Sample Questions on NOSQL, OODBMS, DW, DDBMS

Sample question Object Database

1. A car has engine of type Engine, chasis has type Chasis, image, colour, height, length and width. There are two method for car: store details() and get details(). An engine has piston of type Piston, tank of type Tank and horsepower of type float. A piston type has attributes length, diameter and ignition type. A Tank type has volume and type.
   a. Define Car type for the above.

```
CREATE TYPE Piston AS (
   length DECIMAL(5, 2),
   diameter DECIMAL(5, 2),
   ignition_type VARCHAR(50)
);

CREATE TYPE Tank AS (
   volume DECIMAL(6, 2),
   type VARCHAR(50)
);

CREATE TYPE Chasis AS (
   serial_number VARCHAR(100)
);

CREATE TYPE Engine AS (
   piston Piston,
   tank Tank,
   horsepower FLOAT
);

CREATE TYPE Car AS (
   engine Engine,
   chasis Chasis,
   image VARCHAR(255),
   colour VARCHAR(50),
   height DECIMAL(5, 2),
   length DECIMAL(5, 2),
   width DECIMAL(5, 2),
   METHOD store_details() RETURNS VARCHAR(4000),
   METHOD get_details() RETURNS VARCHAR(4000)
) NOT FINAL;
```

   b. Create table for car type and insert a tuple into that table.

```sql
CREATE TABLE CarTable OF Car (
   PRIMARY KEY (REF IS car_ref SYSTEM GENERATED)
);

INSERT INTO CarTable (engine, chasis, image, colour, height, length, width)
VALUES (
   Engine(
      Piston(10.5, 8.0, 'Spark'),
      Tank(60.0, 'Gasoline'),
      300.0
   ),
   Chasis('VIN123ABC456DEF789'),
   '/images/cars/sedan_red.jpg',
   'Red',
   1.45,
   4.88,
   1.84
);
```

2. Explain the similarities and differences of OODBMS and RDBMS.

- **Similarities between OODBMS and RDBMS**

Feature	OODBMS	RDBMS
Data Storage	Stores data persistently	Stores data persistently
Querying	Supports queries (some SQL-like or OQL)	Supports queries (SQL)
ACID Properties	Maintains ACID properties	Maintains ACID properties
Security	Offers user authentication and access control	Offers user authentication and access control
Backup and Recovery	Provides backup and recovery options	Provides backup and recovery options

- **Differences between OODBMS and RDBMS**

Feature	OODBMS	RDBMS
Data Model	Object-oriented model (objects, classes, inheritance)	Relational model (tables, rows, columns)
Complex Data Types	Directly supports complex types (lists, arrays, etc.)	Requires normalization and table joins
Integration with Programming Languages	Tight integration with languages like Java, C++	Separate from application code (requires mapping)
Query Language	Object Query Language (OQL) or native methods	Structured Query Language (SQL)

Schema Evolution	Easier to modify schemas as objects evolve	Schema changes can be complex and rigid
Performance	High for complex objects and navigation-based queries	High for large-scale transaction processing
Use Cases	CAD/CAM, telecommunications, multimedia, AI	Banking, finance, HR systems, data warehousing

3. A tuple is given for customer relation as follows

c-id	**Name (first-name, middle-name, last-name)**	**Phones (maximum three numbers)**	**Choices (set of strings)** Maximum 10
C-00001	[Mohammad, Sajid, Abdullah]	01556111111 01715111111 01930111111	{PC, laptop, smartphone}

Create types for the above and constructor function for each type. Create table using the types. Insert the above data into the table using the constructor functions.

```
CREATE TYPE Name_Type AS (
   first_name  VARCHAR(50),
   middle_name VARCHAR(50),
   last_name   VARCHAR(50)
);

CREATE TYPE Customer_Type AS (
   c_id    VARCHAR(10),
   name    Name_Type,
   phones  VARCHAR(15)[3],     -- max 3 phone numbers
   choices VARCHAR(50)[10]     -- max 10 choices
);

CREATE TABLE Customer_Table OF Customer_Type (
   PRIMARY KEY (c_id)
);

INSERT INTO Customer_Table VALUES (
   'C-00001',
   Name_Type('Mohammad', 'Sajid', 'Abdullah'),
   ARRAY['01556111111', '01715111111', '01930111111'],
   ARRAY['PC', 'laptop', 'smartphone']
);
```

4. Create the above table using row type.

```
CREATE TABLE Customer_Using_RowType (
   c_id    VARCHAR(10) PRIMARY KEY,
   name    ROW (
            first_name  VARCHAR(50),
            middle_name VARCHAR(50),
            last_name   VARCHAR(50)
         ),
   phones  VARCHAR(15)[3],
   choices VARCHAR(50)[10]
);

INSERT INTO Customer_Using_RowType VALUES (
   'C-00001',
   ROW('Mohammad', 'Sajid', 'Abdullah'),
   ARRAY['01556111111', '01715111111', '01930111111'],
   ARRAY['PC', 'laptop', 'smartphone']
);
```

5. Explain type inheritance and table inheritance with examples.

- Type Inheritance Example:

```
CREATE OR REPLACE TYPE person_typ AS OBJECT (
     idno NUMBER,
     name VARCHAR2(30),
     phone VARCHAR2(20)) NOT FINAL;

CREATE TYPE student_typ UNDER person_typ (
     dept_id NUMBER,
     major VARCHAR2(30) NOT FINAL;


CREATE OR REPLACE TYPE employee_typ UNDER person_typ (
     emp_id NUMBER,
     mgr VARCHAR2(30));

CREATE TYPE part_time_student_typ UNDER student_typ (
number_hours NUMBER);

CREATE TABLE person_obj_table OF person_typ;
INSERT INTO person_obj_table VALUES (person_typ(12, 'Bob Jones', '650-555-0130'));
INSERT INTO person_obj_table VALUES (student_typ(51, 'Joe Lane', '1-650-555-0140', 12,
'HISTORY'));
INSERT INTO person_obj_table VALUES (employee_typ(55, 'Jane Smith', '1-650-555-0144', 100,
'Jennifer Nelson'));
```

A base object type `person_typ` is defined with common attributes (`idno`, `name`, `phone`). Subtypes are then created using the `UNDER` keyword: `student_typ` and `employee_typ` inherit from `person_typ`, gaining its attributes and adding their own (`dept_id`, `major` for students, and `emp_id`, `mgr` for employees). A further subtype `part_time_student_typ` inherits from `student_typ`, adding `number_hours`. The table `person_obj_table` is created from the base type but can store instances of all its subtypes, demonstrating polymorphism—allowing different but related object types in the same table.

- Table Inheritance

```
CREATE OR REPLACE TYPE Person AS OBJECT (
   idno NUMBER,
   name VARCHAR2(30),
   phone VARCHAR2(20)
) NOT FINAL;

CREATE OR REPLACE TYPE Student UNDER Person (
   dept_id NUMBER,
   major VARCHAR2(30)
);

CREATE OR REPLACE TYPE Teacher UNDER Person (
   emp_id NUMBER,
   subject VARCHAR2(30)
);

CREATE TABLE people OF Person;
CREATE TABLE students OF Student UNDER people;
CREATE TABLE teachers OF Teacher UNDER people;

INSERT INTO people VALUES (Person(1, 'Alice Johnson', '123-456-7890'));
INSERT INTO students VALUES (Student(2, 'Bob Smith', '555-123-4567', 10, 'Computer Science'));
INSERT INTO teachers VALUES (Teacher(3, 'Dr. Karen Wu', '555-987-6543', 2001, 'Mathematics'));
```

A base object type `Person` is defined and extended by two subtypes: `Student` and `Teacher`, each adding specific attributes. Three tables are then created: `people` for the base type, and `students` and `teachers` as subtables using the `UNDER` clause, inheriting from `people`. This setup allows each subtype to be stored in its own table while maintaining a shared structure and enabling queries across the hierarchy—demonstrating object-oriented design within SQL.

6. Write SQL to find id and first name of all customers whose choice is ‘laptop’

```
SELECT c.c_id, c.first_name
FROM Customer c
JOIN Choice ch ON c.c_id = ch.c_id
WHERE ch.choice = 'laptop';
```

7. Using the reference type, define type Department (id, dept-name, location and head) and type Teacher (Tid, name and salary). Head will be a reference type and Dr. Abdul Matin of teacher table will be the head of ECE.

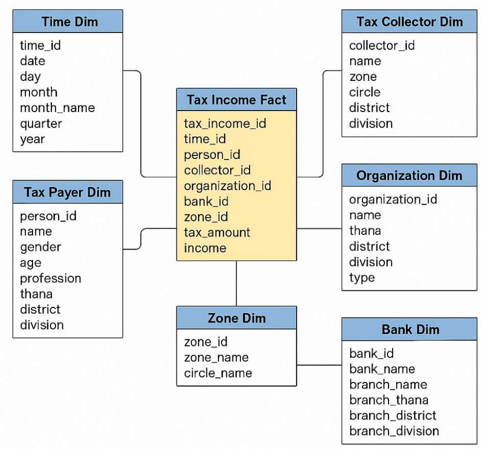
```
CREATE TYPE Teacher AS OBJECT (
  Tid VARCHAR(20),
  name VARCHAR(50),
  salary NUMBER
);

CREATE TABLE teachers OF Teacher (
  PRIMARY KEY (Tid)
);

CREATE TYPE Department AS OBJECT (
  id VARCHAR(20),
  dept_name VARCHAR(50),
  location VARCHAR(50),
  head REF Teacher
);

CREATE TABLE departments OF Department (
  PRIMARY KEY (id),
  SCOPE FOR (head) IS teachers
);

INSERT INTO teachers VALUES (
  'T01',
  'Dr. Abdul Matin',
  90000
);
```

9.

a. Design a star schema for a data warehouse for income tax and income analytics of Bangladesh. The measure attributes are tax amount and income. The dimensional attributes are time id, person id, collector id, organization id, bank id. You have to choose the attributes of dimension tables as per analytical requirements. You need to analyze thana, district, division, profession of tax payer, organization's location (thana, district, division), bank and branch wise, zone and circle wise analytics etc.
b. Perform rollup operations on your developed schema for the dimensional table time (date, day, month, year) and explain.
c. Perform cube operations on attributes (year, tax_payer_district, bank_name) and total tax. Explain the cube.
d. How will you use the star schema for decision support of National tax analysis?

10. Design star schema for telecom call record data for financial analytics, caller, callee analytics based on location (thana, district, division), operator analytics etc. The measure attributes are each call time and call cost.

a. Answer:



Income Tax and Income Analytics Diagram

b. Answer:

```
SELECT
    t.year,
    t.month,
    t.day,
    t.date,
    SUM(f.tax_amount) AS total_tax
FROM
    Tax_Income_Fact f
JOIN
    Time_Dim t ON f.time_id = t.time_id
GROUP BY
    ROLLUP (t.year, t.month, t.day, t.date)
ORDER BY
    t.year, t.month, t.day, t.date;
```

c. Answer:

```
SELECT
    TD.year,
    TPD.district AS tax_payer_district,
    BD.bank_name,
    SUM(TIF.tax_amount) AS TotalTax
FROM
    Tax_Income_Fact TIF
JOIN
    Time_Dim TD ON TIF.time_id = TD.time_id
JOIN
    Tax_Payer_Dim TPD ON TIF.person_id = TPD.person_id
JOIN
    Bank_Dim BD ON TIF.bank_id = BD.bank_id
GROUP BY
    CUBE (TD.year, TPD.district, BD.bank_name)
```

```
ORDER BY
    TD.year, TPD.district, BD.bank_name;
```

The CUBE operation applied to the attributes year, tax_payer_district, and bank_name calculates the aggregate TotalTax for every possible combination of these dimensions. This powerful function generates not only the total tax for the most detailed level (a specific year, district, and bank) but also includes all possible subtotals (e.g., total tax by year and district across all banks, total tax by bank across all years and districts, total tax just by year across all districts and banks, etc.) and the overall grand total, providing a comprehensive multi-dimensional summary in a single result set

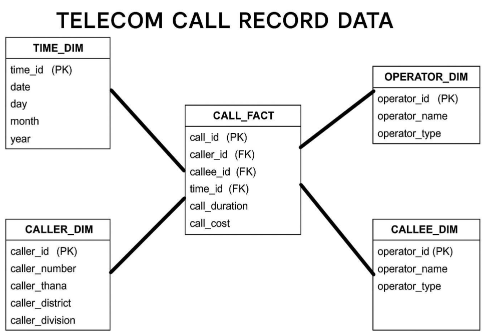
d. Answer:

```
-- Quick Aggregation: Total tax by Taxpayer District
SELECT
    TPD.district,
    SUM(TIF.tax_amount) AS TotalTaxAmount
FROM
    Tax_Income_Fact TIF
JOIN
    Tax_Payer_Dim TPD ON TIF.person_id = TPD.person_id
GROUP BY
    TPD.district
ORDER BY
    TotalTaxAmount DESC;

-- Trend Analysis: Annual Tax Collection Trend
SELECT
    TD.year,
    SUM(TIF.tax_amount) AS AnnualTotalTax
FROM
    Tax_Income_Fact TIF
JOIN
    Time_Dim TD ON TIF.time_id = TD.time_id
GROUP BY
    TD.year
ORDER BY
    TD.year;

-- Performance Monitoring: Total Tax Collected by Tax Collector
SELECT
    TCD.name AS CollectorName,
    SUM(TIF.tax_amount) AS TotalTaxCollected
FROM
    Tax_Income_Fact TIF
JOIN
    Tax_Collector_Dim TCD ON TIF.collector_id = TCD.collector_id
GROUP BY
```

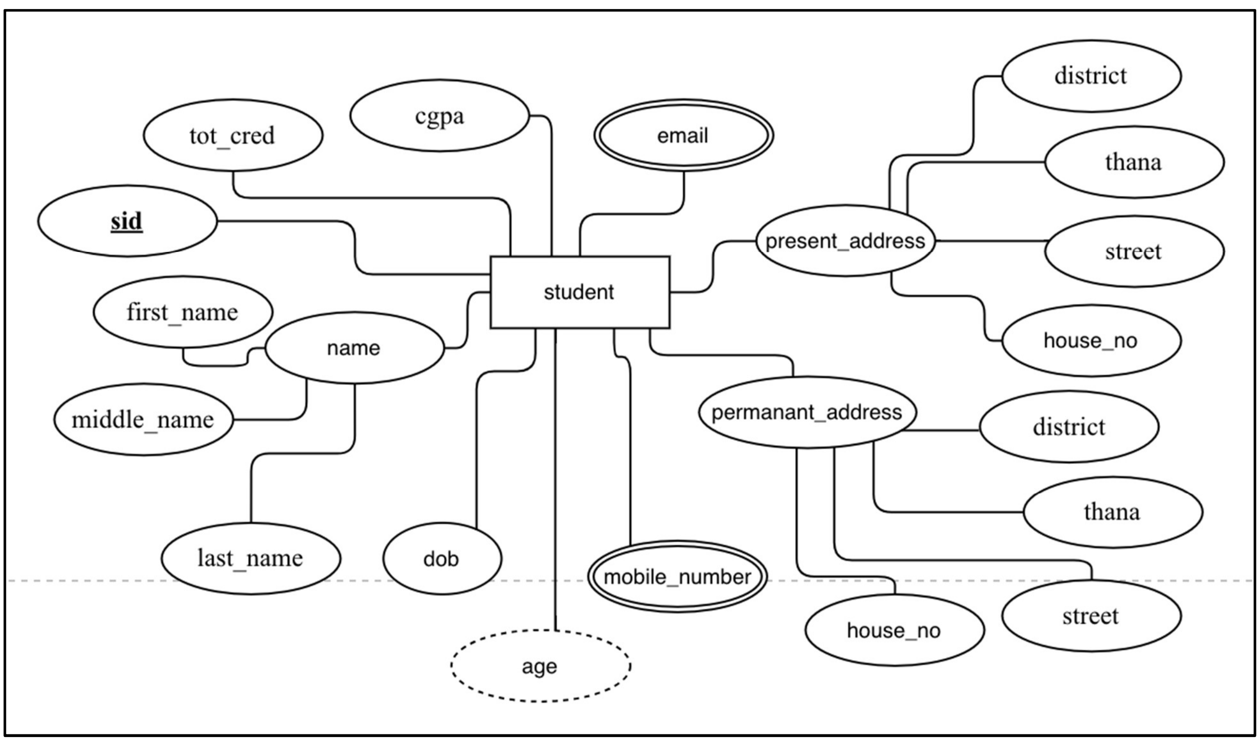
```
    TCD.name
ORDER BY
    TotalTaxCollected DESC;

-- Fraud Detection (Basic Example): Identify records with low tax relative to high income (e.g., less than 5% of income)
-- NOTE: A real fraud detection system would be more complex, involving ratios, thresholds, and potentially machine learning.
SELECT
    TIF.tax_income_id,
    TD.date,
    TPD.name AS TaxPayerName,
    TIF.income,
    TIF.tax_amount
FROM
    Tax_Income_Fact TIF
JOIN
    Time_Dim TD ON TIF.time_id = TD.time_id
JOIN
    Tax_Payer_Dim TPD ON TIF.person_id = TPD.person_id
WHERE
    TIF.income > 100000 -- Example: Focus on higher income individuals
    AND TIF.tax_amount < (TIF.income * 0.05); -- Example: Where tax paid is less than 5% of income

-- Policy Planning: Average Tax per Taxpayer Profession (Can help identify professions contributing significantly)
SELECT
    TPD.profession,
    AVG(TIF.tax_amount) AS AverageTaxAmount
FROM
    Tax_Income_Fact TIF
JOIN
    Tax_Payer_Dim TPD ON TIF.person_id = TPD.person_id
GROUP BY
    TPD.profession
ORDER BY
    AverageTaxAmount DESC;

-- Reporting: This encompasses many of the above. The CUBE query provided previously (9.c.) is a prime example for comprehensive reporting, showing aggregates across multiple dimensions simultaneously.
-- Example: A report showing total tax by taxpayer division and year
SELECT
    TD.year,
    TPD.division,
    SUM(TIF.tax_amount) AS TotalTax
FROM
    Tax_Income_Fact TIF
JOIN
    Time_Dim TD ON TIF.time_id = TD.time_id
```

JOIN
  Tax_Payer_Dim TPD ON TIF.person_id = TPD.person_id
GROUP BY
  TD.year, TPD.division
ORDER BY
  TD.year, TPD.division;

10.a. Answer:



11. The ERD given as follows:



Implement name, mobile number in

a. Relational model.
   CREATE TABLE STUDENT (

```
    sid INT PRIMARY KEY,
    first_name VARCHAR(50),
    middle_name VARCHAR(50),
    last_name VARCHAR(50),
);

CREATE TABLE STUDENT_MOBILE (
    student_sid INT,
    mobile_number VARCHAR(20),
    PRIMARY KEY (student_sid, mobile_number), same number listed twice
    FOREIGN KEY (student_sid) REFERENCES STUDENT(sid)
);
```

b. ORDBMS

```
CREATE TYPE name_t AS OBJECT (
    first_name VARCHAR2(50),
    middle_name VARCHAR2(50),
    last_name VARCHAR2(50)
);

CREATE TYPE mobile_varray_t AS VARRAY(5) OF VARCHAR2(20);
CREATE TYPE mobile_table_t AS TABLE OF VARCHAR2(20);

CREATE TABLE STUDENT_ORDBMS (
    sid INT PRIMARY KEY,
    student_name name_t,
    mobile_numbers mobile_varray_t,
)
```

c. NoSQL

```
{
 "_id": "<ObjectId>",
 "sid": 12345,
 "name": {
   "first_name": "John",
   "middle_name": "Fitzgerald",
   "last_name": "Doe"
 },
 "mobile_numbers": [
   "555-0101",
   "555-0202",
   "555-0303"
 ]
}
```

12. NoSQL is schema less DBMS. Explain it using the following example:

c-id	Name (first-name, middle-name, last-name)	Phones (maximum three numbers)	Choices (set of strings) Maximum 10
C-00001	[Mohammad, Sajid, Abdullah]	01556111111 01715111111 01930111111	{PC, laptop, smartphone}

c-id	Name (first-name, middle-name, last-name)	Phones (maximum three numbers)	hobby (set of strings)
C-00002	[Mohammad, Sajid, Abdullah]	01556111111 01715111111 01930111111	{football, cricket}

c-id	Name (first-name, middle-name, last-name)	email (maximum three numbers)	hobby (set of strings)
C-00003	[Mohammad, Sajid, Abdullah]	abc@gmail.com, abc@northsouth.edu	{football, cricket}

```
{
  "c-id": "C-00001",
  "name": {
    "first-name": "Mohammad",
    "middle-name": "Sajid",
    "last-name": "Abdullah"
  },
  "phones": [
    "01556111111",
    "01715111111",
    "01930111111"
  ],
  "choices": [
    "PC",
    "laptop",
    "smartphone"
  ]
}
```

```
{
  "c-id": "C-00002",
  "name": {
    "first-name": "Mohammad",
    "middle-name": "Sajid",
    "last-name": "Abdullah"
  },
  "phones": [
    "01556111111",
    "01715111111",
    "01930111111"
  ],
  "hobby": [
    "football",
    "cricket"
  ]
}
```

```
{
  "c-id": "C-00003",
  "name": {
    "first-name": "Mohammad",
    "middle-name": "Sajid",
    "last-name": "Abdullah"
  },
  "email": [
    "abc@gmail.com",
    "abc@northsouth.edu"
  ],
  "hobby": [
    "football",
    "cricket"
  ]
}
```

In NoSQL databases, data is stored without a fixed schema, allowing each record to have a different structure. For example, in this case, one customer has phones and choices, another has phones and hobbies, and another has emails and hobbies — each with different fields. Unlike SQL, where a rigid table must be created with all possible columns (leading to many empty or NULL values), NoSQL stores only the available fields for each record, making it highly flexible, efficient, and better suited for handling diverse and evolving data.

13. Given relations and the corresponding sites as follows.

Relation schema	DDL	Site	No. of tuples
Project (p-id, name, budget)	Create table project (p-id char(10) primary key, name varchar2(50) budget number (10, 2))	S1	1000
Employee (e-id, name, street, city, salary)	Create table employee (e-id char(10) primary key, name varchar2(50), street varchar2(30), city varchar2(30), salary number (10, 2))	S2	10000
Works-for(e-id, p-id, start-date, end-date)	Create table works-for (e-id char(10), p-id char(10), s-date date, e-date date)	S3	500000

9. Find the transmission cost for the queries as follows.

Project $\bowtie$ works-for $\bowtie$ employee at site S1

Consider the following cases:

i. Transfer all relations to the query originating site (S1)
ii. Transfer relations in sequential order e.g., S1 → S3 → S2→S1

i. ans.
Size of Employee = (10000X50)/4000 = 125 blocks
transmission cost of Employee = 125X0.2= 25 ms
Size of Work-for = (500000X100)/4000 = 12500 Blocks
Transmission cost of Work-for = 12500X0.2 = 2500 ms
Total transmission cost = 25 + 2500 = 2525 ms

ii. ans.
Cost of r1 = ((1000X500) / 4000)X0.2 = 25 ms
Size of temp1= (((500+100-10)X1000X500000)/4000)= 73750000 blocks
Transmission time temp1= 73750000X0.2 = 14750000ms
Size of temp2= (((500+50 +100-10-10)X10000)/4000) = 1575 blocks
Transmission time temp2= 1575X0.2 = 315ms
Total transmission time = 25+14750000+315= 14750340ms

10: Find the transmission cost for the query as follows using semi-join operation.

Project ⋈ works-for at site S1

Size of temp1 = (10 X 1000)/4000 = 2.5 blocks
Cost of temp1 = 2.5 X 0.1 = 0.25ms

Size of temp3 = (500000X(50+10-10))/4000 = 6250 blocks
Cost of temp3 = 6250 X 0.1 = 625ms

Total cost = 0.25 + 625 = 625.25ms

Show the steps of semi-join operations.