Q.2 a. In large organizations many people are involved in design, use and maintenance of large databases with hundreds of users. Describe in detail the actors on the scene and workers behind the scene. (8)

Answer:

Actors on the scene

1. **Database administrator**: In any organization where many persons use the same resources, there is a need for a chief administrator to oversee and manage these resources. In a database environment, the primary resource is the database itself and the secondary resource is the DBMS and related software. Administering these resources is the responsibility of the database administrator (DBA). The DBA is responsible for authorizing access to the database, for coordinating and monitoring its use, and for acquiring software and hardware resources as needed. The DBA is accountable for problems such as breach of security or poor system response time. In large organizations, the DBA is assisted by a staff that helps carry out these functions.
2. **Database designers** are responsible for identifying the data to be stored in the database and for choosing appropriate structures to represent and store this data. These tasks are mostly undertaken before the database is actually implemented and populated with data. It is the responsibility of database designers to communicate with all prospective database users, in order to understand their requirements, and to come up with a design that meets these requirements. In many cases, the designers are on the staff of the DBA and may be assigned other staff responsibilities after the database design is completed. Database designers typically interact with each potential group of users and develop a view of the database that meets the data and processing requirements of this group. These views are then analyzed and integrated with the views of other user groups. The final database design must be capable of supporting the requirements of all user groups. **5**
3. **End Users:** End users are the people whose jobs require access to the database for querying, updating, and generating reports; the database primarily exists for their use. There are several categories of end users:
   a. Casual end users occasionally access the database, but they may need different information each time. They use a sophisticated database query language to specify their requests and are typically middle- or high-level managers or other occasional browsers.
   b. Naive or parametric end users make up a sizable portion of database end users. Their main job function revolves around constantly querying and updating the database, using standard types of queries and updates—called canned transactions—that have been carefully programmed and tested.
   c. Sophisticated end users include engineers, scientists, business analysts, and others who thoroughly familiarize themselves with the facilities of the DBMS so as to implement their applications to meet their complex requirements.
   d. Standalone users maintain personal databases by using ready-made program packages that provide easy-to-use menu- or graphics-based interfaces. An example is the user of a tax package that stores a variety of personal financial data for tax purposes

Workers behind the scene

1. **DBMS system designers and implementers** are persons who design and implement the DBMS modules and interfaces as a software package. A DBMS is a complex software system that consists of many components or modules, including modules for implementing the catalog, query language, interface processors, data access, concurrency control, recovery, and security. The DBMS must interface with other system software, such as the operating system and compilers for various programming languages. **3**
2. **Tool developers include persons who design and implement tools—**the software packages that facilitate database system design and use, and help improve performance. Tools are optional packages that are often purchased separately. They include packages for database design, performance monitoring, natural language or graphical interfaces, prototyping, simulation, and test data generation. In many cases, independent software vendors develop and market these tools.
3. **Operators and maintenance personnel** are the system administration personnel who are responsible for the actual running and maintenance of the hardware and software environment for the database system.

(5 marks for actors on the scene and 3 marks for actors behind the scene)

b. What is three-schema architecture? Discuss its role in data independence. (8)

Answer:

The goal of the three-schema architecture is to separate the user applications and the physical database. In this architecture, schemas can be defined at the following three levels:

1. The internal level has an internal schema, which describes the physical storage structure of the database. The internal schema uses a physical data model and describes the complete details of data storage and access paths for the database.
2. The conceptual level has a conceptual schema, which describes the structure of the whole database for a community of users. The conceptual schema hides the details of physical storage structures and concentrates on describing entities, data types, relationships, user operations, and constraints. A high-level data model or an implementation data model can be used at this level.
3. The external or view level includes a number of external schemas or user views. Each external schema describes the part of the database that a particular user group is interested in and hides the rest of the database from that user group. A high-level data model or an implementation data model can be used at this level. **4**

The three-schema architecture can be used to explain the concept of data independence, which can be defined as the capacity to change the schema at one level of a database system without having to change the schema at the next higher level. We can define two types of data independence:

1. Logical data independence is the capacity to change the conceptual schema without having to change external schemas or application programs. We may change the conceptual schema to expand the database (by adding a record type or data item), or to reduce the database (by removing a record type or data item). In the latter case, external schemas that refer only to the remaining data should not be affected. Only the view definition and the mappings need be changed in a DBMS that supports logical data independence. Application programs that reference the external schema constructs must work as before, after the conceptual schema undergoes a logical reorganization. Changes to constraints can be applied also to the conceptual schema without affecting the external schemas or application programs. **4**
2. Physical data independence is the capacity to change the internal schema without having to change the conceptual (or external) schemas. Changes to the internal schema may be needed because some physical files had to be reorganized—for example, by creating additional access structures—to improve the performance of retrieval or update. If the same data as before remains in the database, we should not have to change the conceptual schema. For example, providing an access path to improve retrieval of SECTION records by Semester and Year should not require a query such as "list all sections offered in fall 1998" to be changed, although the query would be executed more efficiently by the DBMS by utilizing the new access path.

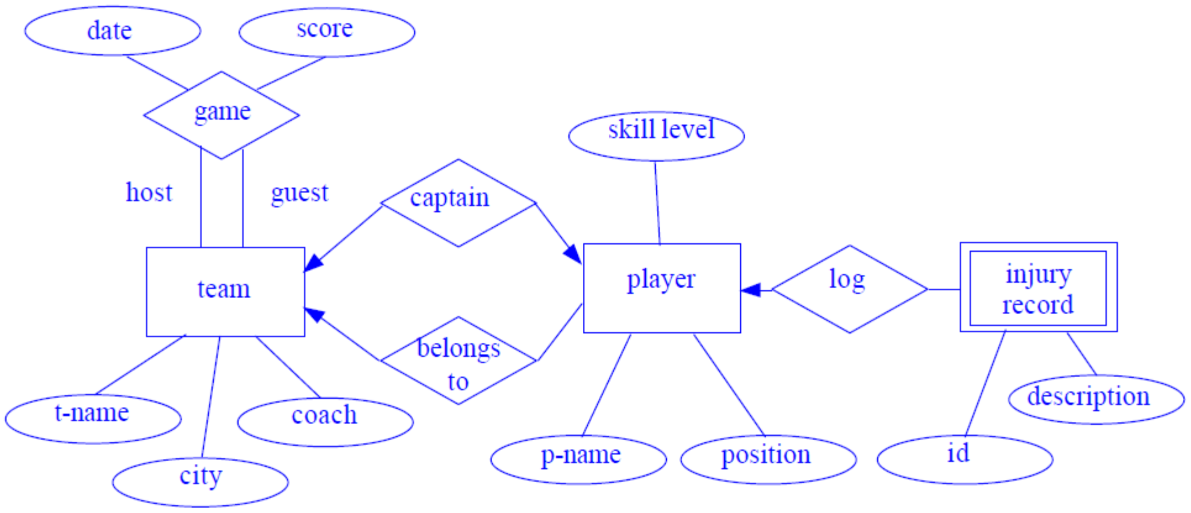
(4 marks for three-schema architecture and 4 marks for role in data independence)

Q.3 Being a database administrator, you are given the following requirements for a simple database of the National Hockey League (NHL): the NHL has many teams, each team has a name, a city, a coach, a captain, and a set of players, each player belongs to only one team, each player has a name, a position, a skill level, and a set of injury records, a team captain is also a player, a game is played between two teams (referred to as host_team and guest_team) and has a date (such as May 11th, 2016) and a score (such as 2 to 4).

Construct an ER diagram for the NHL database and clearly indicate the cardinality mappings as well as role indicators in your ER diagram. (16)

Answer:

Divide marks on the basis of entities identified and correct relationships and attributes



Primary keys (3), Attributes (3), Entity set (3), Relationships (3), Weat Entity Set (2), Connections (one to one) (one to many) (2)

Q.4 a. Discuss the binary relational operations JOIN and DIVISION with examples. (12)

Answer:

The **JOIN** operation, denoted by ⋈ , is used to combine *related tuples* from two relations into single tuples. This operation is very important for any relational database with more than a single relation, because it allows us to process relationships among relations. To illustrate join, suppose that we want to retrieve the name of the manager of each department. To get the manager's name, we need the department tuple. We do this by using the JOIN operation, and then projecting the result over the necessary attributes, as follows:

DEPT_MGR←DEPARTMENT⋈ $_{\text{MGRSSN=SSN}}$ EMPLOYEE

RESULT←$\pi_{\text{DNAME, LNAME, FNAME}}$(DEPT_MGR)

The general form of a JOIN operation on two relations R(A_1, A_2, . . ., A_n) and S(B_1, B_2, . . ., B_m) is:

7

R⋈<join condition>S

The result of the JOIN is a relation Q with n + m attributes Q(A_1, A_2, . . ., A_n, B_1, B_2, . . ., B_m) in that order; Q has one tuple for each combination of tuples—one from R and one from S—*whenever the combination satisfies the join condition.* θThe join condition is specified on attributes from the two relations R and S and is evaluated for each combination of tuples. Each tuple combination for which the join condition evaluates to true is included in the resulting relation Q *as a single combined tuple.*

A general join condition is of the form:

<condition> AND <condition> AND . . . AND <condition>

where each condition is of the form A_i θ B_j, A_i is an attribute of R, B_j is an attribute of S, A_i and B_j have the same domain, and θ (theta) is one of the comparison operators {=, <, 1, >, , }. A JOIN operation with such a general join condition is called a **THETA JOIN.** Tuples whose join attributes are null *do not* appear in the result. In that sense, the join operation does *not* necessarily preserve all of the information in the participating relations.

The most common JOIN involves join conditions with equality comparisons only. Such a JOIN, where the only comparison operator used is =, is called an **EQUIJOIN.** In the result of an EQUIJOIN we always have one or more pairs of attributes that have *identical values* in every tuple. Because one of each pair of attributes with identical values is superfluous, a new operation called **NATURAL JOIN**—denoted by *—was created to get rid of the second (superfluous) attribute in an EQUIJOIN condition . The standard definition of NATURAL JOIN requires that the two join attributes (or each pair of join attributes) have the same name in both relations. If this is not the case, a renaming operation is applied first. In the following example, we first rename the DNUMBER attribute of DEPARTMENT to DNUM—so that it has the same name as the DNUM attribute in PROJECT—then apply NATURAL JOIN:

PROJ_DEPT←PROJECT * ρ(DNAME, DNUM,MGRSSN,MGRSTARTDATE)(DEPARTMENT)

The attribute DNUM is called the **join attribute.** In the PROJ_DEPT relation, each tuple combines a PROJECT tuple with the DEPARTMENT tuple for the department that controls the project, but *only one join attribute* is kept.

The DIVISION Operation

The DIVISION operation denoted by ÷ is useful for a special kind of query that sometimes occurs in database applications. An example is "Retrieve the names of employees who work on *all* the projects that 'John Smith' works on." To express this query using the DIVISION operation, proceed as follows. First, retrieve the list of project numbers that 'John Smith' works on in the intermediate relation SMITH_PNOS:

SMITH←σFNAME='John' AND LNAME='Smith'(EMPLOYEE)

SMITH_PNOS←π PNO(WORKS_ON ⋈ESSN=SSN SMITH)

5

Next, create a relation that includes a tuple <PNO, ESSN> whenever the employee whose social security number is ESSN works on the project whose number is PNO in the intermediate relation SSN_PNOS:

SSN_PNOS←π ESSN,PNO (WORKS_ON)

Finally, apply the DIVISION operation to the two relations, which gives the desired employees' social security numbers:

SSNS(SSN) ←SSN_PNOS ÷ SMITH_PNOS

RESULT← π FNAME, LNAME(SSNS * EMPLOYEE)

In general, the DIVISION operation is applied to two relations R(Z) ÷ S(X), where X ⊆Z. Let Y = Z - X (and hence Z = X D Y); that is, let Y be the set of attributes of R that are not attributes of S. The result of DIVISION is a relation T(Y) that includes a tuple t if tuples t_R appear in R with t_R[Y] = t, and with t_R[X] = t_S *for every tuple* t_S in S. This means that, for a tuple t to appear in the result T of the DIVISION, the values in t must appear in R in combination with *every* tuple in S

(7 marks for Join operation with examples and 5 marks for division operation with examples. Deduct 50% marks if examples not given)

b. Outline the approaches used for mapping of binary 1:1 relationship in ER to relational database schema. (4)

Answer:

For each binary 1:1 relationship type R in the ER schema, identify the relations S and T that correspond to the entity types participating in R. There are three possible approaches:

1. Foreign Key approach: Choose one of the relations-say S-and include a foreign key in S the primary key of T. It is better to choose an entity type with total participation in R in the role of S. Example: 1:1 relation MANAGES is mapped by choosing the participating entity type DEPARTMENT to serve in the role of S, because its participation in the MANAGES relationship type is total.
2. Merged relation option: An alternate mapping of a 1:1 relationship type is possible by merging the two entity types and the relationship into a single relation. This may be appropriate when both participations are total.
3. Cross-reference or relationship relation option: The third alternative is to set up a third relation R for the purpose of cross-referencing the primary keys of the two relations S and T representing the entity types.

4

4 Marks for any of two options

Q.5 a. Explain with an example, how SQL implement the entity integrity and referential integrity constraints of the relational data model. (8)

Answer:

The entity integrity constraint states that no primary key value can be null. This is because the primary key value is used to identify individual tuples in a relation; having null values for the primary key implies that we cannot identify some tuples. For example, if two or more tuples had null for their primary keys, we might not be able to distinguish them.

2

Key constraints and entity integrity constraints are specified on individual relations. The referential integrity constraint is specified between two relations and is used to maintain the consistency among tuples of the two relations. Informally, the referential integrity constraint states that a tuple in one relation that refers to another relation must refer to an existing tuple in that relation. For example, the attribute DNO of EMPLOYEE gives the department number for which each employee works; hence, its value in every EMPLOYEE tuple must match the DNUMBER value of some tuple in the DEPARTMENT relation. **2**

To define referential integrity more formally, we first define the concept of a foreign key. The conditions for a foreign key, given below, specify a referential integrity constraint between the two relation schemas R1 and R2. A set of attributes FK in relation schema R1 is a foreign key of R1 that references relation R2 if it satisfies the following two rules: **2**

1. The attributes in FK have the same domain(s) as the primary key attributes PK of R2; the attributes FK are said to reference or refer to the relation R2.
2. A value of FK in a tuple t1 of the current state r1(R1) either occurs as a value of PK for some tuple t2 in the current state r2(R2) or is null. In the former case, we have t1[FK] = t2[PK], and we say that the tuple t1 references or refers to the tuple t2. R1 is called the referencing relation and R2 is the referenced relation.

In a database of many relations, there are usually many referential integrity constraints. Referential integrity constraints typically arise from the relationships among the entities represented by the relation schemas. For example, In the EMPLOYEE relation, the attribute DNO refers to the department for which an employee works; hence, we designate DNO to be a foreign key of EMPLOYEE, referring to the DEPARTMENT relation. This means that a value of DNO in any tuple t1 of the EMPLOYEE relation must match a value of the primary key of DEPARTMENT—the DNUMBER attribute—in some tuple t2 of the DEPARTMENT relation, or the value of DNO can be null if the employee does not belong to a department. Thus the tuple for employee 'John Smith' references the tuple for the 'Research' department, indicating that 'John Smith' works for this department. **2**

(4 marks for the entity integrity and referential integrity constraints each with examples. Deduct 50% marks if examples not given)

b. Discuss with example how GROUP BY clause works. What is the difference between the WHERE and HAVING clause? (8)

Answer:

A query in SQL can consist of up to six clauses, but only the first two—SELECT and FROM—are mandatory. The clauses are specified in the following order, with the clauses between square brackets [. . .] being optional:

```
SELECT <attribute and function list>
FROM <table list>
[WHERE <condition>]
[GROUP BY <grouping attribute(s)>]
[HAVING <group condition>]
[ORDER BY <attribute list>];
```

The SELECT-clause lists the attributes or functions to be retrieved. The FROM-clause specifies all relations (tables) needed in the query, including joined relations, but not those in nested queries. The WHERE-clause specifies the conditions for selection of tuples from these relations, including join conditions if needed. GROUP BY specifies grouping attributes, whereas HAVING specifies a condition on the groups being selected rather than on the individual tuples. The built-in aggregate functions COUNT, SUM, MIN, MAX, and AVG are used in conjunction with grouping, but they can also be applied to all the selected tuples in a query without a GROUP BY clause. For example, we may want to find the average salary of **4** employees in each department or the number of employees who work on each project. In these cases we need to group the tuples that have the same value of some attribute(s), called the **grouping attribute(s),** and we need to apply the function to each such group independently. SQL has a **GROUP BY-clause** for this purpose. The GROUP BY-clause specifies the grouping attributes, which should *also appear in the SELECT-clause*, so that the value resulting from applying each function to a group of tuples appears along with the value of the grouping attribute(s). For example

Query 1:For each department, retrieve the department number, the number of employees in the department, and their average salary.

SELECT DNO, **COUNT** (*), **AVG** (SALARY)

FROM EMPLOYEE

GROUP BY DNO;

Or

Query 2:For each project, retrieve the project number, the project name, and the number of employees who work on that project.

SELECT PNUMBER, PNAME, **COUNT** (*)
FROM PROJECT, WORKS_ON
WHERE PNUMBER=PNO
GROUP BY PNUMBER, PNAME;

$\boxed{2}$

Sometimes we want to retrieve the values of these functions only for *groups that satisfy certain conditions.* For example, suppose that we want to modify Query 2 so that only projects with more than two employees appear in the result. SQL provides a **HAVING-clause,** which can appear in conjunction with a GROUP BY-clause, for this purpose. HAVING provides a condition on the group of tuples associated with each value of the grouping attributes; and only the groups that satisfy the condition are retrieved in the result of the query as shown in query 3.

Query 3: For each project *on which more than two employees work*, retrieve the project number, the project name, and the number of employees who work on the project.
SELECT PNUMBER, PNAME, **COUNT** (*)
FROM PROJECT, WORKS_ON
WHERE PNUMBER=PNO
GROUP BY PNUMBER, PNAME
HAVING COUNT (*) > 2;

$\boxed{2}$

Thus a query is evaluated conceptually by applying first the FROM-clause (to identify all tables involved in the query or to materialize any joined tables), followed by the WHERE-clause, and then GROUP BY and HAVING.

(Deduct 50% marks if examples not given)

Q.6 An invoice management system stores the invoice details as follows:

Invoice No.	Invoice date	Order no	Challan no	Cust no	Cust name	Item no	Item desc.	QTY sold	rate	Discount	Invoice value
112	12/8/2014	1	1	C1	SRIKANT	I1	PEPSI	2	25	NIL	75
112	12/8/2014	2	1	C1	SRIKANT	I2	BUTTER	1	60	NIL	75
113	16/8/2014	1	1	C4	KAVITA	I4	BREAD	1	22	NIL	22
114	16/8/2014	1	1	C1	SRIKANT	I8	BISCUIT	2	60	NIL	92
114	16/8/2014	2	1	C1	SRIKANT	I2	PEPSI	4	25	NIL	92

Apply normalization until you cannot decompose the invoice relational table further. State reasons behind each decomposition. (16)

Answer:
First Normal form

Invoice Table

Invoice No.	Invoice date	Order no	Challan no	Cust no	Cust name	Invoice value
112	12/8/2014	1	1	C1	SRIKANT	75
113	16/8/2014	1	1	C4	KAVITA	22
114	16/8/2014	1	1	C1	SRIKANT	92

4

Invoice Items

Invoice No.	Item no	Item desc.	QTY sold	rate	Discount
112	I1	PEPSI	2	25	NIL
112	I2	BUTTER	1	60	NIL
113	I4	BREAD	1	22	NIL
114	I8	BISCUIT	2	60	NIL
114	I2	PEPSI	4	25	NIL

Second Normal Form

Invoice Table

Invoice No.	Invoice date	Order no	Challan no	Cust no	Cust name	Invoice value
112	12/8/2014	1	1	C1	SRIKANT	75
113	16/8/2014	1	1	C4	KAVITA	22
114	16/8/2014	1	1	C1	SRIKANT	92

Invoice Items

Invoice No.	Item no	QTY sold	Discount
112	I1	2	NIL
112	I2	1	NIL
113	I4	1	NIL
114	I8	2	NIL
114	I2	4	NIL

4+2

Items table

Item no	Item desc.	rate
I1	PEPSI	25
I2	BUTTER	60
I4	BREAD	22

I8	BISCUIT	60
I2	PEPSI	25

Third Normal Form

Invoice Table

Invoice No.	Invoice date	Order no	Challan no	Cust no	Invoice value
112	12/8/2014	1	1	C1	75
113	16/8/2014	1	1	C4	22
114	16/8/2014	1	1	C1	92

Customer Table

Cust no	Cust name
C1	SRIKANT
C4	KAVITA
C1	SRIKANT

Invoice Items

Invoice No.	Item no	QTY sold	Discount
112	I1	2	NIL
112	I2	1	NIL
113	I4	1	NIL
114	I8	2	NIL
114	I2	4	NIL

Items table

Item no	Item desc.	rate
I1	PEPSI	25
I2	BUTTER	60
I4	BREAD	22
I8	BISCUIT	60
I2	PEPSI	25

$\boxed{4+2}$

[4 marks for each normal form and 4+2 marks for justification for all]

Q.7 a. What is multivalued dependency? When does it arise and what type of constraints does it specify? Explain with example. (8)

Answer:

Definition: A **multivalued dependency** (**MVD**) *X* —>> *Y* specified on relation schema *R*, where *X* and *Y* are both subsets of *R*, specifies the following constraint on any relation state *r* of *R*: If two tuples t_1 and t_2 exist in *r* such that $t_1[X] = t_2[X]$, then two tuples t_3 and t_4 should also exist in *r* with the following properties, where we use *Z* to denote (*R* 2 (*X* υ *Y*)):

$\boxed{2}$

- $t_3[X] = t_4[X] = t_1[X] = t_2[X]$.

- $t_3[Y] = t_1[Y]$ and $t_4[Y] = t_2[Y]$.
- $t_3[Z] = t_2[Z]$ and $t_4[Z] = t_1[Z]$.

An MVD X —>> Y in R is called a **trivial MVD** if (a) Y is a subset of X, or (b) X υ $Y = R$.

Multivalued dependencies are a consequence of first normal form (1NF) which disallowed an attribute in a tuple to have a set of values. If we have two or more multivalued independent attributes in the same relation schema, we get into a problem of having to repeat every value of one of the attributes with every value of the other attribute to keep the relation state consistent and to maintain the independence among the attributes involved. This constraint is specified by a multivalued dependency.

3 + 2

For example, consider the relation EMP shown in Figure 15.04(a). A tuple in this EMP relation represents the fact that an employee whose name is ENAME works on the project whose name is PNAME and has a dependent whose name is DNAME. An employee may work on several projects and may have several dependents, and the employee's projects and dependents are independent of one another (Note 6). To keep the relation state consistent, we must have a separate tuple to represent every combination of an employee's dependent and an employee's project. This constraint is specified as a multivalued dependency on the EMP relation. Informally, whenever two *independent* 1:N relationships *A:B* and *A:C* are mixed in the same relation, an MVD may arise.

(4 marks for definition and 4 marks for next part)

b. Define Join dependencies. Explain fifth normal form with the help of an example. (8)

Answer:

Definition: A **join dependency** (**JD**), denoted by JD(R_1, R_2, ..., R_n), specified on relation schema R, specifies a constraint on the states r of R.

■ The constraint states that every legal state r of R should have a non-additive join decomposition into R_1, R_2, ..., R_n; that is, for every such r we have

2

■ $*(\pi_{R1}(r), \pi_{R2}(r), ..., \pi_{Rn}(r)) = r$

A join dependency JD(R_1, R_2, ..., R_n), specified on relation schema R, is a **trivial JD** if one of the relation schemas R_i in JD(R_1, R_2, ..., R_n) is equal to R.

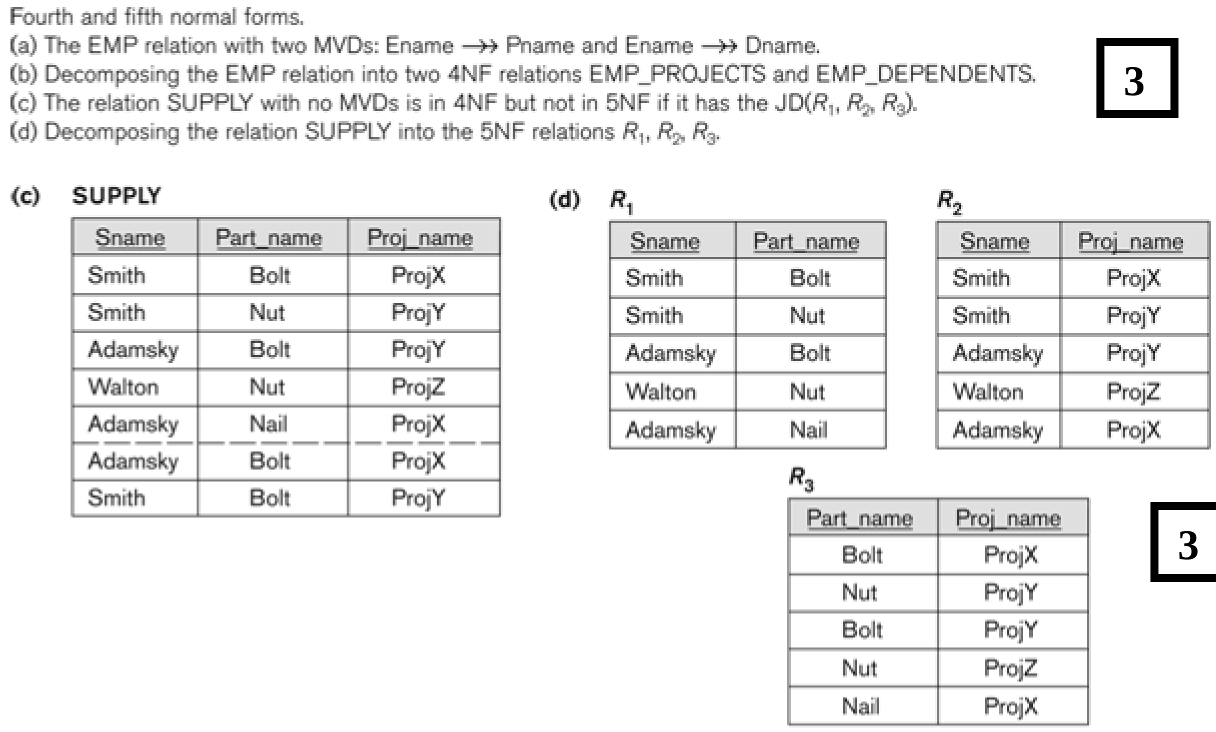
Definition: A relation schema R is in **fifth normal form** (**5NF**) (or **Project-Join Normal Form** (**PJNF**)) with respect to a set F of functional, multivalued, and join dependencies if,

■ for every nontrivial join dependency JD(R_1, R_2, ..., R_n) in F^+ (that is, implied by F),

3

■ every R_i is a superkey of R.

AnyExample



Fourth and fifth normal forms.
(a) The EMP relation with two MVDs: Ename →→ Pname and Ename →→ Dname.
(b) Decomposing the EMP relation into two 4NF relations EMP_PROJECTS and EMP_DEPENDENTS.
(c) The relation SUPPLY with no MVDs is in 4NF but not in 5NF if it has the JD(R_1, R_2, R_3).
(d) Decomposing the relation SUPPLY into the 5NF relations R_1, R_2, R_3.

3

(c) SUPPLY

Sname	Part_name	Proj_name
Smith	Bolt	ProjX
Smith	Nut	ProjY
Adamsky	Bolt	ProjY
Walton	Nut	ProjZ
Adamsky	Nail	ProjX
Adamsky	Bolt	ProjX
Smith	Bolt	ProjY

(d) R_1

Sname	Part_name
Smith	Bolt
Smith	Nut
Adamsky	Bolt
Walton	Nut
Adamsky	Nail

R_2

Sname	Proj_name
Smith	ProjX
Smith	ProjY
Adamsky	ProjY
Walton	ProjZ
Adamsky	ProjX

R_3

Part_name	Proj_name
Bolt	ProjX
Nut	ProjY
Bolt	ProjY
Nut	ProjZ
Nail	ProjX

3

(3 marks for definition and 5 marks for next part]

Q.8 a. Discuss the techniques that allow a hash file to expand and shrink dynamically. Highlight advantages and disadvantages of each. (8)

Answer:
Hashing techniques are adapted to allow the dynamic growth and shrinking of the number of file records. These techniques include the following: **dynamic hashing, extendible hashing**, and **linear hashing.** Both dynamic and extendible hashing use the **binary representation** of the hash value h(K) in order to access a **directory**. In dynamic hashing the directory is a binary tree. In extendible hashing the directory is an array of size 2^d where d is called the **global depth**.

Extendible Hashing : In extendible hashing, a type of **directory—**an array of 2*d* bucket addresses**—** is maintained, where *d* is called the **global depth** of the directory. The integer value corresponding to the first (high-order) *d* bits of a hash value is used as an index to the array to determine a directory entry, and the address in that entry determines the bucket in which the corresponding records are stored. However, there does not have to be a distinct bucket for each of the 2*d* directory locations. Several directory locations with the same first *d'* bits for their hash values may contain the same bucket address if all the records that hash to these locations fit in a single bucket. A **local depth *d'*—** stored with each bucket**—**specifies the number of bits on which the bucket contents are based. 2

The main advantage of extendible hashing that makes it attractive is that the performance of the file does not degrade as the file grows, as opposed to static external hashing where collisions increase and the corresponding chaining causes additional accesses. In addition, no space is allocated in extendible hashing for future growth, but additional buckets can be allocated dynamically as needed. The space overhead for the directory table is negligible. The maximum directory size is 2*k*, where *k* is the number of bits in the hash value. Another advantage is that splitting causes minor reorganization in most cases, since only the records in one bucket are redistributed to the two new buckets. The only time a reorganization is more expensive is when the directory has to be doubled (or halved). A disadvantage is that the directory must be searched before accessing the buckets 4 themselves, resulting in two block accesses instead of one in static hashing. This performance penalty is considered minor and hence the scheme is considered quite desirable for dynamic files.

Linear Hashing: The idea behind linear hashing is to allow a hash file to expand and shrink its number of buckets dynamically *without* needing a directory. Suppose that the file starts with *M* buckets numbered 0, 1, . . . , *M* - 1 and uses the mod hash function *h(K)* = *K* mod *M;* this hash function is called the initial hash function . Overflow because of collisions is still needed and can be handled by maintaining individual overflow chains for each bucket. However, when a collision leads to an overflow record in *any* file bucket, the *first* bucket in the file—bucket 0—is split into two buckets: the original bucket 0 and a new bucket *M* at the end of the file. The records originally in bucket 0 are distributed between the two buckets based on a different hashing function (*K*) = *K* mod 2*M*. A key property of the two hash functions and is that any records that hashed to bucket 0 based on will hash to either bucket 0 or bucket *M* based on ; this is necessary for linear hashing to work. As further collisions lead to overflow records, additional buckets are split in the *linear* order 1, 2, 3, ... If enough overflows occur, all the original file buckets 0, 1, . . . , *M* - 1 will have been split, so the file now has 2*M* instead of *M* buckets, and all buckets use the hash function . Hence, the records in overflow are eventually redistributed into regular buckets, using the function via a *delayed split* of their buckets. There is no directory; only a value *n*—which is initially set to 0 and is incremented by 1 whenever a split occurs—is needed to determine which buckets have been split. To retrieve a record with hash key value *K,* first apply the function to *K;* if (K) < *n,* then apply the function on *K* because the bucket is already split. Initially, *n* = 0, indicating that the function applies to all buckets; *n* grows linearly as buckets are split. $\boxed{2}$

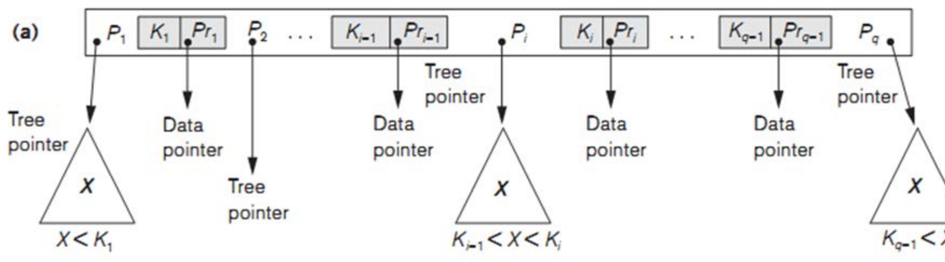
Advantage: No directory concept, Split may not happen at the bucket at which overflow happens. Gives good results for equality search (under uniform distribution assumption). Disadvantage: Poor performance under skewed distribution – as many buckets may be nearly empty $\boxed{2}$

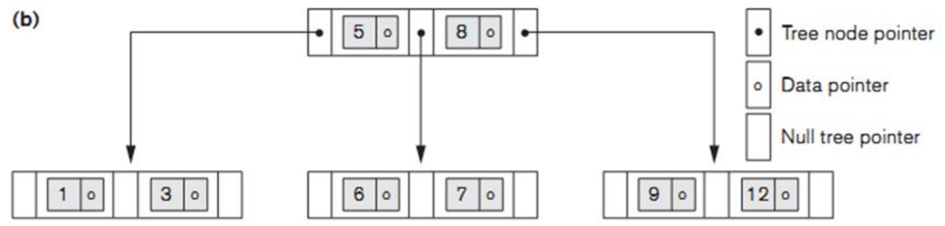
b. **Explain how does a B-tree differ from a B$^+$ tree with the help of an example? Why is a B$^+$ tree usually preferred as an access structure to a data file? (8)**

Answer:

Because of the insertion and deletion problem, most multi-level indexes use B-tree or B+-tree data structures, which leave space in each tree node (disk block) to allow for new index entries. These data structures are variations of search trees that allow efficient insertion and deletion of new search values. In B-Tree and B+-Tree data structures, each node corresponds to a disk block. Each node is kept between half-full and completely full. An insertion into a node that is not full is quite efficient; if a node is full the insertion causes a split into two nodes. Splitting may propagate to other tree levels. A deletion is quite efficient if a node does not become less than half full. If a deletion causes a node to become less than half full, it must be merged with neighboring nodes. $\boxed{3}$ In a B-tree, pointers to data records exist at all levels of the tree whereas in a B+-tree, all pointers to data records exists at the leaf-level nodes. A B+-tree can have less levels (or higher capacity of search values) than the $\boxed{1}$

corresponding B-tree, thus it is more preferred. Example

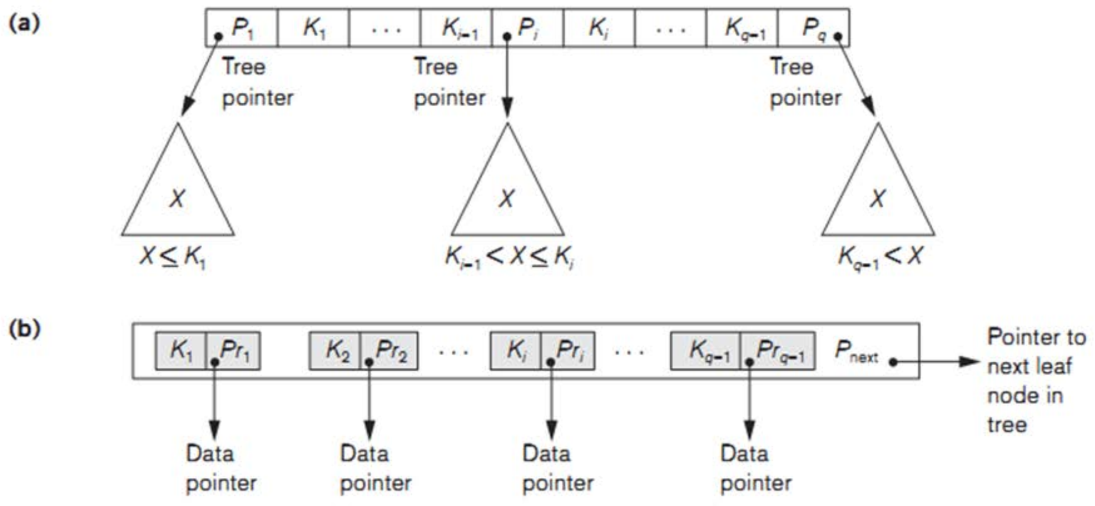




B-tree structures. (a) A node in a B-tree with $q - 1$ search values. (b) A B-tree of order $p = 3$.The values were inserted in the order 8, 5, 1, 7, 3, 12, 9, 6.

4

The nodes of a B$^+$-tree. (a) Internal node of a B$^+$-tree with $q - 1$ search values.
(b) Leaf node of a B$^+$-tree with $q - 1$ search values and $q - 1$ data pointers.


(a)
P_1 K_1 ... K_{i-1} P_i K_i ... K_{q-1} P_q
Tree pointer
Tree pointer
Tree pointer
X
$X \leq K_1$
X
$K_{i-1} < X \leq K_i$
X
$K_{q-1} < X$
(b)
K_1 Pr_1 K_2 Pr_2 ... K_i Pr_i ... K_{q-1} Pr_{q-1} P_{next}
Pointer to next leaf node in tree
Data pointer
Data pointer
Data pointer
Data pointer

Q.9 a. What is meant by the term heuristic optimization? Discuss the main heuristics that are applied during query optimization. (6)

Answer:

Cost-based optimization is expensive, even with dynamic programming. Systems may use heuristics to reduce the number of choices that must be made in a cost-based fashion. Heuristic optimization transforms the query-tree by using a set of rules that typically (but not in all cases) improve execution performance:

2

- Perform selection early (reduces the number of tuples)
- Perform projection early (reduces the number of attributes)

- Perform most restrictive selection and join operations before other similar operations.

Some systems use only heuristics, others combine heuristics with partial cost-based optimization.
Direction in heuristics is to reduce the size of intermediate results.
The **main heuristic** is to first apply the operations that reduce the size of intermediate results
E.g., Apply SELECT and PROJECT operations before applying the JOIN or other binary operations.
General heuristic optimization Algorithm

- **1- Push selections** down
- 2- Apply more **restrictive selections** first
  - Selectivity estimated by DBMS
- 3- Combine **cross products and selections** to become joins
- **4- Push projections** down

4

The main heuristics that are applied during query optimization.

1. Deconstruct conjunctive selections into a sequence of single selection operations (Equiv. rule 1.).
2. Move selection operations down the query tree for the earliest possible execution (Equiv. rules 2, 7a, 7b, 11).
3. Execute first those selection and join operations that will produce the smallest relations (Equiv. rule 6).
4. Replace Cartesian product operations that are followed by a selection condition by join operations (Equiv. rule 4a).

5. Deconstruct and move as far down the tree as possible lists of projection attributes, creating new projections where needed (Equiv. rules 3, 8a, 8b, 12).
6. Identify those subtrees whose operations can be pipelined, and execute them using pipelining). (6)

b. Discuss the cost components for a cost function that is used to estimate query execution cost. Which cost components are used most often as the basis for cost functions? Discuss the use of cost function with the help of an example. (10)

Answer:

The cost of executing a query includes the following components:

1. *Access cost to secondary storage:* This is the cost of searching for, reading, and writing data blocks that reside on secondary storage, mainly on disk. The cost of searching for records in a file depends on the type of access structures on that file, such as ordering, hashing, and primary or secondary indexes. In addition, factors such as whether the file blocks are allocated contiguously on the same disk cylinder or scattered on the disk affect the access cost.
2. *Storage cost:* This is the cost of storing any intermediate files that are generated by an execution strategy for the query.
3. *Computation cost:* This is the cost of performing in-memory operations on the data buffers during query execution. Such operations include searching for and sorting records, merging records for a join, and performing computations on field values.
4. *Memory usage cost:* This is the cost pertaining to the number of memory buffers needed during query execution.
5. *Communication cost:* This is the cost of shipping the query and its results from the database site to the site or terminal where the query originated.

3

For large databases, the main emphasis is on minimizing the access cost to secondary storage. Simple cost functions ignore other factors and compare different query execution strategies in terms of the number of block transfers between disk and main memory. For smaller databases, where most of the data in the files involved in the query can be completely stored in memory, the emphasis is on minimizing computation cost. In distributed databases, where many sites are involved, communication cost must be minimized also. It is difficult to include all the cost components in a (weighted) cost function because of the difficulty of assigning suitable weights to the cost components. That is why some cost functions consider a single factor only—disk access. $\boxed{3}$

A simple example to illustrate how estimates cost may be used. Suppose that the EMPLOYEE file has = 10,000 records stored in = 2000 disk blocks with blocking factor = 5 records/block and the following access paths:

1. A clustering index on SALARY, with levels = 3 and average selection cardinality = 20.
2. A secondary index on the key attribute SSN, with = 4 (= 1).
3. A secondary index on the nonkey attribute DNO, with = 2 and first-level index blocks = 4. There are = 125 distinct values for DNO, so the selection cardinality of DNO is sDNO = = 80.
4. A secondary index on SEX, with = 1. There are = 2 values for the sex attribute, so the average selection cardinality is = = 5000.

$\boxed{4}$

We illustrate the use of cost functions with the following examples:

(OP1): sSSN='123456789'(EMPLOYEE)

(OP2): sDNO>5(EMPLOYEE)

(OP3): sDNO=5(EMPLOYEE)

(OP4): sDNO=5 AND SALARY>30000 AND SEX='F'(EMPLOYEE)

The cost of the brute force (linear search) option S1 will be estimated as = = 2000 (for a selection on a non key attribute) or = = 1000 (average cost for a selection on a key attribute). For OP1 we can use either method S1 or method S6a; the cost estimate for S6a is = + 1 = 4 + 1 = 5, and it is chosen over Method S1, whose average cost is = 1000. For OP2 we can use either method S1 (with estimated cost = 2000) or method S6*b* (with estimated cost = + + = 2 + (4/2) + (10,000/2) = 5004), so we choose the brute force approach for OP2. For OP3 we can use either method S1 (with estimated cost = 2000) or method S6*a* (with estimated cost = + = 2 + 80 = 82), so we choose method S6a.

Finally, consider OP4, which has a conjunctive selection condition. We need to estimate the cost of using any one of the three components of the selection condition to retrieve the records, plus the brute force approach. The latter gives cost estimate = 2000. Using the condition (DNO = 5) first gives the cost estimate = 82. Using the condition (SALARY > 30,000) first gives a cost estimate = + = 3 + (2000/2) = 1003. Using the condition (SEX = 'F') first gives a cost estimate = + = 1 + 5000 = 5001. The optimizer would then choose method S6a on the secondary index on DNO because it has the lowest cost estimate. The condition (DNO = 5) is used to retrieve the records, and the remaining part of the conjunctive condition (SALARY > 30,000 AND SEX = 'F') is checked for each selected record after it is retrieved into memory.

(2+3+5 marks for each part)

<u>TEXT BOOK</u>

I Fundamentals of Database Systems, Elmasri, Navathe, Somayajulu, Gupta, Pearson Education, 2006