Unit II

Relational Query Languages

Relational Query Languages

- **Objective:**
- What is Relational Algebra?
- Relational Calculus:
  - Tuple Relational Calculus
  - Domain Relational Calculus

Relational Query Languages

- The *Relational Algebra* which is an algebraic notation, where queries are expressed by applying specialized operators to the relations.
- The *Relational Calculus* which is a logical notation, where queries are expressed by formulating some logical restrictions that the tuples in the answer must satisfy.

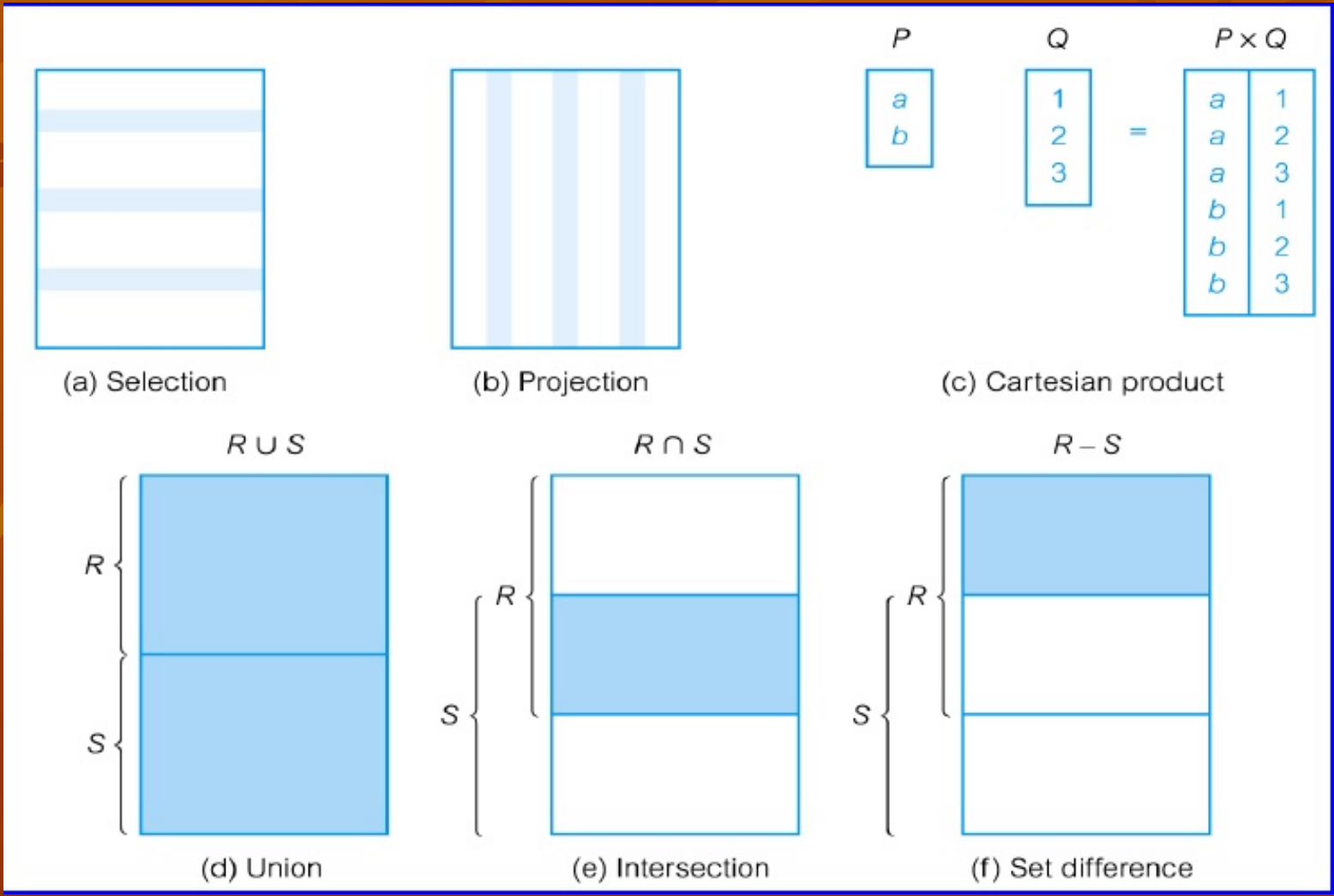
Relational Algebra

- Relational algebra operations work on one or more relations to define another relation without changing the original relations.
- Both operands and results are relations, so output from one operation can become input to another operation.
- Allows expressions to be nested, just as in arithmetic. This property is called closure.

Relational Algebra

- Five basic operations in relational algebra: Selection, Projection, Cartesian product, Union, and Set Difference.
- These perform most of the data retrieval operations needed.
- Also have Join, Intersection, and Division operations, which can be expressed in terms of 5 basic operations

Relational Algebra operations



Selection (or Restriction)

- Predicate (R) :
- Works on a single relation R and defines a relation that contains only those tuples (rows) of R that satisfy the specified condition (predicate).
- Example:
- List all staff with a salary greater than £10,000.

Selection (or Restriction)

$\sigma_{salary > 10000}(Staff)$

staffNo	fName	lName	position	sex	DOB	salary	branchNo
SL21	John	White	Manager	M	1-Oct-45	30000	B005
SG37	Ann	Beech	Assistant	F	10-Nov-60	12000	B003
SG14	David	Ford	Supervisor	M	24- Mar-58	18000	B003
SG5	Susan	Brand	Manager	F	3-Jun-40	24000	B003

In general the select operation is denoted as

σ <selection condition>(R)

Projection

- col1, . . . , coln(R)
- Works on a single relation R and defines a relation that contains a vertical subset of R, extracting the values of specified attributes and eliminating duplicates.
- Example:
- Produce a list of salaries for all staff, showing only staffNo, fName, lName, and salary details.

Projection

ΠstaffNo, fName, lName, salary (Staff)

staffNo	fName	lName	salary
SL21	John	White	30000
SG37	Ann	Beech	12000
SG14	David	Ford	18000
SA9	Mary	Howe	9000
SG5	Susan	Brand	24000
SL41	Julie	Lee	9000

The general form of the PROJECT operation is
Π<attribute list>(R)

Union

- R U S
- Union of two relations R and S defines a relation that contains all the tuples of R, or S, or both R and S, duplicate tuples being eliminated.
- R and S must be union-compatible.
- If R and S have I and J tuples, respectively, union is obtained by concatenating them into one relation with a maximum of (I +J) tuples.

Example - Union

- List all cities where there is either a branch office or a property for rent.
- **Π** City (Branch) U **Π** city(PropertyForRent)

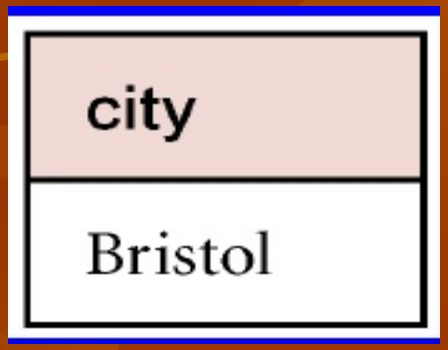
city
London
Aberdeen
Glasgow
Bristol

Set Difference (MINUS)

- R– S
- Defines a relation consisting of the tuples that are in relation R, but not in S.
- R and S must be union-compatible.

Example - Set Difference

- List all cities where there is a branch office but no properties for rent.
- **Π** city(Branch) – **Π**city(PropertyForRent)

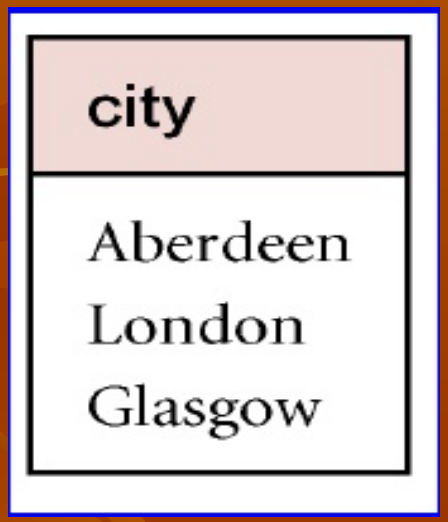


Intersection

- $R \cap S$
- Defines a relation consisting of the set of all tuples that are in both R and S.
- R and S must be union-compatible.

Example - Intersection

- List all cities where there is both a branch office and at least one property for rent.
- **Π** city(Branch) ∩ **Π** city(PropertyForRent)



city
Aberdeen
London
Glasgow

Cartesian product

- R X S
- Defines a relation that is the concatenation of every tuple of relation R with every tuple of relation S.
- If R has n_R tuples and S has n_s tuples, then R X S will have n_R * n_S tuples
- It is also known as cross product or cross join

Cartesian product

- Example - Cartesian product
- List the names and comments of all clients who have viewed a property for rent.

  (**Π** clientNo, fName, lName (Client)) X (**Π** clientNo, propertyNo, comment (Viewing))

Example - Cartesian product

client.clientNo	fName	lName	Viewing.clientNo	propertyNo	comment
CR76	John	Kay	CR56	PA14	too small
CR76	John	Kay	CR76	PG4	too remote
CR76	John	Kay	CR56	PG4	
CR76	John	Kay	CR62	PA14	no dining room
CR76	John	Kay	CR56	PG36	
CR56	Aline	Stewart	CR56	PA14	too small
CR56	Aline	Stewart	CR76	PG4	too remote
CR56	Aline	Stewart	CR56	PG4	
CR56	Aline	Stewart	CR62	PA14	no dining room
CR56	Aline	Stewart	CR56	PG36	
CR74	Mike	Ritchie	CR56	PA14	too small
CR74	Mike	Ritchie	CR76	PG4	too remote
CR74	Mike	Ritchie	CR56	PG4	
CR74	Mike	Ritchie	CR62	PA14	no dining room
CR74	Mike	Ritchie	CR56	PG36	
CR62	Mary	Tregear	CR56	PA14	too small
CR62	Mary	Tregear	CR76	PG4	too remote
CR62	Mary	Tregear	CR56	PG4	
CR62	Mary	Tregear	CR62	PA14	no dining room
CR62	Mary	Tregear	CR56	PG36	

Join

- The join operation is used to combine related tuples from two relations into single tuples
- The general form of a join operation on two relations R(A1,A2,…An) and S(B1,B2,…Bm) is $R \bowtie_{<\text{join condition}>} S$
- The result of the join is a relation Q with n+m attributes
- Q has one tuple for each combination of tuples- one from R and one from S whenever the combination satisfies the join condition

Example – Join

- Find the name of the manager of each department.
- To get the manager's name, we need to combine each department tuple with the employee tuple whose ssn value matches the mgr_ssn value in the department tuple
- Dept_Mgr $\leftarrow$ Dept$\bowtie_{\text{mgr_ssn=ssn}}$ **Employee**
- Result $\leftarrow$ $\Pi_{\text{dname,Lname, Fname}}$(Dept_Mgr)

Relational calculus

- Relational calculus query specifies what is to be retrieved rather than how to retrieve it.
- No description of how to evaluate a query.
- Non procedural language
- When applied to databases, relational calculus has forms : tuple and domain.

Tuple Relational calculus

- Interested in finding tuples for which a predicate is true. Based on use of tuple variables.
- Tuple variable is a variable that ‘ranges over’ a named relation: i.e., variable whose only permitted values are tuples of the relation.
- A simple tuple relational calculus query is of the form {t | COND(t)}

  where t is a tuple variable and COND(t) is a conditional expression involving t
- The result is set of all tuples t that satisfy COND(t)

Tuple Relational calculus example

- To find set of all tuples S such that Staff(S) is true:
- {S | Staff(S)}
- To find details of all staff earning more than £10,000:
- {S | Staff(S) AND S.salary > 10000}
- To find a particular attribute, such as salary, write:
- {S.name, S.salary | Staff(S) AND S.salary > 10000}

Example Tuple Relational Calculus

- List the names of all managers who earn more than £25,000.
- {S.fName, S.lName | Staff (S) AND S.position = ‘Manager’ AND S.salary > 25000}
- Retrieve the birth date and address of the employee whose name is John B. Smith
- {t.Bdate, t.Address | Employee(t) AND t.Fname= ‘John’ AND t.Mname=‘B’ AND t.Lname=‘Smith’}

Tuple Relational Calculus

- Expressions can generate an infinite set.
- For example:
- {S | ~Staff(S)}
- It is unsafe because it yields all tuples in the universe that are not Employee tuples
- To avoid this, add restriction that all values in result must be values in the domain of the expression.

Domain Relational Calculus

- Uses variables that take values from domains instead of tuples of relations.
- If F(d1,d2, . . . ,dn) stands for a formula composed of atoms and d1,d2, . . . ,dn represent domain variables, then:
- {d1,d2, . . . ,dn | F(d1,d2, . . . ,dn)} is a general domain relational calculus expression.

Example Domain Relational Calculus

- Find the names of all managers who earn more than £25,000.
- {fN, lN | (sN, posn, sex, DOB, sal, bN) (Staff (sN, fN, lN, posn, sex, DOB, sal, bN) AND posn = ‘Manager’AND sal > 25000)}

Example Domain Relational Calculus

- List the birth date and address of the employee whose name is ‘John B. Smith’.
- { u, v | (q, r, s, t, w , x, y, z)(Employee(q, r, s, t, u, v, w, x, y, z) AND q=‘John’ AND r=‘B’ AND s=‘Smith’)}
- Or
- {u, v | Employee (‘John’, ’B’, ’Smith’, t, u, v, w, x, y, z)}

Relational Query Languages

- Any query that can be expressed in the relational algebra can also be expressed in the domain or tuple relational calculus
- And , any safe expression in the domain or tuple relational calculus can be expressed in the relational algebra

Other Languages

- Transform-oriented languages are non-procedural languages that use relations to transform input data into required outputs (e.g. SQL).
- Graphical languages provide user with picture of the structure of the relation. User fills in example if what is wanted and system returns required data in that format (e.g. QBE).

Introduction to SQL

- Characteristics and advantages
- SQL Data Types and Literals,
- SQL languages: DDL, DML, DCL, DQL
- SQL Operators
- SELECT Query and clauses
- Aggregate Functions
- Nested Queries
- Predicates and Joins
- PL/SQL: Functions, Procedures and Triggers

SQL

- SQL is the principal language used to describe and manipulate relational databases
- SQL is a keyword based language. Each statement begins with a unique keyword. SQL statements consist of clauses which begin with a keyword.
- SQL syntax is not case sensitive.
- It’s non procedural language

SQL

- SQL enables end user and system persons to deal with a number of database management systems where it is available
- The language while being simple and easy to learn can cope with complex situations
- The results to be expected are well defined in SQL.
- **Standard independent language:** SQL is an open language, involving it is not owned or controlled by a single company.

Languages in SQL

- Data-Definition → for declaring database schemas
- Data-Manipulation → for querying and modifying database
- Transaction-Control → for saving changes in database
- Data-Query → selecting data from database
- Data Control Language → Grant or Revoke the permission

Attribute Data Types in SQL

- All attributes must have a data type

  1) Character strings of fixed or varying length

     CHAR(n) → a fixed-length string of up to *n* characters

     OR

     VARCHAR (n)

Attribute Data Types in SQL

2) Bit strings of fixed or varying length

BIT(n) →bit strings of length *n*

BIT VARYING(n) → bit strings of length up to *n*

3) The type BOOLEAN denotes an attribute whose value is logical

- The possible values are TRUE or FALSE and unknown if presence of null

Attribute Data Types in SQL

4) The type INT or INTEGER denotes typical integer values

- The type SHORTINT also denotes integers, but the number of bits permitted may be less depending on the implementation

5) The type FLOAT denotes floating-point numbers

- Real numbers with a fixed decimal point
- DECIMAL (n,d) allows values that consist of *n* decimal digits, with the decimal point assumed to be *d* positions from the right

Data Types

6) Dates and times can be represented by the data types DATE and TIME respectively. It is used to store date and time information. Default format is DD-MON-YY.

7) NUMBER (P,S)

It is used to store fixed or floating point

P - Precision or total number of digits range 1 to 38.

S - Scale or number of digits to right of decimal point.

DDL

Create Command:

```
CREATE TABLE Table_name
(   Column_name 1 data_type  [column constraint],
Column_name 2 data_type      [column constraint],
                _                    _                    _
                _                    _                    _
Column_name n   data_type    [column constraint],
[Table_constraints]                          );
```

Simple Table Declarations

```
CREATE TABLE Movies (
        title           CHAR(100),
        year            INT,
        length          INT,
        genre           CHAR(10),
        studioName      CHAR(30),
        producerC#      INT
);
```

Declaring Primary Keys

- CREATE TABLE MovieStar (
  name CHAR(30) PRIMARY KEY ,
  address VARCHAR(255),
  gender CHAR(1),
  birthdate DATE
  );

Declaring Primary Keys

- CREATE TABLE MovieStar (
  name CHAR(30),
  address VARCHAR(255),
  gender CHAR(1),
  birthdate DATE,
  PRIMARY KEY (name)
  );

Applying Constraints

```
CREATE TABLE EMP(EMPNO
number(5) PRIMARY KEY,EMPNAME
CHAR (30) UNIQUE,SAL
decimal(7,2)not null
check(sal>25000),COMM
decimal(7,2),DEPTNO int default 1,
FOREIGN KEY(deptno)REFERENCES
DEPT (deptno));
```

Altering the Definition of Table

- **Alter Command:**
- The structure of a table can be modified by using the ALTER table command
- Allows changing the structure of an existing table
- It is possible to **add** or **delete** columns, **change data types** of existing columns,, **add constraints** or **rename** columns or tables

Altering the Definition of Table

- ALTER TABLE Table_name ADD colname datatype [column_constraint], MODIFY colname data_type [column_constraint] , Drop column_name;
- ALTER TABLE *tablename* RENAME [COLUMN] *column* TO *newcolumn* ;

Altering the Definition of Table

- **Example:**
- ALTER TABLE MovieStar ADD phone CHAR(16);
- ALTER TABLE MovieStar ADD phone CHAR(16) DEFAULT ‘unlisted’;
- Alter table employee ADD primary key (empno);

Altering the Definition of Table

- Oracle:
  ALTER TABLE ORDERS
  ADD FOREIGN KEY (customer_sid)
  REFERENCES CUSTOMER(SID);
- alter table employee add constraint fk foreign key(deptno) references cust(custno);

Altering the Definition of Table

- Alter Table Employee **modify** age number(5,2);
- Alter table student modify name varchar(30) not null;
- alter table employee modify salary number(10,2) default 10000;

Altering the Definition of Table

- ALTER TABLE employees RENAME COLUMN address TO mailing_address;
- ALTER TABLE MovieStar DROP column birthdate;
- Alter table employee DROP PRIMARY KEY;
- Alter table employee DROP FOREIGN KEY fk_symbol;

Dropping Database Objects:

- Drop **Command**

- DROP TABLE Table_Name;
- E.G. Drop table emp;

- DDL commands automatically commits

Renaming tables

- Oracle allows renaming of tables
- No one can access the tables while the rename process is running
- Syntax:
- Rename <table_name> to <new table_name>;
- Example:
- Rename emp to employee;

DDL

- **Truncate command:**
- Truncate table empties a table completely
- Logically equivalent to DELETE statement that deletes all rows
- Syntax:
- Truncate table <Table_name>;
- Example:
- Truncate table employee;

Data manipulation language

- **INSERT Command:**
- The INSERT INTO statement is used to insert a new row in a table.
- **Syntax:**
- INSERT INTO TABLE_NAME VALUES (data value 1, data value 2);
- INSERT INTO TABLE_NAME (column 1, column 3) VALUES (data value 1, data value 3);

Data manipulation language

- INSERT INTO EMP VALUES (1001, 'Sharma', NULL,NULL, 3000, NULL);
- INSERT INTO employee (id, name, dept, age, salary) VALUES (105, 'Srinath', 'Aeronautics', 27, 33000);

Data manipulation language

- **Delete command:**
- It deletes rows from the tables that satifies the condition provided by its where clause and returns the number of records deleted
- Syntax:
- DELETE FROM Table1
  WHERE Some_Column = Some_Value ;
- DELETE FROM Table1;

Data manipulation language

- Example:
- Delete from Student;
- DELETE FROM StarsIn

  WHERE movieTitle = ‘The Maltese Falcon’ AND movie Year = 1942 AND starName = ‘Sydney Greenstreet’;

Data manipulation language

- **UPDATE Statement**
- The UPDATE statement is used to update records in a table.
- Syntax:
- UPDATE table_name
  SET column1=value, column2=value2,...
  WHERE some_column=some_value ;

Data manipulation language

- Example:
- Update student set city=‘Bombay’;
- UPDATE Customers
  SET Phone = '626 555-5555'
  WHERE LastName = 'Smith' ;
- UPDATE Customers
  SET DOB = '5/10/1974'
  WHERE LastName = 'Goldfish' AND
  FirstName = 'Steven' ;

Some Other commands

- **Finding out the tables created by the user**
- Select * from tab; (oracle)
- Show tables;
- **Describing the table structure**
- Describe <table name>;
- Desc <table name>;

Data Query language

- **Select command:**
- The SELECT statement is used to select data from a database
- **SQL SELECT Syntax:**
- SELECT column_name (s) FROM table_name;
- SELECT * FROM table_name;
- **Example:**
- SELECT store_name FROM Store_Information;
- Select * from employee;

Select with where clause

- The WHERE clause is used to extract only those records that fulfill a specified criterion.
- **SQL WHERE Syntax:**
- SELECT "column_name(s)“ FROM "table_name“ WHERE "condition“ (column_name operator value);
- Example:
- SELECT * FROM Persons WHERE FirstName=‘Amit' ;
- SELECT store_name FROM Store_Information WHERE Sales > 1000;

Operators Allowed in the WHERE Clause

Operator	Description
=	Equal
<> Or !=	Not equal
>	Greater than
<	Less than
>=	Greater than or equal
<=	Less than or equal
BETWEEN	Between an inclusive range
LIKE	Search for a pattern
IN	If you know the exact value you want to return for at least one of the columns

SQL BETWEEN

- The **SQL BETWEEN & AND** keywords define a range of data between 2 values.
- Syntax:
- SELECT Column1, Column2, Column3 FROM Table1WHERE Column1 BETWEEN Value1 AND Value2 ;
- Example:
- SELECT *FROM Customers WHERE DOB BETWEEN '1/1/1975' AND '1/1/2004'

SQL IN Operator

- The IN operator allows you to specify multiple values in a WHERE clause.
- Syntax:
- SELECT Column1, Column2, …FROM Table1 WHERE Column1 IN (Valu1, Val2..) ;
- Example:
- SELECT *FROM Emp WHERE Hiredate IN ('5/6/2004', '5/7/2004') ;

SQL LIKE

- The **SQL LIKE** clause is very useful when you want to specify a search condition within your SQL WHERE clause, based on a part of a column contents
- The '%' is a so called wildcard character and represents any string in our pattern.
- Another wildcard character is ‘-' representing any single character’
- Example:
- SELECT * FROM Customers WHERE FirstName LIKE 'J%‘;
- SELECT * FROM Customers WHERE Phone LIKE '416%' ;
- SELECT * FROM Customers WHERE FirstName LIKE ‘A_I%‘;

SQL AND & OR Operators

- The AND operator displays a record if both the first condition and the second condition is true.
- SELECT * FROM Persons WHERE FirstName=‘amit‘ AND LastName=‘patil' ;
- The OR operator displays a record if either the first condition or the second condition is true.
- SELECT * FROM Persons WHERE FirstName = ‘amit‘ OR FirstName=‘sumit' ;
- Combining AND & OR
- SELECT * FROM Persons WHERE LastName=‘patil‘ AND (FirstName=‘amit' OR FirstName=‘sumit') ;

SQL SELECT DISTINCT Statement

- The DISTINCT keyword can be used to return only distinct (different) values
- Syntax:
- SELECT DISTINCT column_name(s)
  FROM table_name;
- Example:
- SELECT DISTINCT City FROM employee;

Sorting data in a table: order by

- The ORDER BY keyword is used to sort the result-set
- sort the records in ascending order by default
- If you want to sort the records in a descending order, you can use the DESC keyword.
- Syntax:
- SELECT column_name(s)
  FROM table_name
  ORDER BY column_name(s) ASC|DESC

Sorting data in a table: order by

- Example:
- Select * from employee order by name;
- SELECT * FROM Customers
  ORDER BY DOB DESC ;

Creating table from a table

- The source table is identified in select statement and target table is identified by create
- Syntax:
- Create table <table name> (col1, col2) as select <colname>, <colname> from <table name>;
- Example:
- Create table employee (empno, ename, sal) as select empno, name, salary from emp;

Inserting data into a table from another table

- Populate a table with data that already exists in another table
- Syntax:
- Insert into <table name> select col1, col2 from tablename;
- Example:
- Insert into emp select empno, ename from employee;

SQL Function

- SQL function serve the purpose of manipulating data items and returning a results
- SQL has many built-in functions for performing calculations on data
- Aggregate Functions
- Numeric functions
- String function
- Conversion function
- Date function

Aggregate function

- AVG:
- The AVG() function returns the average value of a numeric column
- Syntax:
- SELECT AVG(column_name) FROM table_name
- Example:
- Select AVG (OrderPrice) AS OrderAverage FROM Orders;
- Find the customers that have an OrderPrice value higher than the average OrderPrice value.
- SELECT Customer FROM Orders WHERE OrderPrice>(SELECT AVG(OrderPrice) FROM Orders) ;

Aggregate function

- **MIN() Function:**
- **The MIN() function returns the smallest value of the selected column**
- **Syntax:**
- **SELECT MIN(column_name) FROM table_name;**
- **Example:**
- **Select** MIN(CURBAL) “Minimum Balance” from account;

Aggregate function

- MAX() Function:
- The MAX() function returns the largest value of the selected column
- Syntax:
- SELECT MAX(column_name) FROM table_name;
- Example:
- Select MAX(curbal) “Maximum Balance” from account;

Aggregate function

- **COUNT() Function:**
- **The COUNT() function returns the number of rows that matches a specified criteria**
- Syntax:
- **SELECT COUNT(column_name) FROM table_name;**
- SELECT COUNT(*) FROM table_name;
- SELECT COUNT(DISTINCT column_name) FROM table_name ;

Aggregate function

- COUNT() Function:
- Examples:
- Select count(ACC_NO) “No of Accounts” from account;
- Select count (*) “No of records” from accounts;
- SELECT COUNT(DISTINCT Customer) “ Number Of Customers” FROM Orders;

Aggregate function

- SUM() Function:
- The SUM() function returns the total sum of a numeric column.
- Syntax:
- SELECT SUM (column_name) FROM table_name;
- Example:
- Select sum (curbal) from account;

Numeric function

- **ABS function:**
- Returns the absolute value of n;
- Example:
- Select ABS(-15) as Absolute from dual;
- O/ P: 15
- **Power:**
- Returns m raised to n th power, n must be an integer
- Example:
- Select power(3,2) from dual;
- O/P: 9

Numeric function

- ROUND() Function:
- **The ROUND() function is used to round a numeric field to the number of decimals specified.**
- **Syntax:**
- **SELECT ROUND (column_name,decimals) FROM table_name;**
- **Example:**
- **Select round(15.19,1) from dual;**
- **O/p: 15.2**
- **SELECT ProductName, ROUND(UnitPrice,0) as UnitPrice FROM Products;**

Numeric function

- **SQRT function:**
- Returns square root of n if n<0 , returns null.
- It returns a real result
- Example:
- Select sqrt(25) as “square root" from dual;
- **Exp function:**
- Returns e raised to the n th power where e=2.71828
- Example:
- Select exp(5) as “exponent” from dual; (148.41)

Numeric function

- **Mod function:**
- Returns the remainder of the first number divided by second number
- If second number is zero, the result is same as the first number
- Example:
- Select mod(15,7), mod(15.7,7)from dual;
- Output: 1 1.7

Numeric function

- **Trunc, Floor, Ceil:**
- Trunc returns a number truncated to a certain number of decimal places
- Select trunc(125.815, 1) “trunc1” , trunc(125.815,-2) “trun 2” from dual; 125.8 100
- Floor returns the smallest integer value that is equal to or than a number
- Select floor(24.8), floor(13.15) from dual; 24 13
- Ceil returns the largest integer value that is equal to a number
- Select ceil(24.8), ceil(13.15) from dual; 25 14

String function

- **Lower:**
- Returns char with all letters in lowercase
- Select lower (name) from emp;
- **Upper:**
- Returns char with all letters in upper case
- Select upper (name) from emp;
- **Initcap:**
- Returns a string with the first letter of each word in upper case
- Select initcap (name) from emp;

String function

- **Ascii :**
- Returns the ascii code for the character
- SELECT ASCII('A') FROM DUAL;
- Result: 65
- **Length:**
- Returns the length of the word
- Select length('SIT LAVALE')as string FROM DUAL;
- Result: 10

String function (mysql)

- MySQL TRIM() function returns a string after removing all prefixes or suffixes from the given string.
- SELECT TRIM(LEADING 'leading' FROM 'l eadingtext');
- SELECT TRIM(TRAILING 'trailing' FROM 't exttrailing');
- SELECT TRIM(BOTH 'leadtrail' FROM 'leadt railtextleadtrail');

String function

- Ltrim:
- Removes characters from the left of char with initial characters removed upto the first character not in set
- Select ltrim(‘nisha’,’n’) from dual; isha
- SELECT LTRIM('xxxDan Morgan','x') FROM DUAL;
- Rtrim:
- Returns char, with final characters removed after the last characters not in the set.
- Set is optional by default it is space
- Select rtrim (‘sunila’,’a’) from dual;

String function

- Lpad:
- Returns char1, left padded to length n with the sequence of characters specified in char2. if char2 is not specified it is blank bydefault
- Select lpad (‘SIT’,5,’*’) from dual; **SIT
- Rpad
- Returns char1 right padded to length n with the characters specified in char2. if char2 is not specified, it is by blank default
- Select rpad (name, 10,’*’) from emp where name=‘amit’; amit******

String function

- **Substring:**
- Returns a portion of characters, beginning at character m and going upto character n.
- If n is ommited, the result returned is upto the last character in the string. The first position of char is 1.
- Substr(<string>,<start position>,<length>);

String function

- **Substr function:**
- Select substr (‘secure’,3,4) from dual;
- Result: cure
- SELECT SUBSTR(store_name, 3) FROM Geography WHERE store_name = 'Los Angeles';
- *Result*: 's Angeles'
- SELECT SUBSTR(store_name,2,4) FROM Geography WHERE store_name = 'San Diego';
- *Result*: 'an D'

String function

- Instr function:
- Returns the location of a substring in a string
- Instr(<string1>,<string2>,[<start position>],[nth app>]);
- Where string1 is the string to search, string2 is a substring to search for in string1. start position is the position in string1 where search start. If ommited by default it is one. nth apper is the nth appearance of string2. if ommited by default it is one.

String function

- Example:
- Select instr (‘symbiosis institute of technology’, ’t’) as instr1 , instr (‘symbiosis institute of technology’,’t’,1 ,4) as instr2 from dual;
- Result: instr1 14 instr2 24

Conversion function

- TO_NUMBER:
- Converts char, a character value expressing a number, to a number datatype.
- Syntax: To_Number (char)
- Example:
- Update account set curbal=curbal+to_number(substr('$100',2,3));
- Here the value 100 will be added to every account's current balance in account table.

Conversion function

- To_CHAR:
- Converts a value of a number datatype to a character datatype, using the optional format string. TO_CHAR () accepts a number (n) and a numeric format in which the number has to appear. If format is ommited, n is converted to a char value exactly long enough to hold all significant digits.
- Syntax: To_Char (n,[fmt])

Conversion function

- Example:
- select to_char(17145,’$099,999’) “char” from dual;
- Result: char $017,145
- Select to_char (dt, ’month DD,YYYY) ”New date format” from trans where trans_no=‘T1’;
- Result: New date format january 05, 2003
- select to_char (sysdate, 'Dy DD-Mon-YYYY HH24:MI:SS') as "Current Time" from dual;
- Result: Current Time Tue 21-Aug-2010 21:18:27

Date Conversion Function

- The value in a column of date is always stored and displayed in a specific default format i.e. ‘DD-MON-YY HH:MI:SS’)
- If data from date column has to be viewed in any other format, to-date function can be used
- The same function can be used for storing a date in a particular format
- Syntax:
- To_date (char, [fmt]);

Date Conversion Function

- TO_DATE function:
- Example:
- Insert into customer (custno, name, dob) values(‘c1,’rahul’,to_date(’25-Jan-1980 10:55 A.M.’,’DD-MON-YY HH:MI A.M.’));
- SELECT * FROM orders WHERE order_date between to_date ('2003/01/01', 'yyyy/mm/dd') AND to_date ('2003/12/31', 'yyyy/mm/dd');

Date function

- ADD_MONTHS:
- Returns date after adding the number of months specified in the function
- Syntax: Add_months (d, n)
- Example:
- Select add_months(sysdate,4) “ add months” from dual;

Date function

- Last_day:
- Returns the last date of the month specified with the function
- Syntax: last_day (d)
- Example:
- Select sysdate, last_day (sysdate) “last day” from dual;
- Result: sysdate 21-Aug-10 last day 31- Aug-10

Date function

- Months_Between:
- Returns number of months between d1 and d2
- Syntax: months_between (d1, d2)
- example:
- Select months_between(’02-feb-10’,’02-jul-10’) ”months” from dual;
- Result: months 5

Date function

- NEXT_DAY:
- Returns the date of the first weekday named by char that is after the date named by name
- Char must be a day of the week
- Syntax: next_day (date , char)
- Select next_day (’21-Aug-10’,’saturday’) “next day” from dual;
- Result: next day 28-Aug-10

Grouping data from tables in SQL

- Group by:
- It group rows based on distinct values that exists for specified columns
- It creates data set, containing several sets of records grouped together based on condition
- Syntax:
- Select <col1>, <col2> , aggr_fun() from tablename where <condition> group by <col1>, <col2>;

Grouping data from tables in SQL

Example 1:
Find out how many employees are there in each department.

```
 mysql> select * from emp ;
+-------+---------+---------+--------+
| empno | ename   | job     | deptno |
+-------+---------+---------+--------+
|     1 | seema   | faculty |     10 |
|     2 | priti   | faculty |     20 |
|     3 | chahita | faculty |     10 |
|     4 | asha    | faculty |     30 |
+-------+---------+---------+--------+
4 rows in set (0.00 sec)

mysql> select deptno, count(empno) "No of employees" from emp group by deptno;
+--------+----------------+
| deptno | No of employees |
+--------+----------------+
|     10 |              2 |
|     20 |              1 |
|     30 |              1 |
+--------+----------------+
3 rows in set (0.00 sec)
```

Grouping data from tables in SQL

Example 2:

Find out the total number of accounts segregated on the basis of account type per branch.

```
mysql> select * from account;
+--------+---------+-----------+
| acc_no | type    | branch_no |
+--------+---------+-----------+
|    100 | saving  |        10 |
|    200 | current |        20 |
|    300 | saving  |        20 |
|    300 | salary  |        20 |
|    400 | current |        30 |
|    500 | current |        10 |
+--------+---------+-----------+
```

Select branch_no, type, count(acc_no) “no of accounts” from account group by branch-no, type;

```
+-----------+---------+---------------+
| branch_no | type    | count(acc_no) |
+-----------+---------+---------------+
|        10 | current |             1 |
|        10 | saving  |             1 |
|        20 | current |             1 |
|        20 | salary  |             1 |
|        20 | saving  |             1 |
|        30 | current |             1 |
```

Grouping data from tables in SQL

Example 3: Find the average annual salary per job in each department.

```
mysql> select * from emp;
+-------+---------+------------+--------+----------+
| empno | ename   | job        | deptno | salary   |
+-------+---------+------------+--------+----------+
|     1 | seema   | faculty    |     10 | 25000.00 |
|     2 | priti   | faculty    |     20 | 10000.00 |
|     3 | chahita | faculty    |     10 | 23000.00 |
|     4 | asha    | faculty    |     30 | 10000.00 |
|     5 | nisha   | instructor |     20 |  5000.00 |
|     6 | geeta   | TA         |     20 |  7000.00 |
|     7 | umesh   | TA         |     10 |  8000.00 |
|     8 | suresh  | instructor |     10 |  9000.00 |
+-------+---------+------------+--------+----------+
8 rows in set (0.00 sec)

mysql> select job,avg(salary) from emp group by
           job;
+------------+--------------+
| job        | avg(salary)  |
+------------+--------------+
| faculty    | 17000.000000 |
| instructor |  7000.000000 |
| TA         |  7500.000000 |
+------------+--------------+
3 rows in set (0.00 sec)

mysql> select deptno, avg(salary) from emp group by
       deptno;
+--------+--------------+
| deptno | avg(salary)  |
+--------+--------------+
|     10 | 16250.000000 |
|     20 |  7333.333333 |
|     30 | 10000.000000 |
+--------+--------------+
3 rows in set (0.00 sec)
mysql> select job,deptno,avg(salary) from emp group by job,
       deptno;
+------------+--------+--------------+
| job        | deptno | avg(salary)  |
+------------+--------+--------------+
| faculty    |     10 | 24000.000000 |
| faculty    |     20 | 10000.000000 |
| faculty    |     30 | 10000.000000 |
| instructor |     10 |  9000.000000 |
| instructor |     20 |  5000.000000 |
| TA         |     10 |  8000.000000 |
| TA         |     20 |  7000.000000 |
+------------+--------+--------------+
7 rows in set (0.00 sec)
```

Grouping data from tables in SQL

- **Having clause:**
- Having clause can be used in conjunction with the group by clause
- It imposes a condition on the group by clause which further filters the groups created by the group by clause
- Each column specification specified in the having clause must occur in the list of columns named in the group by clause

Having clause examples

Example 2:

Find out the number of accounts at a branch having more than one account.

```
mysql> select * from account;
+--------+---------+-----------+-----------+
| acc_no | type    | branch_no | acc_fd_no |
+--------+---------+-----------+-----------+
|    100 | saving  |        10 |      NULL |
|    200 | current |        20 |      NULL |
|    300 | saving  |        20 |      NULL |
|    300 | salary  |        20 |      NULL |
|    400 | current |        30 |      NULL |
|    500 | current |        10 |      NULL |
+--------+---------+-----------+-----------+
6 rows in set (0.00 sec)

mysql> select branch_no, count(acc_no) from account group by branch_no having count(acc_no)>1;
+-----------+---------------+
| branch_no | count(acc_no) |
+-----------+---------------+
|        10 |             2 |
|        20 |             3 |
+-----------+---------------+
2 rows in set (0.00 sec)
```

Having clause examples

Compute the average, minimum and maximum salaries of those groups of employees having job as faculty or TA, Display the job as well

```
mysql> select * from emp;
+-------+---------+------------+--------+----------+
| empno | ename   | job        | deptno | salary   |
+-------+---------+------------+--------+----------+
|     1 | seema   | faculty    |     10 | 25000.00 |
|     2 | priti   | faculty    |     20 | 10000.00 |
|     3 | chahita | faculty    |     10 | 23000.00 |
|     4 | asha    | faculty    |     30 | 10000.00 |
|     5 | nisha   | instructor |     20 |  5000.00 |
|     6 | geeta   | TA         |     20 |  7000.00 |
|     7 | umesh   | TA         |     10 |  8000.00 |
|     8 | suresh  | instructor |     10 |  9000.00 |
+-------+---------+------------+--------+----------+
8 rows in set (0.00 sec)

mysql> select job, avg(salary), min(salary), max(salary) from emp group by job h
aving job='TA' or job='faculty';
+---------+-------------+------------+------------+
| job     | avg(salary) | min(salary) | max(salary) |
+---------+-------------+------------+------------+
| faculty | 17000.000000 |    10000.00 |    25000.00 |
| TA      |  7500.000000 |     7000.00 |     8000.00 |
+---------+-------------+------------+------------+
2 rows in set (0.00 sec)
```

Subqueries

- A subquery is a form of an SQL statement that appears inside another SQL staement
- It is also term as nested query
- The statement containing a subquery is called a parent statement
- The parent statement uses the rows returned by the subquery

Subqueries

- Retrieve the address of a customer named ‘Kunal’.
- Tables: customer, address_detail
- Columns: 1. customer: custno, name, prodid
  2. address_detail : codeno, area, city, state, pincode
- Techniques : suqueries, IN operator, where clause
- Solution: select codeno, area, city, state, pincode from address_detail where codeno in (select custno from customer where name=‘Kunal’) ;

Subqueries

- List customers holding in the bank of amount more than 5000.
- Tables: customer: custno, fanme, lname

  account_detail: custno, amount
- Solution: select (fname|| ‘ ‘||lname “customer” from customer where custno in (select custno from account_detail where amount>5000);

Subqueries

- Find out all the customers having same name as the employee.
- Select name from customer where name in (select name from employee);
- Find out department name in which smith is working.
- Select dname from dept where deptno = (select deptno from emp where ename=‘smith’);

Joins

- The **SQL JOIN** clause is used whenever we have to select data from 2 or more tables.
- To be able to use **SQL JOIN** clause to extract data from 2 (or more) tables, we need a relationship between certain columns in these tables.
- Tables in the database can be related to each other with keys
- The join operator specifies how to relate tables in the query

Types of joins:

1)Inner Join

2)Cross Join

3)OuterJoin

a) Left Outer Join

b) Right Outer Join

c) Full Outer Join

4)Natural Join

5)Equi Join

6) Self join

Joins

- Inner joins:
- The where statement generally compares two columns from two tables with the operator =.
- Outer join:
- It is similar to inner joins, but give a bit more flexibility when selecting data from related tables.
- It is used where it is desired to select all rows from the table on the left or right or both regardless the other table has values in common

Joins

- Inner join example:
- List the employee details along with branch names to which they belongs.
- Tables: emp, branch
- Columns:
- emp: empno, name, dept, desg, branchno

  branch: name, branchno
- Technique : join: inner join , where clause
- Query: select e.empno, b.name, e.dept, e.desg from emp e inner join branch b on b.branchno=e.branchno;
- select e.empno, b.name, e.dept, e.desg from emp e, branch b where b.branchno=e.branchno;

Joins

- Outer join example:
- List the employee details along with the contact details if any.
- Tables: emp, contacts
- Columns:
- emp: empno, name, dept, desg

  contacts: code_no, con_type, data
- Technique : join: outer join , where clause
- Query:select e.name, e.dept, c.con_type, data from emp e, contacts c where e.empno=c.code_no(+);

Joins

- **Joining a table to itself (Self Joins)**
- In some situation, it is necessary to join a table to itself as though joining two separate tables
- This is referred to as a self join
- In it , two rows from the same table combine to form a result row
- From <table name> [alias 1], <table name> [alias 2]……

Joins

- **Self join example:**
- Retrieve the name of the employee and the name of their respective managers from employee table.
- Table: emp: emp_no, ename, mgr_no, sal.
- Technique: self join , where clause
- Select e.ename “employee”, m.ename “manager” from emp e, emp m where e.mgr_no=m.emp_no;

Views

- A view is a logical table that allows you to access data from other tables and views. A view contains no data itself.
- The tables upon which a view is based are called **base tables**.
- **Views are used to:**
- Provide additional level of table security, by restricting access to a predetermined set of rows/columns of a base table.
- Hide data complexity
- To reduce redundant data to the minimum possible

Views

- A view is mapped, to a select sentence
- **Creating view:**
- Create view <viewname> as select <col1>, <col2> from <table name> where cond;
- Examples:
- Create view customer as select * from cust_mstr;
- Create view employee as select name, dept from emp;
- CREATE VIEW dept20 AS SELECT ename, sal FROM EMP WHERE DEPTNO = 20;

Views

- CREATE VIEW clerk (id number, person, department, position) AS SELECT EMPNO, ENAME, DEPTNO, JOB FROM EMP WHERE JOB = ‘CLERK’ WITH CHECK OPTION;
- **WITH CHECK OPTION**
- Specifies that inserts and updates performed through the view must result in rows for which the WHERE clause of the view is true.
- In the second example, you cannot insert a new row if the new employee is not a clerk.
- You cannot perform insert, updates or deletes on a view that is based on a join of multiple tables.

Views

- Consider the following example of a view that contains a join query:
- CREATE VIEW orderreport AS SELECT orders.id, orders.orderdate, items.order_id, items.line_id, items.partid FROM orders, items

  WHERE items.order_id = orders.id ;
- When users query the ORDERREPORT view, ORACLE joins the related row data from ORDERS and ITEMS tables:

Views

- **Selecting a data set from a view:**

- SELECT * FROM orderreport;

ORDER_ID	ORDERDATE	LINE_ID	PART_ID
1	28-Jan-97	1	4
1	28-Jan-97	2	2
1	28-Jan-97	3	3
2	28-Jan-97	1	4

Views

- **Updatable Views:**
- Views can be used for data manipulation
- View on which data manipulation can be done are called updatable views
- Views defined from single table
- **Inserts a row into the key-preserved ORDERS table**
- INSERT INTO order report (order_id, orderdate) VALUES (23, SYSDATE)
- **Updates a column of the key-preserved ORDERS table**
- UPDATE orderreport SET orderdate = SYSDATE WHERE order_id = 22;

Views

- **Delete a row from a view**
- Delete from order_report where orderid=5;
- Destroying a view:
- Syntax: drop view < view name>;
- Example:
- Drop view Branch;

Views

- **Common restrictions on updatable views:**
- For the view definition must not include:
- Aggregate functions
- Distinct, group by or having clause
- Subqueries
- If a view is defined from another view , the second view should be updatable

Indexes

- Indexing a table is an access strategy that is a way to sort and search records in the table
- Indexes are essential to improve the speed with which records can be located and retrieved from a table
- An index is an ordered list of the contents of a column or group of columns of a table
- Indexing involves forming a two dimensional matrix completely independent of the table on which the index is being created

Indexes

- **Creating Indexes For Table Columns**
- CREATE INDEX index1 ON table (column1, column2, ...);
- **Dropping Indexes**
- DROP INDEX <index name>

PL/SQL

- SQL does not have any procedural capabilities such as looping and branching nor does it have any conditional checking capabilities vital for data testing before storage.
- For this, Oracle provides PL/SQL. Programmers can use it to create programs for validation and manipulation of table data.

PL/SQL

- PL/SQL is basically an extension of SQL. Thus it permits the use of:
- SQL data manipulation statements like INSERT, SELECT, UPDATE and DELETE
- SQL transaction processing statements like COMMIT, ROLLBACK
- SQL functions
- SQL predicates

- Comparison operators like BETWEEN.. AND, EXISTS etc.

- Logical operators AND, OR and NOT

-Combined with programming constructs like:

a) Assignment statements (A:= B - C)

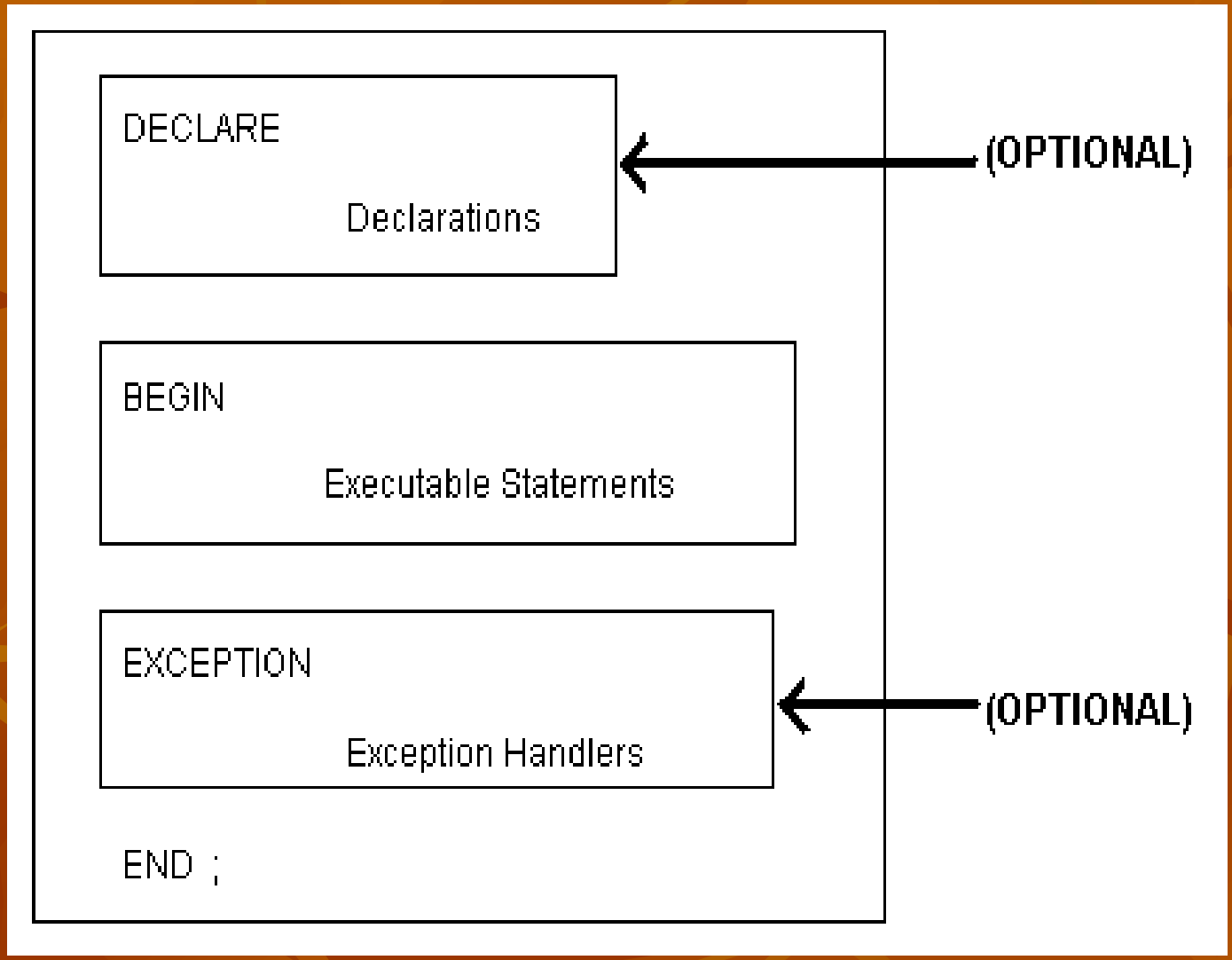
b) Flow of control statements (IF...THEN..ELSE...)

c) Iterative statements (FOR..LOOP, WHILE..LOOP).

PL/SQL

- **PL/SQL Block**
- A standard PL/SQL code segment is called a block. A standard PL/SQL block is made up of three sections.
- 1. **Declaration Section**
- 2. **Executable Statements**
- 3. **Exception Handling Section**

PL/SQL Block Structure



Advantages of PL/SQL

- Procedural Capabilities
- Improved Performance
- Enhanced Productivity
- Portability
- Integration with RDBMS

Data Types: PL/SQL

- NUMBER Type
- DEC
- DECIMAL
- DOUBLE PRECISION
- FLOAT
- INTEGER
- INT
- NUMERIC
- REAL
- LONG RAW Type
- Constants
- CHAR Type
- VARCHAR2 Type
- BOOLEAN Type
- DATE Type
- SMALLINT
- LONG Type

Declaration Using Attributes

- **%TYPE attribute**
- This attribute provides the data type of a variable, constant or column.
- **Example:**
- m_empno emp.empno%TYPE;
- In the above example the variable m_empno has the same data type and size as the column empno in table emp.

Operators

- **Comparison**
- Relational Operators
- <
- >
- =
- <=
- >=
- !=

- **Is NULL Operator / Not NULL**
- Like Operator
- BETWEEN Operator
- IN Operator
- **Logical**
- AND
- OR
- NOT

Conditional Control

```
if (condition) then
   statements
else
   statements
end if;
```

```
if (condition) then
        statements
elsif (condition) then
        statements
else
        statements
end if;
```

- Example:
- IF TRANTYPE = 'd'
- THEN TOT: = TOT + AMT;
- ELSIF TRANTYPE = 'W'
- THEN TOT: = TOT - AMT;
- END IF;

Iterative Control

1) Simple Loop

- LOOP

  statements

  .....

  END LOOP;
- LOOP and END LOOP are the keywords.
- **Examples:**

1) LOOP

ctr:= ctr + 1;

IF (ctr > 5) THEN

END IF;

END LOOP;

2) LOOP

ctr:= ctr + 1;

EXIT WHEN ctr > 5; EXIT ;

END LOOP;

2. For Loop

```
FOR i IN 1..10
LOOP
statements
.....
END LOOP;
```

3. While Loop

```
WHILE  (condition)
LOOP
.........
statements
.........
END LOOP;
```

Example:

```
WHILE i < 100
LOOP
INSERT INTO TEMP
VALUES(i);
i:=i+1;
END LOOP;
```

REVERSE: Reversing the loop

```
DECLARE
      loop_start Integer := 1;
BEGIN
   FOR i IN REVERSE loop_start..5
   LOOP
   DBMS_OUTPUT.PUT_LINE('Loop counter is ' || i);
   END LOOP;
END;
```

Pl/SQL database objects

- Functions
- Stored Procedure
- Database triggers

Pl/SQL database objects

- A stored procedure or function is a logically grouped set of SQL and PL/SQL statements that perform a specific task
- It has been compiled and stored in one of the oracle engine's system tables
- Procedure and functions are made up of
  - A declarative part
  - An executable part
  - An optional exception handling part

Advantages of a procedure or function

- Security: Enforces data security
- Performance: Amount of information sent over a network is less
- Memory allocation: The amount of memory used reduces as stored procedures or functions have shared memory capability
- Productivity: By writing procedures and functions redundant coding can be avoided, increasing productivity
- Integrity: a procedure or function needs to be tested only once to guaranty that it returns an accurate result

Creating A Function

- Syntax:

```
Create or replace Function <Fname>
        (< argument> <data type>)
        Return <data type>  AS
        <variable> declaration;
        <constant> declaration;
Begin
        <Pl/SQL subprogram body>;
Exception
        <exception Pl/SQL block>;
End;
```

Creating stored procedures

- Syntax:

```
CREATE OR REPLACE PROCEDURE proc_name (
argument {IN,OUT,IN OUT} data type )
AS
    variable declaration
BEGIN
    <Pl/SQL subprogram body>;
Exception
    <exception Pl/SQL block>;
End;
```

Creating stored procedures

- Replace: recreates the procedure if it already exists. If a procedure is redefined the oracle engine recompiles it
- In: indicates that the parameters will accepts a value from the user
- Out: indicates that the parameter will return a value to the user
- IN Out: indicates that the parameter will either accept a value from the user or return a value to the user

Function Example

```
create or replace function binary (no number)
return varchar as
a varchar(10);
b varchar(10);
C NUMBER(6);
begin
c := no;
    while c >= 1
    loop
            a:= mod(c,2);
            c := trunc(c/2);
            b := a||b;
    end loop;
    return b;
end;
```

- select binary(15) from dual;

Procedure Example

```
CREATE OR REPLACE PROCEDURE my_first_proc
      (p_name IN VARCHAR2 := 'Lewis',
       p_address IN VARCHAR2 := '123 Mockingbird Ln',
       p_an_in_out_parameter IN OUT NUMBER,
       p_an_out_parameter OUT DATE )
AS

 v_a_variable VARCHAR2(30);

BEGIN
 IF p_name = 'Lewis'
 THEN
   DBMS_OUTPUT.PUT_LINE( p_name || ': ' ||
         p_address );
 END IF;

 v_a_variable := 99;
 p_an_in_out_parameter := v_a_variable;
 p_an_out_parameter := SYSDATE;

END;
```

```
DECLARE
  v_employee VARCHAR2(30) := 'BillyBob';
  v_number NUMBER := 22;
  v_date DATE;

BEGIN    my_first_proc(
      p_name => v_employee,
      p_an_in_out_parameter => v_number,
      p_an_out_parameter => v_date );

       DBMS_OUTPUT.PUT_LINE(
         v_employee || ', ' ||
         to_Char(v_number) || ', ' ||
         to_char(v_date) );

     my_first_proc(       p_an_in_out_parameter => v_number,
      p_an_out_parameter => v_date );

  DBMS_OUTPUT.PUT_LINE( v_employee || ', ' ||
        to_Char(v_number) || ', ' ||
        to_char(v_date) );

END;
```

Procedures verses functions

- A function must return a value back to the caller. A function can return only one value to the calling PL/SQL block
- A stored procedure, returning value is not mandatory. There is no return statement. By defining multiple out parameters in a procedure, multiple values can be passed to the caller.
- A function is called as

  select function_name(value) from dual;
- A procedure is called as
- Call procedure_name(values);

Database Triggers

- Database triggers are the database objects created on the client and stored on the server in the oracle engine's system table
- These consists of the following distinct sections:
- A named database event
- A PL/SQL block that will execute when the event occurs
- The occurrence of the database event is strongly bound to table data being changed

Uses of database Triggers

- It provides a highly customizable database management system
- A trigger can permit DML statements against a table only if they are issued, during regular business hours or on predetermined weekdays
- It can be used to keep an audit trail of the table
- It can be used to prevent invalid transactions
- Enforce complex security authorization

Database Triggers

- When a trigger is fired, an SQL statement inside the trigger's Pl/SQL code block can also fire the same or some other trigger. This is called cascading triggers
- Excessive use of triggers for customizing the database can result in complex interdependencies between the triggers, which may be difficult to maintain in a large application

Database Triggers V/S Procedures

- Triggers do not accept parameters whereas procedures can
- A triggers is executed implicitly by the oracle engine itself upon modification of an associated table or its data
- To execute a procedure, it has to be explicitly called by a user

How to apply database triggers

- A trigger has three basic parts:
- **Triggering Event or statement:**
- It is a SQL statement that causes a trigger to be fired
- It can be insert, update or delete statement for a specific table

How to apply database triggers

- **Trigger Restriction:**
- A trigger restriction specifies a Boolean expression that must be true for the trigger to fire
- It is an option available for triggers that are fired for each row
- Its function is to conditionally control the execution of a trigger
- A trigger restriction is specified using a when clause

How to apply database triggers

- **Trigger Action:**
- A trigger action is the Pl/SQL code to be executed when a triggering statement is encountered and any trigger restriction evaluates to true
- The PL/SQL block can contain SQL and PL/SQL statements, can define Pl/SQL language constructs and can call stored procedures

Types of Triggers

- Row trigger
- Statement trigger
- Before trigger
- After trigger
- Combination trigger
  - Before statement trigger
  - Before row trigger
  - After statement trigger
  - After row trigger

Syntax for creating trigger

Create or replace trigger [schema.] <TriggerName>
{Before, After}
{delete, insert, update [of column,….]}
On [schema.] <table name>
[Referencing {OLD as old, NEW as new}]
[For each row [When condition]]
Declare
    <variable declaration>;
    <variable declaration>;
Begin
    <PL/SQL subprogram body>;
Exception
    <Exception PL/SQL block>;
End;

Keyword and parameters

- Replace: recreates the trigger if it already exists.
- Schema: which contain the trigger
- Before: indicates that the oracle engine fires the trigger before executing the triggering statements
- After: indicates that the oracle engine fires the trigger after executing the triggering statements
- Delete: indicates that oracle engine fires the triggers whenever a delete statement removes a row from the table
- Insert: indicates that oracle engine fires the triggers whenever an insert statement adds a row to a table

Keyword and parameters

- Update: indicates that oracle engine fires the triggers whenever an update statement changes a value in one of the columns specified
- On: specifies the schema and name of the table, which the trigger is to be created
- Referencing: specifies correlation names
- For each row: designates the trigger to be a row trigger
- When: specifies the trigger restriction
- Pl/SQL block: the oracle engine executes when the trigger is fired

Trigger example (mysql)

Create a trigger for table emp. The trigger must keep track of the records that are being deleted. When a record deleted, the original record details and database (user) and the date of operation are stored in the demp table.

```
mysql> desc emp;
+--------+-------------+------+-----+---------+-------+
| Field  | Type        | Null | Key | Default | Extra |
+--------+-------------+------+-----+---------+-------+
| empno  | int(11)     | NO   | PRI |         |       |
| ename  | varchar(20) | YES  |     | NULL    |       |
| job    | varchar(20) | YES  |     | NULL    |       |
| deptno | int(11)     | YES  |     | NULL    |       |
| salary | decimal(7,2)| YES  |     | NULL    |       |
| comm   | decimal(7,2)| YES  |     | NULL    |       |
+--------+-------------+------+-----+---------+-------+
6 rows in set (0.00 sec)
```

```
mysql> desc demp;
    -> //
+--------+-------------+------+-----+---------+-------+
| Field  | Type        | Null | Key | Default | Extra |
+--------+-------------+------+-----+---------+-------+
| empno  | int(11)     | NO   | PRI |         |       |
| ename  | varchar(20) | YES  |     | NULL    |       |
| job    | varchar(20) | YES  |     | NULL    |       |
| deptno | int(11)     | YES  |     | NULL    |       |
| salary | decimal(7,2)| YES  |     | NULL    |       |
| comm   | decimal(7,2)| YES  |     | NULL    |       |
| uname  | varchar(20) | YES  |     | NULL    |       |
| dtime  | datetime    | YES  |     | NULL    |       |
+--------+-------------+------+-----+---------+-------+
8 rows in set (0.00 sec)
```

Trigger example (mysql)

- delimiter //
  create trigger t1
  after delete on emp for each row
  begin
  insert into demp values (old.empno, old.ename, old.job, old.deptno, old.salary, old.comm, database(), curdate());
  end
  ;
- mysql> delete from emp where empno=1;
- Query OK, 1 row affected (0.03 sec)
- mysql> select * from demp;
- | 1 | seema | faculty | 10 | 25000.00 | NULL | test | 2010-09-25 00:00:00 |

Query languages

- A database query language is an integral part of a DBMS enabling users to interact with the system
- A query language must be simple to learn & to use
- Different relational query languages are SQL, QBE, QUEL.

Introduction to QBE and QUEL

- QUEL is the language based on tuple relational calculus which actually uses the range variables as in tuple calculus
- Syntax:
- RANGE OF <variable name> IS <relation name>
- Then it uses
  - RETRIEVE <list of attributes from range variables> WHERE <conditions>

Introduction to QBE and QUEL

- Another variation of relational calculus called the domain relational calculus, or simply, domain calculus is equivalent to tuple calculus and to relational algebra.
- The language called QBE (Query-By-Example) that is related to domain calculus was developed almost concurrently to SQL at IBM Research, Yorktown Heights, New York.
  - Domain calculus was thought of as a way to explain what QBE does.
- Domain calculus differs from tuple calculus in the type of variables used in formulas:
  - Rather than having variables range over tuples, the variables range over single values from domains of attributes.
- To form a relation of degree n for a query result, we must have n of these domain variables— one for each attribute.

QBE: A Query Language Based on Domain Calculus

- The language is very user-friendly, because it uses minimal syntax.
- QBE was fully developed further with facilities for grouping, aggregation, updating etc. and is shown to be equivalent to SQL.
- The language is available under QMF (Query Management Facility) of DB2 of IBM and has been used in various ways by other products like ACCESS of Microsoft, PARADOX.