Unit 3: Managing Database Objects

- Working with Tables and Constraints; Working with Indexes, Views, Synonyms, and Sequences;
- Partitioning and Materialized Views,
- Introduction of PLSQL, Stored Procedure, Functions,
- Trigger, package

Table

- In Relational database model, a table is a collection of data elements organized in terms of rows and columns

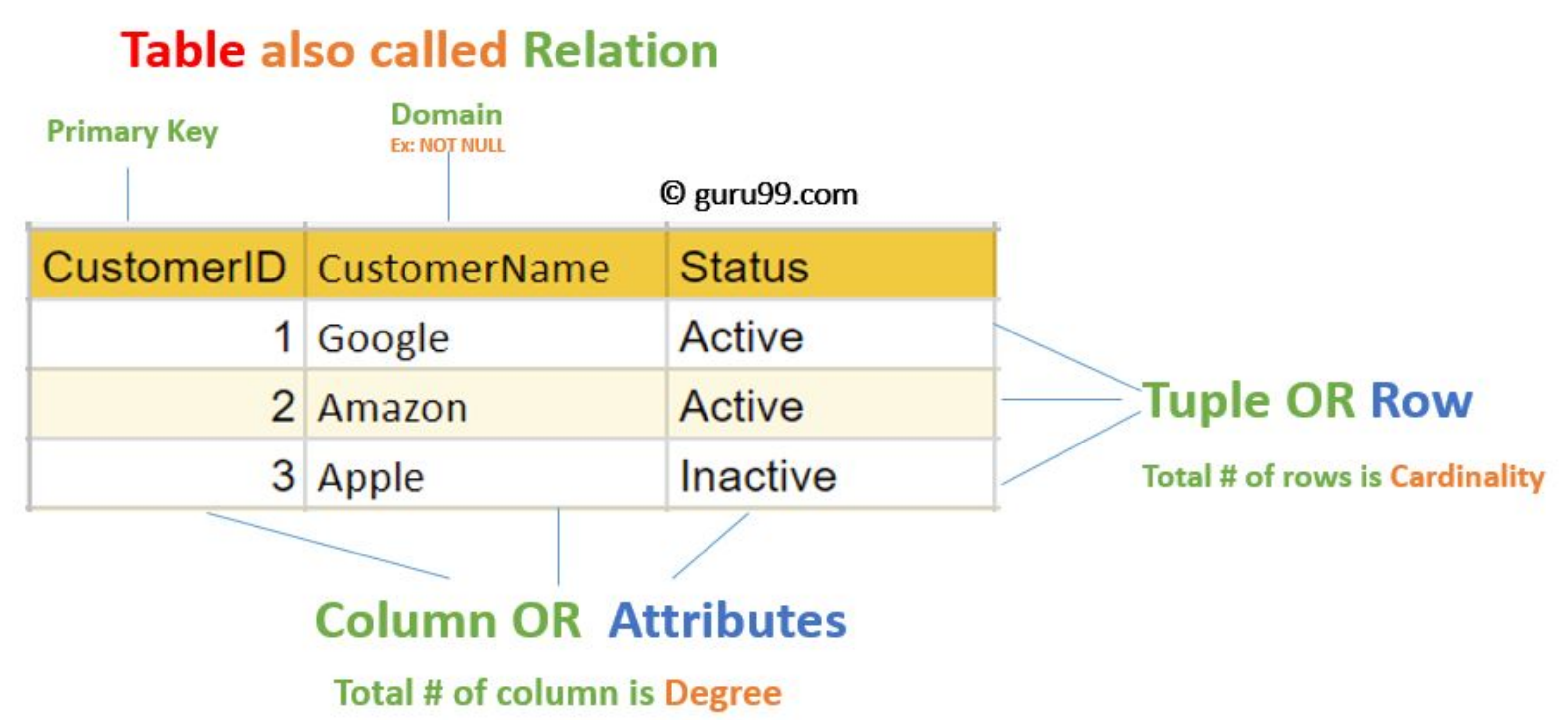
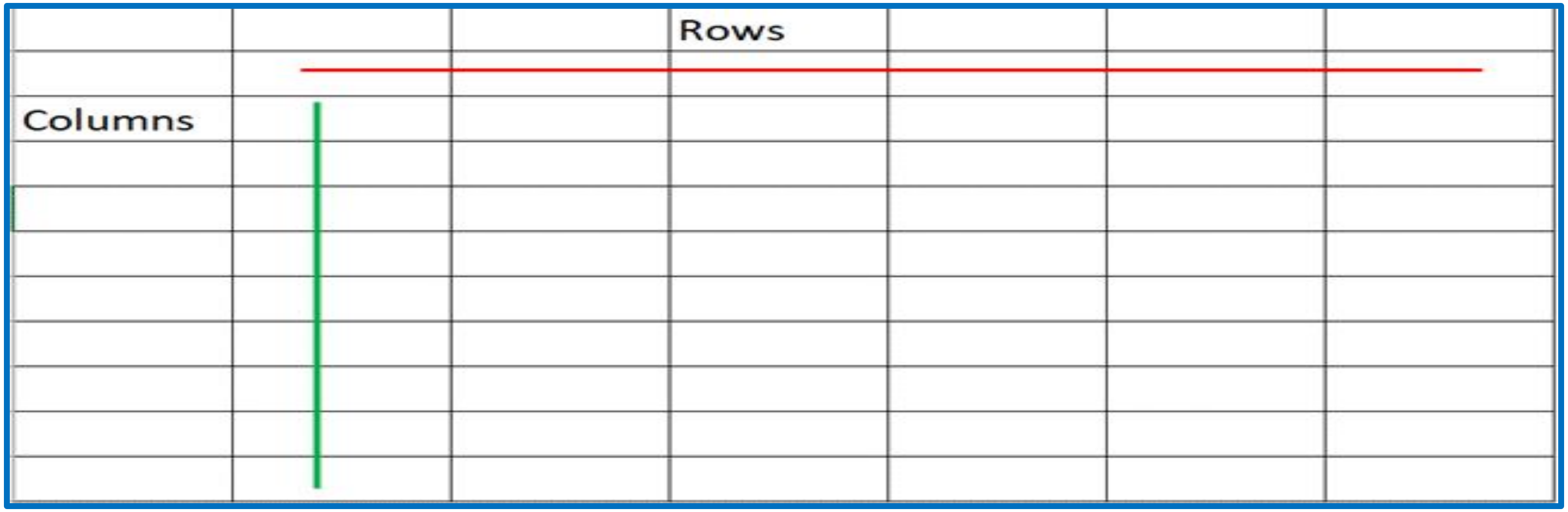


Table also called Relation
Primary Key
Domain
Ex: NOT NULL
© guru99.com
Tuple OR Row
Total # of rows is Cardinality
Column OR Attributes
Total # of column is Degree

CustomerID	CustomerName	Status
1	Google	Active
2	Amazon	Active
3	Apple	Inactive

- **Row:** A row is a series of data placed out horizontally in a table. It is a horizontal arrangement of the objects, words, numbers, and data.
- **Column**: A column is a vertical series of cells in a table. It is an arrangement of figures, facts, words, etc.



- **Properties of tables:**
  - Values are atomic.
  - Column values are of the same kind.
  - Each row is unique.
  - The sequence of columns is insignificant.
  - The sequence of rows is insignificant.
  - Each column must have a unique name.

Constraints

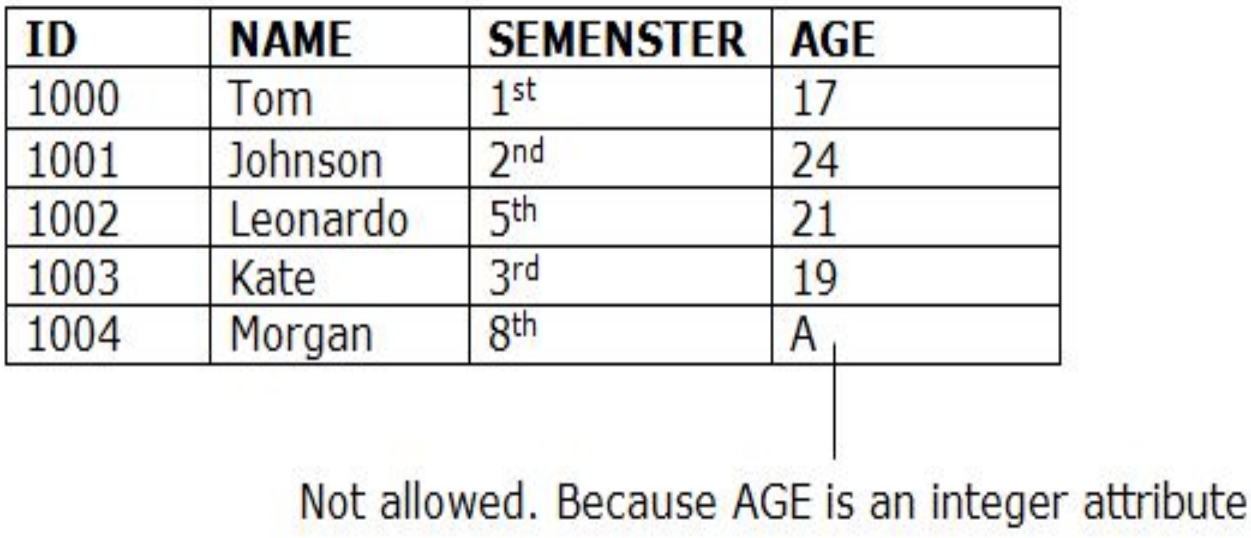
- SQL constraints are used to specify rules for the data in a table.
- Constraints are used to limit the type of data that can go into a table. This ensures the accuracy and reliability of the data in the table. If there is any violation between the constraint and the data action, the action is aborted.
- Constraints can be column level or table level. Column level constraints apply to a column, and table level constraints apply to the whole table.

Types of Constraints

- Domain Constraints
  - Not null constraints
  - Default constraints
  - Check constraints
- Primary key constraints
- Referential constraints
- Unique key constraints

Domain Constraints

- 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.

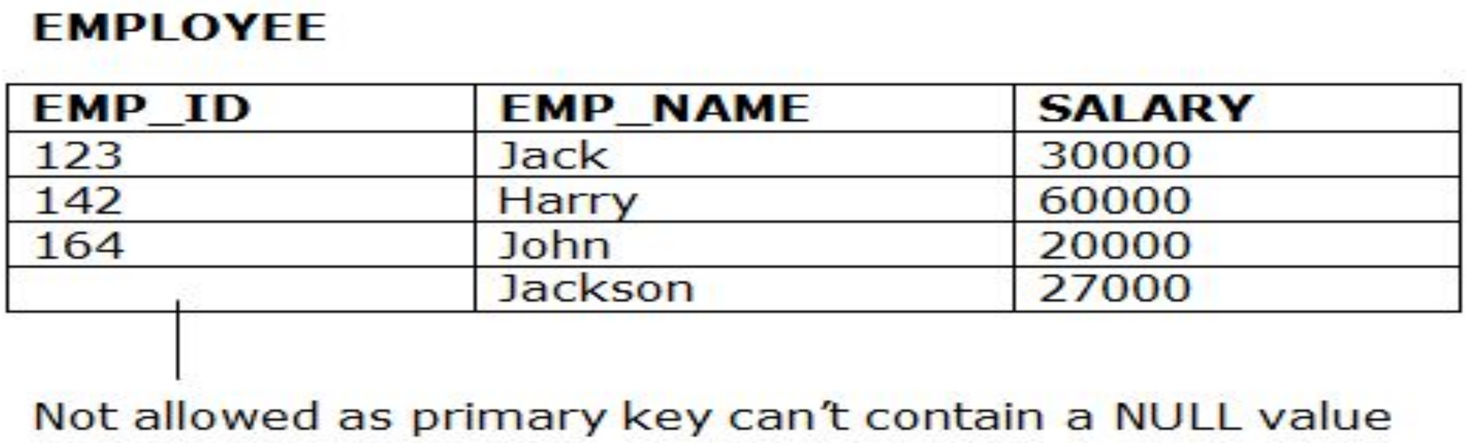


ID	NAME	SEMENSTER	AGE
1000	Tom	1st	17
1001	Johnson	2nd	24
1002	Leonardo	5th	21
1003	Kate	3rd	19
1004	Morgan	8th	A

Not allowed. Because AGE is an integer attribute

Primary key constraints

- 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.



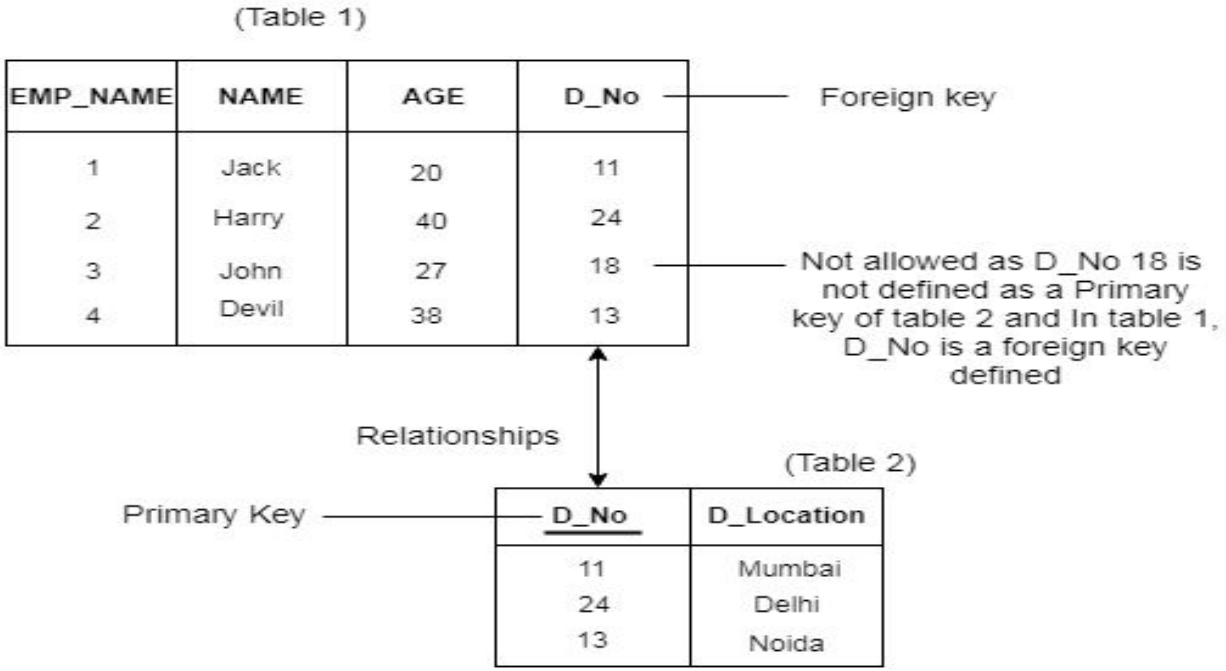
EMPLOYEE

EMP_ID	EMP_NAME	SALARY
123	Jack	30000
142	Harry	60000
164	John	20000
	Jackson	27000

Not allowed as primary key can't contain a NULL value

Referential constraints

- 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.



(Table 1)

EMP_NAME	NAME	AGE	D_No
1	Jack	20	11
2	Harry	40	24
3	John	27	18
4	Devil	38	13

Foreign key

Not allowed as D_No 18 is not defined as a Primary key of table 2 and In table 1, D_No is a foreign key defined

Relationships

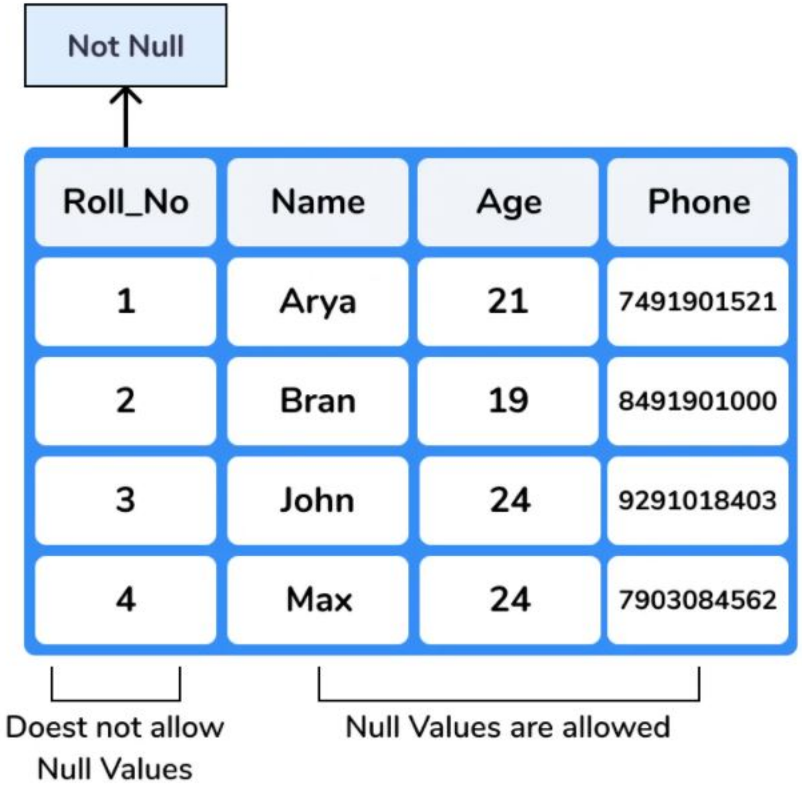
(Table 2)

Primary Key

D_No	D_Location
11	Mumbai
24	Delhi
13	Noida

Not null constraints

- By default, a column can hold NULL values.
- The NOT NULL constraint enforces a column to NOT accept NULL values.

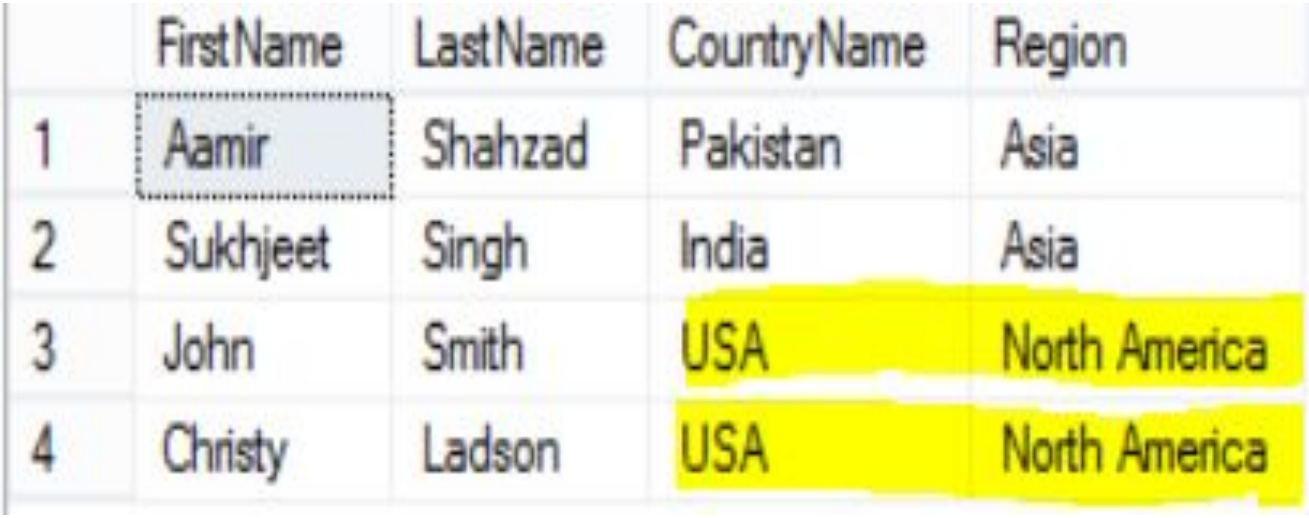


Roll_No	Name	Age	Phone
1	Arya	21	7491901521
2	Bran	19	8491901000
3	John	24	9291018403
4	Max	24	7903084562

Not Null
Doest not allow Null Values
Null Values are allowed

Default constraints

- Provides a default value for a column when none is specified.



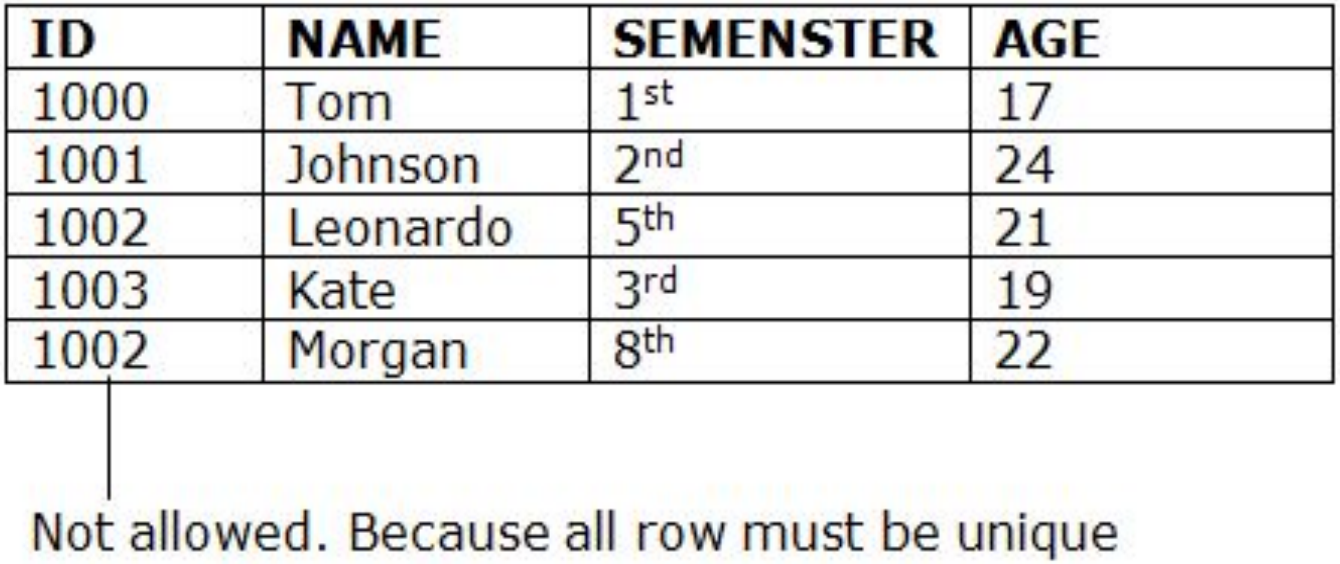
	FirstName	LastName	CountryName	Region
1	Aamir	Shahzad	Pakistan	Asia
2	Sukhjeet	Singh	India	Asia
3	John	Smith	USA	North America
4	Christy	Ladson	USA	North America

Check constraints

- The CHECK constraint ensures that all the values in a column satisfies certain conditions

Unique key constraints

- 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.



ID	NAME	SEMENSTER	AGE
1000	Tom	1st	17
1001	Johnson	2nd	24
1002	Leonardo	5th	21
1003	Kate	3rd	19
1002	Morgan	8th	22

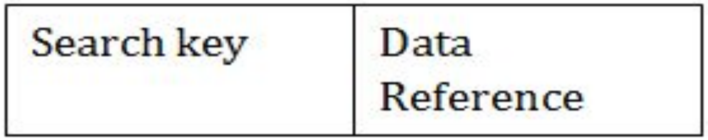
Not allowed. Because all row must be unique

Indexing

- **Indexing** is a data structure technique which allows you to quickly retrieve records from a database file.
- An Index is a small table having only two columns. The first column comprises a copy of the primary or candidate key of a table. Its second column contains a set of pointers for holding the address of the disk block where that specific key value stored.
- An index -
  - Takes a search key as input
  - Efficiently returns a collection of matching records.

Indexing Structure

- Indexes can be created using some database columns.

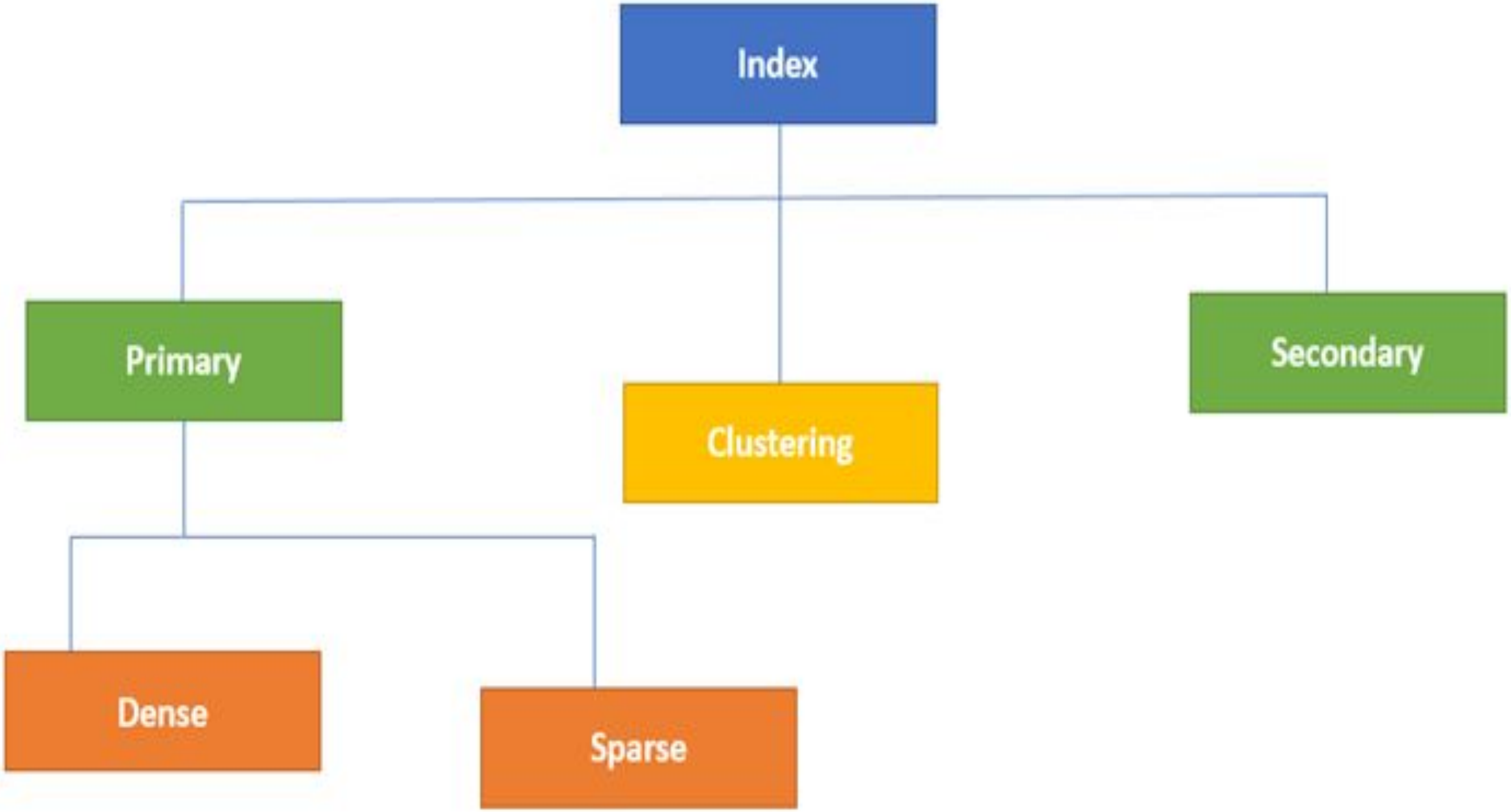


Search key	Data Reference

Fig: Structure of Index

- The first column of the database is the search key that contains a copy of the primary key or candidate key of the table. The values of the primary key are stored in sorted order so that the corresponding data can be accessed easily.
- The second column of the database is the data reference. It contains a set of pointers holding the address of the disk block where the value of the particular key can be found.

Types of Indexing

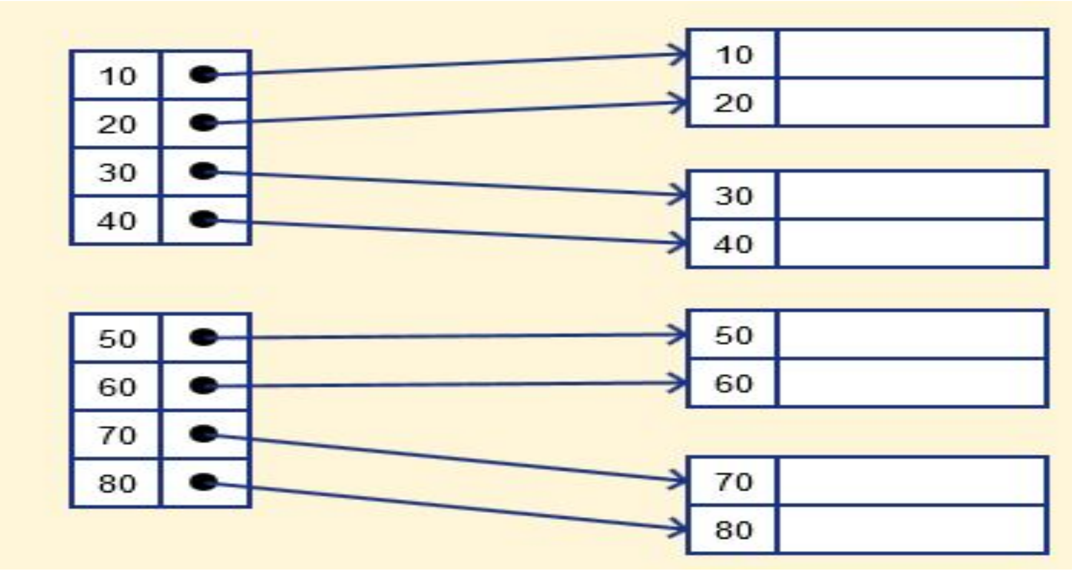


Primary Index

- Primary Index is an ordered file which is fixed length size with two fields. The first field is the same a primary key and second, filed is pointed to that specific data block. In the primary Index, there is always one to one relationship between the entries in the index table.
- The primary Indexing in DBMS is also further divided into two types.
  - Dense Index
  - Sparse Index

Dense Index

- In a dense index, a record is created for every search key valued in the database. This helps you to search faster but needs more space to store index records. In this Indexing, method records contain search key value and points to the real record on the disk.

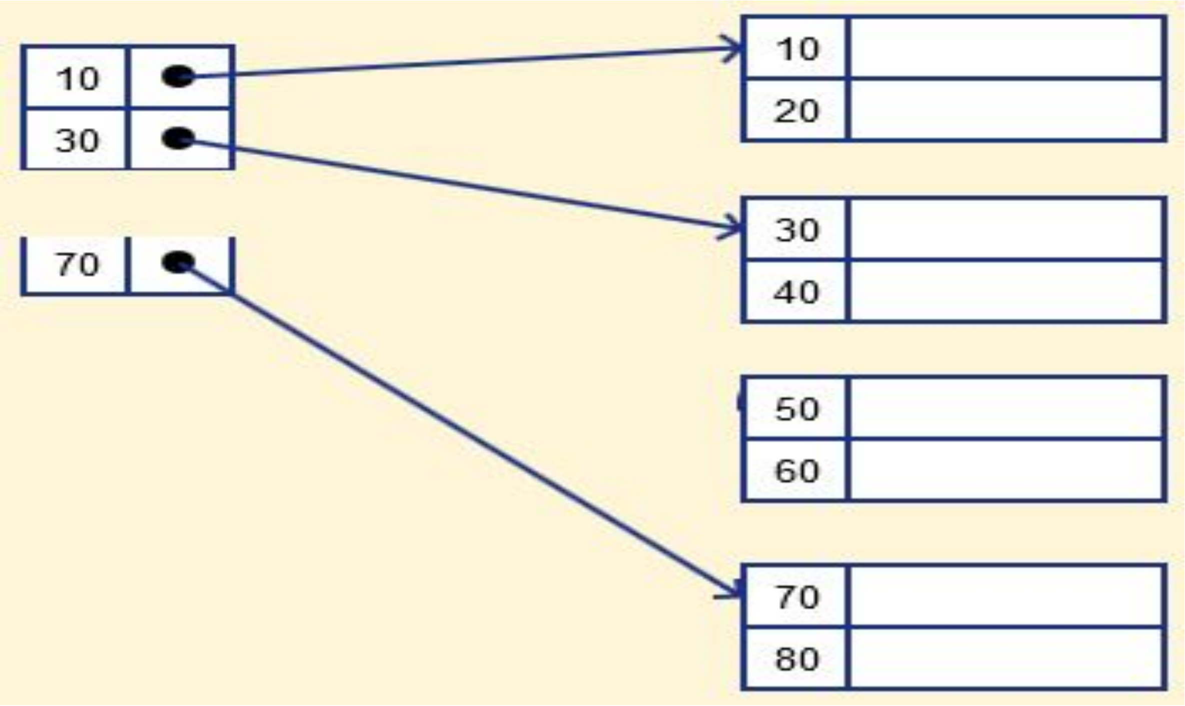


Sparse Index

- It is an index record that appears for only some of the values in the file. Sparse Index helps you to resolve the issues of dense Indexing in DBMS. In this method of indexing technique, a range of index columns stores the same data block address, and when data needs to be retrieved, the block address will be fetched.
- However, sparse Index stores index records for only some search-key values. It needs less space, less maintenance overhead for insertion, and deletions but It is slower compared to the dense Index for locating records.

Sparse Index

- Below is an database index Example of Sparse Index



Secondary Index

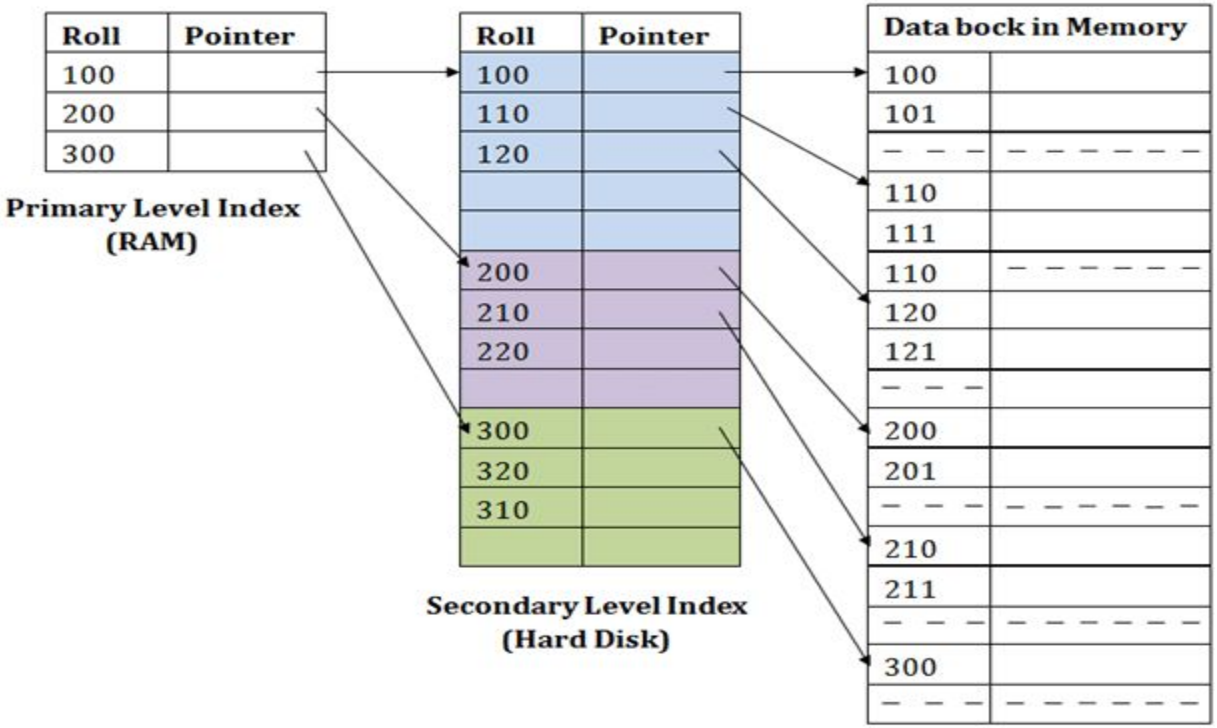
- A secondary index, in the context of a database management system (DBMS), is an additional data structure that provides an alternative way to quickly access and retrieve data from a database table. Unlike the primary index (typically based on the primary key), a secondary index is created based on one or more non-primary key columns.
- Here are some key points about secondary indexes:
- **Purpose:** The primary purpose of a secondary index is to enhance the efficiency of data retrieval operations, such as SELECT queries, that involve columns other than the primary key. It allows for faster data access based on the values in the indexed columns.
- **Non-Unique:** Unlike the primary key index, a secondary index may contain duplicate values because it is based on non-unique columns. This is because multiple rows in a table can have the same value in the indexed column(s).
- **Multiple Indexes:** A database table can have multiple secondary indexes, each created on different columns or combinations of columns. This allows for various query optimization options.

Secondary Index

- **Trade-Offs:** While secondary indexes improve query performance, they come with trade-offs. They consume additional storage space and can slow down data modification operations (e.g., INSERT, UPDATE, DELETE) because the indexes must be updated whenever the corresponding table data changes.
- **Query Optimization:** Secondary indexes enable the DBMS to quickly locate rows that meet specific criteria in queries, reducing the need for full table scans. This results in faster query execution times.
- **Index Selection:** The DBMS query optimizer decides whether to use a secondary index or perform a full table scan based on factors like query complexity, selectivity (the percentage of rows that match the query conditions), and the size of the table.
- **Maintenance:** Regular maintenance and periodic reindexing may be necessary to keep secondary indexes optimized, especially in situations where the data distribution in the indexed columns changes significantly over time.
- Overall, secondary indexes play a crucial role in database performance optimization by allowing for efficient data retrieval on non-primary key columns. However, their creation and management require careful consideration to balance the benefits of improved query performance against the overhead of maintaining additional data structures.

Example of secondary Indexing

- Let's understand secondary indexing with a database index example:
- In a bank account database, data is stored sequentially by acc_no; you may want to find all accounts in of a specific branch of ABC bank.
- Here, you can have a secondary index in DBMS for every search-key. Index record is a record point to a bucket that contains pointers to all the records with their specific search-key value.



Roll	Pointer
100	
200	
300	

Primary Level Index
(RAM)

Roll	Pointer
100	
110	
120	
200	
210	
220	
300	
320	
310	

Secondary Level Index
(Hard Disk)

Data bock in Memory	
100	
101	
— — —	— — — — — — —
110	
111	
110	— — — — — — —
120	
121	
— — —	
200	
201	
— — —	— — — — — — —
210	
211	
— — —	— — — — — — —
300	
— — —	— — — — — — —

Clustering Index

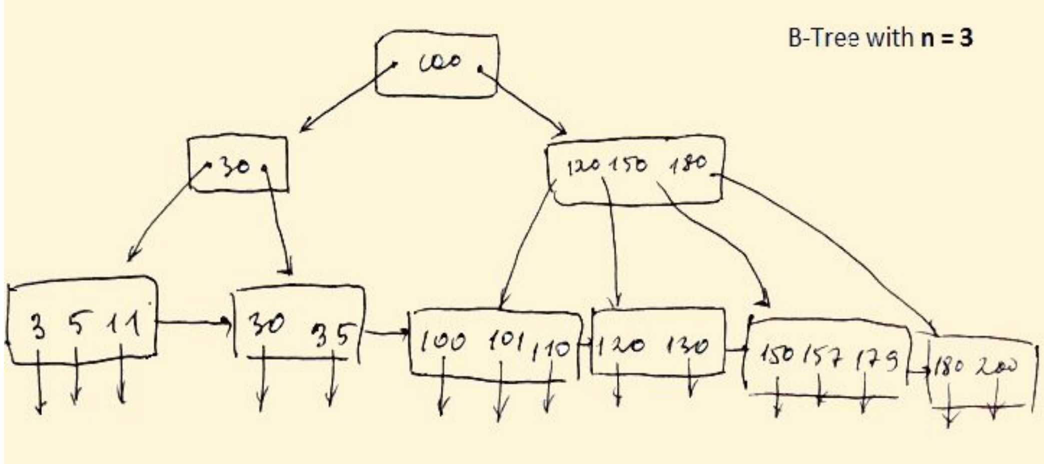
- In a clustered index, records themselves are stored in the Index and not pointers. Sometimes the Index is created on non-primary key columns which might not be unique for each record. In such a situation, you can group two or more columns to get the unique values and create an index which is called clustered Index. This also helps you to identify the record faster.
- **Example:**
  - Let's assume that a company recruited many employees in various departments. In this case, clustering indexing in DBMS should be created for all employees who belong to the same dept.
  - It is considered in a single cluster, and index points point to the cluster as a whole. Here, Department _no is a non-unique key.

B-Tree Index

- B-tree index is the widely used data structures for tree based indexing in DBMS. It is a multilevel format of tree based indexing in DBMS technique which has balanced binary search trees. All leaf nodes of the B tree signify actual data pointers.
- Moreover, all leaf nodes are interlinked with a link list, which allows a B tree to support both random and sequential access.

B-Tree Index

- Lead nodes must have between 2 and 4 values.
- Every path from the root to leaf are mostly on an equal length.
- Non-leaf nodes apart from the root node have between 3 and 5 children nodes.
- Every node which is not a root or a leaf has between n/2] and n children.



Views

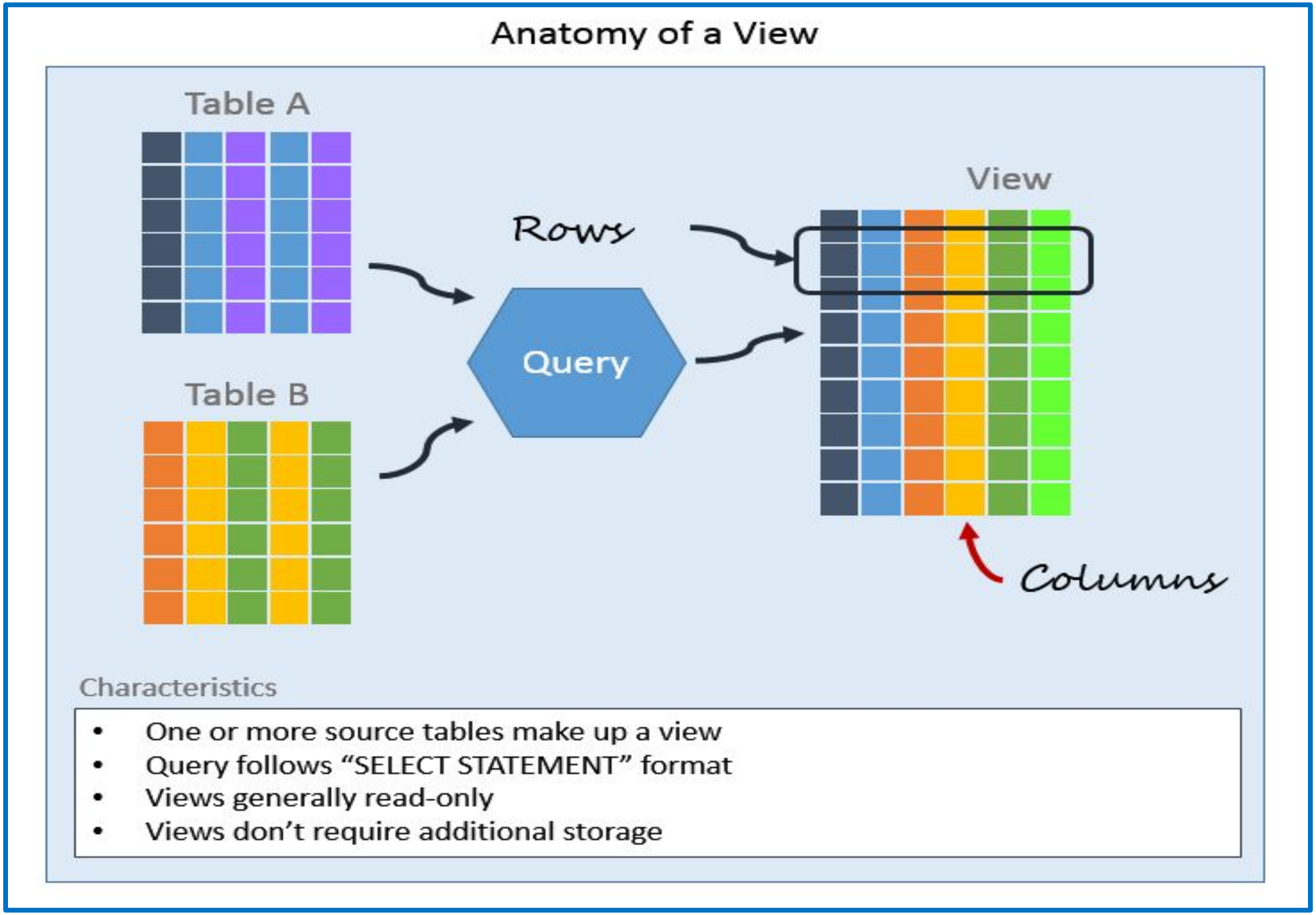
- **VIEWS** are virtual tables that do not store any data of their own but display data stored in other tables.
- A VIEW does not require any storage in a database because it does not exist physically.
- The syntax to create a VIEW is as follows:

CREATE VIEW Name AS

Select column1, Column2...Column N From tables

Where conditions;

- Views can hide complexity
- Views can be used as a security mechanism
- Views can simplify supporting legacy code

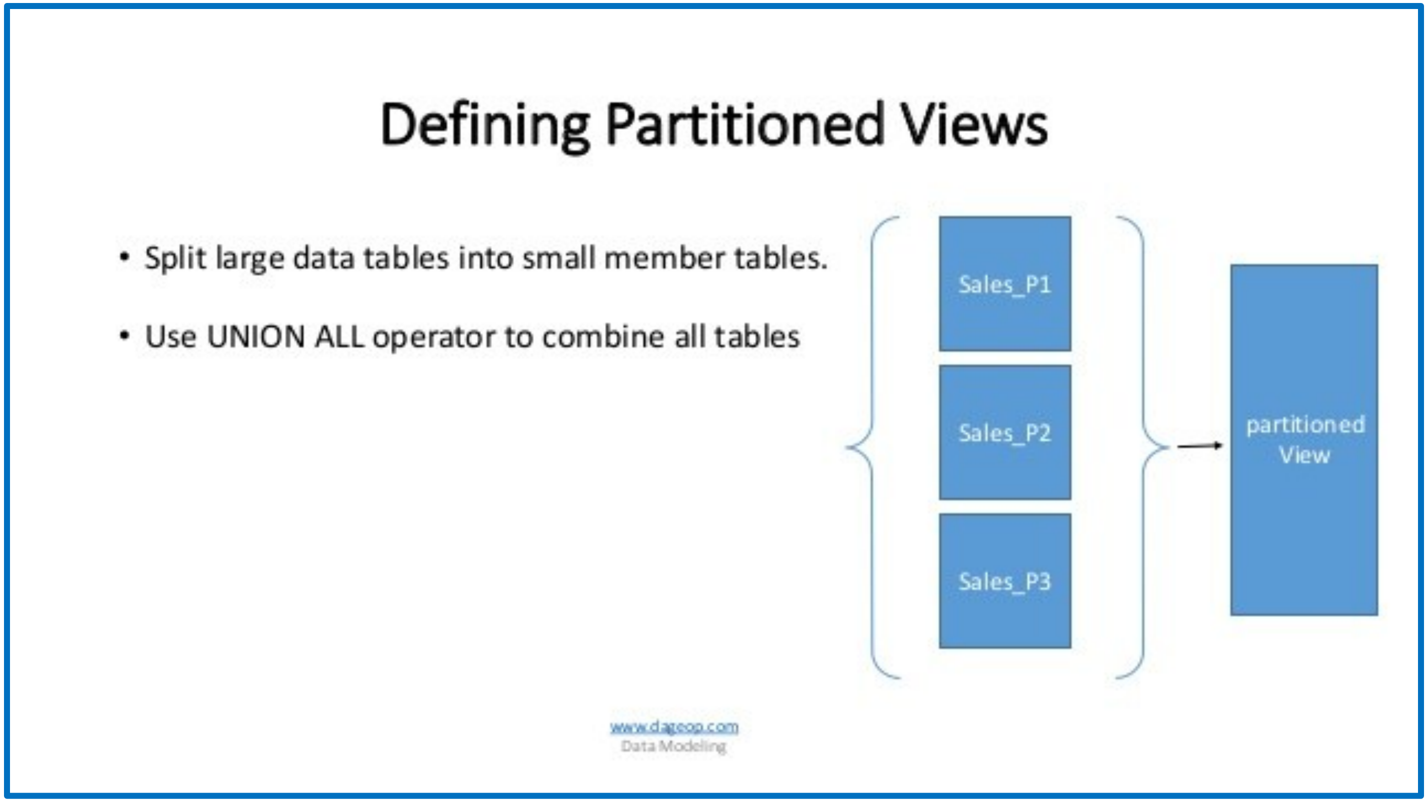


Anatomy of a View

Characteristics

- One or more source tables make up a view
- Query follows "SELECT STATEMENT" format
- Views generally read-only
- Views don't require additional storage

- A materialized view is a table on disk that contains the result set of a query.
- Materialized views are primarily used to increase application performance when it isn't feasible or desirable to use a standard view with indexes applied to it. Materialized views can be updated on a regular basis either through triggers or by using the ON COMMIT REFRESH option. This does require a few extra permissions, but it's nothing complex. ON COMMIT REFRESH has been in place since at least Oracle 10.



Defining Partitioned Views

- Split large data tables into small member tables.
- Use UNION ALL operator to combine all tables

Sales_P1

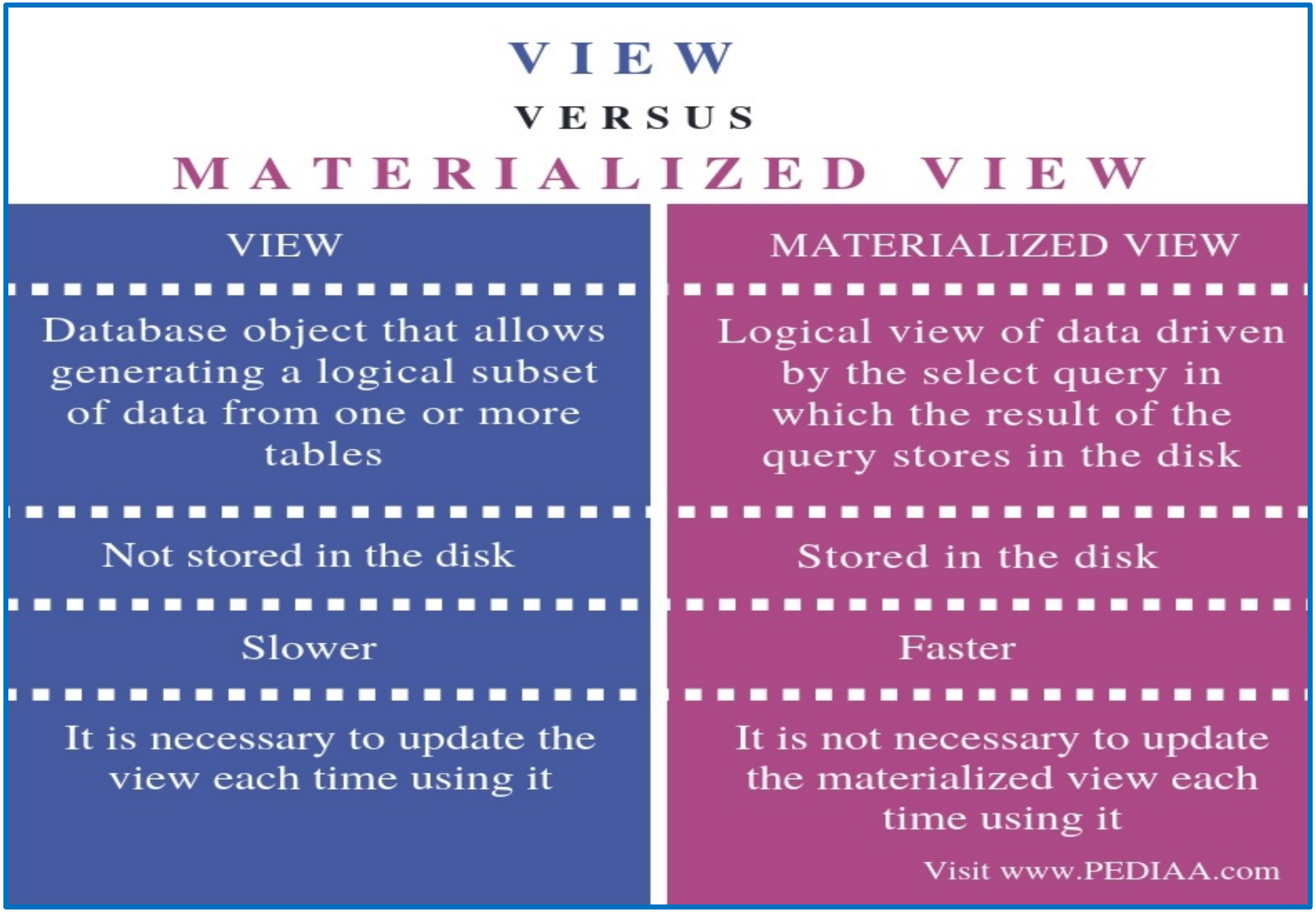
Sales_P2

Sales_P3

partitioned View

www.dageop.com

Data Modeling



VIEW VERSUS MATERIALIZED VIEW

VIEW	MATERIALIZED VIEW
Database object that allows generating a logical subset of data from one or more tables	Logical view of data driven by the select query in which the result of the query stores in the disk
Not stored in the disk	Stored in the disk
Slower	Faster
It is necessary to update the view each time using it	It is not necessary to update the materialized view each time using it

Visit www.PEDIAA.com

Managing Oracle Users:

- To access the Oracle database , a user must specified a valid database user account and successfully authenticate as required by that user accounts
- Each database user has a unique database account
- Oracle recommends this to avoid potential security holes and provide meaningful data for certain audit activities.
- Each database user account has
  - A unique user name
  - Password for database authentication
  - Default tablespace for database objects
  - Default temporary tablespace for query processing work space
  - A user profile
  - An account status

Managing Oracle Users:

- When you create a user, you are also implicitly creating a schema for that user. A **schema is a logical container for the database objects (such as tables, views, triggers, and so on) that the user creates.**
- When you drop (delete) a user, you must either first drop all the user's schema objects, or use the cascade feature of the drop operation, which simultaneously drops a user and all of his schema objects.

Creating Users

- Before creating a user, determine the following:
  - Whether or not you want to permit the user to create database objects in his own schema. If so, grant the RESOURCE role or grant individual create object system privileges
  - Whether or not you want to grant the user DBA privileges. If so, grant the DBA role. Because DBA privileges include the ability to create database objects in any schema, if you grant the DBA role, you do not need to grant the RESOURCE role or individual create object system privileges.

    Caution: Granting the DBA role to a user has security implications, because the user can modify objects in other users' schemas.

Creating Users …

- Whether or not to create the user with an expired password. When you do this, the password that you assign the user is used only for the user's first login. Upon first login, the user is prompted to select a new password.
- Example:

```
CREATE USER username
IDENTIFIED BY Password123
PROFILE CSIT
DEFAULT TABLESPACE  users
TEMPORARY TABLESPACE temp ;
```

Viewing Users

- You can view database user by using a SQL command:
  - **select * from all_users;**
- After viewing a list of users, you can then select an individual users to alter or drop (delete).

Altering User:

- Altering a user means changing some of his user attributes. You can change all user attributes except the user name, default tablespace, and temporary tablespace.
- If you want to change the user name, you must drop the user and re-create him with a different name.
- One of the attributes that you can alter is the user password. To do this you can either assign new password to the user, or request the new password from the user and then enter it. An easier and more secure way to cause a password change is to expire the password.
- When you **expire a password, the user is prompted to change his password the next time that he logs in.**
- Example:
- SQL> ALTER USER username IDENTIFIED BY Password;

Locking and Unlocking User

- To temporarily deny access to the database for a particular user, you can lock the user account. If the user then attempts to connect, the database displays an error message and disallows the connection. You can unlock the user account when you want to allow database access again for that user.
- Note:
  - Many internal user accounts are locked (or both expired and locked). You should not attempt to log in with these locked user accounts.
  - The HR user account, which contains a sample schema, is initially expired and locked. You must log in as SYSTEM, unlock the account, and assign a password before you can log in as HR.
- SQL> ALTER USER HR ACCOUNT UNLOCK;

Dropping User:

- Dropping a user removes the user from the database. Before you can drop a user, you must first drop all the user's schema objects. Or, you can use the cascade feature of the drop operation, which simultaneously drops a user and all his schema objects.
- The following are two alternatives to dropping a user and losing all the user's schema objects:
  - To temporarily deny access to the database for a particular user while preserving the user's schema objects, you can lock the user account.
  - To drop a user but retain the data from the user's tables, export the tables first.
- *Caution :Under no circumstances should you attempt to drop the SYS or SYSTEM users, or any other internal user account. Doing so could cause Oracle Database to malfunction.*

Profiles:

- Profile, which is a set of limits on database resources. If you assign the profile to a user , then that user can not exceed these limits.
- It controls resource consumption.
- Manage account status and password expiration.

Creating a profile:

- You can create many profiles in the database that specify both resource management parameters and password management parameters. However, you can assign a user only one profile at any given time.
- To create a profile, use CREATE PROFILE command by assigning a name of profile and then specifying the parameter names and their values separated by space(s).
- Profiles only take effect when **resource limits** are "turned on" for the database as a whole.

Creating a profile:

- To see status of resource limit:
- SQL> show parameter resource_limit;
- To enable resource limit, use the ALTER SYSTEM statement and set RESOURCE_LIMIT=TRUE.
- SQL> Alter system set resource_limit =true
- SQL> CREATE PROFILE csit LIMIT SESSIONS_PER_USER 2

```
IDLE_TIME 5
CONNECT_TIME 10;
```

Assigning Profiles:

- Profile can be assign in two ways either during USER creation or by using ALTER statement.
- SQL> ALTER USER shraddha PROFILE csit;

Altering Profiles:

- To alter profile use ALTER PROFILE command. A DBA must have the ALTER PROFILE system privilege to use this command. You can change any parameter in profile by using this command. The changes takes effect next time the user connects to the database.
- SQL> ALTER PROFILE csit LIMIT COMPOSITE_LIMIT 1500 PASSWORD_LOCK_TIME 5;

- **Dropping Profile:**
- Profiles that are no longer needed can be dropped using DROP PROFILE command. If the profile you want to delete is assigned to a user, trying to delete that profile will return error. For such profile use CASCADE command which will delete the profile and assigns default profile to the respective user.
- SQL> DROP PROFILE CSIT;
  - *ERROR at line 1:ORA-02382: profile CSIT has users assigned, cannot drop without CASCADE
- SQL> DROP PROFILE csit CASCADE;

Managing User Privileges and Roles:

- A user **privilege** is a right to execute a particular type of SQL statement, or a right to access another user's object.
- With proper privileges, user can create, drop, or modify objects in their own schema or in another user's schema. Privileges also determines the data to which a user should have access.
- There are two main types of user privileges:
  - System privileges
  - Object privileges
- You can grant privileges to a user by two methods:
  - Assign directly to user
  - Assign privilege to role and then assign role to the user.

Object Privilege:

- An object privilege is a right to perform a particular action on a specific schema object. Different object privileges are available for different types of schema objects. The privilege to delete rows from the CUSTOMER table is an example of an object privilege.
- Object privileges are granted on a specific object. The owner of the object has all privileges on the object. The owner can grant privileges on that object to any other users of the database or can also authorize another user in the database to grant privileges on the object to other users.

Object Privilege:

- Example:
- SQL> GRANT SELECT, UPDATE ON CUSTOMER TO SUSHANT;
  - Here SUSHANT can only query and update rows in the CUSTOMER table but cannot insert or delete, or grant privilege to another user in the database.
- SQL> GRANT SELECT, UPDATE ON CUSTOMER TO SUSHANT WITH GRANT OPTION;
  - Now, SUSHANT can grant the privilege SELECT and UPDATE on CUSTOMER to other users.
  - Some of the object privileges that can be granted to users are: SELECT, UPDATE, DELETE, INSERT, EXECUTE, ALTER, etc.

System Privilege:

- A system privilege is the right to perform a particular action, or to perform an action on any schema objects of a particular type. For example, the privileges to create tables and to delete the rows of any table in a database are system privileges.
- System privileges are the privileges that enable the user to perform an action; that are not specified on any particular object. Like object privilege, system privilege can also be granted to a user, a role, or PUBLIC. When a privilege is specified with ANY, the user is authorized to perform the actions on any schema in the database.
- The use of WITH ADMIN OPTION clause gives privilege to a user to the grant privilege to another user.
- Example:
- GRANT SELECT ANY TABLE TO SUSHANT WITH ADMIN OPTION;
- Some of the system privileges that can be granted to users are: SELECT ANY TABLE, ALTER DATABASE, CREATE SESSION, BACKUP ANY TABLE, etc.

Revoking Privileges:

- User's object privileges and system privileges can be revoked by using REVOKE statement.
- Example :
- To revoke UPDATE privilege granted to SUSHANT on CUSTOMER:
- SQL> REVOKE UPDATE ON CUSTOMER FROM SUSHANT;
- To revoke multiple privileges, use comma to separate multiple privileges: SQL> REVOKE CREATE SESSION, SELECT ANY TABLE FROM SUSHANT;
- To revoke REFERENCES privilege, specify CASCADE CONSTRAINTS clause, which will drop the referential integrity constraints created using the privileges. You must use this clause if any constraints exists.
- SQL> REVOKE REFERENCES ON CUSTOMER FROM SUSHANT CASCADE CONSTRAINTS;
- To revoke all privileges:
- SQL> REVOKE ALL ON CUSTOMER FROM SUSHANT;

Roles:

- **Role** is a named set of related privileges that are granted to users or to other roles, which eases the management of privileges and hence improve security.
- **Role Characteristics**
  - Privileges are granted to and revoked from roles as if the role were a user.
  - Roles can be granted to and revoked from users or other roles as if they were system privileges.
  - A role can consist of both system and object privileges.
  - A role can be enabled or disabled for each user who is granted the role.
  - A role can require a password to be enabled.
  - Roles are not owned by anyone; and they are not in any schema.

- **Benefits of Roles**
  - Easier privilege management.
  - Dynamic privilege management.
  - Selective availability of privileges.
  - Can be granted through the operating system.
- **Easier Privilege Management**
  - Use roles to simplify privilege management. Rather than granting the same set of privileges to several users, you can grant the privileges to a role, and then grant that role to each user.
- **Dynamic Privilege Management**
  - If the privileges associated with a role are modified, all the users who are granted the role acquire the modified privileges automatically and immediately.
- **Selective Availability of Privileges**
  - Roles can be enabled and disabled to turn privileges on and off temporarily. Enabling a role can also be used to verify that a user has been granted that role.

Creating Role:

- Using CREATE ROLE command creates the role. No user owns the role; it is owned by the database. When roles are created, no privileges are associated with it. Privileges will be granted to roles later.
- Example:
- SQL> CREATE ROLE MANAGER;
- SQL> GRANT SELECT ON CUSTOMER TO MANAGER;
- SQL> GRANT INSERT, UPDATE ON CUSTOMER TO MANAGER;
- Similar to user, role can also be authenticated. The default is NOT IDENTIFIED which means no authorization is required.

Two authorization methods are available:

- Database:
  - Database authorizes role by using a password associated with the role. Whenever such roles are enabled, user is prompted for a password.
  - Example:
  - SQL> CREATE ROLE MANAGER IDENTIFIED BY MNGR123;
- Operating System:
  - Role is authorized by Operating System when the OS can associate its privileges with the application privileges, and information about each user is configured in operating system files. To enable OS role authorization, set the parameter OS_ROLES=TRUE;
  - Example.
  - SQL> CREATE ROLE APPLICATION_USER IDENTIFIED EXTERNALLY;

- **Removing Roles:**
- Use DROP ROLES statement to remove role from database. Dropping a role will cause all privileges, that users had through the role, to get lost.
- Example:
  - SQL> DROP ROLE admin;

Querying Role Information:

- The data dictionary view DBA_ROLES lists the roles defined in the database. The column PASSWORD specifies the authorization method.
  - SQL> SELECT * FROM DBA_ROLES;
- The view SESSION_ROLES lists the roles that are enabled in the current session
  - SQL> SELECT * FROM SESSION_ROLES;
- The view DBA_ROLE_PRIVS (or USER_ROLE_PRIVS) lists all the roles granted to users and roles.
  - SQL> SELECT * FROM DBA_ROLE_PRIVS;
- The view ROLE_ROLE_PRIVS lists the roles granted to the roles, ROLE_SYS_PRIVS lists the system privileges granted to role and ROLE_TAB_PRIVS shows information on the object privileges granted to role.
  - SQL> SELECT * FROM ROLE_ROLE_PRIVS WHERE ROLE='DBA';

Database Security and Auditing

Database Security:

- Database administrator should follow best practices and continuously monitor database activity for a secure system.
- A secure system ensures the confidentiality of the data it contains. There are several aspects of security:
  - Restricting access to data and services.
  - Authenticating users.
  - Monitoring for suspicious activity.

Database Security and Auditing

Restricting Access to Data and Services:

- All users should not have access to all data. Depending on what is stored in your database, restricted access can be mandated by business requirements, customer expectations, and increasingly by legal restrictions. Credit card information, health care data, identity information, and more must be protected from unauthorized access. Oracle provides extremely fined-grained authorization controls to limit database access.
- Restricting access should include applying the principal of least privilege.

Authenticating User:

- The most basic form of user authentication is by challenging the user to provide something they know such as a password. Ensuring that passwords by following simple rules can greatly increase the security of your system.
- Stronger authentication methods include requiring the user to provide something they have, such as a token or Public Key Infrastructure (PKI) certificate. An even stronger form of authentication is to identify the user through a unique biometric characteristic such as a fingerprint, iris scan, bone structure patterns, and so on. Oracle supports advanced authentication techniques such as token-, biometric-, and certificate-based identification through the Advanced Security Option. User accounts that are not in use should be locked to prevent attempts to compromise authentication.

Monitoring for Suspicious Activity:

- Even authorized, authenticated users can sometimes compromise your system. Identifying unusual database activity such as an employee who suddenly begins querying large amounts of credit card information, research results, or other sensitive information, can be the first step to detecting information theft. Oracle provides a rich set of auditing tools to track user activity and identify suspicious trends.
- Monitoring or auditing should be an integral part of your security procedures. Oracle's built-in audit tools include:
  - Standard Database auditing
  - Value-based auditing
  - Fine-grained auditing (FGA)

- **Standard database auditing** captures several pieces of information about an audited event including that the event occurred, when it occurred, the user who caused the audited event, and which client machine the user was on when the event happened.
- **Use value-based auditing** to audit changes to data (inserts, updates, deletes). Value-based auditing extends standard database auditing, capturing not only that the audited event occurred, but the actual values that were inserted, updated, or deleted. Value-based auditing is implemented through database triggers.
- **Use fine-grained auditing (FGA)** to audit SQL statements. FGA extends standard database auditing, capturing the actual SQL statement that was issued rather than only that the action occurred.

Auditing Database:

- Auditing is storing information about database activity. You can use auditing to monitor suspicious database activity and to collect statistics on database usage. On creation of database, Oracle creates a SYS.AUD$ table, known as the audit trail, which stores the audited records.
- To enable auditing, set the initialization parameter AUDIT_TRAIL to TRUE or DB. When this parameter is set to OS, Oracle writes the audited records to an OS file instead of inserting them to SYS.AUD$ table. Use the AUDIT command to specify the audit actions.

Oracle has three types of auditing capabilities:

- **Statement auditing:** Audits SQL statements. (AUDIT SELECT BY SUSHANT audits all SELECT statements performed by SUSHANT)
- **Privilege auditing:** Audits privileges (AUDIT CREATE TRIGGER audits all user who exercises their CREATE TRIGGER privileges)
- **Object auditing:** Audits the use of specific object. (AUDIT SELECT ON HR.EMPLOYEES monitors the SELECT statement performed on the EMPLOYEES table)

- **BY clause:** restricts the auditing scope by specifying the user list in it.
- **WHENEVER SUCCESSFUL clause:** to specify that only successful statements are to be audited.
- **WHENEVER NOT SUCCESSFUL clause:** to limit auditing to failed statements.
- **BY SESSION clause:** specifies that one audit record is inserted for one session, regardless of number of times statement is executed.
- **BY ACCESS clause:** specifies that one audit record is inserted each time the statement is executed.

Example:

- To audit connections and disconnections to the database:
  - AUDIT SESSION;
- To audit only successful logins:
  - AUDIT SESSION WHENEVER SUCCESSFUL;
- To audit only failed logins:
  - AUDIT SESSION WHENEVER NOT SUCCESSFUL;
- To audit successful logins for specific users:
  - AUDIT SESSION BY SUSHANT, RAM WHENEVER SUCCESSFUL;
- To audit successful updates and deletes on the EMPLOYEES table:
  - AUDIT UPDATE, DELETE ON HR.EMPLOYEES BY ACCESS WHENEVER SUCCESSFUL;

Creating and managing DB objects (Tables, indexes, triggers, stored procedures, etc.)

Creating Tables:

- Use CREATE TABLE command to create table. You can create a table under the username used to connect to the database or with proper privileges; you can create a table under another username.
- CREATE TABLE STUDENT(

```
    SSN NUMBER,
    NAME VARCHAR2(20),
    GRADE VARCHAR(10),
    AGE NUMBER
    );
```

- Some of the data types used for creating table are: CHAR(<size>[BYTE | CHAR]), VARCHAR(<size>[BYTE | CHAR]), NUMBER(<precision>,<scale>), DATE, etc.

Altering Tables:

- Use ALTER TABLE command to change the table's storage settings, add or drop columns, or modify the columns characteristics such as default value, datatypes, and length.

Dropping table:

- If a table is no longer used, you can drop it to free up space. Once dropped, it cannot be undone. The syntax is :
- DROP TABLE [schema.] table_name [CASCADE CONSTRAINTS]

Truncating a table:

- It is similar to DROP, but it doesn't remove the structure of table, so none of the indexes, constraints, triggers, and privileges on the table are dropped. You cannot roll back a truncate operation. The syntax is:
- TRUNCATE {TABLE | CLUSTER} [<schema>.] table_name [{DROP | REUSE} STORAGE].
- You cannot truncate the parent table of an enabled referential integrity constraint. You must first disable constraint and then truncate the table even

- **Stored Procedures** are created to perform one or more DML operations on Database. It is nothing but the group of SQL statements that accepts some input in the form of parameters and performs some task and may or may not returns a value.
- Syntax

```
CREATE PROCEDURE stored_procedure_name
AS
BEGIN
   sql_statement
END;
```

Triggers

- Triggers are the SQL statements that are **automatically executed** when there is any change in the database. The triggers are executed **in response to certain events**(INSERT, UPDATE or DELETE) in a particular table. These triggers help in maintaining the integrity of the data by changing the data of the database in a systematic fashion.
- **Syntax**

create trigger **Trigger_name**
(before | after)
[insert | update | delete] on [table_name]
[**for** each row]
[trigger_body]

- **CREATE TRIGGER:** These two keywords specify that a triggered block is going to be declared.
- **TRIGGER_NAME:** It creates or replaces an existing trigger with the Trigger_name. The trigger name should be unique.
- **BEFORE | AFTER:** It specifies when the trigger will be initiated i.e. before the ongoing event or after the ongoing event.
- **INSERT | UPDATE | DELETE**: These are the DML operations and we can use either of them in a given trigger.
- **ON[TABLE_NAME]:** It specifies the name of the table on which the trigger is going to be applied.
- **FOR EACH ROW:** Row-level trigger gets executed when any row value of any column changes.
- **TRIGGER BODY:** It consists of queries that need to be executed when the trigger is called.

Advantages of Triggers

- Triggers provide a way to check the integrity of the data. When there is a change in the database the triggers can adjust the entire database.
- Triggers help in keeping User Interface lightweight. Instead of putting the same function call all over the application you can put a trigger and it will be executed.

Thank You