SQL for Oracle 8/8i

SQL Basics

Course Objectives

- **Overview of SQL**
- **Concepts and syntax for PL/SQL**
- **Introduction to Object Oriented Approach**
- **Understanding sqlplus editor settings**

Prerequisites for the course

- **Little bit knowledge about database will be helpful**
- **Knowledge on Object-Oriented Programming**

Session Plan

Day1

- SQL Overview
- Data Types
- DDL, DML and DCL
- Operators
- Functions

Day2

- Object relational features
- Collections (Varrays, Nested tables)
- Database Objects (Triggers, Stored procedures)
- System tables
- SQL editor
- Case Study

DAY1: SQL Basics

- **SQL - An Overview**
- **Datatypes**
- **Tables**
- **Integrity Constraints**
- **Synonyms**
- **Indexes**
- **Database Sequences**
- **Data Manipulation language**
- **Data Control Language**
- **Arithmetic Operators**
- **Concatenation operator**
- **Comparison Operators**

DAY1: SQL Basics (continued..)

- **Set Operators**
- **User Defined Operators**
- **Arithmetic Functions**
- **Character Functions**
- **Date Functions**
- **Aggregate Functions**
- **Joins**
- **Subquery**
- **Views**

SQL - An Overview

The original version was developed by San Jose Research Labs.

SQL is an ANSI (American National Standards Institute) standard for accessing database systems

It has several parts along with Object Relational Features:

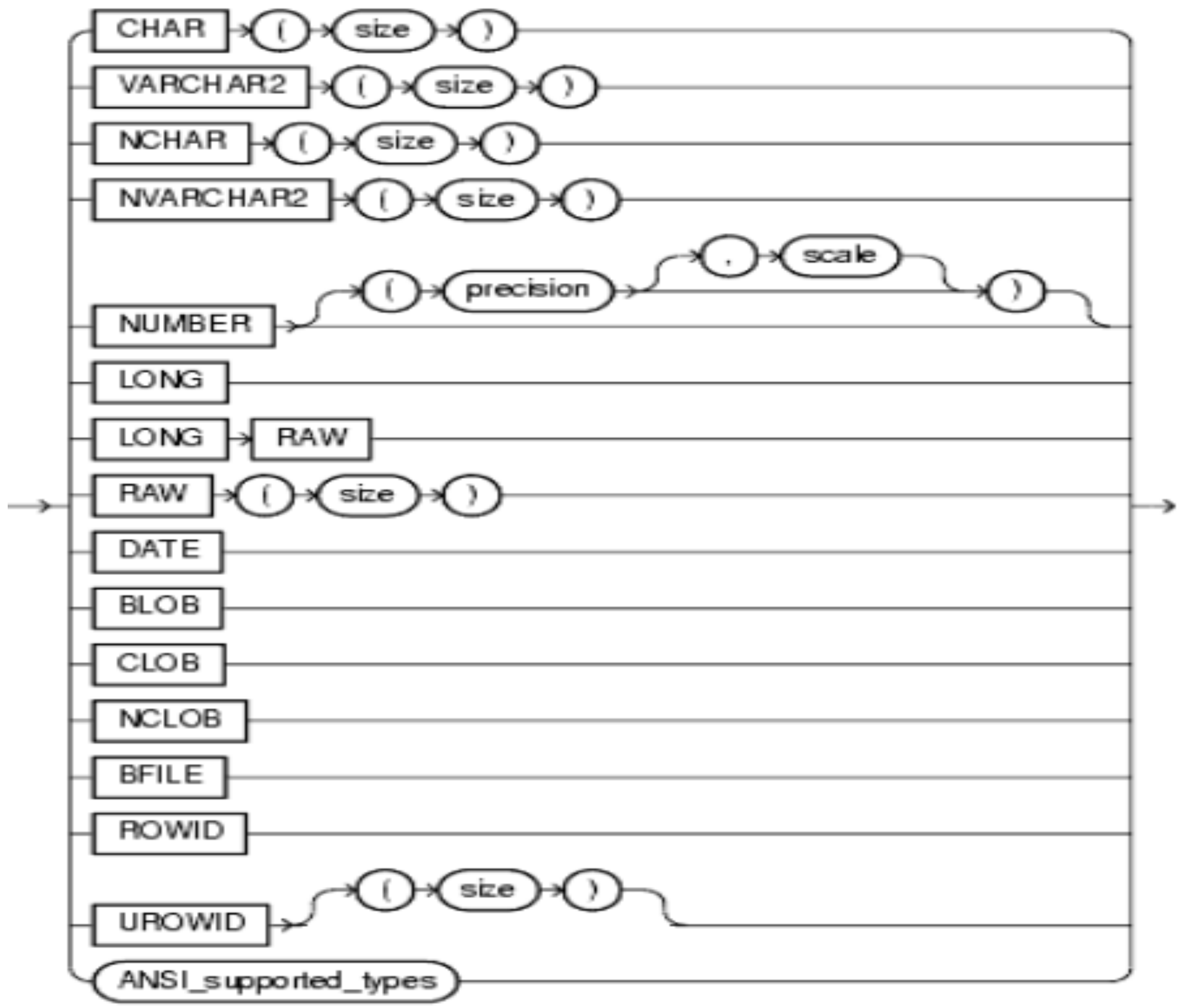
- **Data Definition Language**
- **Data Query Language**
- **Data Manipulation Language**
- **Data Control Language**
- **Transaction Control Language**
- **Introduction To Database Objects**

RDBMS vis a vis SQL

RDBMS Features	<u>How SQL Supports them</u>
Entity / Entity Sets	Schemas/Tables
Attributes	Columns
Relationships	Primary Key/Foreign Key
Integrity Constraints	Entity integrity/ Ref. Integrity/Domain Constraints
Fundamental D’base Operations	SQL Statements

- Selection
- Projection
- Cartesian Product
- Rename
- Union
- Difference
- Join

Datatypes


CHAR
(
size
)
VARCHAR2
(
size
)
NCHAR
(
size
)
NVARCHAR2
(
size
)
NUMBER
(
precision
,
scale
)
LONG
LONG
RAW
RAW
(
size
)
DATE
BLOB
CLOB
NCLOB
BFILE
ROWID
UROWID
(
size
)
ANSI_supported_types

Datatypes

Datatype	Description
NUMBER	Numeric data having precision p and scale s. The precision p can range from 1 to 38. The scale s can range from -84 to 127.
DATE	Valid date range from January 1, 4712 BC to December 31, 9999 AD. A date value without a time component sets the default time to 12:00 AM midnight.Default format is ‘DD-MON-YY’
CHAR	Fixed-length character data of length size bytes. Maximum size is 2000 bytes. Default and minimum size is 1 byte

Datatypes

Datatype	Description
VARCHAR2	Variable-length character string having maximum length size bytes. Maximum size is 4000, and minimum is 1. SIZE must be specified for VARCHAR2.
BLOB	Binary LOB; binary data,up to 4GB in length, stored in the database
CLOB	Character LOB; character data, up to 4GB in length, stored in the database

Datatypes

Datatype	**Description**
BFILE	Binary File; contains a locator to a large binary read only file stored outside the database. Enables byte stream I/O access to external LOBs residing on the database server, the length of which is limited by the operating system
NCLOB	A CLOB column that supports a multibyte character set
RAW	A column can contain data in any form, including binary

Datatypes

A LONG data type column can contain any printable character and can be up to 2 Gigabytes in size.

- **Conditions**
  - **Used with: `SELECT SET UPDATE VALUES`**
  - **`LONG cannot appear in:WHERE,GROUP BY,CONNECT BY,DISTINCT in SELECT,SQL Funcs,Expressions, Conditions`**

Data Definition Language

Creating Databases/ Schemas

Creating Users

Creating Tables / Synonyms/ Views / Materialized Views /Snapshots

Creating Indexes/ Sequences/Triggers

Creating Libraries/ Packages/ Procedures/ Functions

Creating Types

Tables

```
CREATE TABLE customers
    (cust_id NUMBER(5),
    cust_name VARCHAR2(20),
    cust_phone VARCHAR2(10));
```

The table name is ‘customers’

Three columns named ‘cust_id, cust_name, cust_phone’

Data types and length of columns have been defined

If we enter the above table creation script in SQL*Plus then the table will be created in our 'Default tablespace'.

Tables

This command can be used to change the format of an existing table.

- ALTER TABLE table_name MODIFY column_definition
- ALTER TABLE table_name ENABLE/DISABLE CONSTRAINT constraint_name
- ALTER TABLE table_name ADD column_definitions
- ALTER TABLE table_name DROP COLUMN column_definition

Marking columns as unused

- ALTER TABLE table_name SET UNUSED COLUMN column_definition

Tables

```
ALTER TABLE students
    ADD (date_of_birth DATE NULL);

ALTER TABLE students
    ADD (gender VARCHAR2(1) NULL);

ALTER TABLE students
    MODIFY (gender NOT NULL);
```

Tables

The DESC command displays the structure of the table created

```
DESC students;
Name            Null?               Type
------------------------------- -------- ----
NAME                                CHAR(20)
GENDER          Not Null            CHAR(20)
```

The DROP command drops the database object permanently from the Database.

```
DROP TABLE table_name ;
```

Data Definition Language

DDL statements also allow you to do the following:

- Change the names of schema objects.

  Ex: - rename <table_name> to <table_name1>

- Gather statistics about schema objects, validate object structure, and list chained rows within objects.

  Ex: dbms_stats.gather_schema_stats(OWNNAME=>,OPTIONS=>'GATHER AUTO',
  estimate_percent=>DBMS_STATS.AUTO_SAMPLE_SIZE);

  dbms_stats.gather_table_stats(’Table Owner',’Table Name')

- Turn auditing options on and off (AUDIT, NOAUDIT).

- Add a comment to the data dictionary.

  Ex: COMMENT ON COLUMN employees.job_id IS 'abbreviated job title'

Exercise

1. Create a table “Player” with the columns player_id, country_id, player_name, nick_name, DOB, active_player, start_date, end_date and specialization.
2. Create a table “match_details” with the columns match_id, match_date, host_team,against_team, venue_played, overs_played, win_country_id and result(W-win, T- Tie, D - drawn,A-Abandoned)

   Note- The host_team and the against_team should be number and have a reference to country id
3. Create a table “Country” with the columns country_id, country_name

Integrity Constraints

Constraints define conditions which Oracle uses to maintain Data Integrity.

Constraints

- reduce effort in maintaining data in applications
- slow down data insertion and updates.

Constraints may be defined at the

- Columns level as part of the column definition known as ‘column level’ constraint and/or
- Table level as part of the CREATE TABLE statement at the end known as ‘table level’ constraint
- Clauses that constrain multiple columns have to be table level constraints.

Integrity Constraints

Domain Integrity Constraints

- NULL / NOT NULL
- CHECK

Entity Integrity Constraints

- UNIQUE
- PRIMARY KEY

Referential Integrity Constraints

- FOREIGN KEY

Integrity Constraints

Null / Not Null

- CREATE TABLE customers
  (cust_id NUMBER(5) NOT NULL,
  cust_name VARCHAR2(20) NOT NULL,
  cust_phone VARCHAR2(10) NULL);
- The NOT NULL in the create table statement is what is called a 'constraint'. The column must contain data or Oracle will not allow the row to be entered or updated.
- These are column level constraints
- Columns with a NOT NULL constraint cannot hold NULL values.

Integrity Constraints

Check

- This constraint checks to see if incoming data meets certain criteria defined in the constraint.
- A column level CHECK cannot reference other columns and cannot use pseudo-columns such as SYSDATE,UID,CurrVal,NextVal,Level or Rownum.
- Multiple columns may be referred by Table level CHECK constraints.
- CREATE TABLE table_name(

```
C1 NUMBER CHECK( C1 BETWEEN 10 AND 50),
C2 VARCHAR2(10),
C3 NUMBER,
CONSTRAINT my_primary_key PRIMARY KEY(C3),
FOREIGN KEY C2 REFERENCES
some_other_table(column_name));
```

Integrity Constraints

Unique

- An candidate key is a combination of one or more columns which uniquely identify every row of the table.
- Candidate keys act as UNIQUE constraints.
- Unique key doesn’t allow duplicates and may be NULL
- A table can have more than one unique keys
- A key comprises of more than one column known as composite key
- CREATE TABLE table_name(

```
        Column1 VARCHAR2(10) UNIQUE,
        ….,
        UNIQUE(column2,column3…)
        );
```

Integrity Constraints

Primary Key

- A primary key is also a candidate key with a few special characteristics
  - ➢ Each table can have exactly one primary key.
  - ➢ A primary key column cannot contain NULL values.
- CREATE TABLE table_name(
  C1 VARCHAR2(10) PRIMARY KEY,
  C2 NUMBER,...);
- CREATE TABLE table_name(
  C1 VARCHAR2(10),
  C2 NUMBER,...
  PRIMARY KEY(C1,C2));

Integrity Constraints

Foreign Key

- This is also known as Referential Integrity Constraint
- This is a combination of columns whose values are based on the primary key values of another table.
- The referential integrity constraint can be defined on the same table as well.
- CREATE TABLE table_name(

  ```
  C1 VARCHAR2(10) PRIMARY KEY,
  C2 NUMBER REFERENCES
  some_other_table(column_name),
  C3  NUMBER(10));
  ```

- The ON DELETE CASCADE clause tells Oracle to delete dependent rows when the parent row is deleted

Integrity Constraints

Naming Constraints

- Constraints can have either
  - Oracle generated names of the type SYS_C###### or
  - User defined names
- Constraint Names can be defined at the time of table creation.
- These names can be used to enable or disable constraints.
- CREATE TABLE table_name(

```
    C1 VARCHAR2(10),
    C2 NUMBER CONSTRAINT uni_constraint UNIQUE,
    C3 NUMBER,
    CONSTRAINT my_constraint PRIMARY KEY(C3)
  );
```

Exercise

1. Create a table “score_details” with the columns match_id, player_id, runs_scored, sixes, boundaries, catches, stumped, run_outs, batted and balls_played.Define the match_id and player_id as a composite primary key. Define the match_id as foreign key pointing to “match_details” table and player_id as foreign key to player_id of “player” table.
2. Create a table “bowling” with the columns match_id, player_id, overs_bowled, runs_given, wickets_taken, extras_bowled. Define the match_id and player_id as a composite primary key. Define the match_id as foreign key pointing to “match_details” table and player_id as foreign key to player_id of “player” table.
3. Alter the “Country” table to make the country_id as primary key and country_name as not null column.
4. Alter the table “player” to have a check constraint on the column specialization to hold only the values “BM” (batsman), “BO” (bowler), “AR” (all rounder), “WK”(wicket keeper), “C” (captain)
5. Alter the match_details table to make the match_id as primary key.
6. Alter the “match_details” table make the country_id as a reference to country table

Synonyms

A synonym provides an alias for a table.

Types of synonyms

- Private
- Public

Synonyms play an important role in distributed databases

Public Synonyms

Public synonyms can only be created by the database administrator.

```
CREATE [PUBLIC] SYNONYM [schema.]synonym
FOR [schema.]object[@dblink]
Eg.    Create synonym inv_298  for stock ;
```

Aliases and Synonyms

Alias - An alias is another way of referring to the same database table/column within a sql script

- An alias is not a database object

-
  ```
  SELECT    order.cust_id cust, item.qty quantity
  FROM      orders order, order_items item
  WHERE     order.order_id = item.order_id;
  ```
- Table/Alias :   orders/order, order_item/item
- Column/Alias: order.cust_id/cust, item.qty/quantity

Synonyms - These are database objects which provides another,more secure way of referring to database tables.

- CREATE SYNONYM ord FOR orders;
- Synonym/Table: ord/orders

Indexes

Indexes are database structures that enable faster data access.

Indexes are used in an SQL database for two primary reasons

- To facilitate the ordering of data based on the contents of the index's field or fields
- To optimize the execution speed of queries

Other types of Indexes

- Descending Indexes
- Bit-Mapped Indexes
- Function Indexes
- Reverse-key Indexes

Indexes

Creating an Index

```
CREATE INDEX index_name
ON table_name(column_name1, [column_name2], ...) TABLESPACE <tbsname>;
```

Example:

```
CREATE INDEX  I_empno ON employee(empno);
CREATE  INDEX  I_eno_nm ON employee(empno , ename);
```

SQL*Plus takes advantage of indexes only when you use an expression involving an indexed column directly i.e., in the ‘WHERE’ clause.

Indexes

Unique Index

- Indexes can guarantee that a column of a table contains unique values. To do this we create a unique index.

Example:

- CREATE UNIQUE INDEX I_empno ON employee(empno);

Creating an index on columns that are frequently used in joins speeds up the operation.

Indexes

Function Indexes

- Facilitate queries that qualify a value returned by a function or expression. The value of the function or expression is pre-computed and stored in the index.

  Ex: - CREATE INDEX area_index ON rivers (area(geo));

- It works only after 8.1.0 or higher versions
- The table must be analyzed after the index is created.
- The query must be guaranteed not to need any NULL values from the indexed expression, since NULL values are not stored in indexes.

Indexes

Reverse-Key Index

- The bytes of the index are reversed before storage.
- Used often in cases where exact matches are required
- Traditional Indexes are more useful in situations where range checking is used.
  - ➢ E.g. Index values 1234 and 1235 are stored as 4321 and 5321.

Dropping an Index

```
DROP INDEX index_name;
```

Data Dictionary Lookup tables for indexes

USER_INDEXES

USER_IND_COLUMNS

Indexes

Guidelines on usage of Indexes

- If a table has more than a few hundred rows, index it.
- For the index to be used in a partial match, the first column (leading-edge) must be used.
- Index only simple columns.
- Try not to create more than two or three indexes per table.
- Index frequently used columns. Especially if the columns are frequently being made use of in joins.
- Specify table space for each index created
- Parallelize Index creation by enabling the parallel degree option
- Calculate index initial, next storage sizes, PCTFREE values appropriately
- Consider Unrecoverable or No logging indexes for large index creation. (you need to backup)
- Plan to re-create indexes if there were two many inserts and deletions
- Drop the indexes that are no longer required.

Indexes

- If the optimizer decides against using the INDEX
- Functions have been applied on indexed columns.
- Cost based optimizers fail to use Indexes if their columns are part of an OR condition
- However, they slow data updates. Keep this in mind when doing many updates in a row with an index.
- LONG and LONG RAW columns cannot be indexed.
- If there are many nulls in a column and you don’t search on not null columns alone.
- If the selection queries select more than 30% of the records in the table, index will be skipped.

Database Sequences

Database sequences are special database objects that are used to generate integer values according to rules defined when the sequence was created.

Sequences are generally used to create primary keys, they can also be used to generate random numbers

```
CREATE SEQUENCE student_id
    START WITH 1
    INCREMENT BY 1
    CACHE 20
    ORDER;
```

Database Sequences

```
SELECT student_id.nextval FROM dual;

SELECT student_id.currval FROM dual;

INSERT INTO students
    (name, date_of_birth, gender, student_id)
    VALUES
    ('Taylor','01-JAN-77','F',student_id.nextval);

DROP SEQUENCE student_id;
```

Exercise

1. Create a synonym for the “player” and “match_details” table
2. Create a index on player_name column on player table and a index on runs_scored in score_details table.
3. Create a sequence for generating player_id and match_id.
4. Create a sequence for generating the country_id. Ensure that there is a difference of atleast 2 between each country id.

Data Manipulation language

These statements deal with the manipulation of data in the database.

- SELECT
- INSERT
- UPDATE
- DELETE

Data Manipulation language

To add new rows into the database

Eg:INSERT INTO employee VALUES(122, ‘SMITH’,’MANAGER’, ‘10-AUG-93’, 9000);

To make changes to existing data in the table.

Eg:UPDATE employee SET sal=sal+ 500 WHERE empno= 122 ;

To remove the specified data from the table.

Eg:DELETE FROM employee WHERE empno = 100;

Data Retrieval

‘SELECT’ command is used to query data from the database.

Eg.

```
SELECT * FROM employee ;
SELECT DISTINCT job FROM employee ;
```

Conditional Retrieval

- SELECT * FROM employee *WHERE empno=102;*
- SELECT ename,sal FROM employee *WHERE empno=102;*

Data Control Language

User access to the database is controlled in SQL by granting and/or revoking privileges.

These privileges control access to the data as well as to the resources of the database

SQL allows two types of privileges

- System privileges- Extend permission to execute
  - ➢ Data Definition commands

    E.g. CREATE TABLE…
  - ➢ Data Control commands

    E.g. ALTER USER…
- Object privileges – Extend permissions to operate on a named database object

Data Control Language

These statements deal with granting and revoking access and privileges to the data structures and resources.

- GRANT
- REVOKE

GRANT privileges {|ON object name}TO username {WITH GRANT OPTION}
REVOKE privileges {|ON object name}FROM username

E.g.

```
GRANT SELECT  TO  user1;
GRANT ALL TO user2;
GRANT UPDATE TO  user3;
GRANT ALL TO user3 WITH GRANT OPTION;
REVOKE UPDATE FROM user3;
```

Data Control Language

The user can grant privileges on any object created by him.

Object Privileges – ALL,ALTER,DELETE,EXECUTE, INDEX, INSERT, READ, REFERENCES,SELECT, UPDATE

INSERT,UPDATE,DELETE are available on materialized views only if the materialized view is Updateable.

Privileges granted on objects are also available to their synonyms.

Allotting privileges ‘WITH GRANT OPTION’ allows users to pass on their privileges to other users.

- E.g. GRANT SELECT ON table_name TO user_name WITH GRANT OPTION.
  - ✓ The SELECT privilege can be passed on by user *user_name* to other users.

Data Control Language

A ROLE is a collection of privileges given to many users at a time.

- CREATE ROLE role_name {NOT IDENTIFIED | IDENTIFIED {BY passwd |EXTERNALLY}}
- Once the role is created privileges may be granted to it.
- Standard Roles: Connect Role,Resource Role, DBA Role
- User defined Roles
  - ➢ CREATE ROLE DML_ROLE;
  - ➢ GRANT SELECT,INSERT TO DML_ROLE;
  - ➢ GRANT DML_ROLE TO <user>

Transaction Control Statements

A transaction is a logical unit of work transactions can occur between any of the following events -

- Connecting to Oracle
- Disconnecting from Oracle
- Committing changes to the database
- Rollback

Commit

A commit ends the current transaction and makes permanent any changes made during that transaction.

```
Eg
statement 1;
............;
.....;
Commit;
........;
statement7.......;
```

Rollback

The Rollback statement does exactly the opposite of commit.

It ends the transaction, but undoes any changes made during the transaction.

```
Statement1;
..........................;
.....................;
Rollback;
.......................;
```

Savepoint

Savepoint marks and save s the current point in the processing of a transaction. Used with the rollback statement, savepoints can undo parts of a transaction.

```
Rollback to savepoint ;
.........;
............;
savepoint1;
...........;
savepoint2;
Rollback to savepoint1;
```

Pseudo Columns

SEQUENCE.NEXTVAL

SEQUENCE.CURRVAL

ROWNUM

ROWID

SYSDATE

Recap

- Where do we use sequences?
- Why do we need synonyms?
- What is materialized view?
- What are the type of indexes?
- What are the advantages and disadvantages of indexes?
- How can you delete the first 5 rows in a table?
- Why do we need Roles?
- What is the use of Save points?

Exercise

1. Insert records into the country, player, match_details, score_details and bowling tables.
2. Update the specialization of few players to all rounder in “player” table.
3. Try to delete a country_id from “country” table for a country_id which has a record in other tables.
4. Select the score details for a player given the player id.
5. Select the player id’s who are active players and are all rounder.
6. Select the winner country id given a venue.
7. Create a role called viewer and assign the select privileges on player, match_details and country. Assign this to a user trg01. Login as trg01and try selecting the records from above tables, bowling and score_details
8. Create a role called owner and assign the select and insert privileges on all tables. Assign this to a user trg02. Login as user trg02 and try selecting the records from all tables. Try to insert records as well into these tables.
9. Revoke the reviewer role to the user trg01 and assign owner role to the user trg01.

Operators

Types of operator

- Aritmetic Operators
- Concatenation Operators
- Comparison Operators
- Logical Operators
- Set Operations

Precedence

- Unary operators
- Binary operators
  - ➢ multiplication, division
  - ➢ addition, subtraction, concatenation
  - ➢ comparison
  - ➢ exponentiation
  - ➢ conjunction(AND)
  - ➢ disjunction(OR)

Arithmetic Operators

+ (Addition), - (Subtraction),

*** (Multiplication), / (Division)**

Eg.

```
SELECT Name, Above, Below, Empty,
      Above + Below As Plus,
      Above - Below As  Minus,
      Above * Below As Times,
      Above / Below AS Divided
FROM Math WHERE Name=‘EXAM’;
NAME   ABOVE  BELOW  EMPTY  PLUS  MINUS   TIMES   DIVIDED
--------  ----------  ---------   ---------  -------  ---------  --------     -----------
EXAM       2           -3                        -1        5             -6          - .666
```

Concatenation operator

|| (Concatenation)

Tells Oracle to concatenate, or stick together, two strings.

The strings can be either column names or literals .

Eg.,

```
SELECT  City || ’,’ || Country from LOCATION;
CITY || ‘,’ || Country
--------------------------------------
ATHENS, GREECE
CHICAGO, UNITED STATES
```

Comparison Operators

=, != , <>, >, >=, <, <=

BETWEEN

IN, NOT IN

LIKE

IS NULL

Logical Operators

AND

OR

NOT

Comparison Operators

Examples:

SELECT empno,ename FROM employee WHERE job = ‘MANAGER’ AND hiredate >= ‘01-JAN-94’;

SELECT empno, ename FROM employee WHERE sal BETWEEN 5000 AND 9000;

SELECT ename,sal FROM employee WHERE job IN (‘CLERK’,’ANALYST’,’SALESMAN’);

Comparison Operators

The Like operator is used to search for a particular pattern.

An underline character (_) represents one space or a character

A percent sign (%) represents any number of spaces or characters

```
SELECT empno, sal FROM employee WHERE ename LIKE ‘M_LLER’;
SELECT empno, sal FROM employee WHERE ename LIKE ‘M%’;
SELECT * FROM friends  WHERE phone LIKE'555-6_6_';
SELECT empno,sal FROM employee WHERE ename  NOT LIKE ‘P%’;
```

Set Operators

Set operators combine the results of two component queries into a single result

Major set operators

- UNION:All rows selected by either query.
- UNION ALL:All rows selected by either query, including all duplicates.
- INTERSECT:All distinct rows selected by both queries
- MINUS:All distinct rows selected by the first query but not the second

Note: - The set operators are not valid on columns of type BLOB, CLOB, BFILE, VARRAY, or nested table.

Set Operators

Eg.

```
SELECT   reg_no,stu_name,total
FROM      school_a
UNION ALL
SELECT   reg_no,stu_name,total
FROM      school_b
```

Number of column names and data type of each column should be same for set operations

Order by clause can come in the last query only

User Defined Operators

Like built-in operators, user-defined operators take a set of operands as input and return a result

Operators can be referenced by index types and by DML and query SQL statements

Operators reference functions, packages, types, and other user-defined objects.

Syntax

CREATE {|OR REPLACE| OPERATOR {|SCHEMA.}operator_name BINDING binding_clause ;

where the BINDING binding_clause is

BINDING {|(param1…)} RETURN return_type implementation_clause

User Defined Operators

Example

```
CREATE OPERATOR scott.merge
BINDING (varchar2, varchar2) RETURN varchar2
        USING text.merge,
   (spatial.geo, spatial.geo) RETURN spatial.geo
        USING spatial.merge;
```

This example

- creates an operator called MERGE in the SCOTT schema
- The first binding is for merging two VARCHAR2 values and returning a VARCHAR2 result.
- The second binding is for merging two geometries into a single geometry.
- The corresponding functional implementations for the bindings are also specified.

note: - text.merge and spatial.merge are functions

ORDER BY Clause

This clause is used to control the order in which the rows are displayed i.e., either ascending or descending

```
SELECT empno , ename , job FROM employee ORDER BY ename desc ;
SELECT empno, ename , job FROM employee ORDER BY ename , job ;
```

Functions

Arithmetic Functions

Character Functions

Date Functions

Aggregate Functions

Arithmetic Functions

ABS

CEIL

FLOOR

COS

COSH

EXP

GREATEST

LEAST

LN

LOG

MOD

POWER

SIGN

SQRT

ROUND

Arithmetic Functions

ABS(n)	**- Returns absolute value.**
CEIL(n)	**- Returns smallest integer greater than the given value**
FLOOR(n)	**- Returns largest integer smaller than the given value**
COS(n)	**- Cosine value in radians**
GREATEST(a1,a2..)	**- Returns the greatest of the given values**
LEAST(a1,a2..)	**- Returns the least of the given values**
MOD(m,n)	**- Returns remainder of m divided by n**
POWER(m,n)	**- m raised to nth power**
SIGN(n)	**- n<0 = -1, n > 0 = 1, n = 0 = 0**
SQRT(n)	**- Returns the square root of n**
ROUND(n[,m])	**- Returns n rounded to m decimal places, m default is 0**

Arithmetic Functions

SELECT ABS(-4) FROM DUAL

```
 ABS(-4)
----------
         4
```

SELECT SIGN(7) FROM DUAL;

```
 SIGN(7)
----------
         1
```

SELECT SIGN(7) FROM DUAL;

```
 SIGN(-7)
----------
        -1
```

Character Functions

Condition Checking

- DECODE
- NVL
- NVL2

Character Functions

- INITCAP
- INSTR
- LENGTH
- LOWER,UPPER
- RPAD,LPAD
- TRIM,LTRIM,RTRIM
- SOUNDEX

Character Functions

DECODE(e,s1,r1,s2,r2..)	- Returns r1 if e matches s1 etc.,
NVL(expr1,expr2)	- Returns expr2 if expr1 is null else expr1
NVL2(expr1,expr2,expr3)	- Returns expr2 if expr1 is not null else expr3
INITCAP(char)	- Returns the initial character capitalized
INSTR(char1,char2[,n[,m]])	- Position of mth occurrence of char2 in char1 starting from n position
LENGTH(char)	- Returns the length of the character
LOWER,UPPER(char)	- Returns lower case or upper case characters
RPAD,LPAD(char1,n[,char2])	- Right or left pad char1 to length n with char2. Default blank
TRIM,LTRIM,RTRIM(char[,set])	- char, with final characters removed after the last character not in set. set defaults to ' '.
SOUNDEX(char)	- A char value representing the sound of the word(s)in char.
REPLACE(ch,srch,rep)	- ch, with every occurrence of srch replaced by rep
SUBSTR(char,m[,n])	- Returns sub string of of char from position m for n bytes
TRANSLATE(char,from,to)	- from will be translated to to in char

Character Functions

SELECT LENGTH('Rama Devi') FROM DUAL;

```
LENGTH('RAMADEVI')
------------------
                 9
```

SELECT RPAD('5000',14,'*') FROM DUAL;

```
RPAD('5000',14
--------------
5000**********
```

SELECT DECODE(sex, 'M', 'Male', 'F', 'Female', 'Unknown')
FROM   employees;

Date Functions

ADD_MONTHS

GREATEST

LEAST

LAST_DAY

MONTHS_BETWEEN

NEW_TIME

NEXT_DAY

SYSDATE

TO_CHAR

Date Functions

SELECT SYSDATE FROM DUAL;

```
SYSDATE
---------
28-JUL-03
```

SELECT ADD_MONTHS(SYSDATE,3) FROM DUAL;

```
ADD_MONTH
---------
28-OCT-03
```

Aggregate Functions

The aggregate functions are applied to each group of rows and a single result row is returned for each group

All aggregate functions other than COUNT and GROUPING ignore NULLs

Major Aggregate functions

- COUNT
- SUM
- MAX
- MIN
- AVG
- GROUPING:Used with GROUP BY

GROUP BY Clause

The Group by Clause groups the selected rows based on the value of expression for each row and returns a single row of summary information for each group.

```
SELECT      deptno , AVG (pay)
FROM        employee
GROUP BY    deptno ;
```

HAVING Clause

It works just like the ’WHERE’ clause except that its logic is only related to results of GROUP BY functions.

```
SELECT          job , count(*) , 12 * AVG(Sal)
FROM            employee
GROUP BY        job
HAVING          count(*)  > 1;
```

CASE WHEN Clause

In a searched CASE expression, Oracle searches from left to right until it finds an occurrence of condition that is true, and then returns return_expr.

If no condition is found to be true, and an ELSE clause exists, Oracle returns else_expr. Otherwise, Oracle returns null.

CASE expressions let you use IF ... THEN ... ELSE logic in SQL statements without having to invoke procedures SELECT ename,

```
                CASE WHEN sal>1000 THEN 'Sal greater than 1000'
ELSE 'Under paid' END salary_status
FROM      employees;
```

Exercise

1.Select the team id’s and venue of match which was held between a given dates

2.Select the player ids, who has scored century against a given country.

3. Select the bowler id’s who have not taken any wickets but bowled 10 overs and given less than 20 runs in a match.

4. Select all the matches played by a country id as host team or against team. Use the union operator to append the output of two queries.

5. Select the nick name of the players who are active for a given country. If nick name is null then display it as “No Nickname”.

6. Find the average score for a given player_id.

7. Find the total runs scored by a given player_id.

8. Find the maximum and minimum runs scored by each players

9. Find the maximum and minimum runs scored by each players. Select only the players who have scored atleast 500 runs.

Joins

A join is a query that combines rows from two or more tables, views, or materialized views.

Oracle performs a join whenever multiple tables appear in the query's FROM clause.

The query's select list can select any columns from any of these tables.

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

Most join queries contain WHERE clause conditions that compare two columns, each from a different table. Such a condition is called a join condition.

In addition to join conditions, the WHERE clause of a join query can also contain other conditions that refer to columns of only one table. These conditions can further restrict the rows returned by the join query.

Joins

Eg.

```
SELECT *
FROM     student, mark

SELECT stu.reg_no, name, mark1, mark2
FROM    student stu, mark ma

SELECT stu.reg_no, name, mark1, mark2
FROM    student stu, mark ma
WHERE mark1 >= 40 AND mark2 >= 40
```

Equi-joins and Nonequi-joins

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

```
SELECT   stu.reg_no, name, mark1, mark2
FROM     student stu, mark ma
WHERE    stu.reg_no = ma.reg_no
```

An non equijoin is a join with a join condition containing other than equality operator.

Self Joins

A self join is a join of a table to itself.

This table appears twice in the FROM clause and is followed by table aliases that qualify column names in the join condition.

The following query will select the employee name and his manager details.

```
SELECT   emp.emp_id, emp.emp_name, emp1.emp_name
FROM     employee emp, employee emp1
WHERE emp.mgr_id = emp1.emp_id
```

Outer Joins

An outer join extends the result of a simple join.

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

```
SELECT   stu.reg_no, ma.mark1
FROM     student stu, mark ma
WHERE    stu.reg_no = ma.reg_no(+)
```

Subquery

A Subquery is a query inside a query and has the following advantages:

- An SQL statement with a subquery is more often the most natural way to express a query because it closely parallels the English language description of a query.
- Subqueries are easier to write because they let us break the query into pieces.
- There are some queries that can only be written using a subquery.

Subquery

Examples:

```
SELECT ename, job FROM employee WHERE job =
(SELECT job FROM employee WHERE ename = ‘MILLER’);

SELECT ename,job FROM employee  WHERE deptno = 10  AND
job IN ( SELECT  job FROM emp WHERE deptno = 30);

SELECT  ename , job, sal FROM employee
WHERE (job , sal ) = (SELECT job , sal FROM employee  WHERE
ename = ‘JONES’);
```

Subquery

Subqueries can also be used to retrieve information from more than one table.

```
SELECT ename , job  FROM employee WHERE  job IN
(SELECT job FROM employee  E, dept  D WHERE D.loc = ‘CHICAGO’
AND  E.deptno = D.deptno) ORDER BY job;
```

Multiple subqueries can be part of a query

```
SELECT ename, job , deptno ,sal FROM employee
WHERE job = (SELECT job FROM employee WHERE ename = ‘SMITH’)
            OR
sal <=  (SELECT sal FROM employee  WHERE ename = ‘KING’)
 ORDER BY job ,sal ;
```

Subquery

Subqueries can have more subqueries nested in it

```
SELECT ename,job FROM emp WHERE deptno = 20 and job IN
(SELECT job FROM emp WHERE deptno =
(SELECT deptno FROM dept WHERE dname = ‘SALES’);
```

The ‘exists’ clause checks whether a subquery produces any rows of query results.

The logical expression ‘exists’ is true if the subquery returns at least one row and false if not.

```
SELECT job, ename , empno ,deptno FROM emp E
WHERE  EXISTS
(SELECT  * FROM emp WHERE E.empno = mgr)
ORDER BY job , ename;
```

Correlated Subquery

Correlated subqueries enable you to use an outside reference to the query.

The subquery will get executed once for every row returned by the main query

Eg.

To find the employees who earn more than the average salary of employees in their own departments.

Part 1

```
SELECT ename ,sal FROM  employee WHERE sal >
(average  salary of employee’s department)
```

Part2

We also need a subquery that calculates average salary of each candidate employee’s department.

```
SELECT avg(sal)  FROM employee  WHERE
deptno =  (candidate  row’s value of deptno)
```

Correlated Subquery

Hence we arrive at:

- SELECT ename , sal FROM employee X WHERE
  sal > (SELECT avg(sal) FROM employee WHERE
  X.deptno = deptno) ORDER BY deptno , sal;

Subquery to delete duplicate records from the table

DELETE FROM table t WHERE rowid > (SELECT Min(rowid) FROM
table t1 WHERE t1.column = t.column);

Subquery with a HAVING clause

- SELECT job, AVG (sal) FROM employee
  GROUP BY job HAVING AVG(sal) >
  (SELECT AVG (sal) FROM emp WHERE deptno = 30);

Views

A view is a kind of a virtual table in the database whose contents are defined by a query.

Creating a View

```
CREATE {OR REPLACE} VIEW <view_name> [(column1, column2...)] AS
SELECT <table_name column_names>
FROM <table_name>
```

Eg. CREATE VIEW sales_persons AS SELECT empno , ename , job FROM employee WHERE job = ‘SALESMAN’;`

Views

Views from complex queries

```
CREATE VIEW empgrade (grade , employee , desig , pay ,site)
AS
SELECT  S.grade ,E.ename , E.job , E.sal , D.deptno
FROM  employee  E , dept D, salgrade S
WHERE  S.sal BETWEEN S.low AND S.high AND E.deptno = D.deptno ;
CREATE OR  REPLACE VIEW salary (name , monthly_pay, annual_pay ) AS
SELECT ename , sal , sal * 12 FROM employee ;
```

Dropping a view

- DROP VIEW view_name;

Views

The following restrictions apply for Modification of views:

- You cannot use ORDER BY in a CREATE VIEW statement
- You cannot INSERT into the underlying tables if they have any NOT NULL columns which are not a part of the view definition
- You cannot modify the view if it references pseudocolumns like ROWNUM or uses GROUP BY or DISTINCT clauses..
- You cannot INSERT/UPDATE a view if the columns being referenced by the INSERT/UPDATE contain functions

Views

Advantages of Views

- Providing user security functions
- Converting between units
- Creating a new virtual table format
- Simplifying the construction of complex queries

Materialized Views

A materialized view is a database object that contains the results of a query. They are local copies of data located remotely.

CREATE MATERIALIZED VIEW <View Name> [REFRESH [FAST|COMPLETE|FORCE]]

[START WITH DATE] [NEXT DATE] [WITH PRIMARY KEY | ROWID] <Query>

ex: **CREATE MATERIALIZED VIEW mv_emp_pk REFRESH FAST START WITH SYSDATE NEXT SYSDATE + 2WITH PRIMARY KEY AS SELECT * FROM emp@remote_db;**

- Refresh option states how the refresh of remote data occurs. Fast options specifies that the modified data be transferred through the materialized view logs (table)
- Refresh Complete states that the data be refreshed completely. Force will choose fast refresh if possible else do complete refresh.
- Start date states from when refresh should start and next date gives the duration
- Primary key or rowid specifies which is the used as the key in master table

Recap

- Types of functions
- Types of joins
- Sub query
- Views

Exercise

1. Select the player name and his captain name using a self join.
2. Select the player name and the country name of all active players.
3. Select the runs, boundaries, sixes. catches, runs given and wickets taken by a player in each match. (Use outer join)
4. Select the top ten batsman and the runs they scored in this year.
5. Select the player names who have scored more than the average runs of that country.
6. Select the player_name and the runs scored by the players who have just batted and never bowled in that match.
7. Create a view for listing the player_id, country_id who are are currently active. Grant select access to viewer role on this table. Login as trg01 and query from this view.

DAY 2

DAY2: Advanced SQL

- **Object-Relational Features**
- **Defining Object Types**
- **Declaring and Defining Methods**
- **Defining Object Tables**
- **Defining relationship between object and relational table**
- **OIDs and REF/DEREF**
- **VARRAYS**
- **Nested Tables**
- **Triggers**
- **Stored Procedures**

Object-Relational Features

The object-relational model is an evolutionary way to introduce object-oriented features to the database without giving up the existing relational features that are used in existing applications.

Oracle 8i is a ORDBMS. Ie. Object Relational Database Management System

Defining Object Types

An object type is a schema object with three kinds of components:

- A name, which identifies the object type uniquely within that schema.
- Attributes, which model the structure and state of the real-world entity. Attributes can be built-in types or object types.
- Methods, functions or procedures that implement operations that mimic ones you can perform on the real-world entity.

Oracle automatically creates a constructor when an object is defined

-
  ```
  CREATE{|OR REPLACE} TYPE address_ty AS OBJECT(
  street VARCHAR2(50),city VARCHAR2(50),
  zip  NUMBER PRIMARY KEY);
  ```

Defining Object Types and Relational Tables

```
CREATE TYPE PointType AS OBJECT (
     x NUMBER,
     y NUMBER
    );
    /
CREATE TYPE LineType AS OBJECT (
    end1 PointType,
    end2 PointType
    );
    /
CREATE TABLE Lines (
    lineID INT,
    line     LineType
    );
```

Defining Object Types and Relational Tables

- **DROP TYPE Linetype;**

- INSERT INTO Lines
  VALUES(27, LineType(PointType(0.0, 0.0),
  PointType(3.0, 4.0)));

Declaring and Defining Methods

A type declaration can also include methods that are defined on values of that type.

The method is declared by MEMBER FUNCTION or MEMBER PROCEDURE in the CREATE TYPE statement, and the code for the function itself (the definition of the method) is in a separate CREATE TYPE BODY statement.

```
CREATE TYPE LineType AS OBJECT ( end1 PointType, end2
    PointType, MEMBER FUNCTION length(scale IN NUMBER)
    RETURN NUMBER, PRAGMA RESTRICT_REFERENCES(length,
    WNDS) ); /
```

PRAGMA

Pragma specifies the purity level of the functions defined. The syntax is

PRAGMA RESTRICT_REFERENCES (function_name, WNDS [, WNPS] [, RNDS] [, RNPS])

WNDS - Writes no database states., Asserts that the function does not modify any
database tables

WNPS - Writes no package states

RNDS - Reads no database states

RNPS - Reads no package states

WNDS level is mandatory in the pragma

Declaring and Defining Methods

Note the ``pragma'' that says the length method will not modify the database (WNDS = write no database state). This clause is necessary if we are to use length in queries

```
CREATE TYPE BODY LineType AS MEMBER FUNCTION
    length(scale NUMBER) RETURN NUMBER IS BEGIN RETURN
    scale * SQRT((SELF.end1.x-SELF.end2.x)*(SELF.end1.x-
    SELF.end2.x) + (SELF.end1.y-SELF.end2.y)*(SELF.end1.y-
    SELF.end2.y) ); END; END; /
```

```
SELECT lineID, ll.line.length(2.0) FROM Lines ll;
```

```
SELECT ll.line.end1.x, ll.line.end1.y FROM Lines ll;
```

Defining Object Tables

An object table is a special kind of table in which each row represents an object.

CREATE TABLE Lines1 OF LineType;

Oracle allows you to view this table in two ways:

- A single-column table in which each row is a LineType object, allowing you to perform object-oriented operations.
- A multi-column table in which each attribute of the object type LineType, namely end1 and end2, occupies a column, allowing you to perform relational operations.

```
INSERT INTO Lines1
    VALUES (LineType( PointType(0.0, 0.0)), …. );
SELECT VALUE(p) FROM Lines1 p
    WHERE p.end1.x = 3;
```

The first instruction inserts a PERSON object into PERSON_TABLE as a multi-column table. The second selects from PERSON_TABLE as a single column table.

Defining relationship between object and relational table

- **Create an object table**
  - CREATE TYPE STU_TYPE AS OBJECT
    (REG NUMBER, NAME VARCHAR2(15));/
  - CREATE TABLE STUDENT OF STU_TYPE
    (REG PRIMARY KEY)
- **Create a relational table with reference**
  - CREATE TABLE MARK
    (STU_MAST REF STU_TYPE SCOPE IS STUDENT,
    MARK1 NUMBER, MARK2 NUMBER);
- **Insert data into the object table**
  - INSERT INTO STUDENT VALUES(101,'Raja');
  - INSERT INTO STUDENT VALUES(102,'Sunil');

Defining relationship between object and relational table

- **Insert data into the relational table with reference**
  - ➢
    ```
    INSERT   INTO MARK
    SELECT   REF(S), 60, 70
    FROM     STUDENT S
    WHERE    REG = 101
    ```
- **Query the reference table**
  - ➢
    ```
    SELECT  STU_MAS, MARK1, MARK2
    FROM    MARK
    ```

OIDs and REF/DEREF

Each row of an Object Table is an Object

Each Object of an object table gets an OID when it is created

Objects that occupy complete rows in object tables are called row objects.

Objects that occupy table columns in a larger row, or are attributes of other objects, are called column objects.

OIDs and REF/DEREF

REF

- A REF is a logical "pointer" to a row object.
- It is an Oracle built-in datatype
- SELECT REF(L) FROM LINES1 L
- This will return the unique OID for each object in this table

DEREF

- Accessing the object referred to by a REF is called dereferencing the REF.
- SELECT DEREF(M.STU_MAS) FROM MARK M

VALUE

- Accessing the value of an object
- SELECT VALUE(M) FROM STUDENT M

Collections

Oracle supports two collection datatypes for modelling one-to-many relationships

- VARRAYS (Varying Arrays)
- NESTED TABLES

For example, a purchase order has an arbitrary number of line items, so you may want to put the line items into a collection.

VARRAYS

An array is an ordered set of data elements and All elements of a given array are of the same datatype.

Each element has an index, which is a number corresponding to the element's position in the array.

The number of elements in an array is the size of the array.

Oracle allows arrays to be of variable size, which is why they are called VARRAYs. You must specify a maximum size when you declare the array type.

VARRAYS

For example,

- CREATE TYPE MARK AS VARRAY(5) OF NUMBER(3);
- CREATE TABLE STUDENT(REG NUMBER(6) PRIMARY KEY,

  NAME VARCHAR2(10) UNIQUE, MARKS MARK);
- INSERT INTO STUDENT VALUES(110789,’Girish’,MARK(60,80,NULL,90));
- SELECT * FROM STUDENT

Creating an array type does not allocate space. It defines a datatype

The individual elements of a VARRAY cannot be referenced by index in DML or SQL statements

Using PL/SQL constructs, we can access individual array elements

VARRAYS

```
CREATE TYPE GAS_LOG_TY AS OBJECT ( GALLONS NUMBER, FILLUP_DATE DATE, GAS_STATION
    VARCHAR2(255));

CREATE TYPE GAS_LOG_VA AS VARRAY(100) OF GAS_LOG_TY;

CREATE TABLE GAS_LOG (VIN NUMBER NOT NULL, GAS_LOG GAS_LOG_VA);

insert into gas_log values (101010101010101,gas_log_va(gas_log_ty(32,sysdate-1,'Shell')));

insert into gas_log values (321321321321321,gas_log_va( gas_log_ty(45,sysdate-10,'Diamond Shamrock'),
    gas_log_ty(31,sysdate-9,'Shell'), gas_log_ty(32,sysdate-8,'Shell'), gas_log_ty(33,sysdate-7,'Texaco'),
    gas_log_ty(34,sysdate-6,'Texaco'), gas_log_ty(35,sysdate-5,'Diamond Shamrock')));

select * from gas_log;

SQL>col gas_station for a40

select a.vin,var.gallons,var.fillup_date,var.gas_station from gas_log a, table(gas_log) var;
```

Nested Tables

A nested table is an unordered set of data elements, all of the same datatype.

It has a single column, and the type of that column is a built-in type or an object type.

If the column in a nested table is an object type, the table can also be viewed as a multi-column table, with a column for each attribute of the object type.

A table type definition does not allocate space. It defines a type

Nested Tables

For example, in the purchase order , the following statement declares the table type used for the nested tables of line items:

- CREATE TYPE lineitem_table AS TABLE OF lineitem;
- CREATE TABLE purchase_order_table OF purchase_order NESTED TABLE lineitems STORE AS lineitems_table;
- The second line specifies LINEITEMS_TABLE as the storage table for the LINEITEMS attributes of all of the PURCHASE_ORDER objects in PURCHASE_ORDER_TABLE.
- SELECT * FROM THE (SELECT lineitem FROM lineitem_table) nested

A convenient way to access the elements of a nested table individually is to use a nested cursor

THE allows us to treat a nested relation as a regular relation

Nested Tables

Example

```
create or replace type item as object (
item_id Number ( 6 ),
descr varchar2(30 ),
quant Number ( 4,2) );

create or replace type items as table of item;

create table bag_with_items (
bag_id number(7) primary key,
bag_name varchar2(30) not null,
the_items_in_the_bag items )
nested table the_items_in_the_bag store as bag_items_nt;
```

Introduction To Database Objects

Triggers

Stored Procedures

Triggers

Database Triggers

- A database trigger is a stored subprogram associated with a table.
- Oracle can automatically fire the database trigger before or after an INSERT, UPDATE, or DELETE statement.

Applications where database triggers are useful

- Verify data integrity on insertion or update
- Implement delete cascade
- Log events transparently
- Enforce complex business rules
- Initiate business process
- Derive column values automatically
- Enforce complex security rules
- Maintain replicated data

Triggers

There are several types of database triggers:

- Triggers are broadly classified as under
  - ➢ Statement Level
  - ➢ Row Level
- The triggers are listed below

	Row level	**Statement level**
• Before insert	Y	Y
• After insert	Y	Y
• Before update	Y	Y
• After update	Y	Y
• Before delete	Y	Y
• After delete	Y	Y

Triggers

```
CREATE [OR REPLACE] TRIGGER [schema.]trigger
   {BEFORE event | AFTER event | INSTEAD OF event}
   referencing_clause WHEN (condition) pl_sql_block
```

event can be one or more of the following (separate multiple events with OR)

```
   DELETE event_ref,   INSERT event_ref,   UPDATE event_ref
   UPDATE OF column, column... event_ref
   ddl_statement ON [schema.] {table|view}
   ddl_statement ON DATABASE
   SERVERERROR,   LOGON,   LOGOFF,    STARTUP,   SHUTDOWN
event_ref:
   ON [schema.]table
   ON [schema.]view
   ON [NESTED TABLE nested_table_column OF] [schema.]view
referencing_clause:
   FOR EACH ROW
   REFERENCING OLD [AS] old [FOR EACH ROW]
   REFERENCING NEW [AS] new [FOR EACH ROW]
   REFERENCING PARENT [AS] parent [FOR EACH ROW]
```

Triggers

```
create ore replace trigger trg_emp
before insert or update on employee
for each row
begin
    :new.modified_date := sysdate;
end;
```

Triggers

Use INSTEAD OF triggers to perform DELETE, UPDATE, or INSERT operations on views, which are not inherently modifiable

The following view involves a join of two tables and the ability to update records in the view is limited

```
CREATE VIEW worker_lodging_manager
AS
SELECT      worker.name,
            lodging.lodging,
            lodging.manager
FROM        worker,lodging
WHERE       worker.lodging = lodging.lodging
```

Instead Of Triggers

If we use an INSTEAD OF trigger, we can tell Oracle how to update, delete, or insert records in tables

```
CREATE OR REPLACE TRIGGER worker_lodging_manager_update
INSTEAD OF UPDATE ON worker_lodging_manager
FOR EACH ROW
BEGIN
        IF :old.name <> :new.name THEN
        UPDATE  worker SET    name = :new.name WHERE   name = :old.name;
        END IF;
        IF :old.lodging <> :new.lodging THEN
        UPDATE  worker SET    lodging = :new.lodging WHERE   name    = :old.name;
        END IF;
        IF :old.lodging <> :new.lodging THEN
        UPDATE  lodging  SET     manager = :new.manager WHERE  lodging  = :old.lodging;
        END IF;
END
```

New Database Triggers - In Oracle 8i

Prior to Oracle 8i, database triggers could be applied to tables only. Essentially, they were table triggers.

Oracle 8i introduces eight new database triggers, which extend beyond previous limitation.

Trigger Event	Executes Before/After	Trigger Description
STARTUP	AFTER	Executes when the database is started
SHUTDOWN	BEFORE	Executes when the database is shut down
SERVERERROR	AFTER	Executes when a server-side error occurs
LOGON	AFTER	Executes when a session connects to the database
LOGOFF	BEFORE	Executes when a session disconnects from the database
CREATE	AFTER	Executes when a database object is created; could be created to apply to the schema or to the entire database
ALTER	AFTER	Executes when a database object is altered; could be created to apply to the schema or to the entire database
DROP	AFTER	Executes when a database object is dropped; could be created to apply to the schema or to the entire database

Stored Procedures

Stored Procedures are database objects.

A stored procedure is a precompiled oracle statement(s) Procedural Language.

```
[CREATE [OR REPLACE]]
PROCEDURE procedure_name[(parameter[, parameter]...)]
  [AUTHID {DEFINER | CURRENT_USER}] {IS | AS}
  [PRAGMA AUTONOMOUS_TRANSACTION;]
  [local declarations]
BEGIN
  executable statements
[EXCEPTION
  exception handlers]
END [name];
```

where parameter stands for the following syntax:

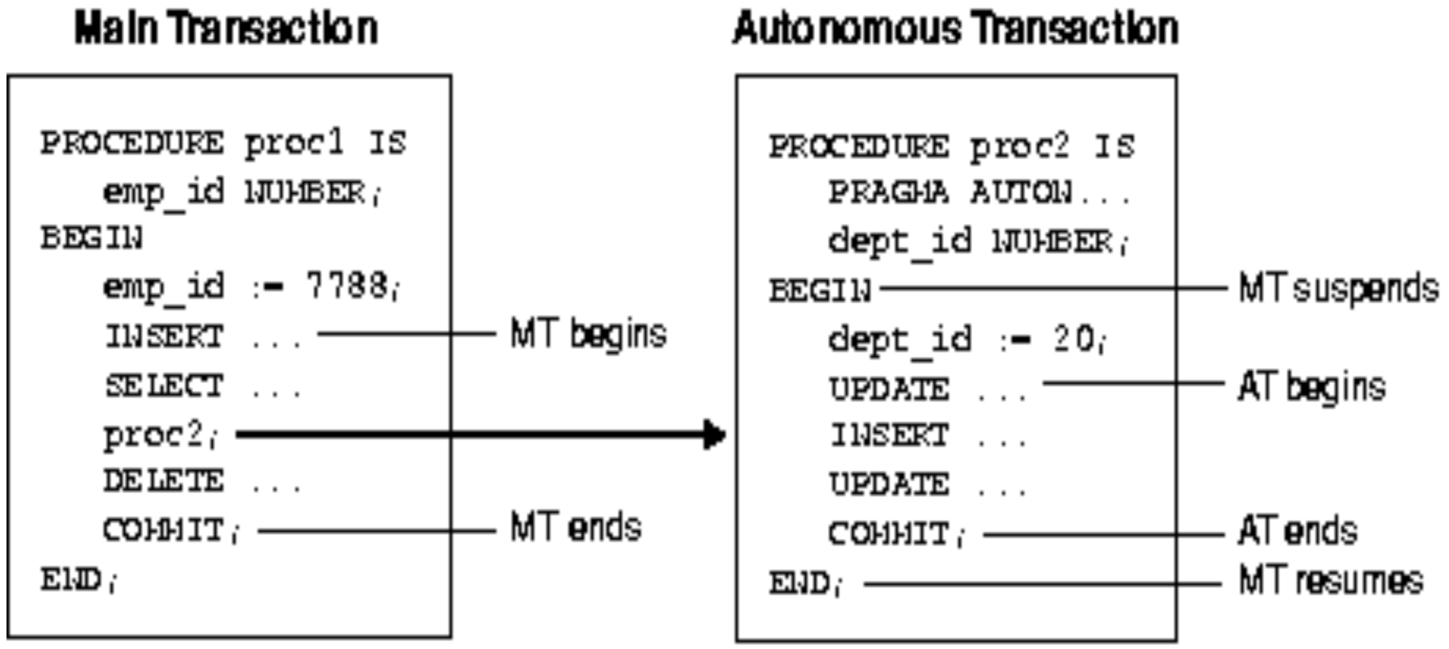
```
parameter_name [IN | OUT [NOCOPY] | IN OUT [NOCOPY]] datatype  [{:= | DEFAULT}
    expression]
```

Stored Procedures

Autonomous Transactions

The pragma AUTONOMOUS_TRANSACTION instructs the PL/SQL compiler to mark a procedure as autonomous (independent).

Autonomous transactions let you suspend the main transaction, do SQL operations, commit or roll back those operations, then resume the main transaction.



```
PROCEDURE proc1 IS
   emp_id NUMBER;
BEGIN
   emp_id := 7788;
   INSERT ...
   SELECT ...
   proc2;
   DELETE ...
   COMMIT;
END;
```

```
PROCEDURE proc2 IS
   PRAGMA AUTON...
   dept_id NUMBER;
BEGIN
   dept_id := 20;
   UPDATE ...
   INSERT ...
   UPDATE ...
   COMMIT;
END;
```

Stored Procedures

NOCOPY Hint

- By default, the OUT and IN OUT parameters are passed by value.
- When the parameters hold large data structures such as collections, records, and instances of object types, all this copying slows down execution and uses up memory.
- To prevent that, you can specify the NOCOPY hint, which allows the PL/SQL compiler to pass OUT and IN OUT parameters by reference.

Stored Procedures

```
CREATE OR REPLACE PROCEDURE raise_salary (emp_id IN NUMBER, increase IN NUMBER) IS
    current_salary REAL;
    salary_missing EXCEPTION;
BEGIN
    SELECT sal INTO current_salary FROM emp
    WHERE empno = emp_id;
    IF current_salary IS NULL THEN
      RAISE salary_missing;
    ELSE
      UPDATE emp SET sal = sal + increase
      WHERE empno = emp_id;
    END IF;
EXCEPTION
    WHEN NO_DATA_FOUND THEN
      INSERT INTO emp_audit VALUES (emp_id, ’No such number’);
    WHEN salary_missing THEN
      RAISE_APPLICATION_ERROR(-20000, ‘Salary is not mentioned for the employee’);
END raise_salary;
```

Note: Composite types such as VARRAYS can also be passed as parameter to Stored Procedure

Stored Procedures

```
CREATE OR REPLACE PROCEDURE cur_sample(employee_id IN number) IS
      var_empid       employee.emp_id%type;
     var_empname employee.emp_name%type;
     CURSOR cur_emp IS select emp_name, emp_id from employee
                                            where emp_id = employee_id;
     cur_var             cur_emp%rowtype;
BEGIN
     FOR cur_var in cur_emp
     LOOP
        var_empid := cur_var.emp_id;
        var_empname := cur_var.emp_name;
        dbms_output.put_line('Employee Name :  ' || var_empid || '  Emp Id =' || var_empname);
     END LOOP;
EXCEPTION
     WHEN NO_DATA_FOUND THEN
         dbms_output.put_line('Invalid Employee id ');
END cur_sample;
```

Stored Procedures with clob

```
DECLARE clob_locator CLOB;
    charbuf VARCHAR2(20);
    read_offset INTEGER;
    read_amount INTEGER;
BEGIN -- First we need to get the lob locator
    SELECT c_lob INTO clob_locator FROM lob_table WHERE id = 1;
    -- Read LOB field contents
    DBMS_OUTPUT.PUT_LINE('CLOB Size: ' || DBMS_LOB.GETLENGTH(clob_locator)); read_offset := 1;
    read_amount := 20;
    dbms_lob.read(clob_locator, read_amount, read_offset, charbuf);
    dbms_output.put_line('CLOB Value: ' || charbuf);
END;
```

Stored Procedures with VARRAY

```
CREATE OR REPLACE PROCEDURE gas_log_insert ( in_vin IN NUMBER, in_gallons IN NUMBER, in_fillup_date
      IN DATE, in_gas_station IN VARCHAR2) AS

pr_gas_log_va gas_log_va := gas_log_va();

BEGIN

EXECUTE IMMEDIATE

'SELECT gas_log FROM gas_log WHERE vin = :1 FOR UPDATE OF gas_log' INTO pr_gas_log_va USING in_vin;

 pr_gas_log_va.EXTEND;

pr_gas_log_va(pr_gas_log_va.LAST) := gas_log_ty(in_gallons,in_fillup_date,in_gas_station);

EXECUTE IMMEDIATE

'UPDATE gas_log SET gas_log = :1 WHERE vin = :2' USING pr_gas_log_va, in_vin;

EXCEPTION

         WHEN NO_DATA_FOUND THEN

                 EXECUTE IMMEDIATE 'INSERT INTO gas_log VALUES (:1,gas_log_va(gas_log_ty(:2,:3,:4)))' USING
      in_vin,in_gallons,in_fillup_date,in_gas_station;

END gas_log_insert;
```

Advantages of Stored Procedures

- Higher Productivity due to elimination of redundant coding.
- Memory Saving. Only one copy of the stored program needs to be loaded into the memory for execution by multiple users.
- Application Integrity can be achieved by developing all the applications around a library of stored programs.Coding errors can be reduced.
- Tighter Security can be achieved by restricting users to specific database operations by granting access only through subprograms.

System tables

ALL_CONSTRAINTS	- Contains constraint definitions on accessible tables
ALL_DB_LINKS	- Contains the db link details like host, owner etc.,
ALL_ERRORS	- Contains all the errors and its details
ALL_INDEXES	- Contains details like owner, last analyzed, extents
ALL_OBJECTS	- Contains all objects details like,procedure, table etc
ALL_SOURCES	- Contains the stored procedure, functions etc.,
ALL_TRIGGERS	- Contains the trigger details present in database
DICT	- Contains all the table names present in schema

System tables

DUAL	- Dummy table can be used with any functions
TABLE_PRIVILEGES	- Contains the grants details on objects
USER_FREE_SPACE	- Contains the table space details
USER_INDEXES	- Contains the user created index details
USER_SYNONYMS	- Contains details of synonym created by user
USER_TAB_COLUMNS	- Contains the table, column details created by user
USER_ROLE_PRIVS	- Contains the roles granted to the users
V$SESSION	- Contains the current oracle session details

SQL editor

AUTO [ON,OFF]	- Sets the auto commit on or off for the session
FEED [ON,OFF]	- Switches the feed back. Ex. 1 row selected
HEAD [ON,OFF]	- Switches the column heading on or off
PAGES [n]	- Sets the page size for the output for headings etc.,
PAUSE[ON,OFF]	- Sets the pause option for the output display
PAUSE [char]	- Sets the string when the screen has paused
TIME [ON,OFF]	- Sets the current time display before the prompt
TIMING[ON,OFF]	- Sets the timings for the pl/sql queries.
SPOOL [ON,OFF]	- Controls the spooling of output to file

SQL editor

LINESIZE[n]	- Sets the number of characters to display in a line
SERVEROUTPUT[ON,OFF]	- Sets the display on the screen
UNDERLINE[char]	- Set the format to the header ex. “=“ or “-”
SQLPROMPT[char]	- Sets the sql prompt to char
EDITFILE[file name]	- Sets the default edit file and directory
VERIFY [ON,OFF]	- Sets display in command prompt for verification
WRAP [ON, OFF]	- Controls the wrapping of display
LONG[n]	- Sets the length of the display on prompt