RDBMS & Oracle SQL

Agenda

1 Introduction to Database

2 Normalization

3 Data Control Language

4 Data Definition Language

5 Data Manipulation Language

6 Transaction Control Language

7 Data Query Language

8 Single & Multiple Row Functions

Agenda

9 Set operations in SQL

10 Joins in SQL

11 SubQueries in SQL

12

13

14

15

16

Introduction to Databases

Objectives

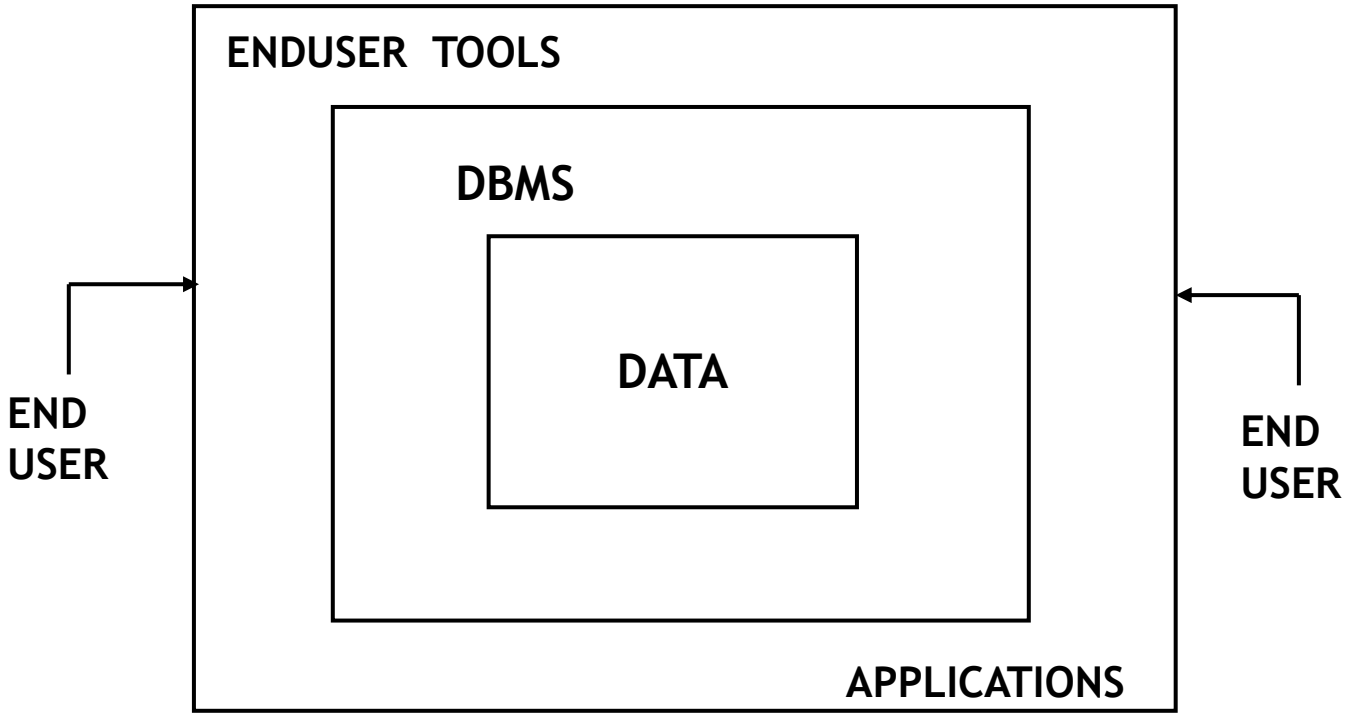
- The objective of this session is to make participants understand the Database Concepts and enable to design the Database Schema

Contents

- Database Fundamentals
- Database Views and Users
- Data Models
- Normalization

Introduction to Database

- DB - A set of inter-related data
- MS - A software that manages the data
- SCHEMA - A set of tables and relationships, which meet a specific need



Database Administrator

- Is a database architect who is responsible for design and implementation of new databases.
- Manages the database centrally
- Decides on the type of data, internal structures and their relationships
- Ensures the security of the database
- Controls access to the data through the user codes and passwords
- Can restrict the views or operations that the users can perform on the database.

Characteristics of DBMS

- Control of Data Redundancy
  - Same data is stored at number of places
  - DBMS helps in removing redundancies by providing means of integration.
- Sharing of Data
  - DBMS allow many applications to share the data.
- Maintenance of Integrity
  - Refers to the correctness, consistency and interrelationship of data with respect to the application, which uses the data.

Characteristics of DBMS

- Support for Transaction Control and Recovery
  - DBMS ensures that updates take place physically after a logical transaction is complete.
- Data Independence
  - In DBMS, the application programs are transparent to the physical organization and access techniques.
- Availability of Productivity Tools
  - Tools like query language, screen and report painter and other 4GL tools are available.

Characteristics of DBMS

- Control over Security
  - DBMS provide tools by which the DBA can ensure security of the database.
- Hardware Independence
  - Most DBMS are available across hardware platforms and operating systems.

Levels of Abstraction

- **Conceptual Level** :-the overall integrated structural organization of the database

- **Physical Level** :- the information about how the database is actually stored in the disk.

- **View / External Level** :- the user view of the database. Different for different user based on application requirement.

The Data Model

- Data model defines the range of data structures supported and the availability of data handling languages

- It is a collection of conceptual tools to describe:
  - Data
  - Data relationships
  - Constraints

- There are different data models:
  - Hierarchical Model
  - Network Model
  - Relational Model

Contd...

- **What is A Data Model**

  - A conceptual representation of the data structures that are required by a database.
  - The data structures include the data objects, the associations between data objects, and the rules which govern operations on the objects.
  - The data model **focuses** on *what data* is required and *how* it should be organized *rather than what* operations will be performed on the data.

- The data model is equivalent to an architect's building plans.
  - A data model is independent of hardware or software constraints
  - The data model focuses on representing the data as the user sees it in the "real world".
  - It serves as a bridge between the concepts that make up real-world events and processes and the physical representation of those concepts in a database.

Contd...

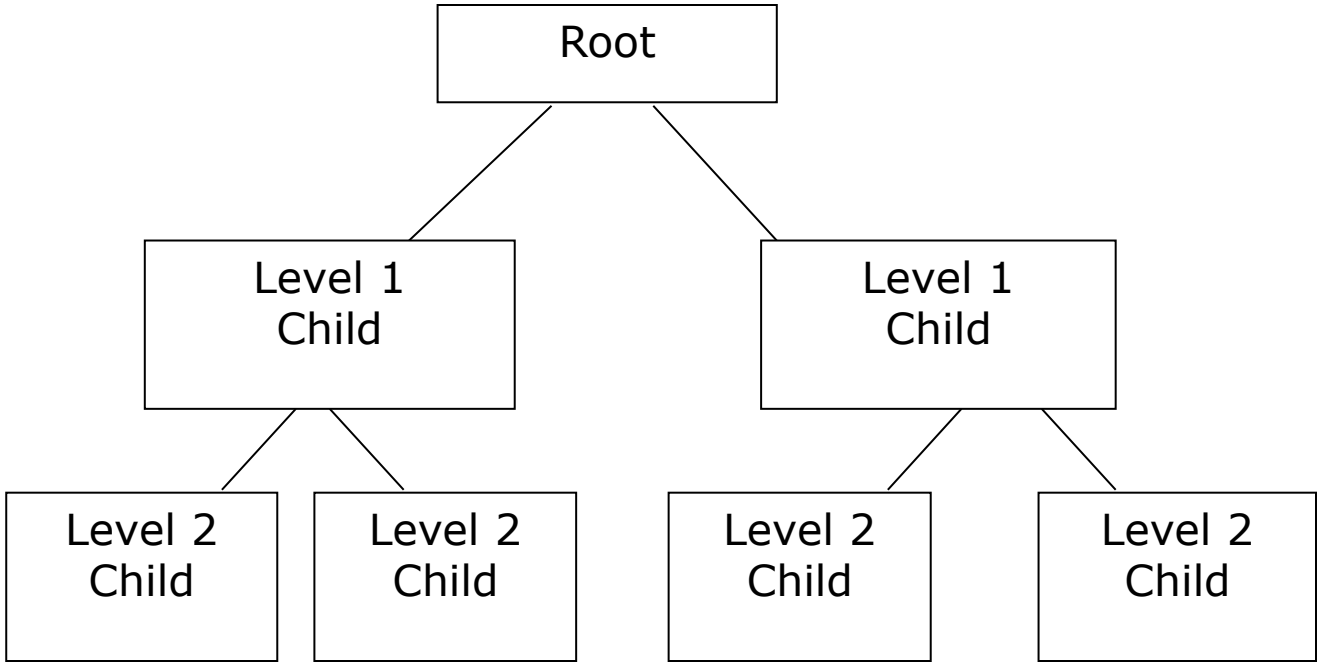
- **Why is Data Modeling Important?**
  - To make sure that the all data objects required by the database are completely and accurately represented.
  - Because the data model uses easily understood notations and natural language , it can be reviewed and verified as correct by the end-users.

Contd...

- The data model is also detailed enough to be used by the database developers to use as a "blueprint" for building the physical database. The information contained in the data model will be used to define the relational tables, primary and foreign keys, stored procedures, and triggers. A poorly designed database will require more time in the long-term. Without careful planning you may create a database that omits data required to create critical reports, produces results that are incorrect or inconsistent, and is unable to accommodate changes in the user's requirements.

Hierarchical Structure

- In this model data is represented by simple tree structure
  - Relationships between entities are represented as parent-child.
  - Many-to-many relationships are not allowed.
  - Parents and children are tied together by links called "pointers"



The Hierarchical Model

- Consider a student course - marks database
- In the hierarchical model a student can register for many courses and gets marks for each course.

- A parent can have many children
- A child cannot have more than one parent
- No child can exist without its parent

Scode	Sname
S1	A

Ccode	Cname	Marks
C1	Physics	65
C2	Chemistry	78
C3	Maths	83
C4	Biology	85

Scode	Sname
S2	B

Ccode	Cname	Marks
C3	Maths	83
C4	Biology	85

Possibilities in Hierarchical Model

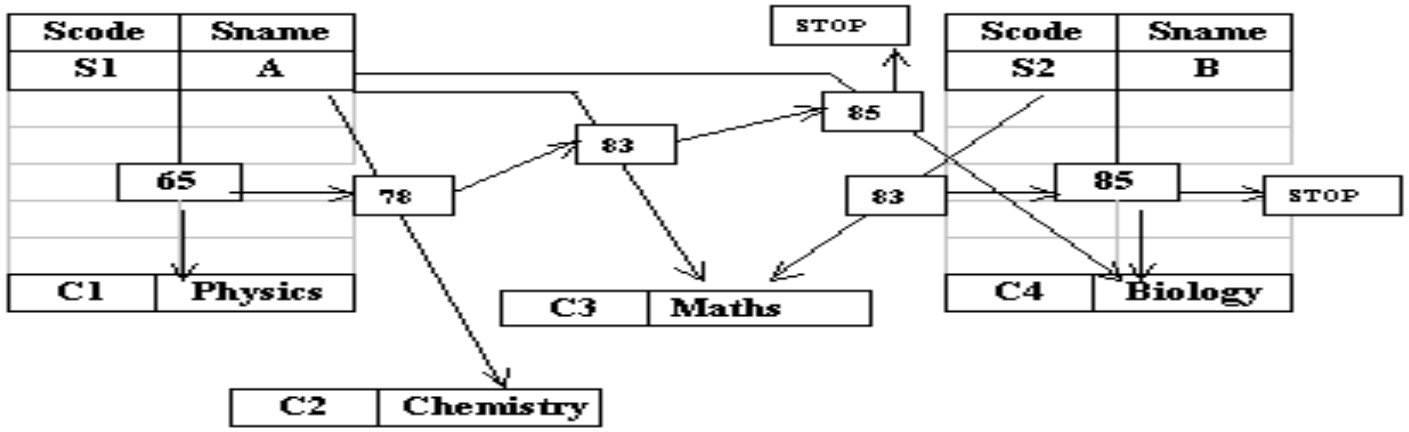
- INSERT
  - Insertion of Dummy student required to introduce a new course.
- DELETE
  - Deleting a student- the only one to take the course deletes course information.
- UPDATE
  - To change the course name of one course, the whole database has to be searched, this may result in data inconsistency.

The Network Model

- Network model solves the problem of data redundancy by representing relationships in terms of sets rather than hierarchy
- A record occurrence may have any number of immediate superiors
- Network model supports many-to-many relationships

The Network Model

- The student and course records are linked together through a marks record
- No restrictions on number of parents
- Record type can have number of parent and child record types
- It is more complex than the hierarchical model because of links
- Is a superset of the hierarchical model



Possibilities in Network Model

- INSERT
  - Inserting a course record or student record poses no problems as they can exist without any connectors till a student takes the course.
- DELETE
  - Deleting any record automatically adjusts the chain
- UPDATE
  - Update can be done only to a particular child record

The Relational Model

- The Relational Model developed out of the work done by Dr. E. F. Codd at IBM in the late 1960s who was looking for ways to solve the problems with the existing models.
- At the core of the relational model is the concept of a table (also called a relation) in which all data is stored.
- Each table is made up of records (horizontal rows also known as tuples) and fields (vertical columns also known as attributes).
- Examples of RDBMS are Oracle, Informix, and Sybase.

The Relational Model

- Because of lack of linkages relational model is easier to understand and implement.

Student Table

Scode	Sname
S1	A
S2	B

Course Table

Ccode	Cname
C1	Physics
C2	Chemistry
C3	Maths
C4	Biology

Marks Table

Ccode	Scode	Marks
C1	S1	65
C2	S1	78
C3	S1	83
C4	S1	85
C3	S2	83
C4	S2	85

Possibilities in Relational Model

- **INSERT**
  - Inserting a course record or student record poses no problems because tables are separate
- **DELETE**
  - Deleting any record affects only a particular table
- **UPDATE**
  - Update can be done only to a particular table

Relational DBMS

Examples of Relational Tables

DEPT table

Deptno	Dname	Loc
10	Accounting	New York
20	Research	Dallas
30	Sales	Chicago
40	Operations	Boston

ß 'row' or 'tuple'

á

'column' or 'attribute'

EMPLOYEE table

Empno	Empname	Job	Mgr	Deptno
7369	Smith	Clerk	7902	20
7499	Allen	Salesman	7839	30
7566	Jones	Manager	7839	20
7839	King	President		10
7902	Ford	Analyst	7566	20

Properties of Relational Data Structures

- Tables must satisfy the following properties to be classified as relational.
  - Entries of attributes are single valued
  - Entries of attribute are of the same kind.
  - No two rows are identical
  - The order of attributes is unimportant
  - The order of rows is unimportant
  - Every column can be uniquely identified.

Data Integrity

- Data integrity is the assurance that data is consistent, correct, and accessible
- Entity Integrity:
  - Entity integrity ensures that no records are duplicated and that no attributes that make up the primary key are NULL
  - It is one of the properties necessary to ensure the consistency of the database.
- Foreign Key and Referential Integrity
  - The referential integrity rule : If a foreign key in table A refers to the primary key in table B, then every value of the foreign key in table A must be null or must be available in table B

Data Integrity (Cont...)

- Unique Constraint:
  - A unique constraint is a single field or combination of fields that uniquely defines a tuple/row
  - Unique constraints ensure that every value in the specified key is unique
  - A table can have any number of unique constraints, with at most one unique constraint defined as a primary key
  - A unique constraint can contain NULL value.

Data Integrity (Cont...)

DEPT table

Deptno	Dname	Loc
10	Accounting	New York
20	Research	Dallas

– **EMPLOYEE table**

Empno	Empname	Job	Mgr	Deptno
7369	Smith	Clerk	7902	20
7499	Allen	Salesman	7839	30

Column Constraints : Specify restrictions on the values a column can take

Normalization

Atos | Syntel

Normalization

- is a technique for designing relational database tables
- minimize duplication of information
- safeguard the database against certain types of problems

Contd...

- Process of efficiently organizing data in a database
- Goals of the normalization process:
  - eliminate redundant data
  - ensure data dependencies make sense
- reduces the amount of space a database consumed
- ensures that data is logically stored.

Problems with Unnormalized Database

- can suffer from logical inconsistencies of various types and from anomalies involving data operations.
- **UPDATE ANOMALY**
- **INSERTION ANOMALY**
- **DELETION ANOMALY**

Example: UPDATE ANOMALY

Employees' Skills

Employee ID	Employee Address	Skill
426	87 Sycamore Grove	Typing
426	87 Sycamore Grove	Shorthand
519	94 Chestnut Street	Public Speaking
519	96 Walnut Avenue	Carpentry

An update anomaly. Employee 519 is shown as having different addresses on different records

Example: INSERTION ANOMALY

Faculty and Their Courses

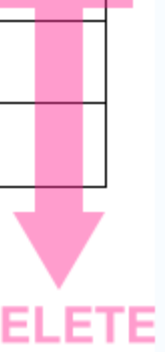
Faculty ID	Faculty Name	Faculty Hire Date	Course Code
389	Dr. Giddens	10-Feb-1985	ENG-206
407	Dr. Saperstein	19-Apr-1999	CMP-101
407	Dr. Saperstein	19-Apr-1999	CMP-201
424	Dr. Newsome	29-Mar-2007	?

An insertion anomaly. Until the new faculty member is assigned to teach at least one course, his details cannot be recorded.

Example: DELETION ANOMALY

Faculty and Their Courses

Faculty ID	Faculty Name	Faculty Hire Date	Course Code
389	Dr. Giddens	10-Feb-1985	ENG-206
407	Dr. Saperstein	19-Apr-1999	CMP-101
407	Dr. Saperstein	19-Apr-1999	CMP-201



A **deletion anomaly**. All information about Dr. Giddens is lost when he temporarily ceases to be assigned to any courses

Background to normalization:

- **Functional dependency**
- **Trivial functional dependency**
- **Transitive dependency**
- **Multivalued dependency**
- **Join dependency**
- **Super key**
- **Candidate key**
- **Non-prime attribute**
- **Primary key**

Purpose of the Normalization

1. no duplication of data

2. no unnecessary data stored within a Database.

3. All the attributes (columns) of a table are completely dependent on the primary key of a table only and not on any other attribute.

Properties of Normalization:

Two important properties of Normalization:

- **a. Lossless Join Property**
- **b. Dependency Preservation Property**

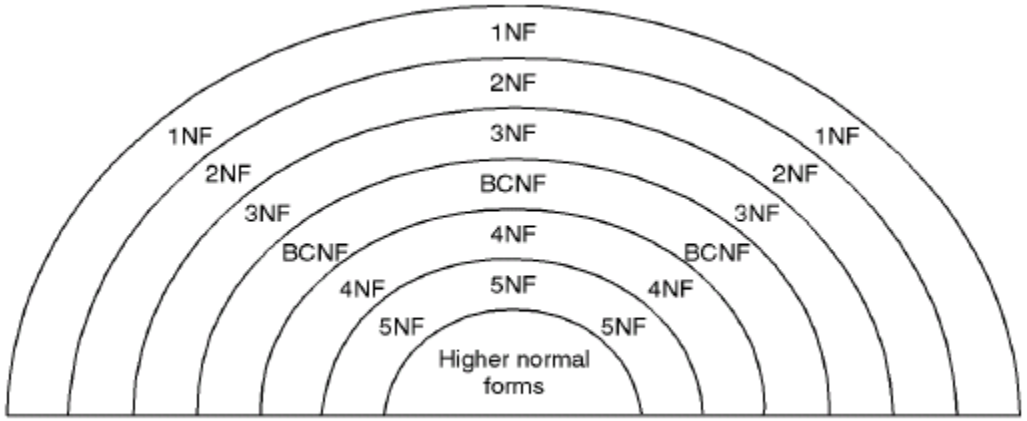
Normal forms (abbrev. NF)

- **provides criteria for determining a table's degree of vulnerability to logical inconsistencies and anomalies.**
- **The higher the NF applicable to a table, the less vulnerable it is to inconsistencies and anomalies**

Normal Forms – Anomalies and Data Redundancies

- A number of NF have been defined for classifying relations.
- has associated with the relation.
- Are used to ensure the various types of anomalies and inconsistencies

Relationship between Normal Forms



Unnormalized Relation

- A table that contains one or more repeating groups.
- To create an unnormalized table transform the data from the information source (e.g. form) into table format with columns and rows.

Unnormalized relation contains non atomic values

- Below unnormalized table shows the preferences that student registers for various classes
- Each row may contain non-atomic values

Student#	Advisor	Adv-Room	Class1	Class2	Class3
1022	Jones	412	101-07	143-01	159-02
4123	Smith	216	201-01	211-02	214-01

First Normal Form (1NF)

- Eliminate Repeating Groups - Make a separate table for each set of related attributes, and give each table a primary key.
- 1NF requires that the values in each column of a table are atomic. By atomic we mean that there are no sets of values within a column.

Contd...

- A table is in **first normal form (1NF)** if and only if it faithfully represents a relation.
- a table with a unique key and without any nullable columns is in 1NF

Example of First Normal form

unnormalized data is normalized to 1NF

Student#	Advisor	Adv-Room	Class#
1022	Jones	412	101-07
1022	Jones	412	143-01
1022	Jones	412	159-02
4123	Smith	216	201-01
4123	Smith	216	211-02
4123	Smith	216	214-01

Second Normal Form (2NF)

- Is based on the concept of Full Functional Dependency.
- *A relation R is in 2NF if and only if it is in 1NF and every non-key attribute is fully functionally dependent on the primary key of the relation R.*

Rules for 2NF

- A table in 1NF is also in 2NF form if the values in every column are functionally or transitively dependent on the complete primary key
- For every relation with a single data item making up the primary key, this rule should always be true. For those with the composite key examine every column and ask whether its value depends on the whole of the composite key or just some part of it
- Remove those that depend only on part of the key to a new relation with that part as the primary key

Example of 2NF form

- The below two tables demonstrate 2NF.
- Note the multiple Class# values for each Student# value in the above table. Class# is not functionally dependent on Student# (primary key), so this relationship is not in 2NF.

Students:

Student#	Advisor	Adv-Room
1022	Jones	412
4123	Smith	216

Registration:

Student#	Class#
1022	101-07
1022	143-01
1022	159-02
4123	201-01
4123	211-02
4123	214-01

Third Normal form

- Eliminate Columns Not Dependent On Key - If attributes do not contribute to a description of the key, remove them to a separate table
- 3nF is based on the concept of transitive dependency.
- *A relation R is in 3NF if and only if it is in 2NF and every non-key attribute is nontransitively dependent on the primary key of the relation R.*

The criteria for third normal form (3NF)

Rules :

- A relation in the second normal form is also in the third normal form, if the values in every non - key column are not transitively dependent on the primary key
- Examine every non - key column and question its relationship with every other non - key column
- If there is a transitive dependency remove both the columns to a new relation

Example of Third Normal form

- In the last example, Adv-Room (the advisor's office number) is functionally dependent on the Advisor attribute.
- move that attribute from the Students table to the Faculty table, as shown below:

Students:

Student#	Advisor
1022	Jones
4123	Smith

Faculty:

Name	Room	Dept
Jones	412	42
Smith	216	42

Contd...

- this relation has redundancies wherein If the name of the employee is changed, the change will have to be made in every tuple of the relation, otherwise inconsistencies will creep up.
- The table will further have to be decomposed to eliminate dependencies between the candidate key columns as shown,

Ecode	ProjectCode	Hours
E001	P2	48
E001	P5	100
E001	P6	15
E004	P6	250

Ecode	EName
E001	Allen
E004	Biju

BENEFITS OF NORMALIZATION

- Greater overall database organization
- Reduction of redundant data
- Data consistency within the database
- A much more flexible database design
- A better handle on database security

DRAWBACKS OF NORMALIZATION

- Reduced database performance
  - The acceptance of reduced performance requires the knowledge that when a query or transaction request is sent to the database
  - requires much more CPU, memory, and I/O to process transactions and database queries

Summary

- RDBMS are primary used for storing transactional data.
- E-R Model is prepared to map the business system to database.
- Normalization is the process of refining the table schema.

Data Control Language

Atos | Syntel

DCL: User Creation

- Creating User Statement
- Creates a new row in the mysql.user system table
- Unspecified properties are set to their default values:

```
CREATE USER [IF NOT EXISTS]
    user [auth_option] [, user [auth_option]] ...
    DEFAULT ROLE role [, role ] ...
    [REQUIRE {NONE | tls_option [[AND] tls_option] ...}]
    [WITH resource_option [resource_option] ...]
        [password_option | lock_option] ...
    [COMMENT 'comment_string' | ATTRIBUTE 'json_object']
```

```
CREATE USER GIRI-PRASAD.P@ATOS.NET
    IDENTIFIED WITH caching_sha2_password BY 'new_PWD'
    PASSWORD EXPIRE INTERVAL 180 DAY
    FAILED_LOGIN_ATTEMPTS 3
    PASSWORD_LOCK_TIME 2;
```

DCL – USER CREATION

- Authentication: The authentication plugin defined by the default_authentication_plugin system variable, and empty credentials
- Default role: NONE
- SSL/TLS: NONE
- Resource limits: Unlimited
- Password management: PASSWORD EXPIRE DEFAULT PASSWORD HISTORY DEFAULT PASSWORD REUSE INTERVAL DEFAULT PASSWORD REQUIRE CURRENT DEFAULT; failed-login tracking and temporary account locking are disabled
- Account locking: ACCOUNT UNLOCK
- An account when first created has no privileges and a default role of NONE. To assign privileges or roles, use the GRANT statement.

DCL: User Creation

- creating a new user account name as Trainer and password as syntel123.

```
CREATE USER trainer IDENTIFIED BY syntel123;
```

- Example for Creating an account that uses the caching_sha2_password authentication plugin and the given password. Require that a new password be chosen every 180 days, and enable failed-login tracking, such that three consecutive incorrect passwords cause temporary account locking for two days:

```
CREATE USER trainer IDENTIFIED WITH
        caching_sha2_password BY 'new_password'
        PASSWORD EXPIRE INTERVAL 180 DAY
        FAILED_LOGIN_ATTEMPTS 3
        PASSWORD_LOCK_TIME 2;
```

DCL: User Creation - AUTH_Options

- IDENTIFIED BY 'auth_string’

```
CREATE USER trainer
  IDENTIFIED BY `syntel123';
```

- IDENTIFIED BY RANDOM PASSWORD
- IDENTIFIED WITH auth_plugin
- IDENTIFIED WITH auth_plugin BY 'auth_string’

```
CREATE USER trainer
  IDENTIFIED WITH mysql_native_password BY `syntel123';
```

- IDENTIFIED WITH auth_plugin BY RANDOM PASSWORD
- IDENTIFIED WITH auth_plugin AS 'auth_string'

DCL: CREATE USER Role Options

- Defines which roles become active when the user connects to the server and authenticates, or when the user executes the SET ROLE DEFAULT statement during a session.
- Each role name uses the format; MySQL role names refer to roles, which are named collections of privileges.
- The host name part of the role name, if omitted, defaults to '%'.
- The DEFAULT ROLE clause permits a list of one or more comma-separated role names. These roles need not exist at the time CREATE USER is executed.

```
CREATE USER 'joe'@'10.0.0.1’
DEFAULT ROLE administrator, developer;
```

DCL: CREATE USER SSL/TLS Options

- MySQL can check X.509 certificate attributes in addition to the usual authentication that is based on the user name and credentials. For background information on the use of SSL/TLS with MySQL.
- To specify SSL/TLS-related options for a MySQL account, use a REQUIRE clause that specifies one or more tls_option values.
- Order of REQUIRE options does not matter, but no option can be specified twice. The AND keyword is optional between REQUIRE options.

```
CREATE USER giri-prasad.p'@'localhost' REQUIRE NONE;
```

```
CREATE USER giri-prasad.p'@'localhost' REQUIRE SSL;
```

```
CREATE USER giri-prasad.p'@'localhost' REQUIRE X509;
```

```
CREATE USER giri-prasad.p'@'localhost' REQUIRE ISSUER
'/C=SE/ST=Stockholm/L=Stockholm/
   O=MySQL/CN=CA/emailAddress=ca@example.com';
```

```
CREATE USER giri-prasad.p'@'localhost' REQUIRE SUBJECT
'/C=SE/ST=Stockholm/L=Stockholm/
   O=MySQL demo client certificate/
   CN=client/emailAddress=client@example.com';
```

DCL: CREATE USER SSL/TLS Options

- The SUBJECT, ISSUER, and CIPHER options can be combined in the REQUIRE clause:

```
CREATE USER giri-prasad.p'@'localhost' CIPHER 'EDH-RSA-
DES-CBC3-SHA';
```

```
CREATE USER giri-prasad.p'@'localhost'
   REQUIRE
         SUBJECT '/C=SE/ST=Stockholm/L=Stockholm/
                  O=MySQL demo client certificate/
                  CN=client/emailAddress=client@example.com'
         AND ISSUER '/C=SE/ST=Stockholm/L=Stockholm/
                  O=MySQL/CN=CA/emailAddress=ca@example.com'
         AND CIPHER 'EDH-RSA-DES-CBC3-SHA';
```

DCL: USER Resource-Limit Options

- To place limits on use of server resources by an account
- Resource-option values:
  - MAX_QUERIES_PER_HOUR count, MAX_UPDATES_PER_HOUR count, MAX_CONNECTIONS_PER_HOUR count
  - MAX_USER_CONNECTIONS count

```
CREATE USER giri-prasad.p'@'localhost’
   WITH MAX_QUERIES_PER_HOUR 500
MAX_UPDATES_PER_HOUR 100;
```

DCL: USER Account-Locking Options

- MySQL supports locking and unlocking user accounts using the ACCOUNT LOCK and ACCOUNT UNLOCK clauses for the CREATE USER and ALTER USER statements:
  - When used with CREATE USER, these clauses specify the initial locking state for a new account. In the absence of either clause, the account is created in an unlocked state.
  - If the validate_password component is enabled, creating an account without a password is not permitted, even if the account is locked.
  - When used with ALTER USER, these clauses specify the new locking state for an existing account. In the absence of either clause, the account locking state remains unchanged.
  - As of MySQL 8.0.19, ALTER USER ... UNLOCK unlocks any account named by the statement that is temporarily locked due to too many failed logins.
- The account-locking capability depends on the presence of the account_locked column in the mysql.user system table.

DCL: GRANT

- The GRANT statement enables system administrators to grant privileges and roles, which can be granted to user accounts and roles. These syntax restrictions apply:
- GRANT cannot mix granting both privileges and roles in the same statement. A given GRANT statement must grant either privileges or roles.
- The ON clause distinguishes whether the statement grants privileges or roles:
- With ON, the statement grants privileges.
- Without ON, the statement grants roles.
- It is permitted to assign both privileges and roles to an account, but you must use separate GRANT statements, each with syntax appropriate to what is to be granted.

DCL: GRANT

- Provide SELECT, INSERT, UPDATE, DELETE privileges on training. Employee table

  to username(Trainer)

```
GRANT SELECT, INSERT, UPDATE, DELETE ON
training.employees TO Trainer;
```

.

- Provide ALL privileges on training. Employee table to username(Trainer)

```
GRANT ALL ON training.departments TO Trainer;
```

DCL: GRANT

- Provide SELECT, INSERT, UPDATE, DELETE privileges on db1 to username(Trainer)

```
GRANT ALL ON db1.* TO giri-prasad.p'@'localhost’;
GRANT 'role1', 'role2' TO giri'@'localhost’, prasad'@'localhost’;
GRANT SELECT ON world.* TO 'role3';
```

.

- Provide ALL privileges on training. invoice table to username(Trainer)

```
CREATE USER giri-p'@'localhost' IDENTIFIED BY 'password’;
GRANT ALL ON db1.* TO syntel123'@'localhost’;
GRANT SELECT ON db2.invoice TO xdda'@'localhost’;
ALTER USER giri-p'@'localhost' WITH MAX_QUERIES_PER_HOUR
90;
```

DCL: GRANT *Privileges*

- Global privileges are administrative or apply to all databases on a given server. To assign global privileges, use ON *.* syntax:

```
GRANT ALL ON *.* TO ‘atos-syntel'@'somehost’;
GRANT SELECT, INSERT ON *.* TO ' atos-syntel'@'somehost';
```

- The effect of GRANT OPTION granted at the global level differs for static and dynamic privileges:
  - » GRANT OPTION granted for any static global privilege applies to all static global privileges.
  - » GRANT OPTION granted for any dynamic privilege applies only to that dynamic privilege.
- Database privileges
- Table Privileges
- Column Privileges
- Stored Routine Privileges
- Proxy User Privileges

DCL: GRANT *Privileges*

- Database privileges

```
GRANT ALL ON mydb.* TO 'sytel123'@'somehost';
GRANT SELECT, INSERT ON mydb.* TO 'sytel123'@'somehost';
```

- Table Privileges

```
GRANT ALL ON mydb.mytbl TO 'sytel123'@'somehost';
GRANT SELECT, INSERT ON mydb.mytbl TO 'sytel123'@'somehost';
```

- Column Privileges

```
GRANT SELECT (col1), INSERT (col1, col2) ON mydb.mytbl TO
'sytel123'@'somehost';
```

- Stored Routine Privileges

```
GRANT CREATE ROUTINE ON mydb.* TO 'sytel123'@'somehost';
GRANT EXECUTE ON PROCEDURE mydb.myproc TO 'sytel123'@'somehost';
```

- Proxy User Privileges

```
GRANT SELECT (col1), INSERT (col1, col2) ON mydb.mytbl TO
'sytel123'@'somehost';
```

Revoke

- DCL:
  - SELECT, INSERT, UPDATE, DELETE privileges on training. Employee table to username(Trainer)

```
REVOKE SELECT, INSERT, UPDATE, DELETE ON
training.employees FROM Trainer;
```

.

  - Revoke ALL privileges on training. Employee table to username(Trainer)

```
REVOKE ALL ON training.departments FROM
Trainer;
```

Data Definition Language

4

Atos | Syntel

Data Definition Language

- Data Definition Language
  - Used to define , modify or drop the database objects like tables , views, sequences, synonyms, etc..

- Some Statements are:
  - CREATE : Creates the database object
  - ALTER : Modify the object
  - DROP : Delete the object

Data Definition Language

- CREATE TABLE:
  - Used to create a table to hold the data
  - Integrity constraints are used to ensure accuracy and consistency of data in the table

```
CREATE TABLE [schema.]table (
        column datatype [DEFAULT expr]
        [column constraints[,…]] [,column datatype [,…]] )
        [ table constraint_[,…]] [ table ref constraint [,…]]
```

- Constraints can be defined at
  - Column Level : Specified immediately after the column definition.
  - Table Level : Specified after all the columns are defined.

Data Type

- Oracle Supports the following data types:

Data Type	Description
Char	A fixed-sized field of characters. Max 2000 bytes/characters.
Varchar2	A variable-sized field of characters. Max 4000 bytes/characters.
Number	A variable-sized number. A NUMBER data type with only one parameter is NUMBER (precision), where the parameter specifies the precision of the number.
Date	A fixed-sized 7 bit field that is used to store dates.
Long	A variable-sized field of characters. Max 2GB.
RAW and Long RAW	A variable-sized field of raw binary data.
BLOB,CLOB	The Binary Large Object and Character Large Object to hold large data

DDL – Create table

- DDL:
  - CREATE TABLE:
    - Example:

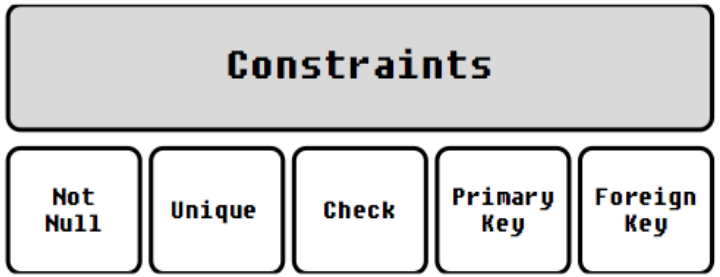
```
CREATE TABLE products (
        product_id INTEGER,
        product_name TEXT,
        cost numeric );
```

```
1   CREATE TABLE products (product_id INTEGER,
2       product_name TEXT,
3       cost numeric );
```

Output

Action Output

#	Time	Action
1	23:08:57	CREATE TABLE products (product_id INTEGER, product_name TEXT, cost numeric)



- DDL:
  - CREATE TABLE:
    - Constraints
      - Defined as the rules to preserve the data integrity in the application.
      - These rules are imposed on a column of a database table, so as to define the basic behavioral layer of a column of the table and check the sanctity of the data flowing into it.
      - The data which violates the rule, fails to pass the constraint layer and oracle raises a predefined exception.
      - They can be defined when a table is first created via the CREATE TABLE statement, or after the table is already created.
      - The key milestones achieved by the usage of constraints are:
        - Validate NULL property of the data
        - Validate uniqueness of the data
        - Validate referential integrity of the data

Constraints

Data Type	Description
CHECK	Used to set one or more conditions that the value in a column must meet before it is stored.
UNIQUE	Ensures that all values in a specified column are distinct, that is, the column cannot contain two identical instances of any value.
NOT NULL	prevents the inclusion of NULL values in a column.
Primary Key	Used to uniquely identify each row in a table. It acts to enforce row-level integrity of the table,
Foreign Key	Defines constraint between two tables in a database The foreign key identifies a column (or set of columns) in one table (the "referencing" table) that refers to set of columns in another table (the "referenced" table).

Create Table – NOT NULL

- Example: NOT NULL

```
CREATE TABLE products (
            product_id NUMBER(5),
            product_name TEXT NOT NULL
            cost numeric );
```

  - **Note :** 'Not Null' constraint can be defined only at the column level.
  - If any NULL values are encountered during insertion, Oracle raises exception 'ORA-01400: cannot insert NULL into [Column description]'.
  - NOT NULL constraint is also active during update operation; violation of the rule results in exception 'ORA-10407: cannot update [Column description] to NULL'.

UNIQUE

- Example: UNIQUE
  - It prevents duplication of the column data, but interestingly allows NULLs. It allows multiple NULLs during creation as well as modification of the column data.

```
CREATE TABLE products (
        product_id NUMBER(5),
        product_name TEXT
UNIQUE, cost numeric );
```

  - Can also be defined at Table-level as :

```
CREATE TABLE products (
        product_id NUMBER(5),
        product_name TEXT ,
        cost numeric,
        CONSTRAINT UN_NAME
UNIQUE(product_name));
```

Primary Key

- PRIMARY KEY
  - It is referred to as the hybrid evolution of NOT NULL and UNIQUE
  - The declaration can be either made at Column level, Table level or using ALTER TABLE command.
  - Unlike other constraints, there can be one and only one Primary Key in a table. It can be either a single column or a composite primary key.
  - Composite primary keys can accommodate maximum of 32 columns.

```
CREATE TABLE products (
        product_id NUMBER(5)
PRIMARY KEY,
        product_name TEXT UNIQUE,
        cost numeric );
```

  - Similar to UNIQUE key, a unique b-tree index is always created whenever a primary key is created, with the same name as that of primary key constraint.

Foreign Key

- FOREIGN KEY
  - Two tables can be connected through a column, where one table acts as Parent table while the other one is the child table.
  - The key column value set in the child table is always the subset of key column value set in the parent table, thus establishing the Parent Child relationship and obeys the referential integrity of data.
  - The key column in child table is known as Foreign Key i.e. its data references an 'external or foreign' set of values.

```
CREATE TABLE Sales (
        sales_id integer PRIMARY KEY,
        product_id integer
              REFERENCES Product(product_id),
        quantity INT CHECK(quantity >0),
           order_date DATE );
```

Foreign Key

- FOREIGN KEY
  - Oracle prevents the deletion of a Parent record, if its corresponding child exists in the child table.
    - This can be solved by adding the CASCADE clause with the constraint :
      - ON DELETE CASCADE : This will remove the record from child table, if that value of foreign key is deleted from the main table.
      - and ON DELETE SET NULL : This will set all the values in that record of child table as NULL, for which the value of foreign key is deleted from the main table.

```
CREATE TABLE Sales (
        sales_id NUMBER(4) PRIMARY KEY,
        product_id NUMBER(5),
        quantity INT CHECK(quantity >0),
        order_date DATE,
        CONSTRAINT FK_PRODUCT FOREIGN KEY
(product_id) REFERENCES Product(product_id) ON
DELETE CASCADE );
```

Foreign Key

- Guidelines for establishing Relationship
  - The key column in the Parent table must be a Primary Key.
  - Multiple Child key columns can refer single Parent key column.
  - Though the Parent table key column is a Primary Key (which does not allows NULLs), foreign key can accommodate NULL values.
  - Oracle prevents the deletion of a Parent record, if its corresponding child exists in the child table.
    - If the constraint is enforced with ON DELETE CASCADE option, then the child record would also be deleted.
    - If the constraint is enforced with ON DELETE SET NULL option, then the child record would not be deleted, but their key column value would be set to NULL
  - Oracle prevents the creation of a Child record, for which value of key column does not exists in the Parent table.

Create AS

- DDL:
  - CREATE TABLE: Creating a Table From an Existing Table

```
CREATE TABLE  AS (SELECT
Query);
```

  - Example:

```
CREATE TABLE  Emp_New AS (
        SELECT * from
HR.Employees);
```

  - The new Table Emp_New will be created with the data populated from HR.Employees.
  - Although the table structure and data is copied the constraints are not copied.

Create AS

- DDL:
  - CREATE TABLE: Creating a Table with the same structure as an existing Table.

```
CREATE TABLE  Emp_New AS (
        SELECT * from HR.Employees where 1=2);
```

  - The structure of a table can be copied by giving in a WHERE condition that evaluates to false

How will you find out the structure of a table in oracle?

Answer: Using DESCRIBE/DESC Command

E.g. DESC Employees;

If an unique key constraint on DATE column is created, will it validate the rows that are inserted with SYSDATE ?

Answer: It won't, Because SYSDATE format contains time attached with it.

- DDL:
  - ALTER TABLE:
    - It is used for alteration of table structures. There are various uses of *alter* command, such as,
      - To add a column to existing table
      - To rename any existing column
      - To change data type of any column or to modify its size.
      - alter is also used to drop a column
      - Rename a Table

ALTER TABLE

- DDL:
  - ALTER TABLE: Columns
    - Add Column to an Existing Table
    - Syntax:

```
ALTER TABLE table_name ADD (column_1 column-definition,
column_2 column-definition, ... column_n column_definition);
```

    - Example:

```
ALTER TABLE Products ADD price NUMBER(4) NOT NULL
```

```
ALTER TABLE supplier ADD (
        supplier_name varchar2(50),
        city varchar2(45));
```

ALTER Table

- Modifying an Existing Column
  - Syntax:

```
ALTER TABLE table_name MODIFY column_name column-
definition[, column_2 column-definition, ... column_n
column_definition]);
```

  - Example:
    - Altering the size of the column

```
ALTER TABLE supplier
        MODIFY supplier_name varchar2(70) not null;
```

    - Altering multiple columns

```
ALTER TABLE supplier MODIFY (
        city varchar2(75)  NOT NULL,
        supplier_name(75)) NOT NULL;
```

ALTER TABLE Column

- Columns
  - Dropping an Existing Column
  - Syntax:

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

  - Example:

```
ALTER TABLE supplier DROP COLUMN supplier_name;
```

ALTER TABLE RENAME

- Renaming a Column
  - Starting in Oracle 9i Release 2, you can now rename a column.

  - Syntax:

```
ALTER TABLE table_name
        RENAME old_column TO new_column_name
```

  - Example:

```
ALTER TABLE supplier RENAME COLUMN city TO supplier_city;
```

ALTER TABLE enable/disable

- Enable / Disable a Constraint
  - Syntax:

```
ALTER TABLE  table_name
        ENABLE/DISABLE constraint  ConstraintName;
```

  - Example:

```
ALTER TABLE Sales
        DISABLE CONSTRAINT FK_PRODUCT
```

ALTER TABLE add / drop

- Add /Drop Constraints

```
ALTER TABLE table_name
        ADD/DROP CONSTRAINT ConstraintName
```

- Example:

```
ALTER TABLE Sales
        DROP CONSTRAINT FK_PRODUCT
```

ALTER – Rename a Table

- Rename a Table

```
RENAME old_table_name  TO New_Table_name
```

- Example:

```
RENAME TABLE Suppliers TO vendors
```

- **Note :** If you change the object's name any reference to the old name will be affected. You have to manually change the old name to the new name in every reference.

DROP TABLE

- DROP TABLE:
  - Used to delete one or more table structure which is no longer in use.
  - When a table is dropped, all the tables' rows, indexes and privileges will also be removed.

```
DROP TABLE table_name
DROP TABLE table_name CASCADE constraints;
DROP TABLE table_name PURGE;
```

  - CASCADE Constraints :
    - Deletes all foreign keys that reference the table to be dropped, then drops the table.

  - PURGE :
    - As of Oracle 10g, when a table is dropped it is moved into the recycle bin unless the PURGE modifier is used. If the PURGE modifier is used then the table is dropped completely from the database and is unrecoverable.

DROP TABLE: Retrieving the Table Back

- DROP TABLE: Retrieving the Table Back
  - With Oracle 10g, the table once dropped, cannot be queried or acted upon and is moved from schema to Oracle Recycle bin.
  - We can see the contents of user's recycle bin:

```
SELECT * FROM recyclebin;
```

  - Table can be retrieved back from the recyclebin using *FLASHBACK* utility as:

```
FLASHBACK table table_name
          TO BEFORE DROP;
```

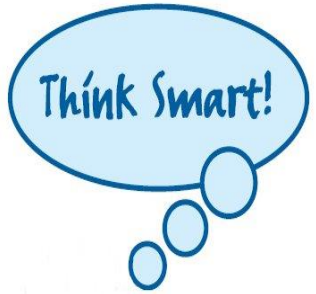
  - To drop the table entirely and release the space, use "PURGE" clause.
    - Unless you purge them, Oracle will leave objects in the recyclebin until the tablespace runs out of space, or until you hit your user quota on the tablespace.
    - At that point, Oracle purges the objects one at a time, starting with the ones dropped the longest time ago, until there is enough space for the current operation.

TRUNCATE

- TRUNCATE :
  - Used to remove (delete) all rows from a table.
  - Also deallocates all of the space used by the removed rows

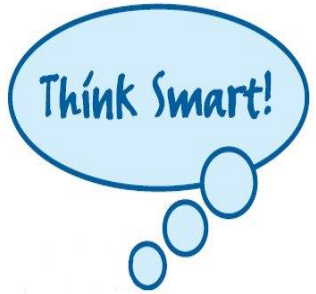
```
TRUNCATE TABLE table_name;
```

How do you force a DROP command to drop the parent table even when a child table is existing?



Answer: Using the CASCADE CONSTRAINTS Option with the DROP Command

Where the integrity constraints are stored in Data Dictionary ?



Answer: The integrity constraints are stored in USER_CONSTRAINTS.

Data Manipulation Language

Atos | Syntel

Oracle SQL

- DML:
  - Data Manipulation Language
  - Used to manipulate the data stored in the database.
    - Inserting Data
    - Updating Data
    - Deleting Data
    - Merging Data

DML – INSERT

- DML:
  - INSERT :
    - Data can be inserted into the table using INSERT Statement

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

    - Example:

```
INSERT INTO suppliers (supplier_id,
supplier_name,supplier_city,phone)
VALUES('S01','UPS','Columbia','0441-782-7892');
```

      - If data is inserted into all the columns of the table the column_name can be skipped

```
INSERT INTO suppliers      VALUES('S01','UPS','Columbia','0441-
782-7892');
```

DML INSERT ALL

- INSERT ALL
  - INSERT ALL can be used for inserting multiple rows at a time
  - Applicable only to Oracle10g and above

```
INSERT ALL
        INTO table_name (column-1, ..) VALUES (value-1... )
        INTO table_name (column-1, ..) VALUES (value-1... )
```

  - Example:

```
INSERT ALL
          INTO Supplier (supplier_name) VALUES ('A')
          INTO Supplier (supplier_name) VALUES ('B')
          INTO Supplier (supplier_name) VALUES ('C')
SELECT * FROM dual;
```

DML INSERT – sub select

- INSERT :
  - We can also use sub-selects in an INSERT Statement

```
INSERT INTO table_name (column-1, column-2, ... column-n)
SELECT column1,column2 FROM table_name …..
```

  - Example:

```
INSERT INTO suppliers (supplier_id, supplier_name) SELECT
account_no, name
FROM customers WHERE city = 'Newark';
```

DML – UPDATE

- UPDATE :
  - Used to modify the existing data in the table.

```
UPDATE table_name
 SET column-1= value-1[, column-2 = value-2, ... column-n)
[WHERE …..]
```

  - Example:

```
UPDATE suppliers
SET name = ‘Samsung' WHERE name = ‘Apple’;
```

    - If the WHERE Clause is ignored it will update all the rows in the table.
  - Example : Updating Multiple Columns

```
UPDATE suppliers
SET name = ‘Nokia', product = ‘Nokia 910'
WHERE name = ‘Samsung';
```

DML - Update

- Records in one table based on values in another table.

```
UPDATE suppliers
SET supplier_name = (SELECT customers.name
                     FROM customers
                     WHERE customers.customer_id =
suppliers.supplier_id)
WHERE EXISTS
                (SELECT customers.name
                 FROM customers
                WHERE customers.customer_id =
suppliers.supplier_id);
```

DML – UPDATE

- Update based on a query returning multiple values

```
UPDATE suppliers
SET (supplier_name,city) = (SELECT customers.name,city
                            FROM customers
                            WHERE customers.customer_id =
suppliers.supplier_id)
WHERE supplier_id=10;
```

DML – DELETE

- DELETE :
  - Used to delete the data in the table
  - DELETE Statement by default deletes all the data from the table unless specified using a WHERE clause

```
DELETE
FROM table_name
[WHERE …..]
```

DML – MERGE

- MERGE :
  - The MERGE statement was introduced in Oracle 9i to conditionally insert or update data in a single atomic statement, depending on the existence of a record.
  - The MERGE statement reduces table scans and can perform the operation in parallel if required.
  - Syntax:

```
MERGE into <target table>
USING  <souce table/view/result of subquery>
ON  <match condition>
WHEN MATCHED THEN
    <update clause>
    <delete clause>
WHEN NOT MATCHED THEN
    <insert clause>
```

DML – MERGE

- Example:

```
MERGE INTO dept d
USING dept_online o
 ON (d.deptno = o.deptno)
WHEN MATCHED THEN
          UPDATE SET d.dname = o.dname, d.loc = o.loc
WHEN NOT MATCHED THEN
           INSERT (d.deptno, d.dname, d.loc) VALUES (o.deptno,
o.dname, o.loc);
```

DML – MERGE

- MERGE :
  - With the release of 10g, Oracle has added many enhancements to MERGE statement
    - It can now UPDATE, DELETE and INSERT with separate conditions for each.
    - It also supports UPDATE-only or INSERT-only operations.
    - Deleting during the merge operation

  - Example: Oracle 10g : UPDATE- only

```
MERGE INTO dept d
USING dept_online o
 ON (d.deptno = o.deptno)
WHEN MATCHED THEN
          UPDATE SET d.dname = o.dname, d.loc = o.loc
```

  - Similarly for INSERT-only operations.

DML – MERGE

- DML:
  - MERGE :
    - With Oracle 10g, we can now apply additional conditions to the UPDATE or INSERT operation within a MERGE.
      - This is extremely useful if we have different rules for when a record is updated or inserted but we do not wish to restrict the ON condition that joins source and target together.
    - Example: Oracle 10g : Conditional dml

```
MERGE INTO dept d
USING dept_online o
 ON (d.deptno = o.deptno)
WHEN MATCHED THEN
        UPDATE SET d.dname = o.dname, d.loc = o.loc
        WHERE d.deptno IN (10,20,30)
```

Oracle SQL

- MERGE :
  - With Oracle 10g, we can conditionally DELETE rows from the target dataset during an UPDATE operation.
    - The DELETE works against conditions on the **target** data, not the source.
    - The DELETE works only on rows that have been updated as a result of the MERGE. Any rows in the target table that are not touched by the MERGE are not deleted, even if they satisfy the DELETE criteria.
  - **Example**: Oracle 10g : Conditional dml

```
MERGE INTO emp e1
USING emp_load e2
ON (e2.empno = e1.empno)
WHEN MATCHED THEN
        UPDATE SET e1.sal = e2.sal DELETE WHERE sal <= 0
WHEN NOT MATCHED THEN
        INSERT (empno, ename, job, mgr, hiredate, sal, comm,
deptno) VALUES (e2.empno, e2.ename, e2.job, e2.mgr, e2.hiredate,
e2.sal, e2.comm, e2.deptno);
```

Transaction Control Language

Atos | Syntel

Oracle SQL

- TCL:
  - Transaction Control Language
    - Properties of TCL
    - Commit
    - Rollback

Oracle SQL

- Properties of TCL:
  - **Atomicity**: This property ensures that either all the operations of a transaction reflect in database or none.
  - **Consistency**: To preserve the consistency of database, the execution of transaction should take place in isolation (that means no other transaction should run concurrently when there is a transaction already running).
  - **Isolation**: For every pair of transactions, one transaction should start execution only when the other finished execution
  - **Durability**: Once a transaction completes successfully, the changes it has made into the database should be permanent even if there is a system failure. The recovery-management component of database systems ensures the durability of transaction.

TCL – COMMIT & ROLLBACK

- TCL:
  - Commit :
    - Save changes permanently made on tables after query is executed successfully

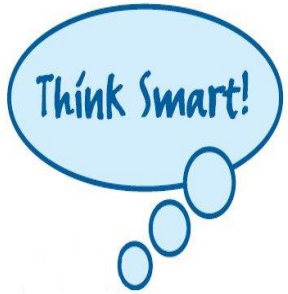
```
UPDATE suppliers
SET name = ‘Samsung' WHERE name = ‘Apple’;
Commit;
```

  - Rollbackl:
    - undo the changes made on tables after query is execution getting failure

```
UPDATE suppliers
SET name = ‘Samsung' WHERE name = ‘Apple’;
rollback;
```

DELETE VS TRUNCATE

How is DELETE different from TRUNCATE?

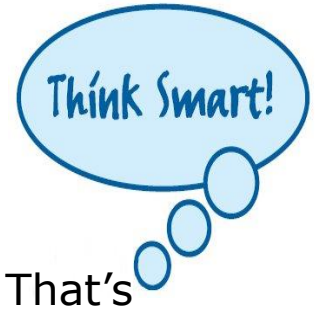


Answer: Delete is a DML Operation. After performing a DELETE we need to COMMIT or ROLLBACK the transaction to make the change permanent or to undo it and this operation will cause all DELETE triggers on the table to fire.

TRUNCATE operation is a DDL Operation. It cannot be rolled back and no triggers will be fired. As far as performance is concerned TRUCATE is faster and doesn't use as much undo space as a DELETE.

DELETE VS TRUNCATE

Why is TRUNCATE faster than DELETE?



Answer: When you type DELETE. All the data get copied into the Rollback Tablespace first and then delete operation is performed. That's how when you type ROLLBACK after deleting a table ,you can get back the data. All this process is time consuming.

Whereas when you type TRUNCATE, it removes data directly without copying it into the Rollback Tablespace and makes TRUNCATE a faster operation in comparison. Once you Truncate the data cannot be retrieved.

Recap

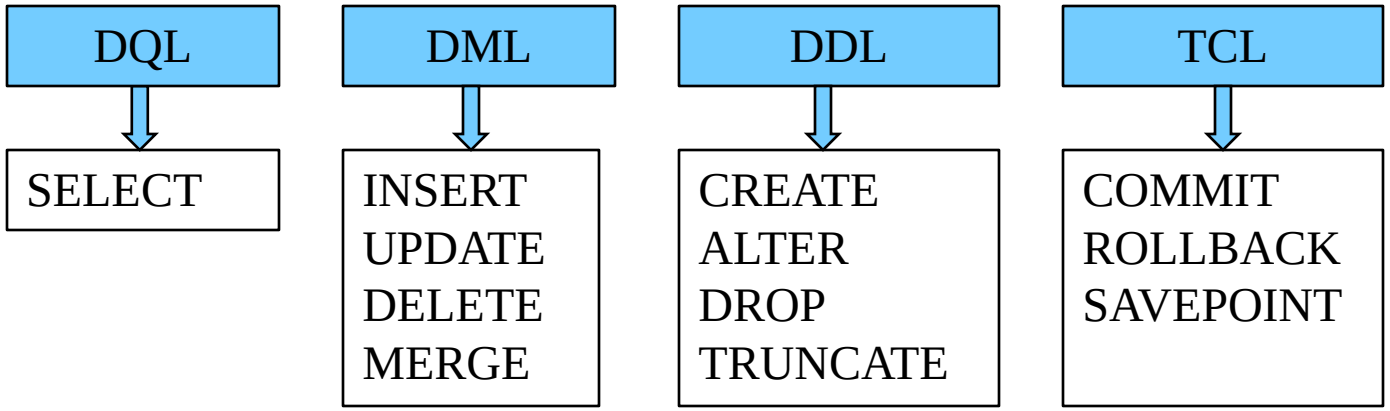
- DDL Statements
  - Help in Defining and Modifying the database structure
  - CREATE TABLE:
    - Creates the Table and also help in providing in the different constraints to maintain the integrity of the data
  - ALTER TABLE:
    - Used to modify the existing data structure which includes adding a new column or constraint, dropping an existing column or constraint, enabling or disabling a constraint,etc…
  - DROP TABLE:
    - Dropping an existing data structure temporarily or permanently and also retrieving the data structure using the FLASHBACK utility of Oracle 10g.
  - TRUNCATE TABLE;
    - Used to remove all the data from the table.
- DML Statements
  - Manipulate the data store in the tables
  - INSERT,UPDATE,DELETE and MERGER

Oracle SQL

- What is SQL?
  - Abbreviated as Structured Query Language.
  - It’s a special purpose programming language to manage the data in a RDBMS.
  - One of the first commercial languages for Relational Model
  - Described as a declarative language(4GL)

Oracle SQL

- Subsets of SQL?
  - Some subsets of SQL are:
    - DQL (Data Query Language) : Querying the Data
    - DML (Data Manipulation Language) : Manipulating the Data
    - DDL (Data Definition Language) : Defining the structure to hold the data
    - TCL (Transaction Control Language) : Controlling the Transactions



Oracle SQL

- Rules for writing SQL Statements
  - Many DBMS systems normally terminate SQL statements with a semi-colon character.
  - Character strings and date values are enclosed in single quotes
  - Character values are case-sensitive and date values are format sensitive.

Oracle SQL

- Getting Connected to Oracle:
  - There are two modes to get connected in Oracle 10g XE
    - Command Line
      - To work with the SQL Command Line go to

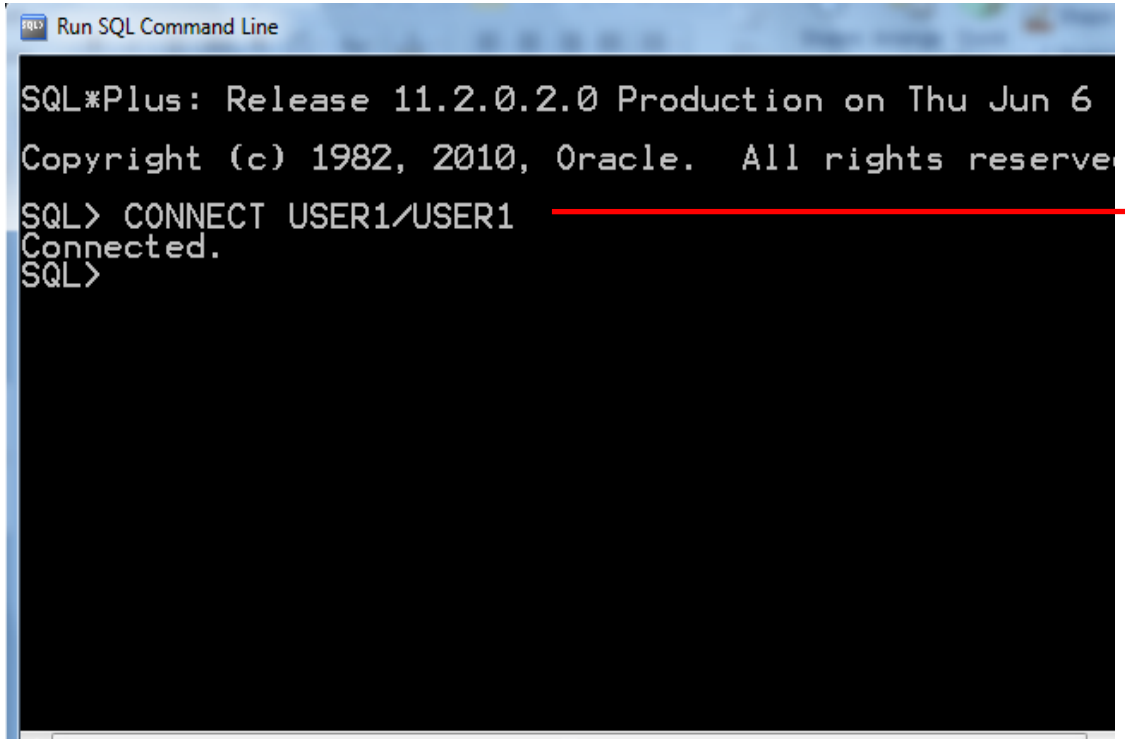
        Start -> Oracle 10g XE -> Run SQL Command Line

    - GU I

      Start -> Oracle 10g XE -> Go to Database HomePage

Oracle SQL

- Writing a Query:
  - Command Line:



To Connect using a username and password the command is:

CONNECT <<username>> / <<password>>

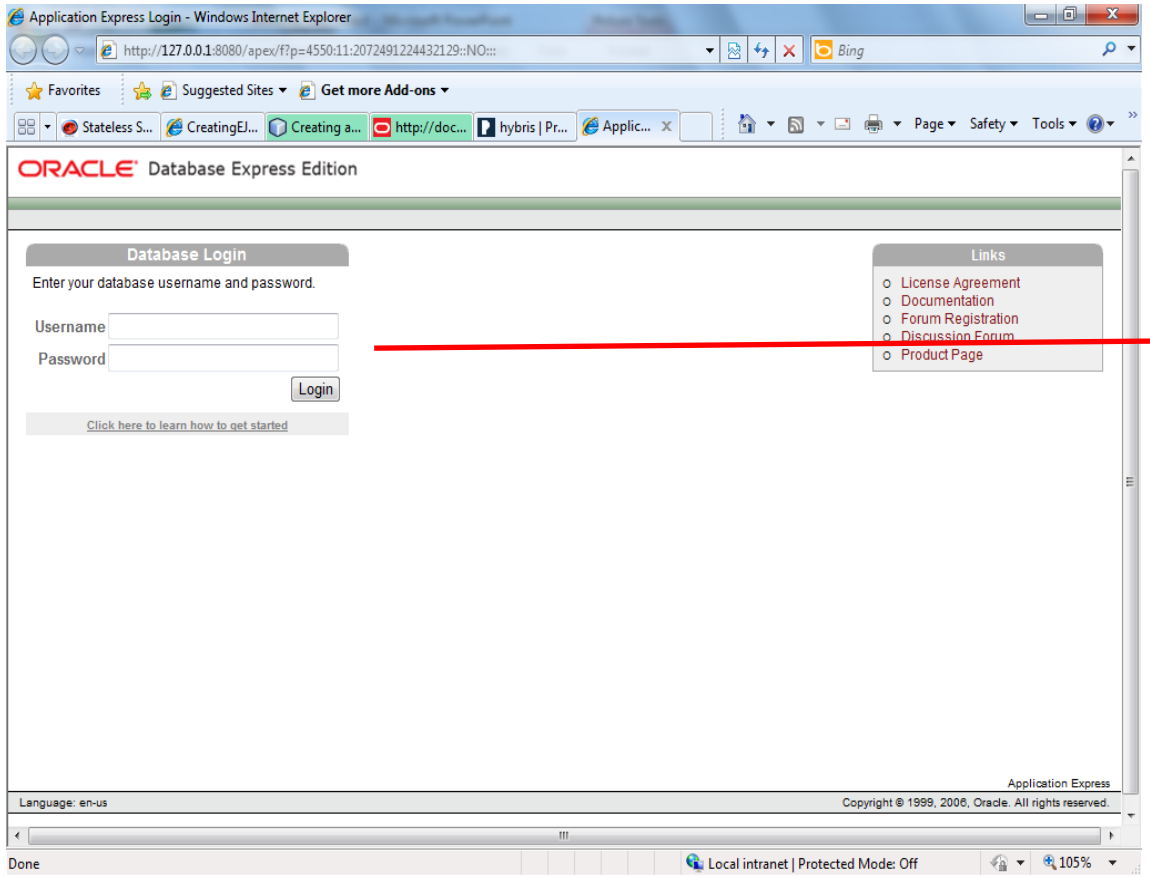
Command Line

- Set of commands to start working with the CLI.

Command	Description
SQL > LIST	Lists the SQL Command in the Buffer
SQL > SAVE <<filename>>	Save the Contents of the Buffer to a file
CLEAR BUFFER	Flushes the buffer
SQL > GET <<filename>>	Load the contents of the file into memory
SQL > @filename or SQL > START <<filename>>	Load the contents of the file into the memory and execute the commands in the file.
SQL > RUN or SQL > /	Executes the commands in the buffer(RUN displays the contents of the buffer before execution)
SQL > SPO[OL] <<filename|OFF|OUT>>	To store/print the queries and the Output.

Oracle SQL

- Writing a Query:
  - GUI Environment:



Specify the username and password

Data Query Language

Data Query Language

- DQL:
  - Data Query Language
  - Used to fetch records from Database.

```
SELECT [DISTINCT ]column_list |*
 FROM table-name
[WHERE Clause]
[GROUP BY clause]
[HAVING clause]
[ORDER BY clause];
```

- DISTINCT : Selecting DISTINCT records from the set
- WHERE : Filtering Records based on some criteria
- GROUP BY : Grouping Records
  - HAVING BY : Applying condition on GROUPS
- ORDER BY : Arranging in Ascending / Descending Order.

DQL - SELECT

- DQL:
  - Fetching Records from the table

```
SELECT *  FROM HR.Employees
SELECT first_name,last_name,department_id
FROM Hr.Employees
```

```
SELECT   DISTINCT job_id FROM HR.Employees
```

  - Expressions in SELECT Statement
    - ` ||’ is the concatenation operator in Oracle

```
SELECT first_name||last_name “Emp Name”
FROM Hr.Employees
                                    OR
SELECT first_name||last_name AS “Emp Name”
FROM Hr.Employees
```

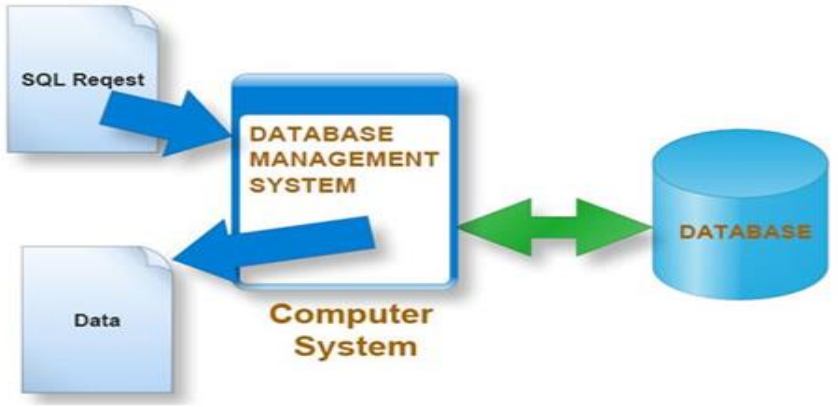
WHERE condition

- DQL:
  - Filtering Information using Where Condition

```
SELECT   * FROM HR.Employees
WHERE Salary  > 5000;
```

- When filtering out String or Date data enclose the data in quotes

```
SELECT   * FROM HR.Employees
WHERE job_id='HR_REP';
```



Logical Conditions & Operators

- DQL:
  - Logical Conditions (Contd..)
    - AND Operator : Returns TRUE if both component conditions are TRUE.

```
SELECT *
FROM HR.Employees
WHERE first_name ='SAMI‘ AND salary=3000;
```

    - OR Operator : Returns TRUE if either component conditions are TRUE.

```
SELECT *
FROM HR.Employees
WHERE first_name = 'SAMI‘  OR salary >= 1000;
```

    - NOT Operator : Returns TRUE if the following condition is FALSE.

```
SELECT *
FROM HR.Employees
WHERE NOT (salary BETWEEN  1000 AND 2000);
```

Logical Conditions & Operators

- DQL:
  - Logical Conditions (Contd..)
    - Order of Precedence of Operators is as follows:

ORDER	OPERATORS
1	Arithmetic + - * /
2	Concatenation ||
3	Comparison <, <=, >, >=, <>
4	IS (NOT) NULL, LIKE, (NOT) IN
5	(NOT) BETWEEN
6	NOT
7	AND
8	OR

SELECT – IN

- DQL:
  - Other Keywords:
    - IN

```
SELECT *
FROM HR.Employees
WHERE deparmentid IN (10,20);
```

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

```
SELECT *
FROM HR.Employees
WHERE first_name NOT IN  ('SCOTT', 'SMITH');
```

SELECT – NULL & LIKE

- DQL:
  - NULL:
    - If a column is empty or no value has been inserted in it then it is called null.
    - NULL doesn't mean 0 (Zero) or blank String " ".

```
SELECT * FROM HR.Employees
WHERE commission_pct IS NULL
```

  - LIKE :
    - Used to search for a pattern
    - Wildcards like % and _ (Underscore) is used to match the pattern.
      - % Matches any number of characters
      - _ (underscore) matches a single character

```
SELECT *  FROM HR.Employees
WHERE job_id LIKE 'HR%'
```

ORDER BY

- DQL:
  - Sorting the Records:ORDER BY
    - Results fetched can be ordered in Ascending / Descending order using the ORDER BY Clause
    - Default order of arranging the Records is Ascending Order.

```
SELECT * FROM HR.Employees
ORDER BY department_id
```

    - Specifying the Descending order can be done by adding the DESC Keyword.

```
SELECT *  FROM HR.Employees
ORDER BY Salary DESC
```

    - More than one ordering criteria can also be enforced:

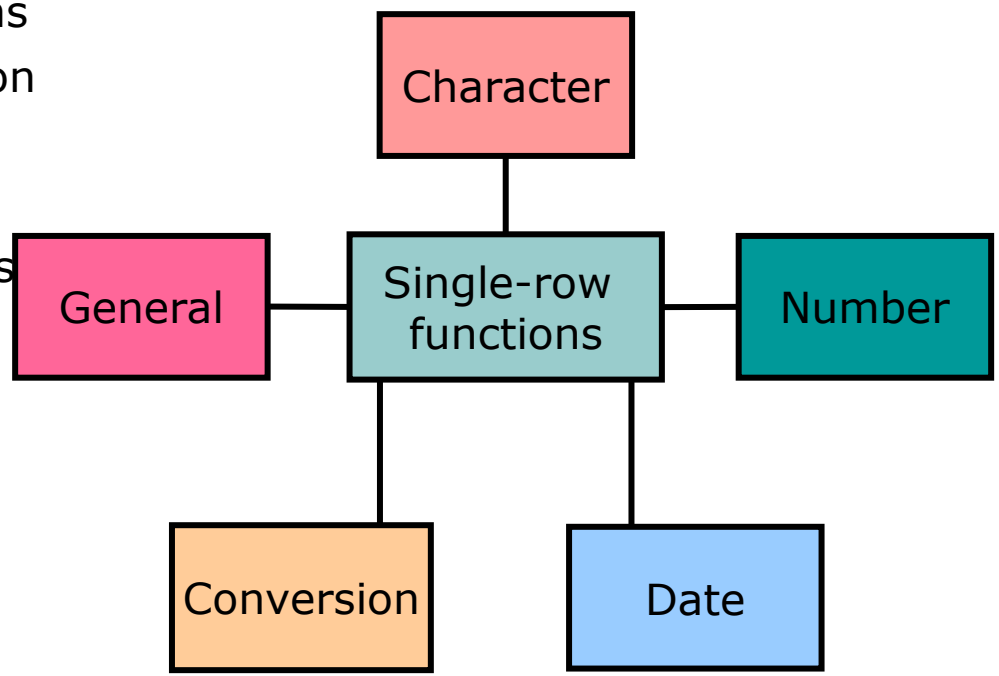
```
SELECT *  FROM HR.Employees
ORDER BY department_id ASC, Salary DESC
```

Single & Multiple Row Funtions

Atos | Syntel

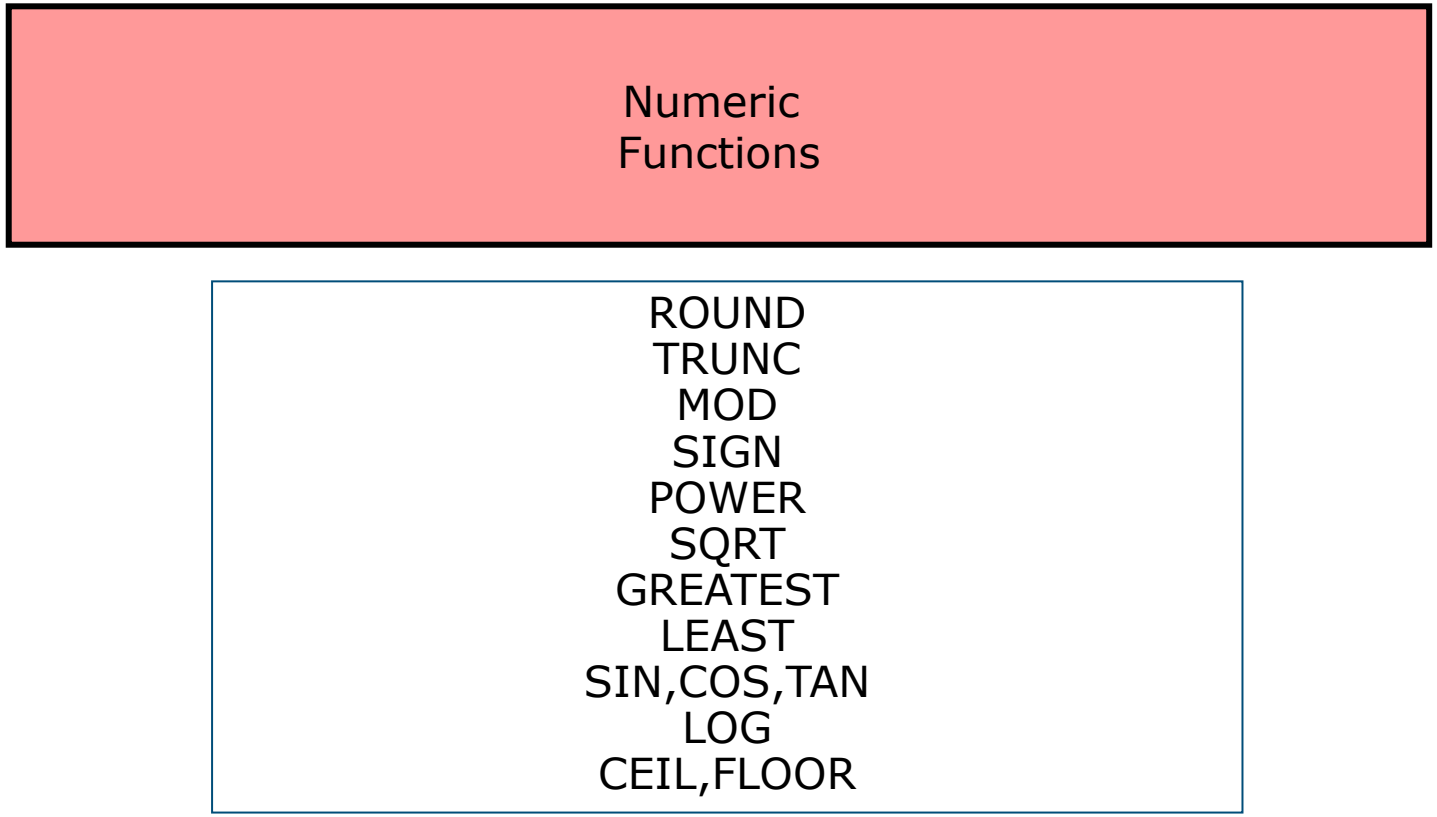
Oracle SQL

- Single Row Functions:
  - Used for performing calculations on data, alter data formats for display, convert data types, etc.
  - Return a single result row for every row of a queried table or view.
  - Can appear in select lists, WHERE clauses, START WITH and CONNECT BY clauses, and HAVING clauses
  - They are categorized as
    - Numeric Functions
    - Character Function
    - Date Functions
    - Conversion
    - General Functions



Oracle SQL

- Single Row Functions:
  - Numeric Function:
    - Some of the numeric functions are :



Single Row Numeric Functions

- Single Row Functions:
  - Numeric Function:

Function	Description
ABS	Returns the absolute of the value
FLOOR	Returns largest integer equal to or less than n.
CEIL	Returns smallest integer equal to or greater than n.
MOD	MOD returns the remainder of n2 divided by n1.
POWER	POWER returns n2 raised to the n1 power
ROUND	ROUND returns n rounded to integer places to the right of the decimal point. If you omit integer, then n is rounded to 0 places. The argument integer can be negative to round off digits left of the decimal point.

Single Row Functions

- Single Row Functions:
  - Numeric Function:

Function	Description
SIGN	SIGN returns the sign of n. This function takes as an argument any numeric datatype, or any nonnumeric datatype that can be implicitly converted to NUMBER, and returns NUMBER. of NUMBER type, the sign is: -1 if n<0, 0 if n=0, 1 if n>0
SQRT	SQRT returns the square root of n.
TRUNC	The TRUNC (number) function returns n1 truncated to n2 decimal places. If n2 is omitted, then n1 is truncated to 0 places. n2 can be negative to truncate (make zero) n2 digits left of the decimal point.

Single Row Functions

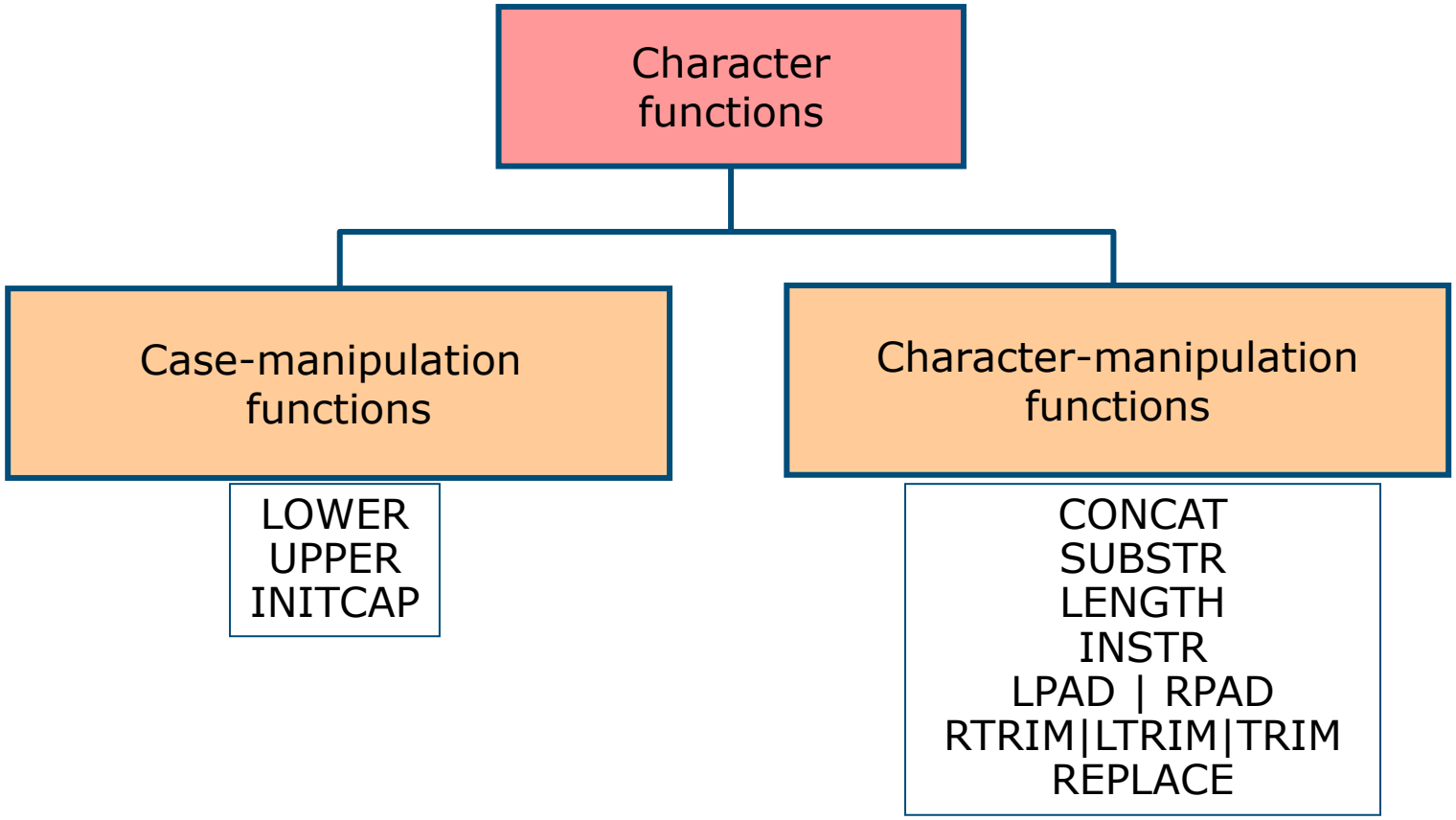
- Single Row Functions:
  - Numeric Functions :
    - Example :

```
SELECT
TRUNC(44.46,1),ROUND(44.46,1),CEIL(44.46),FLOOR(44.46)
FROM Dual;
```

    - Output
      - TRUNC : 44.4
      - ROUND : 44.5
      - CEIL : 45
      - FLOOR : 44

Single Row Character Functions:

- Single Row Functions:
  - Character Function:
    - Some of the character functions are :



Single Row Character Functions:

- Single Row Functions:
  - Character Functions :

Function	Description
CONCAT	Returns the concatenation of 2 strings. You can also use the || command for this
SUBSTR	Returns a part of the String
LENGTH	Returns the length of the String
TRIM|LTRIM|RTRIM	Removes the spaces from the Left or right or both sides of the String.
INSTR	Returns the position of a String within a String.
LPAD|RPAD	Add characters to the left/right of a string until a fixed number is reached

Single Row Character Functions:

- Single Row Functions:
  - Character Functions :

Function	Description
REPLACE	Replaces every occurrence of a search_string with a new string.
REVERSE	Reverses the characters of a String
UPPER|LOWER |INITCAP	Transform a string to all upper case or lower case or sentence case characters

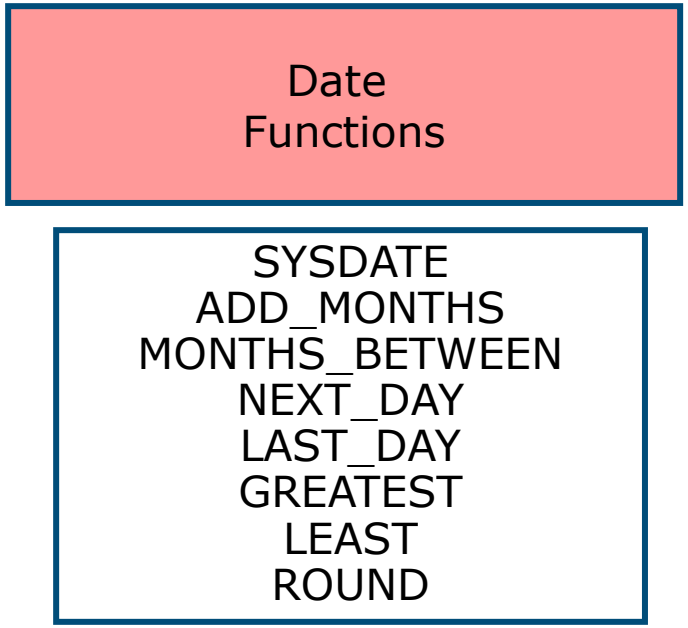
Single Row Character Functions:

- Single Row Functions:
  - Character Functions :
    - Example : Display the data for those employees whose last names end with the character “n”.

```
SELECT employee_id, CONCAT(first_name, last_name) NAME,
LENGTH (last_name), INSTR(last_name, 'a') "Contains 'a'?"
FROM   employees
WHERE  SUBSTR(last_name, -1, 1) = 'n';
```

Single Row Date Functions:

- Single Row Functions:
  - Date Functions:
    - Some of the date functions are :



Single Row Date Functions:

- Single Row Functions:
  - Date Function:

Function	Description
SYSDATE	returns the current date and time in the default Oracle date format. The default format for the date returned is MM-DD-YY
MONTHS_BETWEEN	returns the months between two dates
LAST_DAY	returns the date of the last day of the month that contains date.
ROUND	Rounds of the date to the nearest month or year . Syntax: ROUND(date,'MONTH/YEAR')
GREATEST and LEAST	Returns the GREATEST or LEAST dates out of the given set of dates.

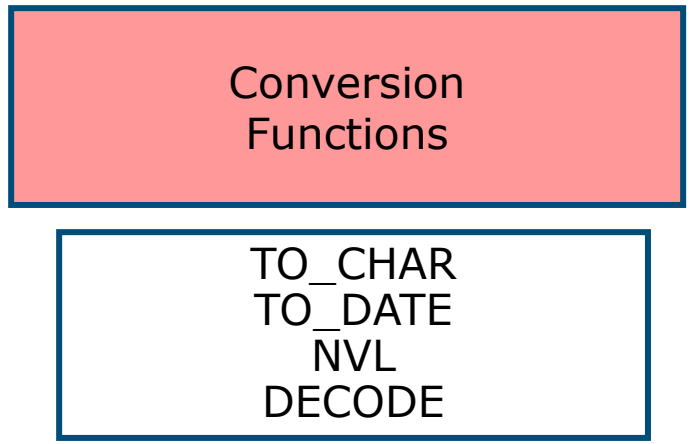
Single Row Date Functions:

- Single Row Functions:
  - Date Functions :
    - Example : Display how many days are left in the current month

```
SELECT SYSDATE,
   LAST_DAY(SYSDATE) "Last",
   LAST_DAY(SYSDATE) - SYSDATE "Days Left“,
   TO_CHAR(NEXT_DAY(sysdate,'MON'),'DD.MM.YYYY') "Next
Monday from now“
FROM DUAL;
```

Single Row Conversion Functions:

- Single Row Functions:
  - Conversion Functions:
    - These are functions that help us to convert a value in one form to another form.
    - Some of the conversion functions are :



Single Row Conversion Functions:

- Single Row Functions:
  - Conversion Function:

Function	Description
TO_CHAR	Converts Numeric and Date values to a character string value. It cannot be used for calculations since it is a string value.
TO_DATE	Converts a valid Numeric and Character values to a Date value. Date is formatted to the format specified by 'date_format'.
TO_NUMBER	Converts a character string containing digits to a number with the optional format model fmt.
NVL(x,y)	If 'x' is NULL, replace it with 'y'. 'x' and 'y' must be of the same datatype.

Single Row Conversion Functions:

- Single Row Functions:
  - Conversion Functions :
    - Example : Display how many days are left in the current month

```
SELECT TO_CHAR(hire_date, 'DD-MON-YYYY')
FROM EMPLOYEES;

SELECT TO_DATE('01-JAN-1985','DD-MON-YYYY')
FROM DUAL;

SELECT first_name, salary, commission_pct,
                    (salary*12)+NVL(commisson_pct,0)
FROM employees;
```

Single Row Conversion Functions:

- Single Row Functions:
  - Conversion Functions :
    - DECODE:
      - Decodes an expression in a way similar to the **IF-THEN-ELSE** logic used in various languages.
      - The DECODE function decodes *expression* after comparing it to each *search* value.
        - » If the expression is the same as ***search, result*** is returned.
        - » If the default value is omitted, a null value is retuned where a search value does not match any of the result values.

```
SELECT   job_id, salary, DECODE(job_id,
                                'ANALYST' , sal*1.1,
                                'CLERK'   , sal*1.15,
                                'MANAGER' , sal*1.20, salary)
Increment
FROM employees ;
```

Recap

- SQL is the stuctured Query Language used to query the data from the database.
  - Consists of some subsets as:
  - DQL - SELECT
  - DDL - CREATE, ALTER, DROP, TRUNCATE
  - DML - INSERT, UPDATE, DELETE
- DQL Queries out the data from the database
- WHERE CLAUSE : Specifies the criteria for selecting the results
- ORDER BY : Sorts the Results fetched
- GROUP BY : Helps in Grouping the Results and HAVING BY filters the grouped results
- Aggregate Functions like COUNT,SUM,MAX,MIN and AVG are mostly used with GROUP BY Clause to enable data analysis.

Recap

- Single- Row Functions
  - Single-row functions can be nested to any level. Single-row functions can manipulate the following
  - Character data:
    - LOWER, UPPER, INITCAP, CONCAT, SUBSTR, INSTR, LENGTH
  - Number data:
    - ROUND, TRUNC, MOD
  - Date data:
    - MONTHS_BETWEEN, ADD_MONTHS, NEXT_DAY, LAST_DAY, ROUND, TRUNC
  - Date values can also use arithmetic operators.
  - Conversion functions can convert character, date, and numeric values
    - TO_CHAR, TO_DATE, TO_NUMBER
  - SYSDATE and DUAL
    - SYSDATE is a date function that returns the current date and time. It is customary to select SYSDATE from a dummy table called DUAL.

Multiple Row Function

- Multiple row function is also called as group function or it is also called as aggregate function.
- Multiple row functions work upon group of rows and return one result for the complete set of rows.
- Group function operate on set of rows to give one result per group
  - These functions are also called “Aggregate functions” (Or) Group functions.
  - All these multi row functions will process multiple records.
  - Applied on group of records and returns one value from the entire group.

Multiple Row Function

- Types of Multi-Row Functions are
  - Maximum(Max)
  - Minimum(MIN)
  - Average(Avg)
  - Sum
  - Count

Multiple Row Function - Maximum(Max)

- MAX ( ) Function in SQL:-
  - Returns the maximum value of a from the given data.
- Syntax :
  - Max(expression)

```
SELECT max(salary) from employee;
```

Output : 87500

- Max() Function on Char
  - Select max (e_name) from employees
  - Output → word {based on Ascii}
- Max() Function on Date
  - Select max(hire_date) from employees

Multiple Row Function - Minimum(Min)

- Min ( ) Function in SQL:-
  - Returns the minimum value of a from the given data.
- Syntax :
  - Min(expression)

```
SELECT min(salary) from employee;
```

Output : 4500

- Min() Function on Char
  - Select min (e_name) from employees
  - Output → word {based on Ascii}
- Min() Function on Date
  - Select min(hire_date) from employees

Multiple Row Function - Average(avg)

- The AVG() is an aggregate function that is used to find out an average of the list of values of columns or an expression.
- The AVG() function is used with the numeric columns only.

```
SELECT AVG(column_name)
FROM table_name
WHERE condition;
```

```
SELECT AVG(salary) AS AVGsal
FROM employee;
```

Output :

AVGsal
21633

Multiple Row Function - SUM

- The SUM() is an aggregate function that is used to return the total sum of a numeric column.
- The SUM() function is used with the numeric columns only.

```
SELECT SUM(column_name)
FROM table_name
WHERE condition;
```

```
SELECT SUM(salary) AS SUMSal
FROM employee;
```

Output :

SUMSal
84750000

Multiple Row Function – COUNT (*)

- COUNT() function returns the number of rows in a table satisfying the criteria specified in the WHERE clause.
- It sets the number of rows or non NULL column values.
- COUNT() returns 0 if there were no matching rows.

```
COUNT(*)
COUNT( [ALL|DISTINCT] expression )
```

```
SELECT COUNT(*)
FROM orders;
Output: No of rows
COUNT(*)
---------
      34
```

Multiple Row Function – COUNT (*)

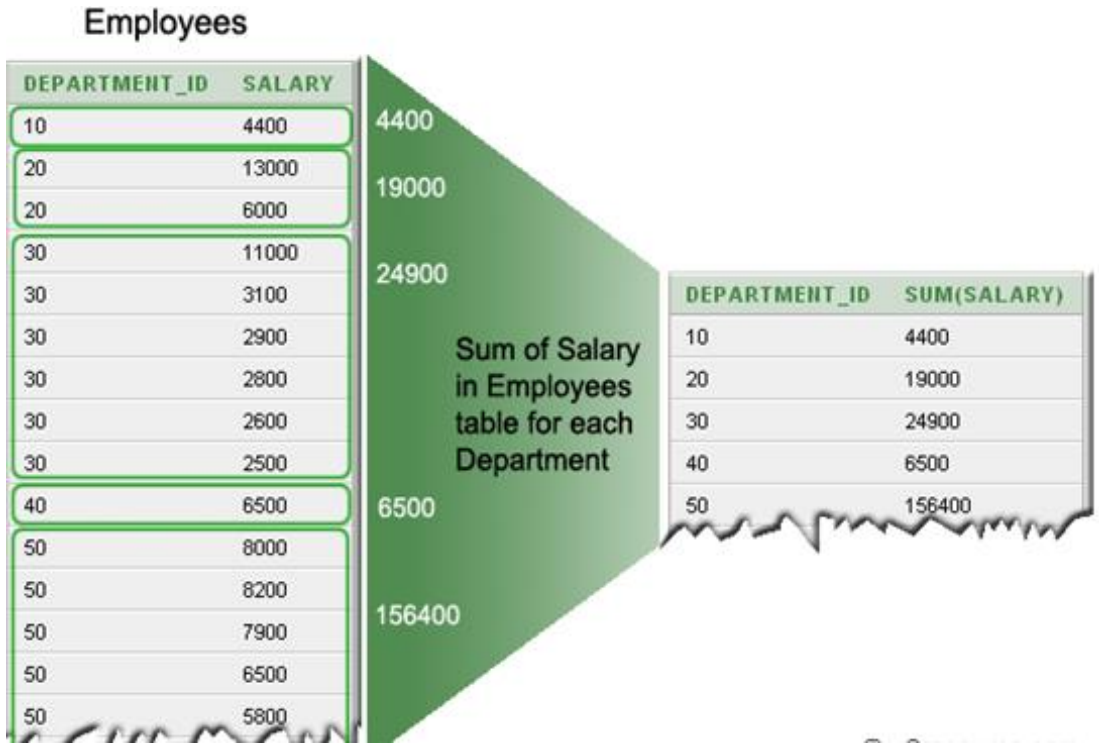
```
SELECT(
     SELECT COUNT(*)
            FROM   employees
            ) AS Total_Employees,
            (SELECT COUNT(*)
            FROM   departments
            ) AS No_Of_Departments
FROM dual
```

Output

```
 TOTAL_EMPLOYEES    NO_OF_DEPARTMENTS
 ---------------    -------------------------
                107                        27
```

GROUP BY

- GROUP BY :
  - Used to divide the rows in a table into smaller groups
  - The grouping can happen after retrieves the rows from a table.
  - The GROUP BY clause is rarely used without an aggregate function.



SELECT – GROUP BY, SUM, AGGREGATE

- GROUP BY : Aggregate Functions
- COUNT()

```
SELECT department_id "Department Code“,COUNT(*) "No of
Employees"
FROM employees
GROUP BY department_id;
```

- SUM()

```
SELECT department_id, SUM(salary)
FROM  employees
GROUP BY  department_id;
```

- Other Aggregate Functions are : MIN,MAX,AVG,…

SELECT – GROUP BY, SUM, AGGREGATE

- GROUP BY : Aggregate Functions
- Grouping more than one column is also permissble

```
SELECT department_id "Department Code", job_id,
SUM(salary) "Total Salary"
FROM  employees
GROUP BY  department_id,job_id;
```

- GROUP BY with HAVING Clause
  - Works with the GROUP BY clause to limit the results to groups that meet the criteria

```
SELECT department_id, count(*) "No. of Employee"
FROM employees
GROUP BY  department_id
HAVING count(*) > 2;
```

GROUPING COLUMNS
SELECT – GROUP BY, SUM, AGGREGATE

- GROUP BY:
- Rules for Grouping columns are:
  - If a select block does have a GROUP BY clause, any column specification specified in the SELECT clause must exclusively occur as a parameter of an aggregated function or in the list of columns given in the GROUP BY clause, or in both.
  - The result of an aggregation function always consists of one value for each group. The result of a column specification on which grouping is performed also always consists of one value per group. These results are compatible. In contrast, the result of a column specification on which no grouping is performed consists of a set of values. This would not be compatible with the results of the other expressions in the SELECT clause.

Rules for Grouping columns

- GROUP BY :
- Although every column included in the SELECT list must also be listed in a GROUP BY clause, this restriction doesn't apply to number and string literals, *constant* expressions (expressions that do not use column values), and functions such as SYSDATE.

```
SELECT COUNT(employee_id), department_id, salary,
          SYSDATE, 'String Literal', 42*37 Expression
FROM Employee
GROUP BY department_id, salary
HAVING (COUNT(employee_id) > 1 OR salary < 100000)
ORDER BY department_id, salary DESC;
```

Rules for Grouping columns

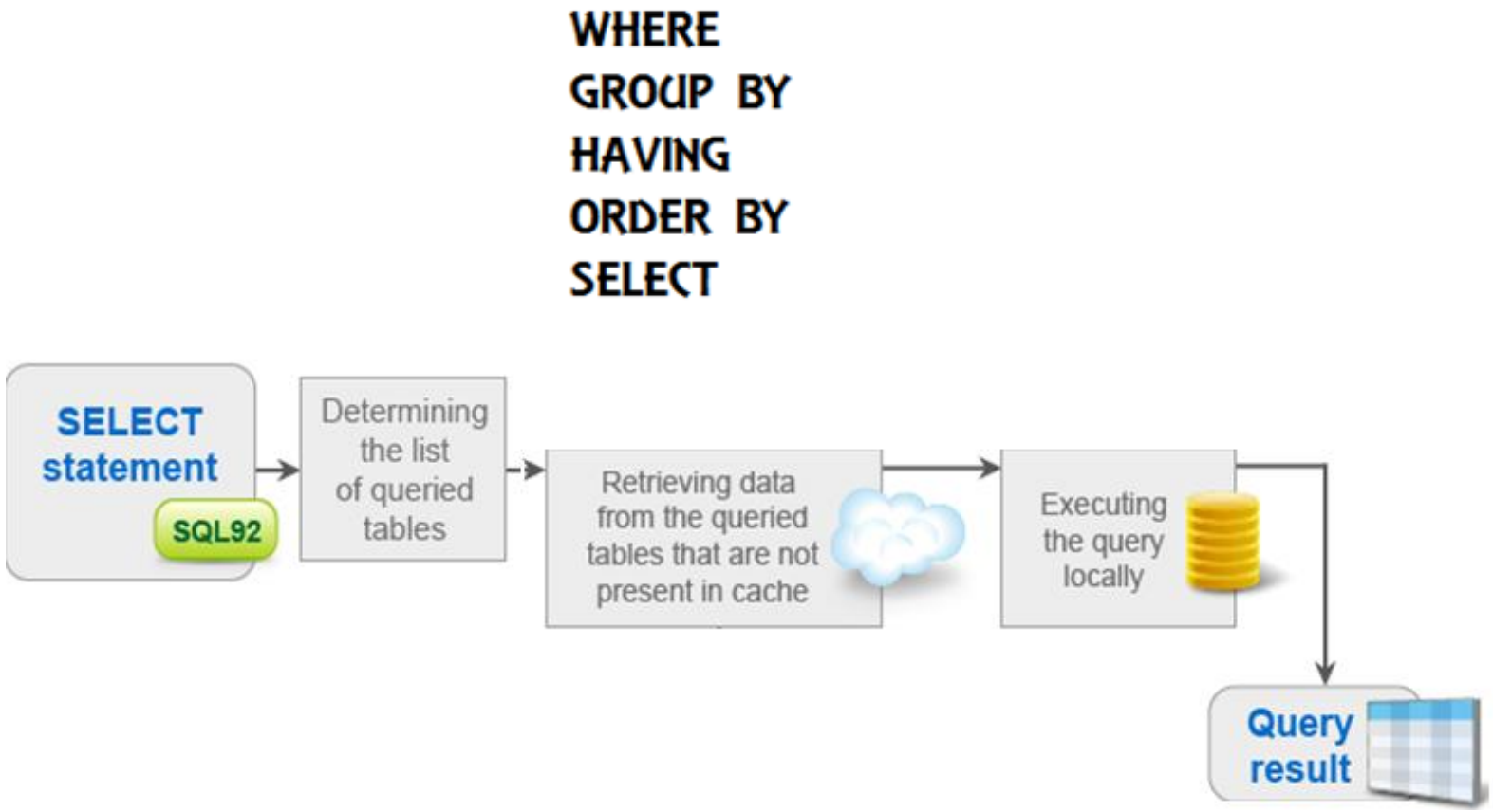
- GROUP BY:
- Rules for Grouping columns are:
  - An expression that is used to form groups can also occur in the SELECT clause within a compound expression.

```
SELECT AMOUNT * 100 AS  AMOUNT_IN_CENTS
FROM PENALTIES
GROUP BY AMOUNT
```

  - For SELECT statements with a GROUP BY clause: DISTINCT (if used outside an aggregation function) that is superfluous when the SELECT clause includes all the columns specified in the GROUP BY clause. The GROUP BY clause groups the rows in such a way that the columns on which they are grouped no longer contain duplicate values.

Execution Order

- Execution Order
  - The Execution order of the various clauses in the SELECT Statement are as follows:



Multiple Row Function – COUNT (*)

- Application of COUNT() function
- COUNT with DISTINCT
  - eliminates the repetitive appearance of the same data. The DISTINCT can come only once in a given select statement.
    - Syntax :
      - COUNT(DISTINCT expr,[expr...])

Multiple Row Function – COUNT (*)

- Application of COUNT() function
- COUNT HAVING
  - can be used to set a condition with the select statement. The HAVING clause is used instead of WHERE clause with SQL COUNT() function.
  - The GROUP BY with HAVING clause retrieves the result for a specific group of a column, which matches the condition specified in the HAVING clause.
    - Example:
      - » To get data of number of agents from the 'agents' table with the following condition -
      - » 1. number of agents must be greater than 3,
      - » the following SQL statement can be used:

```
SELECT COUNT( * ) FROM agents
HAVING COUNT(*)>3;
```

Multiple Row Function – COUNT (*)

- Application of COUNT() function
- COUNT with GROUP BY
  - is useful for characterizing our data under various groupings. A combination of same values (on a column) will be treated as an individual group.

    - Example:
    - To get data of 'working_area' and number of agents for this 'working_area' from the 'agents' table with the following condition

      - 1. 'working_area' should come uniquely,
      - the following SQL statement can be used :

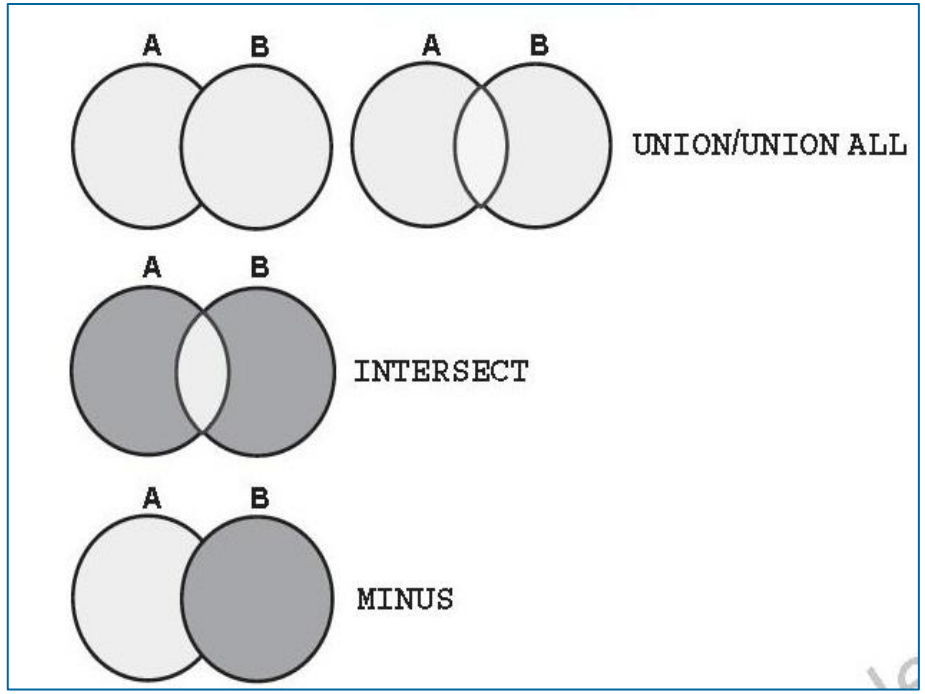
```
SELECT working_area, COUNT(*)
FROM agents
GROUP BY working_area;
```

Set operations in SQL

Atos | Syntel

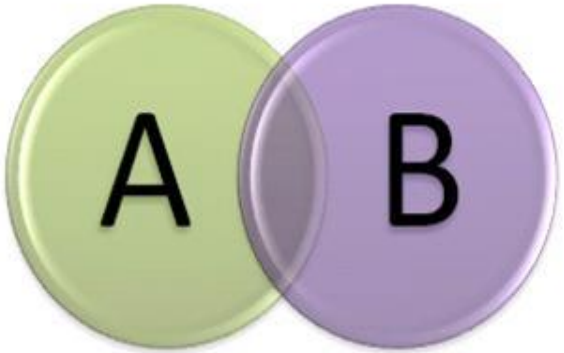
Oracle SQL

- Set Operations:
  - Oracle provides SET operators to combine the result of more than one SELECT statements into one result set in a logical manner.
- Set Operators are:
  - UNION ALL and UNION
  - INTERSECT
  - MINUS



Set Operators – UNION & UNION ALL

- Set Operators:
  - UNION ALL:
    - operator simply unites the result sets of two SELECT queries, without eliminating duplicates from the combined result set.
    - In addition, it doesn't sorts the final result set in any order.



  - UNION :
    - operator simply unites the result sets of two SELECT queries, eliminating duplicates from the combined result set.

Set Operators –UNION ALL

- Set Operators:
  - UNION ALL
    - Example

```
SELECT JOB_ID
FROM EMPLOYEEs
WHERE DEPARTMENT_ID = 10
UNION ALL
SELECT JOB_ID
FROM EMPLOYEEs
WHERE DEPARTMENT_ID = 20
```

    - Ordering of the UNION ALL query result set can be achieved using positional ordering mechanism.

Set Operators –UNION ALL

- Set Operators:
  - UNION ALL
    - Order BY Clause in Set Operation
    - Example :

```
SELECT employee_id, job_id,salary
FROM employees
UNION
SELECT employee_id, job_id,0
FROM job_history
ORDER BY 2;
```

    - If you omit the ORDER BY, then by default the output will be sorted in the ascending order of employee_id. You cannot use the columns from the second query to sort the output.

Set Operators – INTERSECT

- Set Operators:
  - INTERSECT :
    - Operator return all distinct rows selected by the first query, but not present in the second query result set (the first SELECT statement MINUS the second SELECT statement).
    - **Syntax**

```
SELECT expression1, expression2, ... expression_n
FROM tables
[WHERE conditions]
INTERSECT
SELECT expression1, expression2, ... expression_n
FROM tables
[WHERE conditions];
```

Set Operators – INTERSECT

- Set Operators:
  - INTERSECT :
    - Operator return all distinct rows selected by the first query, but not present in the second query result set (the first SELECT statement MINUS the second SELECT statement).
    - Example: Display the employee IDs of those employees who have not changed their jobs even once.

```
SELECT employee_id, job_id, department_id
FROM employees
INTERSECT
SELECT employee_id, job_id, department_id
FROM job_history;
```

Oracle SQL

- Set Operators:
  - MINUS :
    - Operator simply returns all rows that are common to multiple queries.
    - **Syntax**

```
SELECT expression1, expression2, ... expression_n
FROM tables
[WHERE conditions]
MINUS
SELECT expression1, expression2, ... expression_n
FROM tables
[WHERE conditions];
```

Oracle SQL

- Set Operators:
  - MINUS :
    - Operator simply returns all rows that are common to multiple queries.
    - Example: Display the employee IDs and job IDs of those employees who currently have a job title that is the same as their previous one

```
SELECT employee_id
FROM employees
MINUS
SELECT employee_id
FROM job_history;
```

Guidelines while using sub queries

- Set Operations:
  - Some Guidelines while using sub queries are:
    - The expressions in the SELECT lists must match in number.
    - The data type of each column in the second query must match the data type of its corresponding column in the first query.
    - ORDER BY clause can appear only at the very end of the statement.

Joins in SQL

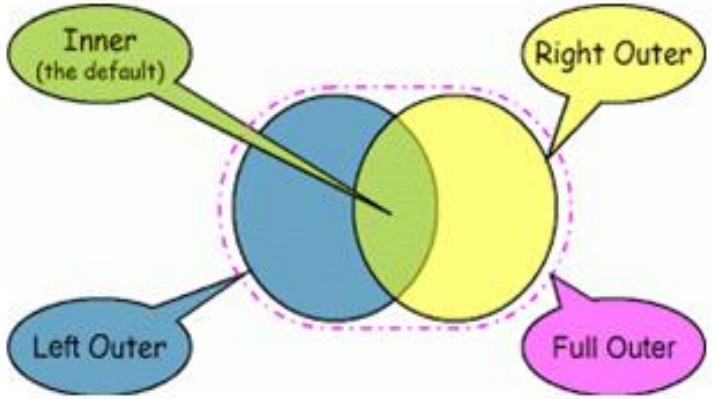
10

Atos | Syntel

Oracle SQL

- Joins:
  - Joins help retrieving data from two or more database tables.
  - The tables are mutually related using primary and foreign keys.
  - A programmer writes a JOIN predicate to identify the records for joining.
    - If the evaluated predicate is true, the combined record is then produced in the expected format, a record set or a temporary table.

- Types of Joins:
  - Inner Join
    - Self Join
  - Outer Join



Why use Joins?

- Why use Joins?
  - A Cartesian product is formed when:
    - A join condition is omitted
    - A join condition is invalid
    - All rows in the first table are joined to all rows in the second table
  - To avoid a Cartesian product, always include a valid join condition in a `WHERE` clause.

R

A	1
B	2
D	3
F	4
E	5

S

A	1
C	2
D	3
E	4

R CROSS S

A	1	A	1
A	1	C	2
A	1	D	3
A	1	E	4
B	2	A	1
B	2	C	2
B	2	D	3
B	2	E	4
D	3	A	1
D	3	C	2
D	3	D	3
D	3	E	4

F	4	A	1
F	4	C	2
F	4	D	3
F	4	E	4
E	5	A	1
E	5	C	2
E	5	D	3
E	5	E	4

JOINS

- Joins:
  - To join the tables:
    - Write the join condition in the `WHERE` clause.
    - Prefix the column name with the table name when the same column name appears in more than one table.
    - Improve performance by using table prefixes.
    - Optionally distinguish columns that have identical names but reside in different tables by using column aliases.
  - Note : To join *n* tables together, you need a minimum of *n*-1 join conditions.
    - For example, to join three tables, a minimum of two joins is required.

Inner Join

- Joins
  - Inner Join:
    - A join in which the joining condition is based on equality between values in the common columns
    - Simplest type of join
    - Also called: Equality join, Equijoin, Natural join

Inner Join

- Joins:
  - Inner Join
    - Example

```
SELECT e.employee_id, e.last_name, e.department_id,
d.location_id
FROM   employees e, departments d
WHERE  e.department_id=d.department_id
```

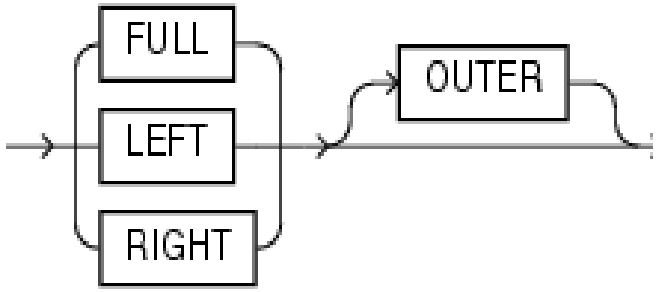
- When you perform an INNER JOIN, only rows that match up are returned. Any time a row from either table doesn't have corresponding values from the other table, it is disregarded.
- The keyword INNER is optional because a JOIN clause will be INNER by default.
- An **equijoin** is a join with a join condition containing an equality operator (=).

Non Equi-Joins

- Joins:
  - Non Equi-Joins
    - Non equi joins is used to return result from two or more tables where exact join is not possible
    - Example

```
SELECT e.last_name, e.salary, j.grade_level
FROM   employees e, job_grades j
WHERE  e.salary BETWEEN
                        j.lowest_sal AND j.highest_sal;
```

Outer Join



- Joins
  - Outer Join:
    - Returns
      - all rows from one table (called inner table) and
      - only matching rows from second table (outer table)
    - Three types:
      - Left
      - Right
      - Full
    - The outer join operator is the plus sign (+).
    - LEFT OUTER JOIN keeps the stray rows from the “left” table (the one listed first in your query statement). In the result set, columns from the other table that have no corresponding data are filled with NULL values.
    - Similarly, the RIGHT OUTER JOIN keeps stray rows from the right table, filling columns from the left table with NULL values.
    - The FULL OUTER JOIN keeps all stray rows as part of the result set.

LEFT Outer-Join

- Joins:
  - Outer-Join
    - Example : LEFT Outer Join

```
SELECT e.last_name, e.department_id,d.department_name
FROM employees e, departments d
WHERE e.department_id(+) = d.department_id;
                                    OR
SELECT e.last_name, e.department_id, d.department_name
FROM employees e
LEFT JOIN departments d
ON e.department_id(+) = d.department_id;
```

RIGHT Outer-Join

- Joins:
  - Outer-Join
    - Example : RIGHT Outer Join

```
SELECT e.last_name, e.department_id, d.department_name
FROM employees e, departments d
WHERE e.department_id = d.department_id(+);
                                    OR
SELECT e.last_name, e.department_id, d.department_name
FROM employees e
RIGHT JOIN departments d
ON e.department_id(+) = d.department_id;
```

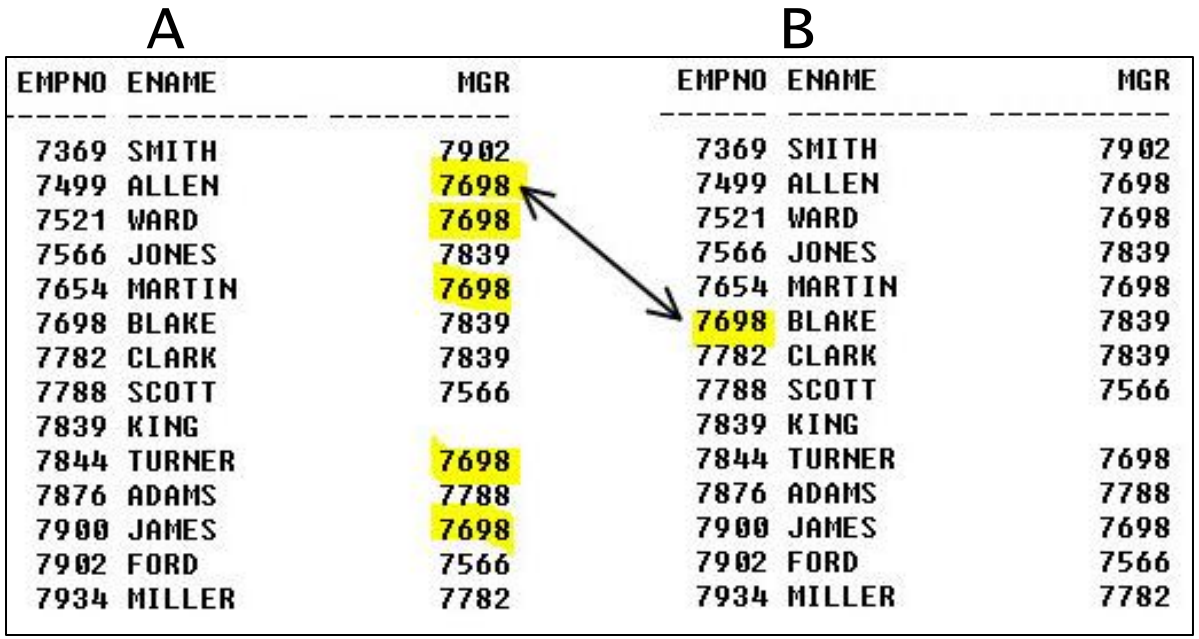
FULL Outer Join

- Joins:
  - Outer-Join
    - Example : FULL Outer Join

```
SELECT e.last_name, e.department_id, d.department_name
FROM employees e
FULL JOIN departments d
ON e.department_id(+) = d.department_id;
```

Self Join

- Joins
  - Self Join:
    - Query that joins 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.
    - Must create table alias
      - Alternate name assigned to table in query's FROM clause

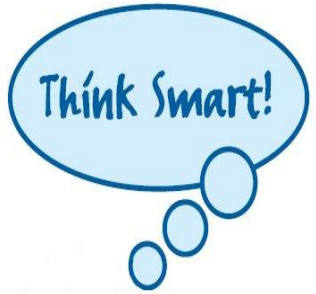


Self Join

- Joins:
  - Self-Join
    - Example

```
SELECT worker.first_name  “FirstName”,
worker.last_name “LastName”,
manager.first_name||manager.last_name
“ManagerName”
FROM   employees worker, employees manager
WHERE  worker.manager_id = manager.employee_id;
```

Suppose that we want to find all employees from employees table and their department_name from dept table. However the problem is some of the employees have a department_id that does not exist in dept table.



So which type of join can help us extract the data?

Answer: Outer Join.

SubQueries in SQL

SubQueries:

- SubQueries:
  - A subquery is a query within another query. The outer query is called as main query and inner query is called as subquery.
  - Subqueries enable you to write queries that select data rows for criteria that are actually developed while the query is executing at *run time*.
  - Using a subquery is equivalent to performing two sequential queries and using the result of the first query as the search value in the second query.
    - The subquery (inner query) executes *before* the main query (outer query).
    - The result of the subquery is used by the main query.
  - The subquery is often referred to as a nested SELECT, sub-SELECT, or inner SELECT statement.

SubQueries:

- SubQueries:
  - A subquery may occur in :
    - SELECT clause
    - FROM clause
    - WHERE clause
  - The subquery can be nested inside a SELECT, INSERT, UPDATE, or DELETE statement or inside another subquery.

SubQuery Types:

- SubQueries:
  - Types of Subqueries:
    - Single row subquery : Returns zero or one row.
    - Multiple row subquery : Returns one or more rows.
    - Multiple column subquery : Returns one or more columns.
    - Correlated subqueries : Reference one or more columns in the outer SQL statement. The subquery is known as a correlated subquery because the subquery is related to the outer SQL statement.
    - Inline Views: An inline view is a query placed inside the FROM clause or inside a WITH clause.

Single row subquery

- SubQueries:
  - Single row subquery :
    - A single row subquery returns all columns for a single row.
    - We can use a single row subquery as the select list of arguments for an INSERT, UPDATE, and DELETE statement.
    - It can be used with the equal comparison operators (=,<,>,<>, etc).
    - Example: Display the employees whose job ID is the same as that of employee 141

```
SELECT last_name, job_id
FROM employees
WHERE job_id =
                    (SELECT job_id
                    FROM employees
                    WHERE employee_id = 141);
```

Single row subquery

- Subquery:
  - Single Row Subquery
    - Example: Display employees who do the same job as "Taylor," but earn more salary than him.

```
SELECT last_name, job_id
FROM employees
WHERE job_id =(SELECT job_id
                 FROM employees
                 WHERE employee_id = 141);
AND salary > (SELECT salary
                FROM employees
                WHERE last_name='Taylor');
```

    - Note: The outer and inner queries can get data from different tables

Single row subquery

- Subquery:
  - Single Row Subquery
    - Example: Group By Functions in a Subquery
      - Display the employee last name, job ID, and salary of all employees whose salary is equal to the minimum salary.

```
SELECT  last_name "Last Name", first_name "First
Name", salary "Salary"
FROM employees
WHERE salary = (SELECT MIN(salary)
                       FROM employees);
```

    - Example : HAVING clause with subqueries
      - Display all the departments that have a minimum salary greater than that of department 50.

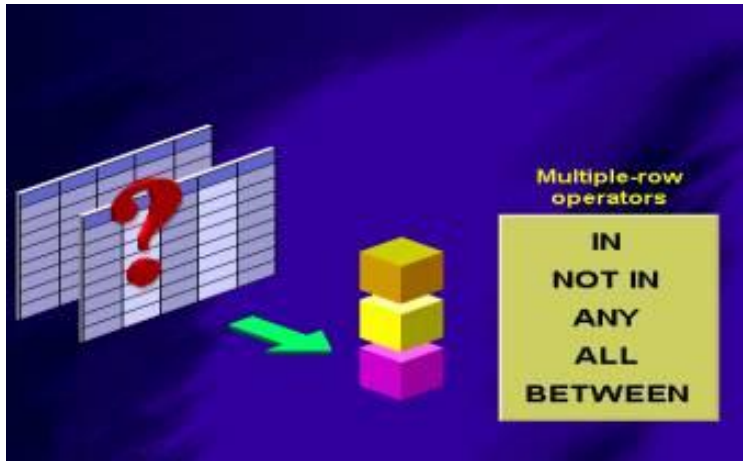
```
SELECT  department_id,MIN(salary)
FROM employees
GROUP BY department_id
HAVING MIN(salary) >= (SELECT MIN(salary)
                       FROM employees
                       WHERE department_id=50);
```

Single row subquery – Common Errors

- Two common errors that occurs when writing sub-queries are:
  - A single-row subquery returns more than one row.
    - If a subquery contains a GROUP BY clause. The GROUP BY clause implies that the subquery returns multiple rows, one value for each group the query encounters.
  - Another error occurs when you write a single-row subquery that returns no rows.

Multiple row subquery

- Subqueries that return more than one row are called multiple-row subqueries.
- You use a multiple-row operator, instead of a single-row operator, with a multiple-row subquery.
- There are five multiple-row comparison operators you can use in a multiple-row subquery to compare a single value to a list of values. You can use IN, NOT IN, ANY, ALL, and BETWEEN.



Multiple row subquery

- SubQueries:
  - Multiple row subquery :
    - Example: Using IN Operator

```
SELECT last_name, salary, department_id
FROM employees
WHERE salary IN (SELECT MIN(salary)
                 FROM employees
                 GROUP BY department_id);
```

    - We can use the IN operator to compare a single value specified in the outer query to any member in a list returned by a subquery.
    - A multiple-row subquery always must be enclosed in parentheses and must appear to the right of a multiple-row comparison operator.

Multiple row subquery

- SubQueries:
  - Multiple row subquery :
    - Example: Using ANY Operator (Display employees who are not IT programmers and whose salary is less than that of any IT programmer.)

```
SELECT employee_id,last_name, job_id,salary
FROM employees
WHERE salary < ALL (SELECT mgr.manager_id
                    FROM employees mgr);
```

    - The ANY operator (and its synonym, the SOME operator) compares a value to ach value returned by a subquery.
    - Here:
      - ANY means less than the maximum.
      - >ANY means more than the minimum.
      - =ANY is equivalent to IN.

Multiple row subquery

- SubQueries:
  - Multiple row subquery :
    - Example: Using ALL Operator (Display employees whose salary is less than the salary of all employees with a job ID of IT_PROG and whose job is not IT_PROG.)

```
SELECT emp.last_name
FROM employees emp
WHERE emp.employee_id < ALL (SELECT salary
                             FROM employees
                             WHERE job_id='IT_PROG');
```

    - The ALL operator compares a value to every value returned by a sub query.
    - Here :
      - ALL means more than the maximum and
      - <ALL means less than the minimum.

Multiple row subquery

- SubQueries:
  - Multiple row subquery :
    - Example: Using NOT Operator
    - The NOT operator can be used with IN, ANY, and ALL operators.
      - Display all employees who do not have any subordinates:

```
SELECT last_name FROM employees
WHERE employee_id NOT IN (SELECT manager_id
                          FROM employees
                          WHERE manager_id IS NOT NULL);
```

    - **Note:** If one of the values returned by the inner query is a null value, the entire query returns no rows.
      - The reason is that all conditions that compare a null value result in a null.
      - So whenever null values are likely to be part of the results set of a subquery, do not use the NOT IN operator. The NOT IN operator is equivalent to <> ALL.

Multiple column subquery

- There are times when you need to use a subquery that compares more than just one column between the parent query and the subquery. This is known as a multiple-column subquery.
- the IN clause is used to compare the outer query's columns to the columns of the subquery.
- A subquery in which more than one column is selected for comparison to the main query using the same number of columns.
- Example:
  - If the columns do not match in number, then an error message is displayed.
  - If the data types do not match, then the results of the query are incorrect.

```
SELECT employee_id, last_name, job_id, salary
FROM employees
WHERE (job_id, salary) IN (select job_id, salary
                           FROM employees
                           WHERE employee_id = 136);
```

Multiple column subquery

- SubQueries:
  - Multiple Column subquery :
    - Example: Display only those employees who have worked the longest time in each job title.

```
SELECT first_name, job_id, hiredate
From employees
WHERE (job_id,hiredate) IN
                    (SELECT job_id,MIN(hiredate_date)
                       FROM employees
                       GROUP BY job_id);
```

Multiple column subquery

- SubQueries:
  - Multiple Column subquery :
    - Example: Display details about employees whose salary and commission match the salary and commission of any employee in Department 30.

```
SELECT first_name, department_id, salary, commission_pct
FROM employees
WHERE (salary, NVL(commission_pct,-1)) IN
                    (SELECT salary, NVL(commission_pct,-1)
                    FROM employees
                    WHERE department_id = 30);
```

Multiple column subquery

- Queries in FROM Clause
  - We can create a logical subset of data by writing a subquery in the FROM clause of a SELECT statement.
  - These subsets are virtual tables that we can use to return specific data to the main query.
  - Use a table alias to identify the result set of the subquery
- Example: Display the names, salaries, department numbers, and average salaries for all employees who earn more than the average salaries in their departments.

```
SELECT a.first_name, a.salary, a.department_id, b.salavg
  FROM employees a,
            (SELECT department_id, AVG(sal) salavg
            FROM employees
             GROUP BY department_id ) b
  WHERE a. department_id =b. department_id
  AND a.salary > b.salavg
```

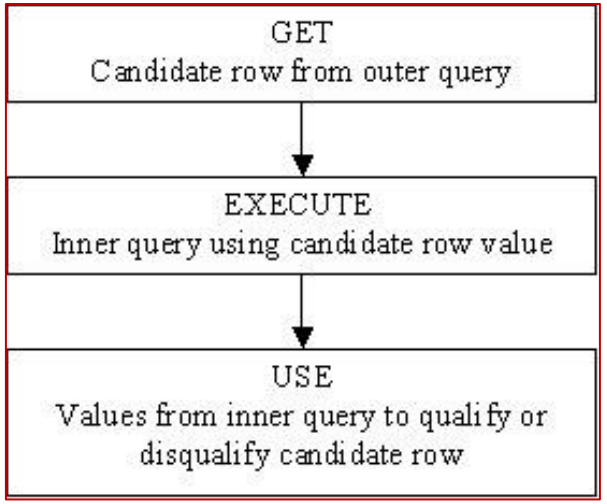
Multiple column subquery

- SubQueries:
  - Multiple Column subquery :
    - Queries in FROM Clause
    - Example: Display details for employees who earn the lowest salary in each job title.

```
SELECT e.job_id, e.first_name, e.salary, sub.low_sal
FROM employees e,
          (SELECT job_id, MIN(salary) low_sal
          FROM employees
          GROUP BY job_id) sub
WHERE e.job_id = sub.job_id
AND e.salary = sub.low_sal;
```

Correlated Subqueries

- Correlated Subqueries:
  - A subquery that contains a reference to a column in the main, or parent, query and is evaluated once for each row of the outer query.
  - A correlated subquery looks very much like other subquery, with one important difference:
    - The correlated subquery references a column in the main query as part of the qualification process to see if a given row will be returned by the query.
    - For each row in the parent query, the subquery is evaluated to see if the row will be returned.
  - It is used for row-by-row processing. Each subquery is executed once for every row of the outer query.



Correlated Subqueries

- SubQueries:
  - Correlated Sub Query :
    - Example: Display employees who earn more than the average for their department across all departments.

```
SELECT employee_id, last_name, department_id, salary
FROM  employees emp
WHERE salary > (SELECT avg(salary)
                FROM employees
                WHERE department_id =
emp.department_id);
```

Correlated Subqueries

- SubQueries:
  - Correlated Sub Query :
    - Example: Display details of those employees who have switched jobs at least twice.

```
SELECT e.employee_id, e.last_name, e.job_id
FROM employees e
WHERE 2 <=
          ( SELECT COUNT(*)
          FROM job_history
          WHERE employee_id = e.employee_id);
```

In-line Views

- The inline view is a construct in Oracle SQL where you can place a query in the SQL FROM, clause, just as if the query was a table name.
- It allows you to use a filtered data set not found in a table like a view, but only in the scope of the query or SQL statement.
- A common use for in-line views in Oracle SQL is
  - to simplify complex queries by removing join operations and
  - condensing several separate queries into a single query.
- Example: Display the employees who earn the highest salary in each department.

```
SELECT a.last_name, a.salary, a.department_id, b.maxsal
FROM employees a,
          ( SELECT department_id, max(salary) maxsal
          FROM employees
          GROUP BY department_id ) b
WHERE a.department_id = b.department_id
AND a.salary = b.maxsal;
```

Sub Queries - Guidelines

- A subquery must be enclosed in parentheses.
- A subquery must be placed on the right side of the comparison operator.
- Sub queries cannot manipulate their results internally, therefore ORDER BY clause cannot be added in to a subquery.
  - However we can use a ORDER BY clause in the main SELECT statement (outer query) which will be last clause.
- Use single-row operators with single-row sub queries.
- If a subquery (inner query) returns a null value to the outer query, the outer query will not return any rows when using certain comparison operators in a WHERE clause.

Recap

- Set Operations:
  - Used to combine the results of more than one Query
  - Different types of set Opearations supported are:
    - UNION, UNION ALL, INTERSECT and MINUS
- Joins:
  - Used to fetch the results from more than one table
  - Different Joins supported are :
    - INNER JOIN,OUTER JOIN and SELF Join
- SubQueries:
  - Used to write a Query within another Query

Thank You

Atos | Syntel