USN | | | | | | | | | | |

Internal Assessment Test II – May 2024

| Sub: | Database Management System | | | | | Sub Code: | BCS/BAD 403 | Branch: | AINDS / CS (DS) | |
|---|---|---|---|---|---|---|---|---|---|---|
| Date: | 11/07/2024 | Duration: | 90 minutes | Max Marks: | 50 | Sem | IV | | OBE | |

| | | Answer any FIVE Questions | MARKS | CO | RBT |
|---|---|---|---|---|---|
| 1 | A | **Explain the informal design guidelines of a database?** | 5 | CO1 | L1 |

Four Informal design guidelines for relation schema are:

**1.Semantics of the Attributes**
**2.Reducing the Redundant Value in Tuples.**
**3.Reducing Null values in Tuples.**
**4.Disallowing spurious Tuples.**

1. **Semantics of the Attributes :-**

Whenever we are going to form relational schema there should be some meaning among the attributes. This meaning is called semantics. This semantics relates one attribute to another with some relation.

| USN No | Student name | Sem |
|---|---|---|

Eg:

**2. Reducing the Redundant Value in Tuples :-**

Mixing attributes of multiple entities may cause problems.
Information is stored redundantly, wasting storage. Problems with update anomalies
Insertion anomalies
Deletion anomalies
Modification anomalies

| | Student name | Sem |
|---|---|---|

Eg:

| Dept No | Dept Name |
|---|---|

**If we integrate these two and is used as a single table i.e Student Table**

| USN No | Student name | Sem | Dept No | Dept Name |
|---|---|---|---|---|

- Here whenever if we insert the tuples there may be 'N' students in one department, so Dept No, Dept Name values are repeated 'N' times which leads to data redundancy.
- Another problem is update anomalies i.e. if we insert new dept that has no students.
- If we delete the last student of a dept ,then whole information about that department will be deleted.
- If we change the value of one of the attributes of particular table the we must update the tuples of all the students belonging to that dept else Database will become inconsistent.

**3. Reducing Null values in Tuples:-**

**Note:** Relations should be designed such that their tuples will have as few NULL values as possible.
Attributes that are NULL frequently could be placed in separate relations (with the primary key)

**Reasons for nulls:**

attribute not applicable or invalid
attribute value unknown (may exist)
value known to exist, but unavailable

**4. Disallowing spurious Tuples**

Bad designs for a relational database may result in erroneous results for certain JOIN operations.
The "lossless join" property is used to guarantee meaningful results for join operations

| | | | | | |
|---|---|---|---|---|---|
| | B | **Explain the types of anomalies with examples?**<br><br>There are three types of anomalies that occur when the database is not normalized. These are – Insertion, update and deletion anomaly.<br><br>Insertion Anomaly<br>Update Anomaly<br>Delete Anomaly<br><br>**Example:** Suppose a manufacturing company stores the employee details in a table named employee that has four attributes: emp_id for storing employee"s id, emp_name for storing employee"s name, emp_address for storing employee"s address and emp_dept for storing the department details in which the employee works.<br><br>emp_id \| emp_name \| emp_address \| emp_dept<br>101 \| Rick \| Delhi \| D001<br>101 \| Rick \| Delhi \| D002<br>123 \| Maggie \| Agra \| D890<br>166 \| Glenn \| Chennai \| D900<br>166 \| Glenn \| Chennai \| D004<br><br>The above table is not normalized. We will see the problems that we face when a table is not normalized<br>• **Update anomaly:** In the above table we have two rows for employee Rick as he belongs to two departments of the company. If we want to update the address of Rick then we have to update the same in two rows or the data will become inconsistent. If somehow, the correct address gets updated in one department but not in other then as per the database, Rick would be having two different addresses, which is not correct and would lead to inconsistent data.<br>• **Insert anomaly:** Suppose a new employee joins the company, who is under training and currently not assigned to any department then we would not be able to insert the data into the table if emp_dept field doesn't allow nulls.<br>• **Delete anomaly:** Suppose, if at a point of time the company closes the department D890 then deleting the rows that are having emp_dept as D890 would also delete the information of employee Maggie since she is assigned only to this department. | 5 | CO1 | L2 |

| | | | | | |
|---|---|---|---|---|---|
| 2 | a,b | **Write an algorithm to find the closure of functional dependency 'F'.**<br>To find the closure of a set of functional dependencies `F`, we need to determine the set of all attributes that can be functionally determined by a given set of attributes. This is typically denoted as `X+` for a set of attributes `X`.<br>Algorithm to Find the Closure of Functional Dependency `F`<br>**Input:**<br>A set of attributes `X`<br>A set of functional dependencies `F`<br>**Output:**<br>The closure of `X` (denoted as `X+`)<br><br>**Steps:**<br>1. **Initialize the closure**:<br>○ Start with the closure `X+` containing all attributes in `X`.<br>2. **Iterate through functional dependencies**:<br>○ For each functional dependency `A → B` in `F`, if `A` is a subset of `X+`, add all attributes in `B` to `X+`.<br>3. **Repeat until no new attributes are added**:<br>○ Continue the process until `X+` no longer changes.<br>4. **Return the closure**:<br>○ The final set `X+` is the closure of `X`.<br><br>**R(ABCDEF) Check the highest Normal Form using closure Algorithm ? FD { AB->C,C->DE,E->F,F->A}.**<br>**Step 1: Candidate Key**<br>First, we need to determine the candidate key(s) for the relation. We can do this by finding the closure of different attribute sets to see which ones determine all attributes in the relation.<br>**Find Closure of AB**:<br>○ Start with AB: {A,B}<br>○ AB→C:{A,B,C}<br>○ C→DE:{A,B,C,D,E}<br>○ E→F:{A,B,C,D,E,F}<br>○ Closure of AB is {A,B,C,D,E,F}. Hence, AB is a candidate key.<br>● Since AB is a candidate key and it alone determines all attributes, no other subsets are candidate keys.<br>● C.K□{AB AB+=ABCDEF<br>FB FB+=FBACDE<br>EB EB+=EBFACD<br>CB} CB+=CBDEFA<br>● **Step 2: Write All Prime Attributes**<br>**{A,B,C,E,F}**<br>● **Step 3: Write All Non-Prime Attributes**<br>**{ D }**<br>● **Step 4:-Finding FD's**<br>**Determine the Highest Normal Form**<br>● **First Normal Form (1NF)**<br>The relation R is in 1NF since all attributes are atomic.<br>● **Second Normal Form (2NF)**<br>The relation R is in 2NF if it is in 1NF and all non-key attributes are fully functionally dependent on the entire candidate key.<br>Here, the candidate key is AB, and all non-key attributes {C,D,E,F} are fully functionally dependent on AB.<br>No partial dependency exists (no non-prime attribute is dependent on a part of the candidate key).<br>● **Third Normal Form (3NF)**<br>The relation R is in 3NF if it is in 2NF and no transitive dependencies exist.<br>Check the dependencies for transitive dependency:<br>○ AB→C is not transitive.<br>○ C→DE and E→F: C→F via E,,F→A: C→A via F. Hence, transitive dependencies exist.<br>● Therefore, the relation is not in 3NF due to transitive dependencies.<br>● **Boyce-Codd Normal Form (BCNF)**<br>The relation R is in BCNF if for every non-trivial functional dependency X→Y,X is a superkey.<br>Since C→DE and F→AF violate this rule (because C and F are not superkeys), the relation is not in BCNF. | 5 | CO1 | L2 |

| | | | | | |
|---|---|---|---|---|---|
| | | | | | |

| FD | AB□C | C□DE | E□F | F□A |
|---|---|---|---|---|
| BCNF | yes | no | no | no |
| 3NF | yes | no | yes | yes |
| 2NF | yes | yes | yes | yes |
| 1NF | yes | Yes | yes | yes |

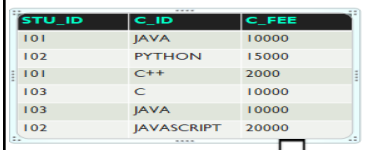
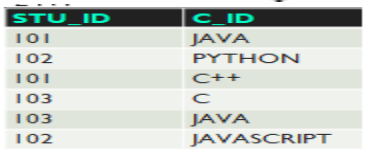
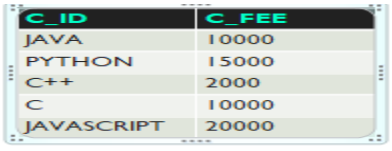
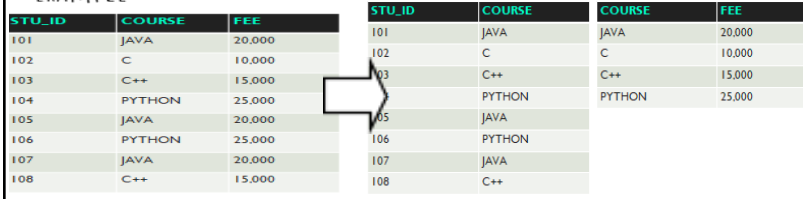
5 | CO2 | L3

- **Conclusion**

The highest normal form of the given relation R(ABCDEF) with the functional dependencies {AB→C,C→DE,E→F,F→A} is **Second Normal Form (2NF)**.

- **To summarize**:

The relation is in 1NF (all attributes are atomic).
The relation is in 2NF (all non-key attributes are fully dependent on the entire candidate key).
The relation is not in 3NF due to transitive dependencies.
The relation is not in BCNF as not all determinants are superkeys.

| | | | | | |
|---|---|---|---|---|---|
| 3 | | **What is the need for Normalization? Explain INF, 2NF, 3NF and BCNF with examples?**<br>• Data redundancy in DBMS means having the same data at multiple places. It is necessary to remove data redundancy because it causes ANOMALIES in a database which makes it very hard for a database administrator to maintain it.<br>• Normalization is a process of organizing the data in a database to avoid data redundancy.<br>• Normalization provides a method to remove the anomalies from the database & bring it to a more Consistent state.<br>• **Types of Normal Forms**<br>• There is further enhancement to the theory` of normalization & it is still developed.<br>• Normalization achieves its best shape in 3rd NF<br>• Here are the most commonly used normal forms:<br>**1) First normal form(1NF)**<br>**2) Second normal form(2NF)**<br>**3) Third normal form(3NF)**<br>**4) Boyce & Codd normal form (BCNF)**<br>**5)Fourth Normal Form (4 NF)**<br>**6) Fifth Normal Form (5F)**<br>**1 Normal Form-Conditions:-**<br>• An attribute (column) of a table cannot hold multiple values. It should hold only atomic values.<br>□ Each attribute should contain atomic(single)values<br>□ A column should contain value from same domain<br>□ Each column should have unique (P.K)name<br>□ No ordering to rows and column<br>□ No duplicate rows<br>**Example 1 –** Relation STUDENT in table 1 is not in 1NF because of multi-valued attribute STUD_PHONE. Its decomposition into 1NF has been shown in table 2<br>STUD_NO \| STUD_NAME \| STUD_PHONE \| STUD_STATE \| STUD_COUNTRY<br>1 \| RAM \| 9716271721, 9871717178 \| HARYANA \| INDIA<br>2 \| RAM \| 9898297281 \| PUNJAB \| INDIA<br>3 \| SURESH \| \| PUNJAB \| INDIA<br>**Table 1**<br>Conversion to first normal form<br>STUD_NO \| STUD_NAME \| STUD_PHONE \| STUD_STATE \| STUD_COUNTRY<br>1 \| RAM \| 9716271721 \| HARYANA \| INDIA<br>1 \| RAM \| 9871717178 \| HARYANA \| INDIA<br>2 \| RAM \| 9898297281 \| PUNJAB \| INDIA<br>3 \| SURESH \| \| PUNJAB \| INDIA<br>**Table 2**<br>**Second normal form (2NF):-**<br>A table is said to be in 2NF if both the following conditions hold:<br>**Conditions**<br>• Relations should be in 1NF<br>• Relations should not have partial functional dependency i.e No-partial dependency.<br>Ex:-<br>  <br>STU_ID \| C_ID \| C_FEE<br>101 \| JAVA \| 10000<br>102 \| PYTHON \| 15000<br>101 \| C++ \| 2000<br>103 \| C \| 10000<br>103 \| JAVA \| 10000<br>102 \| JAVASCRIPT \| 20000<br>2 NF<br>  <br>STU_ID \| C_ID<br>101 \| JAVA<br>102 \| PYTHON<br>101 \| C++<br>103 \| C<br>103 \| JAVA<br>102 \| JAVASCRIPT<br>  <br>C_ID \| C_FEE<br>JAVA \| 10000<br>PYTHON \| 15000<br>C++ \| 2000<br>C \| 10000<br>JAVASCRIPT \| 20000<br>**Third Normal Forms:-**<br>**Condition:**<br>• The relation should be in 2NF<br>• No transitive dependencies for non-prime attributes<br>EXAMPLE<br>  <br>STU_ID \| COURSE \| FEE<br>101 \| JAVA \| 20,000<br>102 \| C \| 10,000<br>103 \| C++ \| 15,000<br>104 \| PYTHON \| 25,000<br>105 \| JAVA \| 20,000<br>106 \| PYTHON \| 25,000<br>107 \| JAVA \| 20,000<br>108 \| C++ \| 15,000<br>STU_ID \| COURSE<br>101 \| JAVA<br>102 \| C<br>103 \| C++<br>104 \| PYTHON<br>105 \| JAVA<br>106 \| PYTHON<br>107 \| JAVA<br>108 \| C++<br>COURSE \| FEE<br>JAVA \| 20,000<br>C \| 10,000<br>C++ \| 15,000<br>PYTHON \| 25,000<br>**Boyce-Codd Normal Form:-**<br>**BCNF** is the extension to the 3NF.It is also known as 3.5NF.BCNF is the advanced version of 3NF.It is stricter than 3NF.<br>**Conditions:**<br>• Relation should be in 3NF<br>• For all functional dependencies<br>X->Y<br>X should be super key/candidate key | 10 | CO2 | L4 |

# BCNF

## Example

| Student (S) | Course (C) | Tutor (T) |
|---|---|---|
| 101 | Java | Tejas |
| 101 | Python | Basha |
| 102 | Java | Tejas |
| 102 | Python | Sachin |

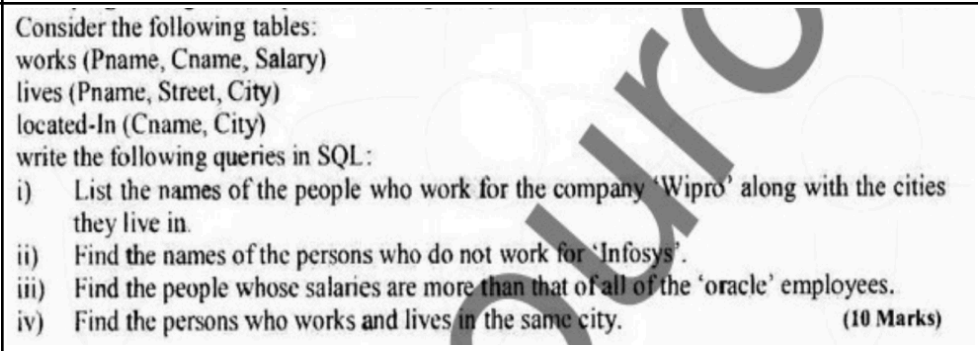
| Student (S) | Tutor (T) |
|---|---|
| 101 | Tejas |
| [illegible] | Basha |
| [illegible] | Tejas |
| 102 | Sachin |

| Course (C) | Tutor (T) |
|---|---|
| Java | Tejas |
| Python | Basha |
| Python | Sachin |

| | | | | | |
|---|---|---|---|---|---|
| 4 | | **Normalize the below relation up to 3NF**<br>| Module | Dept | Lecturer | Text |<br>|---|---|---|---|<br>| M1 | D1 | L1 | T1 |<br>| M1 | D1 | L1 | T2 |<br>| M2 | D1 | L1 | T1 |<br>| M2 | D1 | L1 | T3 |<br>| M3 | D1 | L2 | T4 |<br>| M4 | D2 | L3 | T1 |<br>| M4 | D2 | L3 | T5 |<br>| M5 | D2 | L4 | T6 |<br>• To normalize the given relation up to Third Normal Form (3NF), we'll follow the normalization process:<br>**Step1:-Identify the functional dependencies.**<br>**Step2:-Ensure the relation is in First Normal Form (1NF).**<br>**Step3:-Transform it to Second Normal Form (2NF).**<br>**Step4:-Transform it to Third Normal Form (3NF)**<br>**Step 1: Identify Functional Dependencies**<br>• From the given table:<br>• **Module** determines **Dept** and **Lecturer**.<br>• **Module** and **Text** are combined to determine all attributes uniquely, as each combination of **Module** and **Text** uniquely identifies a record.<br>• Therefore, we have the following functional dependencies:<br>**Module → Dept, Lecturer**<br>**Module, Text → Dept, Lecturer, Text**<br>**Step 3: Second Normal Form (2NF)**<br>• A table is in 2NF if it is in 1NF and all non-key attributes are fully functionally dependent on the primary key. We need to ensure that there are no partial dependencies on a composite key.<br>• **Current Candidate Key:** (Module, Text)<br>• To move to 2NF, we decompose the table to remove partial dependencies.<br>• **Decomposition into 2NF:**<br>**Table 1: Modules**<br>| Module | Dept | Lecturer |<br>|---|---|---|<br>| M1 | D1 | L1 |<br>| M2 | D1 | L1 |<br>| M3 | D1 | L2 |<br>| M4 | D2 | L3 |<br>| M5 | D2 | L4 |<br>**Table 2: Module_Text**<br>| Module | Text |<br>|---|---|<br>| M1 | T1 |<br>| M1 | T2 |<br>| M2 | T1 |<br>| M2 | T3 |<br>| M3 | T4 |<br>| M4 | T1 |<br>| M4 | T5 |<br>| M5 | T6 |<br>**Step 4: Third Normal Form (3NF)**<br>• A table is in 3NF if it is in 2NF and all the attributes are functionally dependent only on the primary key and there are no transitive dependencies.<br>• **Table 1: Modules** is already in 3NF since all non-key attributes (Dept, Lecturer) depend only on the primary key (Module).<br>• **Table 2: Module_Text** is in 3NF since there are no transitive dependencies.<br>**Final Normalized Tables:**<br>**Table 1: Modules**<br>| Module | Dept | Lecturer |<br>|---|---|---|<br>| M1 | D1 | L1 |<br>| M2 | D1 | L1 |<br>| M3 | D1 | L2 |<br>| M4 | D2 | L3 |<br>| M5 | D2 | L4 |<br>**Table 2: Module_Text**<br>| Module | Text |<br>|---|---|<br>| M1 | T1 |<br>| M1 | T2 |<br>| M2 | T1 |<br>| M2 | T3 |<br>| M3 | T4 |<br>| M4 | T1 |<br>| M4 | T5 |<br>| M5 | T6 | | 10 | CO1 | L2 |

| | | | | | |
|---|---|---|---|---|---|
| | | These two tables are now in 3NF, ensuring no redundancy and eliminating partial and transitive dependencies. | | | |

| | | | | | |
|---|---|---|---|---|---|
| 5 | | **Demonstrate the following constraints in SQL with suitable example:**<br>**i) NOT NULL ii) Primary key iii) Foreign key iv) Default v)Unique**<br><br>**i) NOT NULL :- Ensures that a column cannot have a NULL value**<br><br>ii) **Primary key :-** A combination of a NOT NULL and UNIQUE. Uniquely identifies each row in a table.<br>• The PRIMARY KEY constraint uniquely identifies each record in a table.<br>• Primary keys must contain UNIQUE values, and cannot contain NULL values.<br>• A table can have only ONE primary key; and in the table, this primary key can consist of single or multiple columns.<br>**SQL PRIMARY KEY on CREATE TABLE**<br>• The following SQL creates a PRIMARY KEY on the "ID" column when the "Persons" table is created:<br>`CREATE TABLE Persons (`<br>`ID int NOT NULL,`<br>`LastName varchar(255) NOT NULL,`<br>`FirstName varchar(255),`<br>`Age int,`<br>`PRIMARY KEY (ID));`<br>**SQL PRIMARY KEY on ALTER TABLE**<br>To create a PRIMARY KEY constraint on the "ID" column when the table is already created, use the following SQL:<br>`ALTER TABLE Persons`<br>`ADD PRIMARY KEY (ID);`<br>**DROP a PRIMARY KEY Constraint**<br>To drop a PRIMARY KEY constraint, use the following SQL:<br>`ALTER TABLE Persons`<br>`DROP PRIMARY KEY;`<br><br>iii) Foreign key:- Prevents actions that would destroy links between tables.<br>• The FOREIGN KEY constraint is used to prevent actions that would destroy links between tables.<br>• A FOREIGN KEY is a field in one table, that refers to the PRIMARY KEY in another table.<br>• The following SQL creates a FOREIGN KEY on the "PersonID" column when the "Orders" table is created:<br>`CREATE TABLE Orders (`<br>`OrderID int NOT NULL,`<br>`OrderNumber int NOT NULL,`<br>`PersonID int,`<br>`PRIMARY KEY (OrderID),`<br>`FOREIGN KEY (PersonID) REFERENCES Persons(PersonID));`<br>**SQL FOREIGN KEY on ALTER TABLE**<br>To create a FOREIGN KEY constraint on the "PersonID" column when the "Orders" table is already created, use the following SQL:<br>`ALTER TABLE Orders`<br>`ADD FOREIGN KEY (PersonID) REFERENCES Persons(PersonID);`<br>**DROP a FOREIGN KEY Constraint**<br>To drop a FOREIGN KEY constraint, use the following SQL:<br>`ALTER TABLE Orders`<br>`DROP FOREIGN KEY FK_PersonOrder;`<br><br>iv) Default:- Sets a default value for a column if no value is specified.<br>• The **DEFAULT** keyword is used to set a default value for a column. When no value is specified for the column during an INSERT operation, the default value is automatically assigned.<br>**SQL DEFAULT on CREATE TABLE**<br>The following SQL sets a DEFAULT value for the "City" column when the "Persons" table is created:<br>`CREATE TABLE Persons (`<br>`ID int NOT NULL,`<br>`LastName varchar(255) NOT NULL,`<br>`FirstName varchar(255),`<br>`Age int,`<br>`City varchar(255) DEFAULT 'Sandnes');`<br>**SQL DEFAULT on ALTER TABLE**<br>• To create a DEFAULT constraint on the "City" column when the table is already created, use the following SQL:<br>`ALTER TABLE Persons`<br>`ALTER City SET DEFAULT 'Sandnes';`<br>**DROP a DEFAULT Constraint**<br>• To drop a DEFAULT constraint, use the following SQL:<br>`ALTER TABLE Persons`<br>`ALTER City DROP DEFAULT;`<br><br>**v)Unique:-** Ensures that all values in a column are different.<br>• The UNIQUE constraint ensures that all values in a column are different.<br>• Both the UNIQUE and PRIMARY KEY constraints provide a guarantee for uniqueness for a column or set of columns.<br>• A PRIMARY KEY constraint automatically has a UNIQUE constraint.<br>• However, many UNIQUE constraints per table, but only one PRIMARY KEY constraint per table.<br>The following SQL creates a UNIQUE constraint on the "ID" column when the "Persons" table is created:<br>`CREATE TABLE Persons (`<br>`ID int NOT NULL,`<br>`LastName varchar(255) NOT NULL,`<br>`FirstName varchar(255),`<br>`Age int,`<br>`UNIQUE (ID));` | 10 | CO3 | L2 |

| | | | | | |
|---|---|---|---|---|---|
| | | **SQL UNIQUE Constraint on ALTER TABLE**<br>• To create a UNIQUE constraint on the "ID" column when the table is already created, use the following SQL:<br>ALTER TABLE Persons<br>ADD UNIQUE (ID);<br>**DROP a UNIQUE Constraint**<br>To drop a UNIQUE constraint, use the following SQL:<br>ALTER TABLE Persons<br>DROP INDEX UC_Person; | | | |
| 6 | |  <br>Consider the following tables:<br>works (Pname, Cname, Salary)<br>lives (Pname, Street, City)<br>located-In (Cname, City)<br>write the following queries in SQL:<br>i) List the names of the people who work for the company 'Wipro' along with the cities they live in.<br>ii) Find the names of the persons who do not work for 'Infosys'.<br>iii) Find the people whose salaries are more than that of all of the 'oracle' employees.<br>iv) Find the persons who works and lives in the same city. (10 Marks)<br><br>i) List the names of the people who work for the company 'Wipro' along with the cities they live in.<br><br>SELECT w.Pname, l.City FROM works w JOIN lives l ON w.Pname = l.Pname WHERE w.Cname = 'Wipro';<br><br>ii) Find the names of the persons who do not work for 'Infosys'.<br><br>SELECT l.Pname FROM lives l WHERE l.Pname NOT IN ( SELECT w.Pname FROM works w WHERE w.Cname = 'Infosys' );<br><br>iii) Find the people whose salaries are more than that of all of the 'Oracle' employees.<br><br>SELECT w1.Pname FROM works w1 WHERE w1.Salary > ALL ( SELECT w2.Salary FROM works w2 WHERE w2.Cname = 'Oracle' );<br><br>iv) Find the persons who work and live in the same city.<br><br>SELECT l.Pname FROM lives l JOIN located_In li ON l.City = li.City JOIN works w ON l.Pname = w.Pname AND w.Cname = li.Cname; | 10 | CO3 | L3 |

CI CCI HOD