Unit 7

Object Oriented Database Vs Object Relational Database

Basis of comparison	Object Oriented Database (OODBMS)	Objected Relational Database (ORDBMS)
Connection between two relations	Relationships are represented by references via the object identifier (OID).	Connections between two relations are represented by foreign key attributes

		in one relation that reference the primary key of another relation
Data storage structure	It employs indexing techniques to locate disk pages that store the object. Therefore, they are able to provide persistent storage for complex-structured objects	It does not specify any data storage structure, each base relation is implemented as separate file and therefore, they are unable to provide persistent storage for complex-structured objects
Quantity of Data	Handles larger and complex data than RDBMS	Handles comparatively simpler data
Constraints	The constraints supported by this system vary from system to system	It has keys, entity integrity and referential integrity.

Data Manipulation Language	The data management language is typically incorporated into a programming language such as #C++.	There are data manipulation languages such as SQL, QUEL and QBE which are based on relational calculus.
Description of Stored Data	Stores data entries are described as object	Stores data in entries is described as tables.
Type of Data	Object oriented database can handle different types of data.	Relational database can handle a single type of data.
Data Storage	The data is stored in the form of objects	Data is stored in the form of tables, which contains rows and columns.

Q) Oids and Reference Types :

In SQL, reference types can be used to define relationships between row types and uniquely identify a row within a

table.

Reference type value can be stored in one table and used as a direct reference to a specific row in some base table defined to be of this type (similar to pointer type in 'C'/C++).

In this way, reference type provides similar functionality as OID of OODBMS.

Thus, references allow a row to be shared among multiple tables, and enable users to replace complex join definitions in queries with much simpler path expressions.

References also give optimizer alternative way to navigate data instead of using value-based joins.

REF IS SYSTEM GENERATED in CREATE TYPE indicates that actual values of associated REF type are provided by the system.

Example

```
CREATE TYPE staff-type UNDER person-type
AS (
    sno VARCHAR(5) NOT NULL UNIQUE,
    position VARCHAR(10) NOT NULL,
    salary NUMBER(7,2),
    next-of-kin REF(person-type) -> Could point to a row in any table containing a person-type!
    bno VARCHAR(3) NOT NULL)
NOT FINAL;
```

CREATE TABLE person OF person_type (
oid REF(person_type) VALUES ARE SYSTEM GENERATED);

To ensure that a REFERENCE is limited to a single table, a SCOPE has to be added to the table using the REFERENCE!

Example:

CREATE TYPE staff_type UNDER person_type AS (
sno VARCHAR(5) NOT NULL UNIQUE,
position VARCHAR(10) NOT NULL,
salary NUMBER(7,2),
next_of_kin REF(person_type) → staff next_of_kin will point to a row in the person table!)
bno VARCHAR(3) NOT NULL)
NOT FINAL;

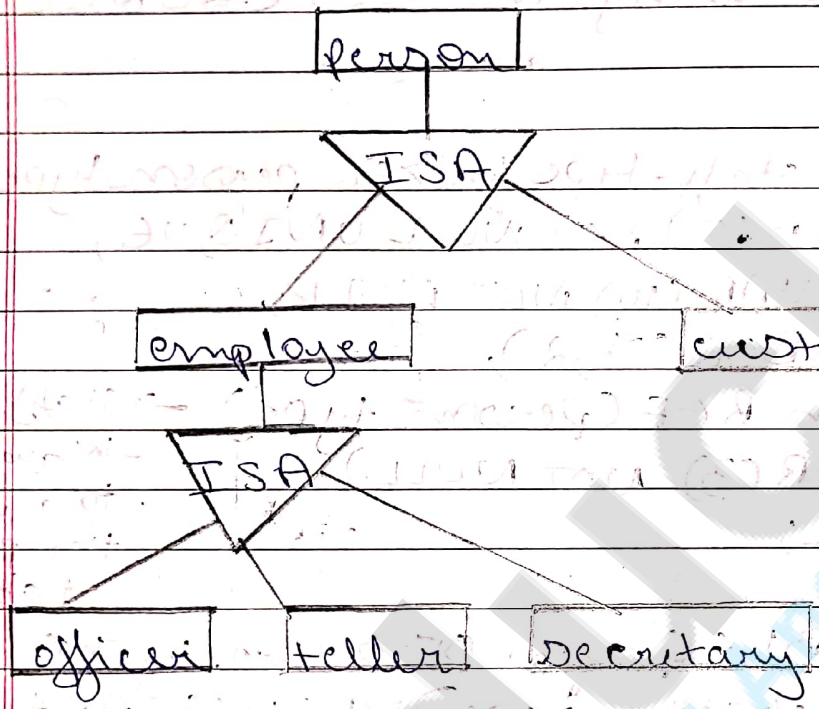
CREATE TABLE person OF person_type(
oid REF(person_type) VALUES ARE SYSTEM GENERATED);

CREATE TABLE staff OF staff_type(
PRIMARY KEY sno,
SCOPE FOR next_of_kin IS person);

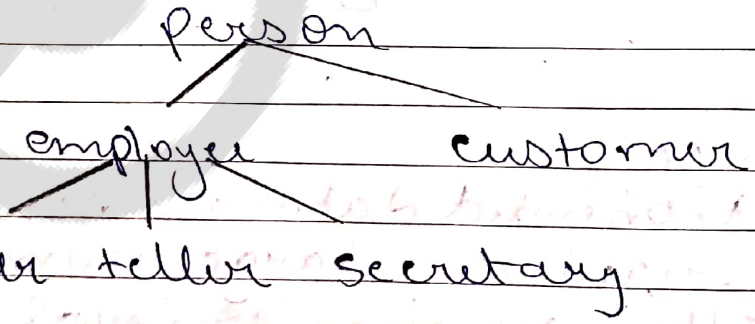
Inheritance:

An object-oriented database schema typically requires a large number of classes. Often, however, several classes are similar. For example, bank employees are similar to customers.

In order to allow the direct representation of similarities among classes, we need to place classes in a specialization hierarchy.
Below diagram is a specialization hierarchy for the ER model.



The concept of a class hierarchy is similar to that of specialization in the ER model. The corresponding class diagram is show below.



The class hierarchy can be defined in pseudo-code in which the variables

associated with each class are as follows.

```
class person{
    string name;
    string address;
};
class customer isa person{
    int credit-rating;
};
class employee isa person{
    date start-date;
    int salary;
};
class officer isa employee{
    int office-number;
    int expense-account-number;
};
class teller isa employee{
    int hours-per-week;
    int station-number;
};
class secretary isa employee{
    int hours-per-week;
    int manager;
};
```

- The keyword isa is used to indicate that a class is a specialization of another class. The specialization of a class are called subclasses. Eg employee is a subclass of person; teller is a subclass of employee. Conversely, employee is a

superclass of teller.
- Class hierarchy and inheritance of properties from more general classes. Eg an object representing an officer contains all the variables of classes officer, employee and person. Methods are inherited in a manner identical to inheritance of variables.
- An important benefit of inheritance in OO systems is the notion of substitutability. Any method of a class, A, can be equally well be invoked with an object belonging to any subclass B of A. This characteristic leads to code-reuse: methods and functions in class A (such as get-name() in class person) do not have to be rewritten again for objects of class B).

Two plausible ways of associating objects with nonleaf classes:

- associate with the employee class, all employee objects including those that are instances of officer, teller and secretary.
- associate with the employee class, only those employee objects that are instances neither officer, nor teller, nor secretary[3].

Typically, the latter choice is made in OO systems. It is possible to determine the set of all employee

objects, in this case by taking the union of those objects associated with all classes in the subtree rooted at employee.

Most OO systems allow specialization to be partial i.e they allow objects that belong to a class such as employee that do not belong to any of that class's subclasses.

Q) Database object

A database object is any defined object in a database that is used to store or reference data. Anything which we make from create command is known as Database Object. It can be used to hold and manipulate the data. Some of the examples of database objects are: view, sequence, indexes, etc.

- Table - Basic unit of storage; composed rows and columns.
- View - Logically represents subsets of data from one or more tables
- Sequence - Generates primary key values
- Index - Improves the performance of some queries
- Synonym - Alternative name for an object

Different database Objects:

1) Table - This database object is used

to create a table in database.

Syntax:

CREATE TABLE [schema.]table (column datatype [DEFAULT expr][, ...]);

Example:

```
CREATE TABLE dept
    (deptno NUMBER(2),
     dname VARCHAR2(14),
     loc VARCHAR2(13));
```

2) View - This database object is used to create a view in database. A view is a logical table based on a table or another view. A view contains no data of its own but is like a window through which data from tables can be viewed or changed. The tables on which a view is based are called base tables. The view is stored as a SELECT statement in the data dictionary.

Syntax:

```
CREATE [OR REPLACE] [FORCE|NO FORCE]
VIEW view
    [(alias [, alias]...)]
  AS subquery
   [WITH CHECK OPTION [CONSTRAINT constraint]]
   [WITH READ ONLY [CONSTRAINT
```

constraint]];

Example:

```
CREATE VIEW salvu50
    AS SELECT employee_id ID_NUMBER,
last_name NAME,
    salary * 12 ANN_SALARY
    FROM employees
    WHERE department_id = 50;
```

3) Sequence - This database object is used to create a sequence in database. A sequence is a user created database object that can be shared by multiple users to generate unique intigers. A typical usage for sequences is to create a primary key value, which must be unique for each row. The sequence is generated and incremented (or decremented) by an internal Oracle routine.

Syntax:

```
CREATE SEQUENCE sequence
    [INCREMENT BY n]
    [START WITH n]
    [{MAXVALUE n | NOMAXVALUE}]
    [{MINVALUE n | NOMINVALUE}]
    [{CYCLE | NOCYCLE}]
    [{CACHE n | NOCACHE}];
```

Example:

```
CREATE SEQUENCE dept_deptid_seq
    INCREMENT BY 10
    START WITH 120
    MAXVALUE 9999
    NOCACHE
    NOCYCLE;
```

Check if sequence is created by:
SELECT sequence_name, min_value, max_value, increment_by, last_number FROM user_sequences;

4) Index - This database object is used to create a indexes in database. An Oracle server index is a schema object that can speed up the retrieval of rows by using a pointer. Indexes can be created explicitly or automatically. If you do not have an index on the column, then a full table scan occurs. An index provides direct and fast access to rows in a table. Its purpose is to reduce the necessity of disk I/O by using an indexed path to locate data quickly. The index is used and maintained automatically by the Oracle server. Once an index is created, no direct activity is required by the user. Indexes are logically and physically independent of the

table they index. This means that they can be created or dropped at any time and have no effect on the base tables or other indexes.

Syntax:

```
CREATE INDEX index
  ON table (column[, column]...);
```

Example:

```
CREATE INDEX emp_last_name_idx
  ON employees(last_name);
```

5) Synonym - This database object is used to creati a index in database. It simplify access to objects by creating a synonym (another name for an object). With synonyms, you can ease referring to a table owned by another user and shorten lengthy object names. To refer to a table owned by another user, you need to prefix the table name with the name of the user who created it followed by a period. Creating a synonym eliminates the need to qualify the object name with the schema and provides you with an alternative name for a table, view, sequence, procedure, or other objects. This method can be especially useful with lengthy object names, such as views.

In the syntax :
PUBLIC : creates a synonym accessible to all users.
synonym: is the name of the synonym to be created
object: identifies the object for which the synonym is created.

Syntax:
CREATE [PUBLIC] SYNONYM Synonym FOR object;
Example:
CREATE SYNONYM d_sum FOR dept_sum_vu;

9] Structured Data Types :-
A structured ~~dt~~ data type is a user defined data type with elements that are not atomic rather they are divisible and can be used either separately or as a single unit as per requirements. It is a form of user defined object that contains a sequence of attributes, each of which has a data type. An attribute is a property that helps to describe an instance of a particular type, for example if we want to define a structured type called address to store addresses, in which city might be one of the attributes of this structured

type:
A structured data type can be used as the type for a column in a regular table, the type for an entire table or an attribute of another structured type. When used as the type for a table, the table is known as typed table.
CREATE TYPE statement is used to create a structured data type and DROP statement is used to delete the structured data type.
SQL allows user to define 'distinct' types but they are confined to relational model as these data types are atomic.
SQL also provides the facility of defining structured types which extends the relation model, that deals with atomic values only. When we create such structured type, SQL creates a constructor function for the type and creates both 'mutator' and 'observer' methods for the attribute of type.
Constructor function has the same name as structured type with which it is associated. The constructor function has no parameter and returns an instance of type with all of its attribute set to null values.
MUTATOR method exists for each attribute of a structured type. When a mutator method is invoked on an ~~instante~~ instance of structured

type and specify a new value for its associated attribute method, returns a new instance with the attribute update to a new value.
When an 'OBSERVER' method is invoked on an instance of structured type, the method returns the value of attribute for that instance. Structured types available in SQL are ROW(f1 t1, f2 t2 ... fn tn) -

1) It represents a row, or a tuple of fields f1, f2, ... fn of types t1, t2, ... tn respectively. 'ROW' data type specifies every table as a collection of rows or every table as set of rows or multi-set of rows, for example, the 'address-t' is declared as of ROW data type as show below which contains area, city and state as its components :-

```
CREATE TTOE address-t AS ROW
(area : varchar(20),
 city : varchar(20),
 state : varchar(20))
```

2) ARRAY[i] : It represents an array of 'i' items of 'base' type for example, the 'objects' field of CLIP table used an array of 10 objects, each of which is of varchar(20) type. A multidimensional array

can not be created in SQL. An array can be used as component in ROW type as shown but not in array type :-
ROW (pno: integer, object : varchar(20) ARRAY [10])

3) list of (base): It represents a list of all items of 'base' type, for example:
PROJECT (Projno: integer, Pname: varchar(25), Empno: list of (integer))

4) set of (base): It represents a set of 'base' type items. A set does not contain duplicate elements unlike lists otherwise it is used in the same manner as list.

5) bag of (base): It represents a bag or a multi-set of base type items.
Collection type or build data types are types using list of, set of, bag of and ARRAY. But SQL does not provide any efficient method for manipulation of these collection type objects.

Q) Operation on Structured Data :-
Structured data can be manipulated using built in methods for types defined using type constructor. These methods are similar to operations used for

data types (atomic) of traditional RDBMS.

1) Operations on Arrays

Array is used in the same manner as in traditional RDBMS. 'Array index' method is used to return the number of elements in the array for example. Suppose we want to find those projects whose clips contain more than 10 items or objects then following query can be used:

```
SELECT P.Pname, P.Projno
FROM project P, Clip C
WHERE CARDINALITY(C.Objects) >
10 AND C.Projno = P.Projno
```

The above query select project name and projectno from "PROJECT" whose clips contain more then 10 items which can calculatid by using CARDINALITY operation.

2) Operations on Rows

Row type is a collection of fields values whose each fields can be accessed by the same traditional notation for example, address-t.city specify the attribute 'city' of the type address-t. when operation is applied on collection of

rows then result obtained is also a collection of values.
If a column or field whose type is ROW (f1 t1, f2 t2, ... fn tn) and c1.fk gives us a list of values whose type is tk. If c1 is a set of rows or a bag of rows then c1.fk give us a set of values of type tk.
Consider 'Emp-Dept' schema in which we have to find the names of those employees who resides in 'Malviya Nagar' of 'New Delhi'.

```
SELECT E.Empno, E.Name
FROM Emp E
WHERE E.Address.area = 'Malviya Nagar'
AND E.Address.city = 'New Delhi'
```

~~AND E.Address.city =~~

3) Operations on Sets and Multi-sets
Set and multisets are used in the traditional manner by using =, <, >, ≤, ≥ comparison operators. An item of a set can be compared by other items using ∈ (belongs to) relation. Two set objects can create a new object using ∪, (Union Operation). They can also create a new object by subtracting a set of elements from other set by using '-' (set difference operator). Multi-set also uses the same operations as used by the sets but the operations are applied on the number of copies of element into

account

4) Operations on lists

list includes operations like 'append', 'concatenate', 'head', 'tail' etc to manipulate the items of list for example, 'concatenate' or 'append' appends one list to another, 'head' returns the first element of list, 'tail' returns the list after removing the first element.