- To see those employees whose name contains ‘a’ in second position
- SQL>select * from emp where ename like ‘_a%’;
- To see those employees whose name contains ‘a’ as last second character
- SQL>select * from emp where ename like ‘%a_’;

SELECT statement varieties with clauses

SELECT – with clauses

- GROUP BY clause
- HAVING clause
- ORDER BY clause

SELECT – with clauses

- GROUP BY clause : You can group query results on some column values. When you give a SELECT statement without group by clause then all the resultant rows are treated as a single group
- we want to see the sum salary of all employees dept wise. Then the following query will achieved the result
- Select deptno,sum(sal) from emp group by deptno;
- Similarly we want to see the average salary dept wise
- Select deptno,avg(sal) from emp group by deptno;
- Similarly we want to see the maximum salary in each department
- Select deptno,max(sal) from emp group by deptno;

SELECT – with clauses

- GROUP BY clause :
- Similarly the minimum salary
- Select deptno,min(sal) from emp group by deptno;
- Now we want to see the number of employees working in each department
- Select deptno,count(*) from emp group by deptno;

SELECT – with clauses

- HAVING clause :

- Now we want to see total salary department wise where the dept wise total salary is above 5000

- For this you have to use HAVING clause. Remember HAVING clause is used to filter groups and WHERE clause is used to filter rows. You cannot use WHERE clause to filter groups

- select deptno,sum(sal) from emp group by deptno
  having sum(sal) >= 5000;

SELECT – with clauses

- HAVING clause :

- We want to see those departments and the number of employees working in them where the number of employees is more than 2

- Select deptno, count(*) from emp group by deptno
  having count(*) >=2;

SELECT – with clauses

- ORDER BY clause :
- To sort query result you can use ORDER BY clause in SELECT statement.Sorting Examples
- The following query sorts the employees according to ascending order of salaries
- select * from emp order by sal;
- The following query sorts the employees according to descending order of salaries
- select * from emp order by sal desc;
- The following query sorts the employees according to ascending order of names
- select * from emp order by ename;

SELECT – with clauses

- ORDER BY clause :
- The following query first sorts the employees according to ascending order of names.If names are equal then sorts employees on descending order of salaries
- select * from emp order by ename, sal desc;
- You can also specify the positions instead of column names. Following query shows employees according to ascending order of their names
- select * from emp order by 2;
- If salaries are equal then sorts employees on ascending order of names
- select * from emp order by 3, 2;

Oracle built in functions

SQL Functions

- SQL Functions are similar to SQL operators Both manipulate data items and return a result
- There are two types of SQL Functions :
  - Single Row (or Scalar) Function
  - Aggregate Functions
- **Single Row Functions** returns a value based on a single row in a query,
  whereas an **aggregate function** returns a value based on all the rows in a query
- **Single Row Functions** can appear in select lists(except in SELECT statements containing GROUP BY) and WHERE clauses
  Aggregate Functions must be provided with an alias used in GROUP BY clause

SQL Functions

- Single Row (or Scalar) Function
  - Character Functions
  - Character Functions returning Number Values
  - Number Functions
  - Date Functions
  - Conversion Functions

SQL Functions

- Character Functions
  - LOWER
  - UPPER
  - INITCAP
  - LPAD
  - RPAD
  - LTRIM
  - RTRIM
  - CONCAT
  - SUBSTR

SQL Functions

- Number Functions
  - CEIL
  - FLOOR
  - ROUND
  - TRUNC
  - MOD

SQL Functions

- Character Functions returning Number Values
  - INSTR
  - LENGTH

SQL Functions

- Date Functions
  - ADD_MONTHS
  - LAST_DAY
  - MONTHS_BETWEEN
  - SYSDATE
  - NEXT_DAY
  - ROUND
  - TRUNC
  - CURRENT_DATE
  - TO_TIMESTAMP

SQL Functions

- Conversion Functions
  - TO_CHAR
  - TO_NUMBER
  - TO_DATE

SQL Functions

- Aggregate Row Functions
  - Grouping Functions
  - Other Functions

SQL Functions

- Grouping Functions
  - AVG
  - SUM
  - COUNT
  - MAX
  - MIN
  - STDDEV
  - VARIANCE

SQL Functions

- Other Functions
  - GREATEST
  - LEAST
  - NVL
  - NVL2
  - NULLIF
  - COALESCE
  - CASE
  - DECODE

Retrieving Data from multiple tables

Joins

- Equi and Non-Equi Joins
- Self Join
- Cartesian Product
- Outer join with modern syntax

Joins

- A join is a query that combines rows from two or more tables, views, or materialized views. Oracle performs a join whenever multiple tables appear in the query's FROM clause. The query's select list can select any columns from any of these tables

- If any two of these tables have a column name in common, you must qualify all references to these columns throughout the query with table names to avoid ambiguity

Join Conditions

Most join queries contain WHERE clause conditions that compare two columns, each from a different table. Such a condition is called a join condition. To execute a join, Oracle combines pairs of rows, each containing one row from each table, for which the join condition evaluates to TRUE. The columns in the join conditions need not also appear in the select list

Joins

- Natural Joins:

- A NATURAL JOIN is a JOIN operation that creates an implicit join clause for you based on the common columns in the two tables being joined. Common columns are columns that have the same name in both tables

- A NATURAL JOIN can be an INNER join, a LEFT OUTER join, or a RIGHT OUTER join. The default is INNER join

```
SELECT * FROM EMP NATURAL JOIN DEPT
```

Joins

- Equi and Non-Equi Joins

- An **equijoin** is a join with a join condition containing an equality operator (=). An equijoin combines rows that have equivalent values for the specified columns

```
select e.empno, e.ename, e.sal, e.deptno, d.dname, d.city from
emp e, dept d where emp.deptno=dept.deptno;

select empno,ename,sal,dname,city from emp,dept where
emp.deptno=dept.deptno;

select * from emp,dept where emp.deptno=dept.deptno;
```

Joins

- Joins with using:
- Use the USING clause to specify the columns for the equijoin where several columns have the same names but not same data types
- Use the USING clause to match only one column when more than one column matches
- The NATURAL JOIN and USING clauses are mutually exclusive
- SELECT * FROM EMP JOIN DEPT using(dept)

Joins

- Equi and Non-Equi Joins:
- **Non equi** joins is used to return result from two or more tables where exact join is not possible
- select e.empno, e.ename, e.sal, s.grade from emp e, salgrade s
  where e.sal between s.lowsal and s.hisal

- Self Joins:

- A self join is a join of a table to itself. This table appears twice in the FROM clause and is followed by table aliases that qualify column names in the join condition. To perform a self join, Oracle combines and returns rows of the table that satisfy the join condition

- To return employee names and their manager names for whom they are working

```
Select e.empno, e.ename, m.ename  “Manager” from emp e,
  emp m where e.mgrid=m.empno
```

Joins

- Inner Joins:

- A self join is a join of a table to itself. This table appears twice in the FROM clause and is followed by table aliases that qualify column names in the join condition. To perform a self join, Oracle combines and returns rows of the table that satisfy the join condition

- To return employee names and their manager names for whom they are working

```
Select e.empno, e.ename, m.ename  “Manager” from emp e,
  emp m where e.mgrid=m.empno
```

Joins

- Cross Join:

- A CROSS JOIN is a JOIN operation that produces the Cartesian product of two tables. Unlike other JOIN operators, it does not let you specify a join clause. You may, however, specify a WHERE clause in the SELECT statement

```
SELECT * FROM CITIES CROSS JOIN FLIGHTS

SELECT * FROM CITIES, FLIGHTS
```

- The following SELECT statements are equivalent:

```
SELECT * FROM CITIES CROSS JOIN FLIGHTS

  WHERE CITIES.AIRPORT = FLIGHTS.ORIG_AIRPORT
```

Joins

- Outer Join:

An outer join returns all rows that satisfy the join condition and also returns some or all of those rows from one table for which no rows from the other satisfy the join condition

It can be used in following three ways:

- LEFT OUTER join
- RIGHT OUTER join
- FULL OUTER join

Joins

- LEFT OUTER join:

- A LEFT OUTER JOIN is one of the JOIN operations that allow you to specify a join clause. It preserves the unmatched rows from the first (left) table, joining them with a NULL row in the shape of the second (right) table

```
SELECT e.empno,e.ename,d.dname,d.deptno

FROM emp e LEFT OUTER JOIN dept d

ON (e.deptno = d.deptno)
```

Joins

- LEFT OUTER join:

```
SELECT e.empno,e.ename,d.dname,d.deptno

FROM dept d LEFT OUTER JOIN emp e

ON (e.deptno = d.deptno)
```

- -- Join with WHERE Clause --

```
SELECT e.empno,e.ename,d.dname,d.deptno

FROM dept d LEFT OUTER JOIN emp e

ON (e.deptno = d.deptno)  WHERE rownum <=3
```

- RIGHT OUTER join:

A RIGHT OUTER JOIN is one of the JOIN operations that allow you to specify a JOIN clause. It preserves the unmatched rows from the second (right) table, joining them with a NULL in the shape of the first (left) table

- SELECT e.empno,e.ename,d.dname,d.deptno

FROM emp e RIGHT OUTER JOIN dept d

ON (e.deptno = d.deptno)

- SELECT e.empno,e.ename,d.dname,d.deptno

FROM dept d RIGHT OUTER JOIN emp e ON (e.deptno = d.deptno)

- FULL OUTER join:

A FULL OUTER JOIN is one of the JOIN operations that allow you to specify a JOIN clause. It performs an outer join and returns all rows from both tables and extended with nulls if they do not satisfy the join condition.

```
SELECT *

FROM emp e FULL OUTER JOIN dept d

ON (e.deptno = d.deptno)
```

SUBQUERIES

SUBQUERIES

- **Nested Subqueries:**

- A query nested within a query is known as nested subquery

You want to see all the employees whose salary is above average salary

```
Select * from emp where sal > (select avg(sal) from emp);
```

- we want to see the name and empno of that employee whose salary is maximum

```
Select * from emp where sal = (select max(sal) from emp);
```

SUBQUERIES

- **Nested Subqueries:**

- A query nested within a query is known as nested subquery

You want to see all the employees whose salary is above average salary

```
Select * from emp where sal > (select avg(sal) from emp);
```

- we want to see the name and empno of that employee whose salary is maximum

```
Select * from emp where sal = (select max(sal) from emp);
```

SUBQUERIES

- Nested Subqueries:

- Similarly to see the Third highest salary

- Select max(sal) from emp where
  sal < (select max(sal) from emp where
  sal < (select max(sal) from emp));

- We want to see how many employees are there whose salary is above average

```
Select count(*) from emp where
  sal > (select max(sal) from emp);
```

SUBQUERIES

- Nested Subqueries:

- We want to see those employees who are working in Hyderabad

```
Select * from emp where deptno
    in (select deptno from dept where city='HYD');
```

- You can also use subquery in FROM clause of SELECT statement

- For example the following query returns the top 5 salaries from employees table

- Select sal from (select sal from emp order sal desc)
  where rownum <= 5;

SUBQUERIES

- **Co-related Subqueries:**

- A correlated subquery is a subquery that uses values from the outer query, requiring the inner query to execute once for each outer query

- With a normal nested subquery the inner SELECT query runs first and executes once returning

- values to be used by the main query

- But a correlated subquery executes once for each row considered by the outer query

- The inner query is driven by the outer query

SUBQUERIES

- Co-related Subqueries:

- **query to return the cheapest products from the products**

- SELECT product_id, product_name, list_price FROM products

WHERE list_price =(

SELECT MIN(list_price) FROM products);

SUBQUERIES

- Co-related Subqueries:

- **query to find all products whose list price is above average for their category.**

```
SELECT product_id, product_name, list_price FROM products p

WHERE list_price > (

      SELECT AVG( list_price ) FROM products  WHERE

         category_id = p.category_id );
```

SUBQUERIES

- Co-related Subqueries:

- **query to return all customers who have no orders:**

```
SELECT customer_id, name FROM customers

WHERE NOT EXISTS (

  SELECT * FROM orders WHERE

        orders.customer_id = customers.customer_id) ORDER BY  name;
```

SUBQUERIES

- **Co-related Subqueries:**

- Correlated subqueries and slow because the sub-query is executed ONCE for each row returned by the outer query

- Correlated subqueries are usually used for EXISTS Booleans, and scalar subqueries

- **Nested Subquery versus Correlated Subquery :**

- With a normal nested subquery the inner SELECT query runs first and executes once returning values to be used by the main query

- But a correlated subquery executes once for each row considered by the outer query. So, the inner query is driven by the outer query

VIEWS

VIEWS

- A view is a logical table based on a Table or another view
- A view contains no data of its own but is like a window through which data from tables can be viewed or changed
- The Tables on which a view is based are called Base Tables
- The view is stored as a select statement in the Database

VIEWS

- Advantages of a View:

- To restrict Data Access - Views restrict access to data because the view can display selected columns from the table

- To make complex queries easy - Views can be used to make simple queries to retrieve the results of complicated queries

For ex. Views can be used to query information from multiple tables without the user knowing how to write a join statement

- To provide Data independence - One view can be used to retrive data from several tables

- To present different views of the same data

VIEWS

- Types of a View:
- Simple View -

- Derives data from only one Table

- Contains no functions or groups of data

- Can perform DML operations through the view

- Complex View -

- Derives data from many tables

- Contains functions or groups of data

- Does not always allow DML operations through the view

VIEWS

- Creating a View:

CREATE [OR REPLACE] [FORCE|NOFORCE] VIEW view_name

[(alias[, alias]...)]

AS <sql_query>

[WITH CHECK OPTION [CONSTRAINT constraint]]

[WITH READ ONLY [CONSTRAINT constraint]];

VIEWS

- Creating a View:

- OR REPLACE - Re-creates the view if it already exists after editing the SQL query
- FORCE - Creates the view regardless of whether or not the base tables exists
- NOFORCE - Creates the view only if the Base Tables exists (This is the default)
- alias - Specifies names for the expressions selected by the view's query
- WITH CHECK OPTION - Specifies that only those rows that are accessible to the view can be inserted or updated
- constraint - It is the name assigned to the CHECK OPTION constraint
- WITH READ ONLY - ensures that no DML operations can be performed on this view

VIEWS

- Creating a Simple View:

- Create view EMPVU20 which contain detail of Employee in Department 20 --

```
CREATE OR REPLACE VIEW empvu20

AS

SELECT empno,ename,sal,deptno

FROM emp

WHERE deptno = 20;

SELECT * from empvu20;
```

VIEWS

- Creating a Complex View:

```
SELECT d.dname,MIN(e.sal),MAX(e.sal),AVG(e.sal)

FROM emp e JOIN dept d

ON (e.deptno = d.deptno)

GROUP BY d.dname
```

PL/SQL

PL/SQL

- PL/SQL stands for a procedural language extension of the structured query language (SQL)
- PL/SQL (Procedural language/structured query language) is a program to executes in Oracle database
- Furthermore, PL/SQL specially designed for database oriented activities
- Also, Oracle PL/SQL allows you to perform data manipulation operation that is safe and flexible

Advantages of PL/SQL

- Reduces network traffic This one is **great advantages** of PL/SQL. Because PL/SQL nature is **entire block** of SQL statements execute into **oracle engine** all at once so it's main benefit is **reducing** the **network traffic**
- Procedural language support PL/SQL is a **development tools** not only for data manipulation futures but also provide the conditional checking, looping or branching operations same as like **other programming language**
- Error handling PL/SQL is dealing with **error handling**, It's permits the smart way **handling the errors** and giving **user friendly** error messages, when the errors are encountered
- Declare variable PL/SQL gives you control to **declare variables** and access them **within the block**. The declared variables can be used at the time of **query processing**

Advantages of PL/SQL

- Intermediate Calculation Calculations in PL/SQL done quickly and efficiently without using Oracle engines. This **improves** the transaction performance
- Portable application Applications are written in PL/SQL are **portable** in any **Operating system**. PL/SQL applications are **independence program** to run any computer

Anonymous Block

- PL/SQL program units organize the code into blocks
- A block without a name is known as an **anonymous block**
- The anonymous block is the simplest unit in PL/SQL
- It is called anonymous block because it is not saved in the Oracle database
- An anonymous block is an only one-time use and useful in certain situations such as creating test units
- **To display database's output on the screen, you need to:**

---First, use the SET SERVEROUTPUT ON command to instruct SQL*Plus to echo database's output after executing the PL/SQL block

---Second, use the DBMS_OUTPUT.PUT_LINE procedure to output a string on the screen

PL/SQL Block Structure

- [DECLARE]

  Declaration statements;

BEGIN

  Execution statements;

 [EXCEPTION]

    Exception handling statements;

END;

/

PL/SQL Block Structure

- [DECLARE]

The declaration section allows you to define data types, structures, and variables. You often declare variables in the declaration section by giving them names, data types, and initial values

BEGIN

The execution section is required in a block structure and it must have at least one statement. It is the place where you put the execution code or business logic code. You can use both procedural and SQL statements inside the execution section

[EXCEPTION]

The exception section is the place that you put the code to handle exceptions. You can either catch or handle exceptions in the exception section

END;

/

Anonymous Block

- SET SERVEROUTPUT ON SIZE 1000000
- BEGIN
- DBMS_OUTPUT.PUT_LINE('Hello PL/SQL');
- END;
- /

PL/SQL Data Types

- Scalar
- Anchor

Scalar Types

- Scalar Type: Scalar type is a data type that holds a single value. Scalar type includes the following categories:
- Character / String
- Number
- Boolean
- Date / Time

Scalar Types

- Character / String
  - Allows PL/SQL character or string types
  - Up to 32 K in size
- Number
  - Allows integer data types
- Boolean
  - Allows TRUE, FALSE or NULL values
- Date / Time
  - Include DATE and TIMESTAMP datatypes
  - DATE type is used to store date
  - TIMESTAMP is an extension of DATE type. Includes year, month, day, hour, minute, seconds and fraction of second.

Scalar Types

- Timestamp

  Provides date and time with fraction of seconds up to 9 places

```
declare
 v_date timestamp(9) := systimestamp;
 begin
   dbms_output.put_line(v_date);
end;
```

Scalar Types

- DECLARE

```
  v_first_name varchar2(20);
  v_last_name varchar2(20);
  n_employee_id number;
  d_hire_date date;
BEGIN
  NULL;
END;
```

Anchor Types

- PL/SQL provides you with a very useful feature called variable anchors
- It refers to the use of the %TYPE keyword to declare a variable with the data type is associated with a column's data type of a particular column in a table

```
DECLARE
 v_first_name  EMPLOYEES.FIRST_NAME%TYPE;
  v_last_name   EMPLOYEES.LAST_NAME%TYPE;
  n_employee_id EMPLOYEES.EMPLOYEE_ID%TYPE;
  d_hire_date   EMPLOYEES.HIRE_DATE%TYPE;
BEGIN
  NULL;
END;
/
```

Anchor Types

```
DECLARE
   v_first_name EMPLOYEES.FIRST_NAME%TYPE;
   v_last_name EMPLOYEES.LAST_NAME%TYPE;
   n_employee_id EMPLOYEES.EMPLOYEE_ID%TYPE;
   d_hire_date EMPLOYEES.HIRE_DATE%TYPE;
BEGIN
   v_first_name := 'Mary';
   v_last_name := 'Jane';
   d_hire_date := to_date('19700101','YYYYMMDD');
END;
/
/
```

SELECT INTO statement

- PL/SQL SELECT INTO statement is the simplest and fastest way to fetch a single row from a table into variables

- Syntax:

```
SELECT   select_list INTO   variable_list
FROM   table_name
WHERE   condition;
```

SELECT INTO statement

```
DECLARE
  l_customer_name customers.name%TYPE;
BEGIN
  -- get name of the customer 100 and assign it to l_customer_name
  SELECT name INTO l_customer_name
  FROM customers
  WHERE customer_id = 100;
  -- show the customer name
  dbms_output.put_line( v_customer_name );
END;
```

Constructs in PL/SQL

- IF THEN
- CASE
- GOTO
- LOOP
- FOR LOOP
- WHILE Loop
- CONTINUE

Constructs in PL/SQL

- IF THEN Statement

The IF statement allows you to either execute or skip a sequence of statements, depending on a condition. The IF statement has the three forms:

– IF THEN

– IF THEN ELSE

– IF THEN ELSIF

Constructs in PL/SQL

- CASE Statement
- The CASE statement chooses one sequence of statements to execute out of many possible sequences
- The CASE statement has two types: **simple CASE** statement and **searched CASE** statement
- Both types of the CASE statements support an optional ELSE clause

Constructs in PL/SQL

- **Simple CASE Statement**

A simple CASE statement evaluates a single expression and compares the result with some values

```
CASE selector
WHEN selector_value_1 THEN
   statements_1
WHEN selector_value_1 THEN
   statement_2
   ...
ELSE
  else_statements
END CASE;
```

Constructs in PL/SQL

- **Searched CASE Statement**

- The searched CASE statement evaluates multiple Boolean expressions and executes the sequence of statements associated with the first condition that evaluates to TRUE
- CASE
- WHEN condition_1 THEN statements_1
- WHEN condition_2 THEN statements_2
- ...
- WHEN condition_n THEN statements_n
- [ELSE
- else_statements]
- END CASE;]

- Searched CASE Statement

- The searched CASE statement follows the rules below:
  - The conditions in the WHEN clauses in are evaluated in order, from top to bottom
  - The sequence of statements associated with the WHEN clause whose condition evaluates to TRUE is executed. If more than one condition evaluates to TRUE, only the first one executes
  - If no condition evaluates to TRUE, the else_statements in the ELSE clause executes. If you skip the ELSE clause and no expressions are TRUE, a CASE_NOT_FOUND exception is raised

Constructs in PL/SQL

- GOTO Statement
- The GOTO statement allows you to transfer control to a labeled block or statement. The following illustrates the syntax of the GOTO statement:

```
GOTO label_name;
```

Constructs in PL/SQL

- NULL Statement
- The NULL statement does nothing except that it passes control to the next statement
- The NULL statement is useful in some cases:
  - Improve code readability
  - Provide a target for a GOTO statement
  - Create placeholders for subprograms

Constructs in PL/SQL

- LOOP Statement
- This basic LOOP statement consists of a LOOP keyword, a body of executable code, and the END LOOP keywords
- The LOOP statement executes the statements in its body and returns control to the top of the loop
- Typically, the body of the loop contains at least one EXIT or EXIT WHEN statement for terminating the loop. Otherwise, the loop becomes an infinite loop

Constructs in PL/SQL

- FOR LOOP Statement
- PL/SQL FOR LOOP executes a sequence of statements a specified number of times. The PL/SQL FOR LOOP statement has the following structure:

```
FOR index IN lower_bound .. upper_bound

LOOP

   statements;

END LOOP;
```

Constructs in PL/SQL

- WHILE LOOP Statement
- PL/SQL WHILE loop statement executes a sequence of statements as long as a specified condition is TRUE
- WHILE condition
- LOOP
- statements;
- END LOOP;

Constructs in PL/SQL

- **CONTINUE Statement**

- The CONTINUE statement allows you to exit the current loop iteration and immediately continue on to the next iteration of that loop

```
BEGIN
  FOR n_index IN 1 .. 10
  LOOP
    -- skip odd numbers
    IF MOD( n_index, 2 ) = 1 THEN
      CONTINUE;
    END IF;
    DBMS_OUTPUT.PUT_LINE( n_index );
  END LOOP;
END;
```

Cursors in PL/SQL

- A cursor is a pointer that points to a result of a query
- PL/SQL has two types of cursors: implicit cursors and explicit cursors

Implicit cursors

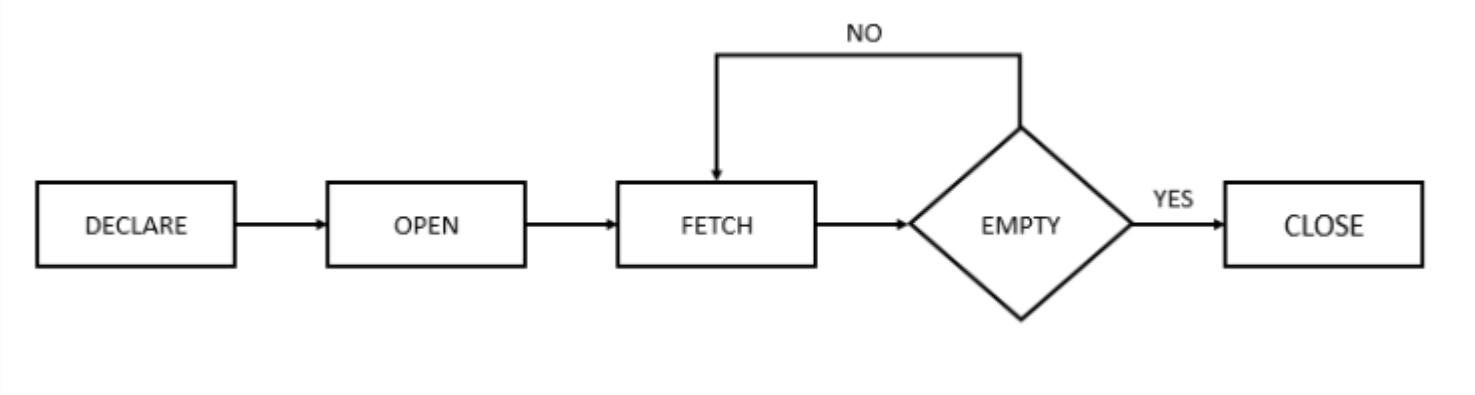
- Whenever Oracle executes an SQL statement such as SELECT INTO, INSERT, UPDATE, and DELETE, it automatically creates an implicit cursor
- Oracle internally manages the whole execution cycle of implicit cursors and reveals only the cursor's information and statuses such as SQL%ROWCOUNT, SQL%ISOPEN, SQL%FOUND, and SQL%NOTFOUND
- The implicit cursor is not elegant when the query returns zero or multiple rows which cause NO_DATA_FOUND or TOO_MANY_ROWS exception respectively

Cursors in PL/SQL

- **Explicit cursors**

- An explicit cursor is an SELECT statement declared explicitly in the declaration section of the current block or a package specification. For an explicit cursor, you have the control over its execution cycle from OPEN, FETCH, and CLOSE

- Oracle defines an execution cycle to execute an SQL statement and associates a cursor with it. The following illustration shows the execution cycle of an explicit cursor:



Cursors in PL/SQL

- **Steps in Explicit cursors**
- Declare a cursor
  - CURSOR cursor_name IS query;
- Open a cursor
  - OPEN cursor_name;
- Fetch from a cursor
  - FETCH cursor_name INTO variable_list;
- Closing a cursor
  - CLOSE cursor_name;

Cursors in PL/SQL

- Explicit Cursor Attributes
- A cursor has four attributes to which you can reference in the following format:
- %ISOPEN
  - This attribute is TRUE if the cursor is open or FALSE if it is not
- %FOUND
  - This attribute has four values:
    - NULL before the first fetch
    - FALSE if a record was fetched successfully
    - TRUE if no row returned
    - INVALID_CURSOR if the cursor is not opened

Cursors in PL/SQL

- Cursor FOR LOOP:
  - The cursor FOR LOOP statement an elegant extension of the numeric FOR LOOP statement
  - It allows you to fetch every row from a cursor without manually managing the execution cycle of the cursor, i.e., OPEN, FETCH, and CLOSE

```
FOR record IN cursor_name

LOOP

    process_record_statements;

END LOOP;
```

Cursors in PL/SQL

- Cursor with Parameters:
  - An explicit cursor may accept a list of parameters. Each time you open the cursor, you can pass different arguments to the cursor, which results in different result sets

```
CURSOR cursor_name (parameter_list)

IS

cursor_query;
```

Cursors in PL/SQL

- Updatable Cursor:
  - Oracle provides the FOR UPDATE clause of the SELECT statement in an updatable cursor to perform this kind of locking mechanism

```
CURSOR cursor_name IS

SELECT select_clause

FROM from_clause

WHERE where_clause

FOR UPDATE;
```

Cursors in PL/SQL

- REF Cursor:
  - A cursor variable is a variable that references to a cursor
  - A cursor variable is not tied to any specific query. Meaning that a cursor variable can be opened for any query
  - It enables passing the result of a query between PL/SQL programs
  - Without a cursor variable, you have to fetch all data from a cursor, store it in a variable e.g., a collection, and pass this variable as an argument
  - With a cursor variable, you simply pass the reference to that cursor

```
DECLARE

  TYPE customer_t IS REF CURSOR RETURN customers%ROWTYPE;

  c_customer customer_t;
```

Modularized programming in PL/SQL

Modularized programming in PL/SQL

- Anonymous v/s Named PL/SQL blocks
- Stored Procedures
- Functions
- Packages

Anonymous v/s Named PL/SQL blocks

- Anonymous v/s Named PL/SQL blocks

Anonymous blocks: In PL/SQL, That's blocks which is not have header are known as anonymous blocks. These blocks do not form the body of a function or triggers or procedure

Named blocks: That's PL/SQL blocks which having header or labels are known as Named blocks. These blocks can either be subprograms like functions, procedures, packages or Triggers

Modularized programming in PL/SQL

- Stored Procedures
- A PL/SQL procedure is a reusable unit that encapsulates specific business logic of the application
- A PL/SQL procedure is a named block stored as a schema object in the Oracle Database

```
CREATE [OR REPLACE ] PROCEDURE procedure_name (parameter_list)
IS
   [declaration statements]
BEGIN
   [execution statements]
   EXCEPTION
      [exception handler]
END [procedure_name ];
```

Modularized programming in PL/SQL

- Functions

- Similar to a procedure, a PL/SQL function is a reusable program unit stored as a schema object in the Oracle Database

```
CREATE [OR REPLACE] FUNCTION function_name (parameter_list)
   RETURN return_type
IS
   [declarative section]
BEGIN
   [executable section]
[EXCEPTION]
   [exception-handling section]
END;
```

Modularized programming in PL/SQL

- **Package**
- a package is a schema object that contains definitions for a group of related functionalities
- A package includes variables, constants, cursors, exceptions, procedures,

  functions, and subprograms
- It is compiled and stored in the Oracle Database
- A PL/SQL package has two parts:
  - package specification and,
  - package body

Modularized programming in PL/SQL

- **Package Exception**

- PL/SQL treats all errors that occur in an anonymous block, procedure, or function as exceptions

- The exceptions can have different causes such as coding mistakes, bugs, even hardware failures

```
BEGIN
   -- executable section
   -- exception-handling section
   EXCEPTION
      WHEN e1 THEN
         -- exception_handler1
      WHEN e2 THEN
         -- exception_handler1
      WHEN OTHERS THEN
         -- other_exception_handler
END;
```

Modularized programming in PL/SQL

- **Trigger**
  - A trigger is a named PL/SQL block stored in the Oracle Database and executed automatically when a triggering event takes place
  - The event can be any of the following:
    - A data manipulation language (DML) statement executed against a table e.g., INSERT, UPDATE, or DELETE. For example, if you define a trigger that fires before an INSERT statement on the customers table, the trigger will fire once before a new row is inserted into the customers table
    - A data definition language (DDL) statement executes e.g., CREATE or ALTER statement. These triggers are often used for auditing purposes to record changes of the schema
    - A system event such as startup or shutdown of the Oracle Database
    - A user event such as login or logout

Modularized programming in PL/SQL

- **Trigger**
- CREATE [OR REPLACE] TRIGGER trigger_name
- {BEFORE | AFTER } triggering_event ON table_name
- [FOR EACH ROW]
- [FOLLOWS | PRECEDES another_trigger]
- [ENABLE / DISABLE]
- [WHEN condition]
- DECLARE
- declaration statements
- BEGIN
- executable statements
- EXCEPTION
- exception_handling statements
- END;


LTI
Let's Solve