Contents

[Key points: 2](#_Toc212069592)

[Char Datatype: 5](#_Toc212069593)

[Numeric Datatype: 6](#_Toc212069594)

[Long Datatype: 6](#_Toc212069595)

[Date Datatype: 6](#_Toc212069596)

[RAW Datatype 7](#_Toc212069597)

[ROW ID DATATYPE 7](#_Toc212069598)

[Classification of SQL Statement. 8](#_Toc212069599)

[Constraints 9](#_Toc212069600)

[Grants and Revoke (DCL) 12](#_Toc212069601)

[Comments 12](#_Toc212069602)

[Creating a table in SQL (DDL) 12](#_Toc212069603)

[Alter Statement in SQL(DDL) 14](#_Toc212069604)

[Dropping a Table (DDL) 15](#_Toc212069605)

[TABLE SPACE AND DATAFILES 15](#_Toc212069606)

[Select Statement (DQL/DML) 19](#_Toc212069607)

[WHERE Clause 19](#_Toc212069608)

[Insert Statement (DML) 21](#_Toc212069609)

[Update Statement (DML) 21](#_Toc212069610)

[Delete Statement (DML) 22](#_Toc212069611)

[Truncate in SQL (DDL) 22](#_Toc212069612)

[TCL (Transaction Control Language) 24](#_Toc212069613)

[Merge Statement 25](#_Toc212069614)

[Order by clause 25](#_Toc212069615)

Introduction to Oracle SQL

# Key points:

**Data** refers to raw facts, figures, or symbols that can be processed and analyzed to extract useful information. Information is obtained by measuring, analyzing, and observing phenomenon. In digital world data is stored in the form of digital bits. Data can be divided into two forms based on its structure; Structured and Unstructured data.

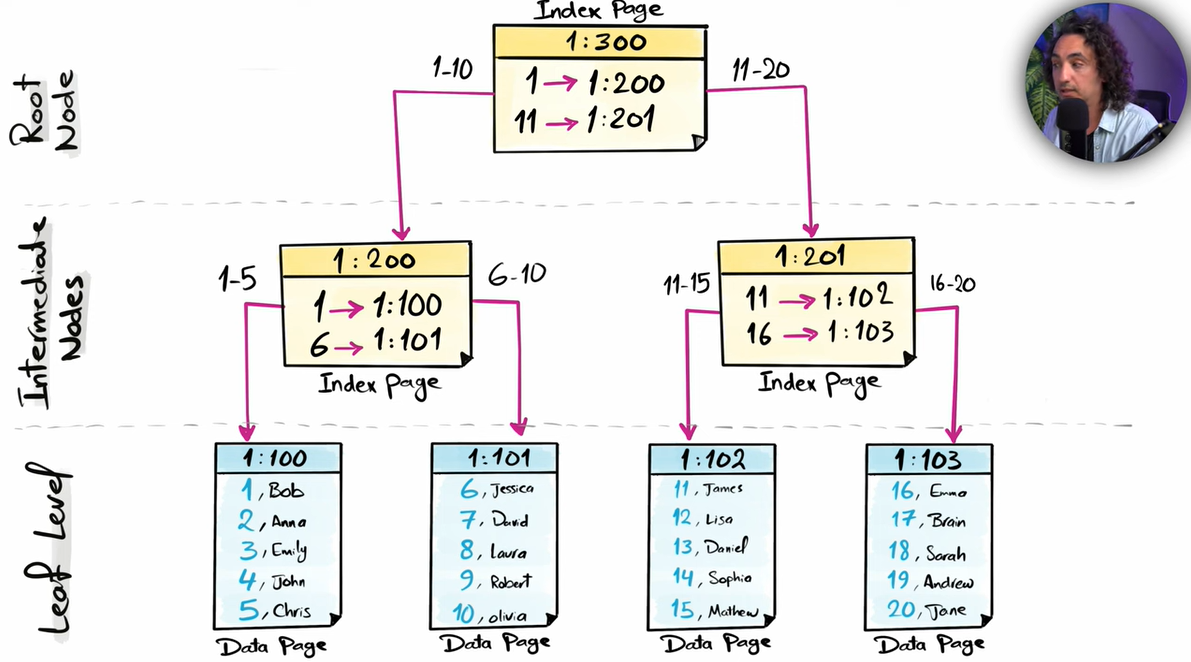
Structured data refers to data that can be organized into rows and columns. Structured data is also referred to as searchable data.

Unstructured data on the other hand is difficult to deal with because it doesn’t follow a specific pattern or doesn’t fit in any predefined model.

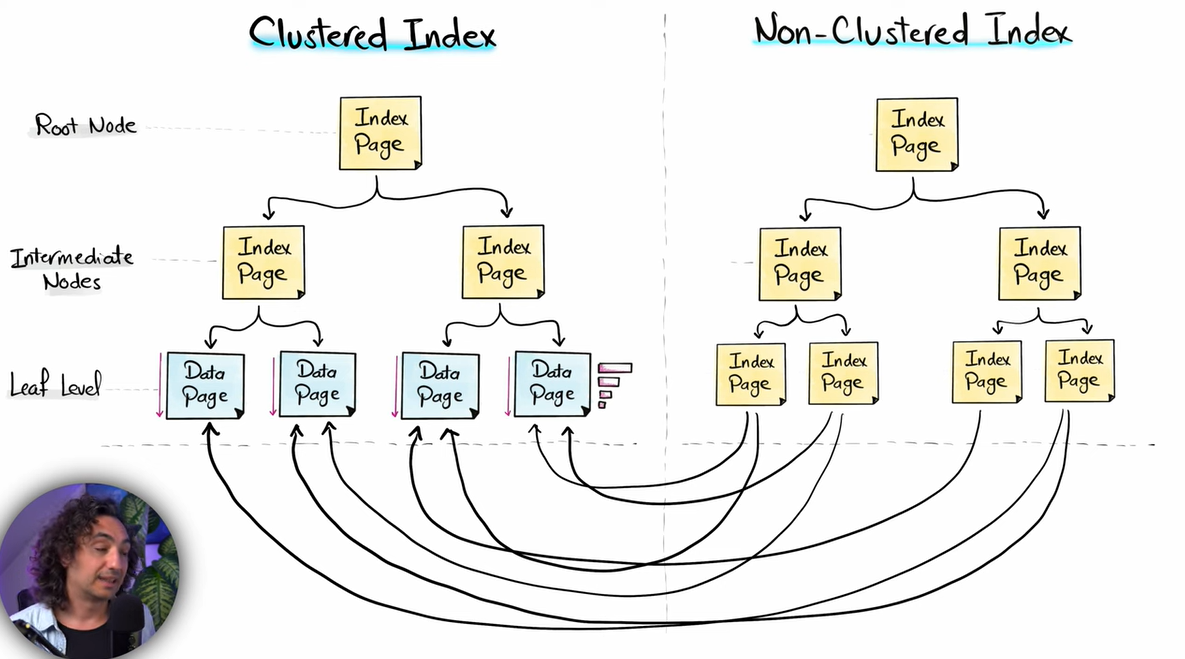
**Database** is something where we store, organize and manage information. Databases are built to store data efficiently and retrieve it quickly. In database data is stored in pages in datafile(datafile.mdf). A page is the smallest unit of data storage in a database(8kb), SQL can store anything inside it. A datafile contains datapages where rows are stored. Oracle Database was co-founded in 1977 by Larry Ellison.

//**Heap:** A table without any clustered index. Because Heaps save data in random order they are fast write, slow read data structures. SQL needs to search entire table to find an entry. //

**Clustered index**: when a clustered index is created in a column, SQL automatically sorts the entries in the column in the data page. Incase of numbers the rows will be arranged from the lowest to the highest. But that’s not all, SQL also creates a Balance tree(B-tree) for the entries; Entries in the B-tree are stored in the leaf notes. Intermediate nodes are usually index pages. Index pages do not contain data rather the contain pointers to data pages where data is located. Root note contains pointers to intermediate index notes. A user can create only one clustered index in a table, because physically the data can only be sorted in one way. Clustered indexes are faster at reading and slower at writing data.



**Non-Clustered indexes:** Unlike clustered indexes, leaf nodes in non clustered indexes contain index pages that point to data pages. Non-Clustered indexes are not concerned wheather the data being accessed isn sorted or not. Leaf nodes in non-clustered indexes usually point to data pages of clustrered indexes. A single table can have multiple non-clustered indexes. Non-clustered indexes are faster at writing and slower at reading data.



**ACID** stands for Atomicity, Consistency, Isolation, Durability. ACID defines a set of properties that ensure consistency of a database.

**Atomicity:** All transactions are performed or none is performed.

**Consistency:** The transaction takes the database from one consistent state to another.

**Isolation:** The effect of a transaction is not visible to other transactions until the transaction is committed.

**Durability:** Changes made by committed transactions are permanent. Database makes sure that changes from the transaction are not lost.

A database contains all the users you’ve created and their data.

**Oracle user and Schema**: In Oracle, Schema is a collection of logical data structures or schema objects. Schema objects are user created structures that directly refer to database objects. A database user owns a database schema with the same name as the user. A Schema is essentially a user’s workspace.

**Relational database** stores information in the form of tables; rows, and columns.

Applications retrieve information from the database, backend acts as a bridge between these two. **Frontend 🡪 Backend 🡪 Database**

There are mainly two types of databases.

* **Relational Database**
* **NoSQL Database**

**Oracle** works on relational databases, and SQL (Structured Query Language) is used to manage these databases.

**NoSQL** databases use a more flexible approach to storing data as compared to relational databases. Information here is stored in documents, and files instead of tables.

**Data Types in SQL:**

* Numeric
* Long
* Char
* Date
* Raw
* Row ID

A datatype imposes some rule/restriction on a particular column. It tells what type of data to store. Datatypes are applied while creating a table.

# Char Datatype:

**Char(n):** Stores fixed length strings, Oracle reserves space for exactly **n** characters, even if the actual value is shorter. Remaining spaces are padded with blank spaces. 2000 bytes max. Each character takes 1 byte.

**VARCHAR2(n):** Stores only the number of characters actually inserted. 4000 bytes max.

**NCHAR(n):** Same as Char, but for Unicode characters. Each character takes 2 bytes.

**NVARCHAR2(n):** Same as NVARCHAR2, but for Unicode characters.

Numeric Datatype:

**Number(p,s):** Commonly used to store numbers in Oracle. p refers to precision (the number of digits input is limited to), and s refers to scale (number of digits after decimal point). If **Number** is used without scale or precision, it can take any input (any number of digits and no limit to numbers after decimal).

e.g Number(4,3) means that it would only store less than 5 digit number either without decimal or decimal included.

**Integer:** A sub-block of **Number**, used to store Integers only. i-e: Number (4)

**Float:** Sub-block of **Number**, used to store decimal numbers regardless of scale.

*Note: if an input exceeds the required limit of a datatype, SQL generates a limit error.*

# Long Datatype:

**LONG:** Like Char it long is used to store text data. Long is mostly used when text is too long and has a large number of characters. LONG could handle up to 2 GB of data.

Limitations: Only one LONG can be used in a column.

Cannot be used with Where, Join and most other functions.

# Date Datatype:

Date: Date datatype is used to store both date and time in Oracle. Oracle stores date in the following format: DD/MM/YYYY /HH/MM/SS

**TIMESTAMP**: Extended version of Date, adds fractional seconds for more precision.

**TIMESTAMP with TIME ZONE:** Precision upto 6-digit fraction of a seconds with any time zone.

**TIMESTAMP with LOCAL TIME ZONE:** Precision upto 6-digit fraction of a seconds with local server time zone.

# RAW Datatype

**RAW:** Raw datatype is used to store binary data (data in bytes not readable text).

Encrypted values, digital signatures, etc. It can store only 2000 byte of data.

Why use RAW instead of CHAR?

Oracle may modify text in order to make it compatible across different platforms, storing binary data in CHAR can be a risk of losing the integrity of stored information. In contrast if data is stored as RAW, Oracle keeps the structure as it is.

**BLOB (Binary Large Object):** Stores very large binary data (Images, Pdfs, videos). Used when a file size exceeds 2000 bytes. Can store upto 128 GBs depending on the system.

**CLOB (Character Large Object):** Stores large amount of text data, it can typically store up to 8 GBs of character data. Unlike LONG multiple CLOB can be used in a column.

**NCLOB (National Large Object):** Same as CLOB, but for multilingual text.

# ROW ID DATATYPE

**ROWID:**  Every row in an Oracle table is assigned a rowid. ROWID is typically used to access a row faster. Oracle automatically assigns rowid to every row in a table which can’t be modified manually.

Each rowid is unique and point to the exact row in the table.

Updating a row does not change the rowid, although operations like deleting and inserting new row, importing or exporting data, or moving the row to another table and rebuild it.

**ROWIDs** are stored inside the data block of the table.

# Classification of SQL Statement.

**SQL (Structured Query Language):** SQL is the standard language used to interact with relational databases. SQL is used to control, and manipulate a database.

SQL is classified into following categories based on their operation:

* DDL (Data Definition Language)
* DML (Data Manipulation Language)
* DCL (Data Control Language)
* TCL (Transaction Control Language)
* DQL (Data Query Language)

**DDL (Data Definition Language):** DDL is used to change the definition/structure of a table. Commands such as **CREATE**, **ALTER** and **DROP** are included in DDL. Database commits the transactions before and after DDL statements. In Oracle database DDL commands come with an implicit commit, which means that its execution will commit all preceding DML commands also. We don’t deal with data at any instance to this point, DDL commands are used to define the structure for data.

**DML (Data Manipulation Language):** DML is usually used to manipulate data within a table. This includes **INSERT**, **UPDATE**, and **DELETE** operations. User needs to execute commit explicitly, to make changes permanent.

**DCL (Data Control Language):** DCL is used to control access and permissions in a table. **GRANT** and **REVOKE** are included in DCL.

**TCL (Transaction Control Language):** TCL is used to when saving or revoking transaction in a database. TCL uses operations such as **COMMIT**, **ROLLBACK**, and **SAVEPOINT**. Rollback and commit act as end points of any transaction. Commit/Rollback by a user changes data for all users using the database.

**DRL/DQL (Data Retrieval Language/ Data Query Language):** DQL is usually used to view data, or retrieve something from the data rather than alter a part of it. **SELECT** command is included in DRL.

# Constraints

Rules defined by user to allow only valid values in a column. Constraints control how and what data should be entered. Constrains are usually applied during column creation. Some constraints are defined below;

**Not Null:** No Null value allowed in the column.

**Unique Constraint:** Do not allow duplicate values in a column, or combination of column. Null values are allowed here, and **one null value is not equal to other.** A table may have multiple unique columns.

**Primary key:** Primary key is one of the most important factors of any table. It is usually defined as a column or group of columns within a table. Primary keys need to be unique between columns and cannot contain null values. A table could only have one primary key.

**Difference between primary and unique Constraint:**

On surface level the mainly differ on the basis of usage within a table, Unique constraints can be applied on multiple columns, on the other hand there can only be one primary key. Alternately, the differ in terms of how these are stored in indexes;

Index gets automatically generated when primary key or unique constraints are defined in a table. The play a vital role in SQL, indexes help improve the overall performance and the data retrieval timing. Keys are cached in a structure that allow SQL servers to access rows attached with these keys efficiently. Indexes are of two types mainly, Clustered and non-clustered indexes. Defining a primary key automatically creates a clustered index, while adding a unique constraint creates a non-clustered index.

**Foreign key:** Foreign key constraint is applied when a column in one table is a primary key of another table. Foreign key acts as a bridge between the tables. The inclusion of foreign key in a table establishes a database constraint rule that the foreign key column in the child table should always be updated with values already present in the parent table of the key. Foreign key help keep the database consistent.

**Candidate Key:** A candidate key is an attribute or a set of attributes in a database table that can uniquely identify each row. There can be multiple candidate keys in a table, and they are called "candidates" because any one of them could be chosen as the primary key.

**Check Constraints:** check constraints are applied to control input values. It specifies a boolean expression that must be true for every row. Check constraints are usually applied to keep values in ranges, checks are applied in number columns to keep values in ranges, and date column to define dates. Check constraint passes unknown results.

Conditions of check constraints cannot contain the following constructs:

* Subqueries and scalar subquery expressions
* Calls to the functions that are **not deterministic** (CURRENT\_DATE,  
  CURRENT\_TIMESTAMP, DBTIMEZONE,  
  LOCALTIMESTAMP, SESSIONTIMEZONE,  
  **SYSDATE**, SYSTIMESTAMP, UID, USER, and USERENV)
* Calls to user-defined functions
* Dereferencing of REF columns (for example, using the DEREF function)
* Nested table columns or attributes
* The pseudocolumns CURRVAL, NEXTVAL, LEVEL, or ROWNUM
* Date constants that are not fully specified

**Default Constraint:** Default constraint applies a default value to a column when no value is applied explicitly, during insertion.

**User defined Constraints:** User defined constraints refer to specific rules defined by user which cannot be implemented using pre-defined constraints. User defined constraints usually involve a series of check statements combined to keep inputs in control.

**Informational Constraints:** Informational constraints are rules which developers follow, these are not specifically implemented by the database itself.

**Defining Constraint in a table**

Constraints can be applied at column level; **Inline constraints**, or they can also be applied at table level; **out-of-line constraints**. Constraints applied on multiple columns should be applied on table level.

Not null constraint must be applied inline.

# Grants and Revoke (DCL)

**Grants** are provided by one user to another, these usually include schema permissions. GRANT command is used to give specific permissions to users, or roles. Grant permissions include SELECT, UPDATE, INSERT, and DELETE. These privileges can be assigned to different database objects, like views, tables and schema’s.

GRANT <List of grants separated by commas> on <TABLE\_NAME> to <USER>

**Revoke:** Revoke as the name signifies, is used to revoke grants/permissions provided to users/roles using GRANT command.

REVOKE <List of grants separated by comma> on <TABLE\_NAME> from <USER>

# Comments

**Comments** acts as notes for future developers. Comments make a code more efficient to understand.

COMMENT ON TABLE <Table\_Name> is “<comment>”;

COMMENT ON COLUMN <Table\_Name.Column.Name> is “<comment>”;

# Creating a table in SQL (DDL)

A table may include following entities;

Table name, Column name, Column data type, Constraints, Grants, Comment on table and column, and datafiles. Table creation is perhaps the most important part of database creation process.

Oracle restrictions on naming table and column.

* Table names cannot be the same in a single schema.
* Table name should not exceed 128 characters.
* A table should have at least one column at the time of creation.
* Table name cannot start with a digit.
* Table should not include a space in its name, if its absolutely necessary then it should be wrapped in double quotes.
* Table name is automatically capitalized by oracle, it can be avoided by double quote wrapping the table name while creating, ideally double quote wrapping in not recommended.

Oracle capitalizes table and column names before executing a query.

CREATE command is used to create a table in SQL.

CREATE <Table Name> (

Column <Datatype>,

Column <Datatype>,

Column <Datatype>

)

CREATE TABLE schema\_name.Table\_name (

Column\_1 <Data\_Type><Default\_value><Column\_level\_constraints>

Column\_2<Data\_Type><Default\_Value><Column\_level\_constraints>

Table\_Constraint

);

*Note: Use all uppercase letters for writing commands.*

# Alter Statement in SQL(DDL)

Alter statement is used for modification in SQL. This includes adding column, removing column, and renaming column. Alternatively, it can be used to modify a columns datatype and removing or modifying constraint in a column.

**Datatypes**

ALTER TABLE <TABLE NAME> ADD <COLUMN NAME> <DATATYPE>;

ALTER TABLE <TABLE NAME> MODIFY <COLUMN NAME> <DATATYPE>;

ALTER TABLE <TABLE NAME> DROP COLUMN <COLUMN NAME>;

ALTER TABLE <TABLE NAME> RENAME <COLUMN NAME > TO <COLUMN NAME>;

**Constraints**

ALTER TABLE <TABLE NAME> ADD CONSTRAINT <CONSTRAINT\_NAME> <change\_expression>;

ALTER TABLE <TABLE NAME> MODIFY <CONSTRAINT\_NAME> <change\_expression>;

ALTER TABLE <TABLE NAME> DROP COLUMN <COLUMN CONSTRAINT>;

ALTER TABLE <TABLE\_NAME> DROP CONSTRAINT <CONSTRAINT NAME>;

ALTER TABLE RENAME COLUMN <COLUMN\_NAME> TO <COLUMN\_NAME>;

ALTER TABLE <TABLE\_NAME> RENAME TO <TABLE\_NAME>;

ALTER TABLE <TABLE\_NAME> MODIFY <COLUMN\_NAME> NOT NULL;

**Renaming a Table/Object**

Renaming can be performed by a database or table owner, however there are certain instances where renaming a table is not permitted, a few of such being:

* When there is some trigger attached to the table.
* When a View is linked to the table.
* Or when a foreign key refers to the table.

Any above case and the renaming operation will generate an error.

RENAME <ExistingTableName> to <NewTableName>

# Dropping a Table (DDL)

DROP command is used for dropping a table, with consequences being the deletion of everything in the table; any view or synonym attached becomes invalid. Only users with permission and DBA owners can perform Drop operation. It should be kept in mind that, unlike previous alter operations DROP operation is irreversible!

DROP TABLE <Table Name>;

Table with special relationships cannot be dropped using just drop command, rather some keywords are used along with drop command for affects to take place;

* **CASCADE CONSTRAINTS**
* **PURGE**

CASCADE CONSTRAINTS Clause is used when dropping a table that has foreign keys referencing its primary or unique key from other tables. Drop command without cascade constraints will fail with an error, if referential integrity constraints exist.

Purge: When purge is specified, the table and its associated objects (like indexes) are immediately and permanently removed from the database. They are not moved to recycle bin and cannot be recovered later.

# TABLE SPACE AND DATAFILES

Oracle **tablespaces** are essential for organizing data in a database. They act like different containers to store data, helping keep everything neat and running smoothly.

A **tablespace** is a logical storage container for organizing data. Think of a tablespace like a big box where different kinds of data are stored. Instead of throwing all your data into one place, Oracle uses tablespaces to help separate and manage different types of information.

**Why Tablespaces?**

* **Organization**: Tablespaces help keep data organized.
* **Efficiency**: By spreading data across different tablespaces, you improve database performance.
* **Management**: It’s easier to allocate and manage disk space.

When you create an Oracle database, two important tablespaces are automatically created: **SYSTEM** and **SYSAUX**.

**SYSTEM Tablespace**

* The **SYSTEM tablespace** is like the brain of the database. It contains the **data dictionary**, which stores vital information about how the database operates — things like user permissions, table definitions, and indexing structures.
* This tablespace is crucial for the database to function, and without it, the database wouldn’t know how to manage its own data.

**SYSAUX Tablespace**

* The **SYSAUX tablespace** acts as a helper for the SYSTEM tablespace. It offloads some of the extra work so the SYSTEM tablespace isn’t overloaded.
* It stores components like:
* **Performance monitoring** data (e.g., Oracle Enterprise Manager repository).
* **Automatic Workload Repository (AWR)** data, which helps collect performance statistics.
* Other auxiliary information that helps the database run efficiently.

Together, SYSTEM and SYSAUX keep the database organized and functioning properly by handling all the important behind-the-scenes data.

**User Tablespaces**

While SYSTEM and SYSAUX handle the database’s internal operations, **user tablespaces** are where you store the data that matters to your applications — things like customer records, sales data, and employee details.

**Why Use User Tablespaces?**

* **Data organization**: You can create separate tablespaces for different types of data or for different departments (e.g., HR, Sales).
* **Security**: You can apply specific security settings to protect sensitive data.
* **Performance**: You can spread data across tablespaces to improve performance and balance the load on the database.

For example, you might create a **user tablespace** called EMPLOYEES to store all employee-related data.

**4. UNDO and TEMP Tablespaces**

Oracle databases also use two special types of tablespaces to manage temporary data and transactions:

**UNDO Tablespace**

* The **UNDO tablespace** is like an eraser. It stores information that allows the database to undo or roll back changes if needed. This is especially useful if a transaction fails, or if you want to reverse a mistake.
* It also helps ensure read consistency so that users querying data see a stable view of the data, even when others are making changes.

**TEMP Tablespace**

* The **TEMP tablespace** is like scratch paper. When Oracle needs to do big calculations (like sorting or joining data), it uses TEMP tablespace to store temporary results. Once the operation is finished, the TEMP space is cleared out, ready for the next task.
* This helps improve performance by giving Oracle space to handle large queries or sorts.

**5. Extents and Segments**

Now, let’s zoom in further to see how data is actually stored inside these tablespaces.

**Segments**

* A **segment** is a collection of data related to a specific object, like a table or an index. Think of a segment as a group of shelves inside a box (tablespace), where each shelf holds a specific type of data (like table rows or index entries).
* **Types of segments**:
* **Table segments**: Store data for tables.
* **Index segments**: Store data for indexes that speed up queries.
* **Undo segments**: Store undo information for transactions.
* **Temporary segments**: Store temporary data during query processing.

**Extents**

* A **segment** is made up of smaller pieces called **extents**. An **extent** is a continuous chunk of storage space inside a tablespace. When a segment (like a table) needs more space, Oracle allocates a new extent for it.
* Think of an extent like a shelf in a cabinet (tablespace) that holds a specific group of items (data). As more data is added, more shelves (extents) can be added to the segment.

So, the flow is like this:

1. **Tablespace**: The overall storage area (box).
2. **Segments**: Groups of related data inside the tablespace (shelves).
3. **Extents**: Smaller pieces of storage within each segment (individual sections of a shelf).

Oracle allocates and manages extents as needed, ensuring that your data objects (tables, indexes) have enough room to grow within the tablespace.

**6. Adding Data Files to a Tablespace**

Just like you might need to add more shelves when you run out of space, you can expand a tablespace by adding **data files**. These are physical files on your disk that store the data from a tablespace. When a tablespace gets full, you can add more data files to give it additional capacity.

Databases 🡪 Table Spaces 🡪 Segments 🡪 Extents 🡪 Data

SELECT \* FROM user\_users;

# Select Statement (DQL/DML)

Select statement is used to retrieve data from one or more tables in the database. User can retrieve desired data by applying different clauses along with SELECT statement.

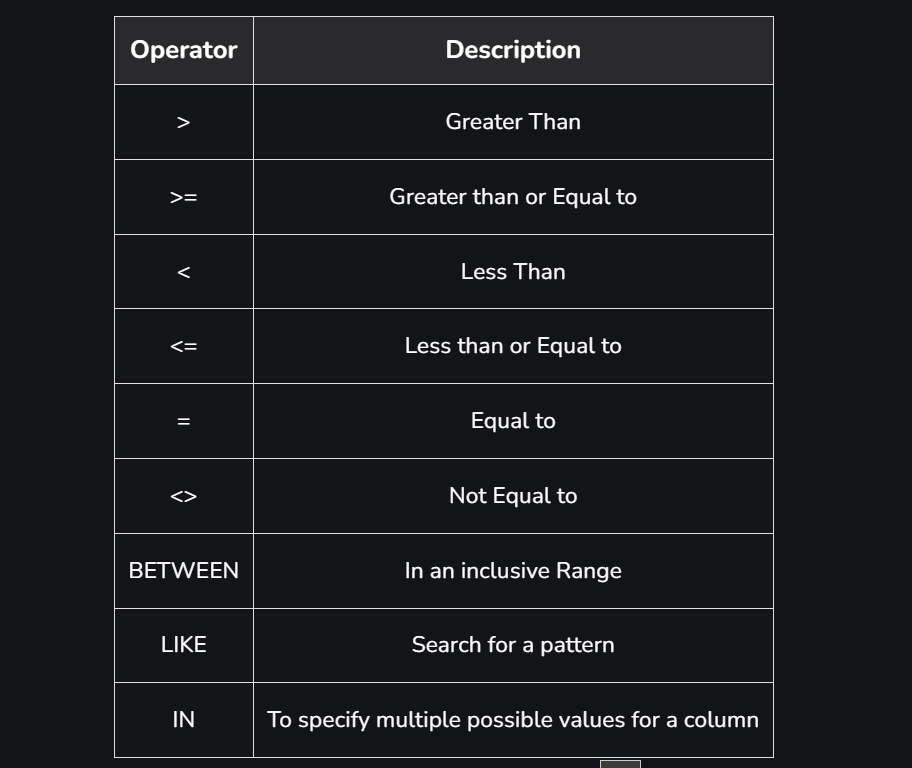
SELECT <Column names separated by comma’s> FROM <Table\_Name>

WHERE <Condition’s separated by comma’s>

# WHERE Clause

The WHERE clause is used to filter rows based on specific conditions. Whether you are retrieving, updating, or deleting data, WHERE ensures that only relevant records are affected. Without it, your query applies to every row in the table! The WHERE clause helps you:

* Filter rows that meet certain conditions
* Target specific data using logical, comparison and pattern-based operators
* Control SELECT, UPDATE, DELETE or even INSERT statements



SELECT \* FROM <TABLE\_NAME>   
WHERE < COLUMN\_NAME > BETWEEN <n> AND <n>;

SELECT \* FROM <TABLE\_NAME> WHERE Name LIKE 'S%';

SELECT Name FROM <TABLE\_NAME> WHERE Age IN (21,23);

# Insert Statement (DML)

Insert statement is used to insert data into a table. It can specify column names and values, or just values if all columns are added in the same order.

INSERT INTO <Table\_Name> ( <Column names separated by comma’s> )

VALUES ( <Values list separated by commas> ).

Above statement is used to add a single record in a table.

Some points to keep in mind while inserting data in table:

* A NOT NULL column should always be passed a value, otherwise the query will fail.
* Referential integrity constraints should also be kept in mind.
* Primary key values should not be repeated!

# Update Statement (DML)

Update statement is Data manipulation command used to modify/update rows/data within a table. It allows you to change the values of one or more columns for one or more rows. WHERE clause is mostly used along UPDATE command to modify certain value in a row.

UPDATE <Table\_Name>

SET <Column\_Name> = <expression/value>,

<Column\_Name> = <expression/value>,

…

WHERE <WHERE Conditions>;

# Delete Statement (DML)

DELETE statement is used to delete/remove rows from a table, while retaining the table structure intact. It is different from drop, which deletes an entire table.

* Removes rows based on conditions.
* Retains table schema, constraints, and indexes.
* Can delete a single row or all rows.
* Useful for cleaning or managing datasets.

WHERE Clause is used along to specify rows that need to be deleted, we can delete a single are multiple rows according to the WHERE Clause. If no WHERE condition is specified, all the rows are deleted.

DELETE FROM <Table\_Name>

WHERE <Condition…>;

Since DELETE is a rollback operation, it can be rolled back when executed in a statement.

Quick Note: Multiple Columns are always separated using commas, whilst multiple conditions are applied using AND or OR statement.

# Truncate in SQL (DDL)

Truncate command is used to delete data in a table, without deleting the table.

Truncate deletes the whole data in one shot at once unlike DELETE that removes data row by row. TRUNCATE command is used for this operation. No WHERE clause is used. Truncate operation is irreversible and can only be performed by table owner or database administrator.

An easier way to put TRUNCATE statement is, that it drops and recreates the table its applied on, rather than deleting the data row by row, thus therefore it is termed as a DDL command.

TRUNCATE TABLE <Table\_Name>

HOW is **TRUNCATE** Different from **DELETE** statement, especially for condition below:  
DELETE FROM <Table\_Name>;

TRUNCATE TABLE <Table\_Name>;

* TRUNCATE is a DDL Command thus once applied it can’t be rolled back, while changes made using DELETE (DML) command can be rolled back.
* DELETE removes data row by row sequentially, thus it is slower in contrast TRUNCATE deletes the data in one go.
* DELETE is slower when dealing with large data because the database has to prepare a copy for a possible rollback, TRUNCATE does not support rollback therefore, it saves resources and time.

TRUNCATE Statement will not work if integrity constraints are present in a table but it **depending on data.** If data doesn’t violate constraint integrity after truncate operation, TRUNCATE is allowed. DROP on the other hand is not allowed at all if integrity constraints are present.

**Integrity constraints and their relation with DROP, TRUNCATE, and DELETE Statements.**

Suppose we have two tables named **sample\_table\_1** and **sample\_table\_2,** with sample\_table\_1 links to sample\_table\_2 using a foreign key.

**Case 1: No data in both tables:**

1. **DROP** is not allowed on table\_sample\_2, since an integrity constraint exists.

Solution:

* Drop parent table(sample\_table\_1) first, then drop child table(sample\_table\_2).
* Remove foreign key constraint in sample\_table\_2, then drop.

1. **TRUNCATE**: Since the tables are both empty and no data to violate integrity constraints, **TRUNCATE** is allowed in this case.
2. **DELETE**: With no data present to delete, DELETE statement is allowed but not applicable in this case.

**Case 2: Data present in both tables:**

1. **DROP:** DROP is not allowed (same as case 1).
2. **TRUNCATE:** Now that there is data in the tables which is linked through integrity constraints, TRUNCATE is not allowed, as it can affect database integrity.
3. **DELETE:** DELETE is allowed in sample\_table\_2.

# TCL (Transaction Control Language)

TCL mainly includes three operations:

Commit, Rollback and Savepoint.

**Commit:** COMMIT Statement is used to permanently save transactions made in a session. Once, committed the changes are visible to all other users. Commit cannot be rolled back.

**Rollback:** Rollback is used to undo certain

# Merge Statement

Merge statement also called UPSERT (Update and Insert) statement is used to combine multiple DML statements (Insert, Delete, update) together, for a variable output. Merge statement typically uses two tables, a target and a source table, which are compared through a common column. Target table is updated using source table according to conditions specified. The target table must be local to the database to which the current session is connected, but you can specify a remote a table as the source table, or in subqueries of the SET clause for UPDATE operations, and in subqueries of the VALUES clause for INSERT operations.

**MERGE INTO** <Table\_Name> <alias>

USING <Table\_Name> <alias>

ON (<merge\_condition>)

**WHEN MATCHED THEN**

UPDATE SET <operation1, operation2, …..>

WHERE <condition>

/DELETE WHERE <Delete\_Condition>

**WHEN NOT MATCHED THEN**

INSERT (col1, col2, …..)

VALUES (value1, value2, …..)

WHERE <conditions……….>

# Order by clause

Order by clause is used with select statement. SELECT statement by default returns unordered data, ORDER BY clause is used along to order data, different expressions like ASC/DESC or NULLS FIRST/NULLS LAST could also be used along the clause according to user requirements.

SELECT \* FROM <Table\_Name>

ORDER BY <Column\_Name>; -------No expression provided, data returned will be ordered in ascending order by default.

SELECT \* FROM <Table\_Name>

ORDER BY <Column\_Name> ASC|DESC; --------Ordering according to expression;

SELECT \* FROM <Table\_Name>

ORDER BY <Column\_1> ASC | DESC , <Column\_2> ASC|DESC;

---------With two columns provided, ordering will be made according to column\_1 first and incase of repetitive values, column\_2 will be considered for ordering.

Note: Instead of column name, column number can also be used.

SELECT \* FROM <Table\_Name>

ORDER BY <1>, <4>;

SELECT \* FROM <Table\_Name>

ORDER BY <1>, <4> NULLS FIRST | NULLS LAST ;

--------------Ordering according to null values.

**Pagination:**

Pagination can be used limit the number of rows returned in the result set. FETCH Clause is used for this purpose, it is usually used along with order by clause to return specified number of rows. FETCH clause is also known as row limiting clause, it can also be combined with offset clause.

SELECT \* FROM <Table\_Name>

ORDER BY <Column\_Name> DESC -------------Displays first row only

FETCH FIRST ROW **ONLY**;

SELECT \* FROM <Table\_Name>

ORDER BY <Column\_Name> DESC ---Displays all matching rows at the position.

FETCH FIRST ROW **WITH TIES**;

**OFFSET Clause**

The OFFSET clause is used to specify a number N of initial query results that should be skipped (not returned to the application). N is computed by an expression that may be a single integer literal, or a single external variable, or any expression which is built from literals and external variables and returns a single non-negative integer.

SELECT \* FROM <Table\_Name>

ORDER BY <Column\_Name> DESC

OFFSET <number of rows to skip> ROWS

FETCH NEXT ROW ONLY | WITH TIES;

NEXT and FIRST keywords return the same result when no OFFSET statement proceeds them. NEXT is usually used when FETCH is followed by OFFSET.

WITH TIES as discussed returns all matching rows that specify condition.