1. List and Explain the ACID properties of database transaction.

Transactions should possess several properties, often called the ACID properties
A Atomicity: a transaction is an atomic unit of processing and it is either performed entirely or not at all.
C Consistency Preservation: a transaction should be consistency preserving that is it must take the database from one consistent state to another.
I Isolation/Independence: A transaction should appear as though it is being executed in isolation from other transactions, even though many transactions are executed concurrently.
D Durability (or Permanency): if a transaction changes the database and is committed, the changes must never be lost because of any failure.

The atomicity property requires that we execute a transaction to completion. It is the responsibility of the transaction recovery subsystem of a DBMS to ensure atomicity.
The preservation of consistency is generally considered to be the responsibility of the programmers who write the database programs or of the DBMS module that enforces integrity constraints.
The isolation property is enforced by the concurrency control subsystem of the DBMS. If every transaction does not make its updates (write operations) visible to other transactions until it is committed, one form of isolation is enforced that solves the temporary update problem and eliminates cascading rollbacks
Durability is the responsibility of recovery subsystem.

2. Demonstrate working of Assertion & Triggers in database? Explain with an example.

Assertions are used to specify additional types of constraints outside scope of built-in relational model constraints. In SQL, users can specify general constraints via declarative assertions, using the CREATE ASSERTION statement of the DDL.Each assertion is given a constraint name and is specified via a condition similar to the WHERE clause of an SQL query.
General form :

CREATE ASSERTION <Name_of_assertion> CHECK (<cond>)

For the assertion to be satisfied, the condition specified after CHECK clause must return true.

For example, to specify the constraint that the salary of an employee must not be greater than the salary of the manager of the department that the employee works for in SQL, we can write the following assertion:

CREATE ASSERTION SALARY_CONSTRAINT
CHECK (NOT EXISTS (SELECT * FROM EMPLOYEE E, EMPLOYEE M, DEPARTMENT D WHERE E.Salary>M.Salary AND
E.Dno=D.Dnumber AND D.Mgr_ssn=M.Ssn));

The constraint name SALARY_CONSTRAINT is followed by the keyword CHECK, which is followed by a condition in parentheses that must hold true on every database state for the assertion to be satisfied. The constraint name can be used later to refer to the constraint or to

modify or drop it. Any WHERE clause condition can be used, but many constraints can be specified using the EXISTS and NOT EXISTS style of SQL conditions.
By including this query inside a NOT EXISTS clause, the assertion will specify that the result of this query must be empty so that the condition will always be TRUE. Thus, the assertion is violated if the result of the query is not empty

A trigger is a procedure that runs automatically when a certain event occurs in the DBMS. In many cases it is convenient to specify the type of action to be taken when certain events occur and when certain conditions are satisfied. The CREATE TRIGGER statement is used to implement such actions in SQL.

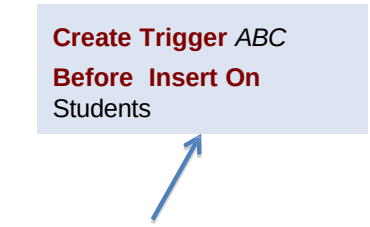
General form:

CREATE TRIGGER <name>
BEFORE | AFTER | <events>
FOR EACH ROW |FOR EACH STATEMENT
WHEN (<condition>)
<action>

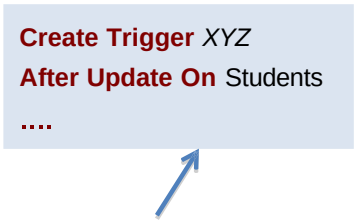
A trigger has three components

1. Event: When this event happens, the trigger is activated Three event types : Insert, Update, Delete
   Two triggering times: Before the event
   After the event
2. Condition (optional): If the condition is true, the trigger executes, otherwise skipped
3. Action: The actions performed by the trigger

When the Event occurs and Condition is true, execute the Action

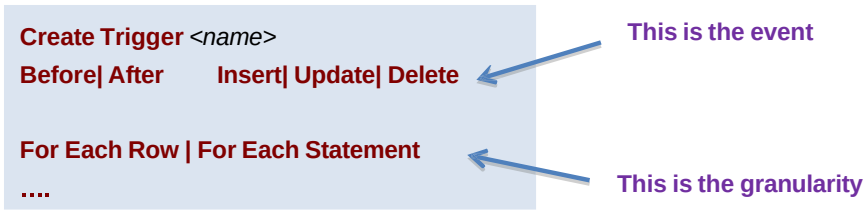


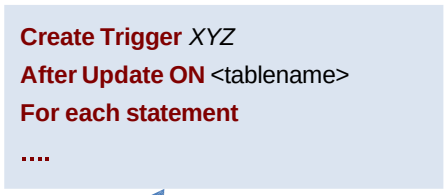
This trigger is activated when an insert statement is issued, but before the new record is inserted



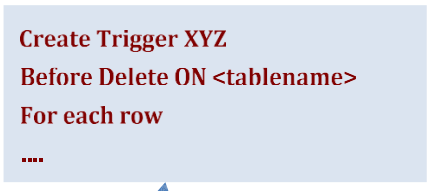
This trigger is activated when an update statement is issued and after the update is executed

Does the trigger execute for each updated or deleted record, or once for the entire statement ?. We define such granularity as follows:





This trigger is activated once (per UPDATE statement) after all records are updated



This trigger is activated before deleting each record

3. Illustrate Stored Procedure Language in SQL with an example

Stored procedures are precompiled SQL statements that are stored in the database and can be executed as a single unit. SQL Stored Procedures are a powerful feature in database management systems (DBMS) that allow developers to encapsulate SQL code and business logic. When executed, they can accept input parameters and return output, acting as a reusable unit of work that can be invoked multiple times by users, applications, or other procedures.

SYNTAX:

```
CREATE PROCEDURE procedure_name
(parameter1 data_type, parameter2 data_type, ...)
AS
BEGIN
  -- SQL statements to be executed
END
```

Example of Creating a Stored Procedure:

In this example, we create a stored procedure called **GetCustomersByCountry**, which accepts a Country parameter and returns the **CustomerName** and **ContactName** for all customers from that country. The procedure is designed to query the **Customers** table, which contains customer information, including their **names**, **contact details**, and **country**.

CustomerID	CustomerName	ContactName	Country
1	Shubham	Thakur	India
2	Aman	Chopra	Australia
3	Naveen	Tulasi	Sri Lanka
4	Aditya	Arpan	Austria
5	Nishant	Jain	Spain

By passing a country as a parameter, the stored procedure dynamically fetches the relevant customer details from the table.

QUERY:

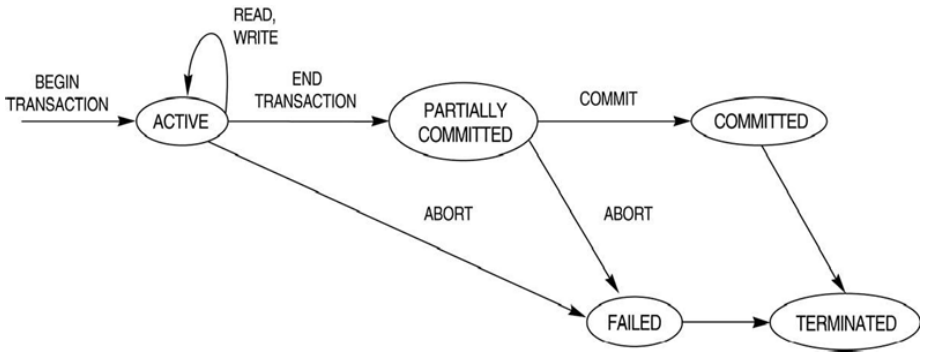
```
-- Create a stored procedure named "GetCustomersByCountry"
CREATE PROCEDURE GetCustomersByCountry
    @Country VARCHAR(50)
AS
BEGIN
    SELECT CustomerName, ContactName
    FROM Customers
    WHERE Country = @Country;
END;

-- Execute the stored procedure with parameter "Sri lanka"
EXEC GetCustomersByCountry @Country = 'Sri lanka';
```

Output:

CustomerName	Contact Name
Naveen	Tulasi

4. Demonstrate the Database Transaction with transaction diagram and explain System Log in detail



A transaction is an atomic unit of work that should either be completed in its entirety or not done at all. For recovery purposes, the system keeps track of start of a transaction, termination, commit or aborts.

BEGIN_TRANSACTION: marks the beginning of transaction execution

READ or WRITE: specify read or write operations on the database items that are executed as part of a transaction

END_TRANSACTION: specifies that READ and WRITE transaction operations have ended and marks the end of transaction execution

COMMIT_TRANSACTION: signals a *successful end* of the transaction so that any changes (updates) executed by the transaction can be safely **committed** to the database and will not be undone

ROLLBACK: signals that the transaction has *ended unsuccessfully*, so that any changes or effects that the transaction may have applied to the database must be **undone**

- A transaction goes into active state immediately after it starts execution and can execute read and write operations.
- When the transaction ends it moves to partially committed state.
- At this end additional checks are done to see if the transaction can be committed or not. If these checks are successful the transaction is said to have reached commit point and enters committed state. All the changes are recorded permanently in the db.
- A transaction can go to the failed state if one of the checks fails or if the transaction is aborted during its active state. The transaction may then have to be rolled back to undo the effect of its write operation.
- Terminated state corresponds to the transaction leaving the system. All the information about the transaction is removed from system tables.

The System Log

Log or Journal keeps track of all transaction operations that affect the values of database items

- This information may be needed to permit recovery from transaction failures.
- The log is kept on disk, so it is not affected by any type of failure except for disk or catastrophic failure

- one (or more) main memory buffers hold the last part of the log file, so that log entries are first added to the main memory buffer
- When the log buffer is filled, or when certain other conditions occur, the log buffer is
- appended to the end of the log file on disk.

In these entries, T refers to a unique **transaction-id** that is generated automatically by the system for each transaction and that is used to identify each transaction:

1. **[start_transaction, T].** Indicates that transaction T has started execution.
2. **[write_item, T, X, old_value, new_value].** Indicates that transaction T has changed the value of database item X from old_value to new_value.
3. **[read_item, T, X].** Indicates that transaction T has read the value of database item X.
4. **[commit, T].** Indicates that transaction T has completed successfully, and affirms that its effect can be committed (recorded permanently) to the database.
5. **[abort, T].** Indicates that transaction T has been aborted.

5. What are the views in SQL? Explain with example.

A view in SQL terminology is a single table that is derived from other tables. other tables can be base tables or previously defined views. A view does not necessarily exist in physical form; it is considered to be a virtual table, in contrast to base tables, whose tuples are always physically stored in the database. This limits the possible update operations that can be applied to views, but it does not provide any limitations on querying a view.
In SQL, the command to specify a view is CREATE VIEW. The view is given a (virtual) table name (or view name), a list of attribute names, and a query to specify the contents of the view.
Example 1:

CREATE VIEW WORKS_ON1
AS SELECT Fname, Lname, Pname, Hours
FROM EMPLOYEE, PROJECT, WORKS_ON
WHERE Ssn=Essn AND Pno=Pnumber;

Example 2:

CREATE VIEW DEPT_INFO(Dept_name, No_of_emps, Total_sal)
AS SELECT Dname, COUNT (*), SUM (Salary)
FROM DEPARTMENT, EMPLOYEE
WHERE Dnumber=Dno
GROUP BY Dname;

View Implementation, View Update and Inline Views:

The problem of efficiently implementing a view for querying is complex. Two main approaches have been suggested.

One strategy, called **query modification**, involves modifying or transforming the view query (submitted by the user) into a query on the underlying base tables. For example, the query
SELECT Fname, Lname
FROM WORKS_ON1
WHERE Pname='Product X';

would be automatically modified to the following query by the DBMS:
SELECT Fname, Lname
FROM EMPLOYEE, PROJECT, WORKS_ON
WHERE Ssn=Essn **AND** Pno=Pnumber
AND Pname='Product X';

In example 1, we did not specify any new attribute names for the view WORKS_ON1. In this case,WORKS_ON1 inherits the names of the view attributes from the defining tables EMPLOYEE, PROJECT, and WORKS_ON.
Example 2 explicitly specifies new attribute names for the view DEPT_INFO, using a one-to-one
correspondence between the attributes specified in the CREATE VIEW clause and those specified in the SELECT clause of the query that defines the view.

6. Explain the Two-Phase Locking Protocol used for Concurrency vControl.

- The concept of locking data items is one of the main techniques used for controlling the concurrent execution of transactions.
- A lock is a variable associated with a data item in the database. Generally there is a lock for each data item in the database.
- A lock describes the status of the data item with respect to possible operations that can be applied to that item.
- It is used for synchronizing the access by concurrent transactions to the database items. A transaction locks an object before using it
- When an object is locked by another transaction, the requesting transaction must wait

Types of Locks and System Lock Tables

Binary Locks

```
lock_item(X):
B: if LOCK(X) = 0 (* item is unlocked *)
    then LOCK(X) ←1 (* lock the item *)
   else
   begin
          wait (until LOCK(X) = 0
          and the lock manager wakes up the transaction);
           go to B
    end;
unlock_item(X):
LOCK(X) ← 0; (* unlock the item *)
if any transactions are waiting
then wakeup one of the waiting transactions ;
```

Shared/Exclusive (or Read/Write) Locks

-
-

```
read_lock(X):
B:  if LOCK(X) = "unlocked"
        then begin LOCK(X) ← "read-locked";
                   no_of_reads(X) ← 1
                   end
    else if LOCK(X) = "read-locked"
        then no_of_reads(X) ← no_of_reads(X) + 1
    else begin
             wait (until LOCK(X) = "unlocked"
                 and the lock manager wakes up the transaction);
             go to B
             end;
write_lock(X):
B:  if LOCK(X) = "unlocked"
        then LOCK(X) ← "write-locked"
    else begin
             wait (until LOCK(X) = "unlocked"
                 and the lock manager wakes up the transaction);
             go to B
             end;
```

```
unlock (X):
    if LOCK(X) = "write-locked"
        then begin LOCK(X) ← "unlocked";
                   wakeup one of the waiting transactions, if any
                   end
    else it LOCK(X) = "read-locked"
        then begin
                   no_of_reads(X) ← no_of_reads(X) –1;
                   if no_of_reads(X) = 0
                       then begin LOCK(X) = "unlocked";
                                  wakeup one of the waiting transactions, if any
                                  end
                   end;
```

-

-

7. Why is Concurrency control needed? Demonstrate with an example

Purpose of Concurrency Control

To enforce Isolation (through mutual exclusion) among conflicting transactions.

To preserve database consistency through consistency preserving execution of transactions.

To resolve read-write and write-write conflicts.

Lock and unlock operations for binary locks.

```
lock_item(X):
B: if LOCK(X) = 0 (* item is unlocked *)
    then LOCK(X) ←1 (* lock the item *)
   else
   begin
         wait (until LOCK(X) = 0
         and the lock manager wakes up the transaction);
          go to B
    end;
unlock_item(X):
LOCK(X) ← 0; (* unlock the item *)
if any transactions are waiting
then wakeup one of the waiting transactions ;
```

8. What is document based NOSQL systems? Explain basic operations CRUD in MongoDB

Document-based NOSQL systems: These systems store data in the form of documents using well-known formats, such as JSON (JavaScript Object Notation). Documents are accessible via their document id, but can also be accessed rapidly using other indexes.

Document-based or document-oriented NOSQL systems typically store data as collections of similar documents. These types of systems are also sometimes known as document stores. The individual documents somewhat resemble complex objects or XML documents, but a major difference between document-based systems versus object and object-relational systems and XML is that there is no requirement to specify a schema rather, the documents are specified as self-describing data. Although the documents in a collection should be similar, they can have different data elements (attributes), and new documents can have new data elements that do not exist in any of the current documents in the collection.

24.3.2 MongoDB CRUD Operations

MongoDb has several CRUD operations, where CRUD stands for (create, read, update, delete). Documents can be created and inserted into their collections using the insert operation, whose format is:

db.<collection_name>.insert(<document(s)>)

The parameters of the insert operation can include either a single document or an array of documents. The delete operation is called remove, and the format is:

db.<collection_name>.remove(<condition>)

The documents to be removed from the collection are specified by a Boolean con- dition on some of the fields in the collection documents. There is also an update operation, which has a condition to select certain documents, and a $set clause to specify the update. It is also possible to use the update operation to replace an existing document with another one but keep the same ObjectId.

For read queries, the main command is called find, and the format is:

db.<collection_name>.find(<condition>)

General Boolean conditions can be specified as <condition>, and the documents in the collection that return true are selected for the query result. For a full discussion of the MongoDb CRUD operations, see the MongoDB online documentation in the chapter references.

(a) **normalized project and worker documents:**

```
{
        _id:                    "P1",
        Pname:                  "ProductX",
        Plocation:              "Bellaire"

 }
 {
        _id:                    "W1",
        Ename:                  "John Smith",
        ProjectId:              "P1",
        Hours:                  32.5
 }

{       _id:                    "W2",
        Ename:                  "Joyce English",
        ProjectId:              "P1",
        Hours:                  20.0
}
```

inserting the documents in (a) into their collections "project" and "worker":

```
db.project.insert( { _id: "P1", Pname: "ProductX", Plocation: "Bellaire" } )

db.worker.insert( [ { _id: "W1", Ename: "John Smith", ProjectId: "P1", Hours: 32.5 },
                    { _id: "W2", Ename: "Joyce English", ProjectId: "P1", Hours: 20.0 } ] )
```

9. What is NOSQL Graph database? Explain Neo4j.

NOSQL Graph database : The data is represented as a graph, which is a collection of vertices (nodes) and edges. Both nodes and edges can be labeled to indicate the types of entities and relationships they represent, and it is generally possible to store data associated with both individual nodes and individual edges.

The data model in Neo4j organizes data using the concepts of nodes and relation- ships. Both nodes and relationships can have properties, which store the data items associated with nodes and relationships. Nodes can have labels; the nodes that have the same label are grouped into a collection that identifies a subset of the nodes in the database graph for querying purposes. A node can have zero, one, or several labels. Relationships are directed; each relationship has a start node and end node as well as a relationship type, which serves a similar role to a node label by identifying similar relationships that have the same relationship type. Properties can

be speci- fied via a map pattern, which is made of one or more “name : value” pairs enclosed in curly brackets; for example {Lname : ‘Smith’, Fname : ‘John’, Minit : ‘B’}.

In conventional graph theory, nodes and relationships are generally called vertices and edges. The Neo4j graph data model somewhat resembles how data is repre- sented in the ER and EER models, but with some notable differences. Comparing the Neo4j graph model with ER/EER concepts, nodes cor- respond to entities, node labels correspond to entity types and subclasses, relation- ships correspond to relationship instances, relationship types correspond to relationship types, and properties correspond to attributes. One notable difference is that a relationship is directed in Neo4j, but is not in ER/EER. Another is that a node may have no label in Neo4j, which is not allowed in ER/EER because every entity must belong to an entity type. A third crucial difference is that the graph model of Neo4j is used as a basis for an actual high-performance distributed data- base system whereas the ER/EER model is mainly used for database design.

Examples in Neo4j using the Cypher language. (a) Creating some nodes.

(a) Creating some nodes for the COMPANY data:

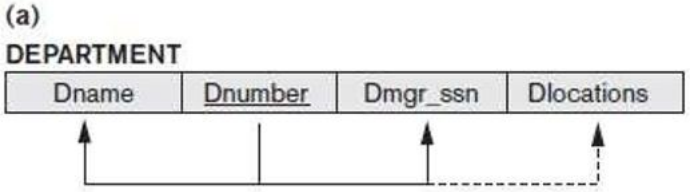
CREATE (e1: EMPLOYEE, {Empid: ‘1’, Lname: ‘Smith’, Fname: ‘John’, Minit: ‘B’}) CREATE (e2: EMPLOYEE, {Empid: ‘2’, Lname: ‘Wong’, Fname: ‘Franklin’}) CREATE (e3: EMPLOYEE, {Empid: ‘3’, Lname: ‘Zelaya’, Fname: ‘Alicia’})
CREATE (e4: EMPLOYEE, {Empid: ‘4’, Lname: ‘Wallace’, Fname: ‘Jennifer’, Minit: ‘S’})
…
CREATE (d1: DEPARTMENT, {Dno: ‘5’, Dname: ‘Research’})
CREATE (d2: DEPARTMENT, {Dno: ‘4’, Dname: ‘Administration’})
…
CREATE (p1: PROJECT, {Pno: ‘1’, Pname: ‘ProductX’}) CREATE (p2: PROJECT, {Pno: ‘2’, Pname: ‘ProductY’})
CREATE (p3: PROJECT, {Pno: ‘10’, Pname: ‘Computerization’}) CREATE (p4: PROJECT, {Pno: ‘20’, Pname: ‘Reorganization’})
…
CREATE (loc1: LOCATION, {Lname: ‘Houston’}) CREATE (loc2: LOCATION, {Lname: ‘Stafford’}) CREATE (loc3: LOCATION, {Lname: ‘Bellaire’}) CREATE (loc4: LOCATION, {Lname: ‘Sugarland’})
…

10. Explain 1NF, 2NF, 3NF and BCNF with example.

First Normal Form:

Defined to disallow multivalued attributes, composite attributes, and their combinations

It states that the domain of an attribute must include only atomic (simple, indivisible) values and that the value of any attribute in a tuple must be a single value from the domain of that attribute.



There are three main techniques to achieve first normal form for such a relation:

1. Remove the attribute Dlocations that violates 1NF and place it in a separate relation DEPT_LOCATIONS along with the primary key Dnumber of DEPARTMENT. The primary

key of this relation is the combination {Dnumber, Dlocation}. A distinct tuple in DEPT_LOCATIONS exists for each location of a department. This decomposes the non-1NF relation into two 1NF relations.

2. Expand the key so that there will be a separate tuple in the original DEPARTMENT relation for each location of a DEPARTMENT. In this case, the primary key becomes the combination {Dnumber, Dlocation}. This solution has the disadvantage of introducing redundancy in the relation

DEPARTMENT

Dname	Dnumber	Dmgr_ssn	Dlocation
Research	5	333445555	Bellaire
Research	5	333445555	Sugarland
Research	5	333445555	Houston
Administration	4	987654321	Stafford
Headquarters	1	888665555	Houston

3. If a maximum number of values is known for the attribute for example, if it is known that at most three locations can exist for a department replace the Dlocations attribute by three atomic attributes: Dlocation1, Dlocation2, and Dlocation3. This solution has the disadvantage of introducing NULL values if most departments have fewer than three locations. Querying on this attribute becomes more difficult; for example, consider how you would write the query: List the departments design.

First normal form also disallows multivalued attributes that are themselves composite. These are called nested relations because each tuple can have a relation within it.

(a)

EMP_PROJ

		Projs	
Ssn	Ename	Pnumber	Hours

The schema of this EMP_PROJ relation can be represented as follows:

EMP_PROJ (Ssn, Ename, {PROJS (Pnumber, Hours)})

Decomposition and primary key propagation yield the schemas EMP_PROJ1 and EMP_PROJ2,

EMP_PROJ1

Ssn	Ename

EMP_PROJ2

Ssn	Pnumber	Hours

EMP_PROJ

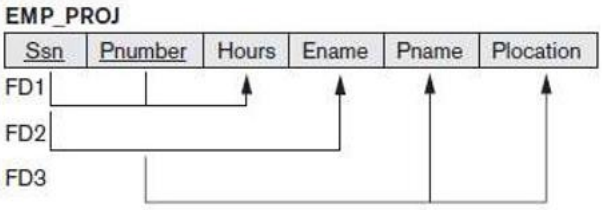
Ssn	Ename	Pnumber	Hours
123456789	Smith, John B.	1	32.5
		2	7.5
666884444	Narayan, Ramesh K.	3	40.0
453453453	English, Joyce A.	1	20.0
		2	20.0
333445555	Wong, Franklin T.	2	10.0
		3	10.0
		10	10.0
		20	10.0
999887777	Zelaya, Alicia J.	30	30.0
		10	10.0
987987987	Jabbar, Ahmad V.	10	35.0
		30	5.0
987654321	Wallace, Jennifer S.	30	20.0
		20	15.0
888665555	Borg, James E.	20	NULL

Second Normal Form

Second normal form (2NF) is based on the concept of full functional dependency

A functional dependency $X \rightarrow Y$ full functional dependency if removal of any attribute A from X means that the dependency does not hold any more; that is, for any attribute $A \varepsilon X$, $(X - \{A\})$ does not functionally determine Y.

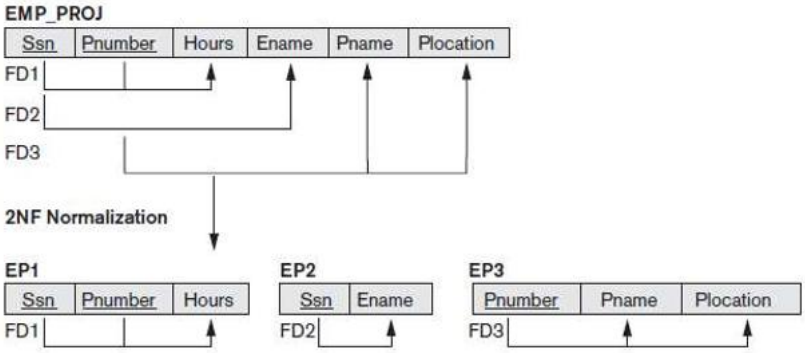
Definition. A relation schema R is in 2NF if every nonprime attribute A in R is fully functionally dependent on the primary key of R



In the above figure , {Ssn, Pnumber} → Hours is a full dependency (neither Ssn → Hours nor Pnumber→Hours holds)

If a relation schema is not in 2NF, it can be second normalized or 2NF normalized into a number of 2NF relations in which nonprime attributes are associated only with the part of the primary key on which they are fully functionally dependent.

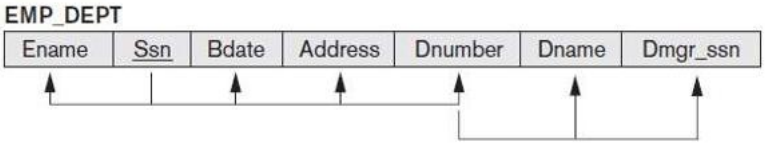
Therefore, the functional dependencies FD1, FD2, and FD3 lead to the decomposition of EMP_PROJ into the three relation schemas EP1, EP2, and EP3 shown in Figure below,
each of which is in 2NF.



Third Normal Form

Transitive functional dependency

A functional dependency $X \rightarrow Y$ in a relation schema R is a transitive dependency if there exists a set of attribute Z that are neither a primary nor a subset of any key of R(candidate key) and both $X \rightarrow Z$ and $Y \rightarrow Z$ holds

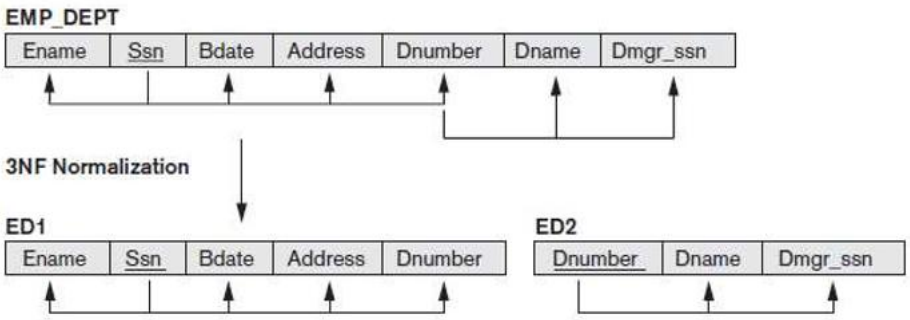


SSN → DMGRSSN is a transitive FD since SSN → DNUMBER and DNUMBER → DMGRSSN hold.

Dnumber is neither a key itself nor a subset of the key of EMP_DEPT

SSN → ENAME is non-transitive since there is no set of attributes X where SSN → X and X → ENAME

Definition: A relation schema R is in third normal form (3NF) if it is in 2NF and no non-prime attribute A in R is transitively dependent on the primary key.



ED1 and ED2 represent independent entity facts about employees and departments.

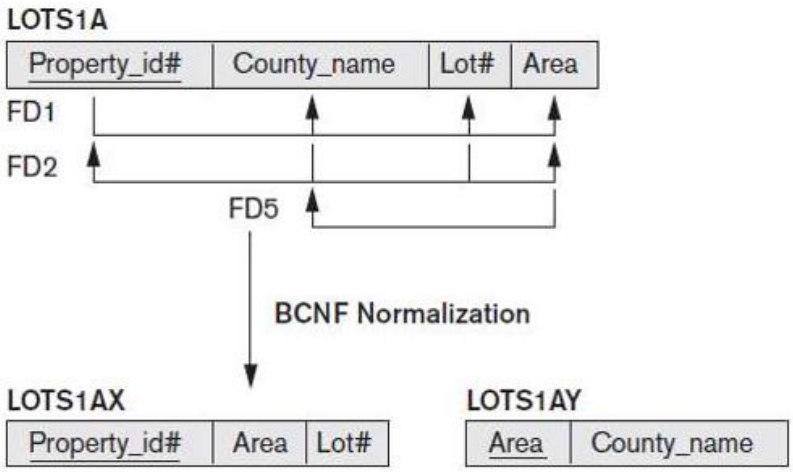
Boyce-Codd Normal Form

Boyce-Codd normal form (BCNF) was proposed as a simpler form of 3NF, but it was found to be stricter than 3NF

Every relation in BCNF is also in 3NF; however, a relation in 3NF is not necessarily in BCNF

Definition. A relation schema R is in **BCNF** if whenever a nontrivial functional

dependency X→A holds in R, then X is a superkey of R



11.Explain Informal Design Guidelines for relational schema design.

Making sure that the semantics of the attributes is clear in the schema
semantics of a relation refers to its meaning resulting from the interpretation of attribute values in a tuple
Whenever we group attributes to form a relation schema, we assume that attributes belonging to one relation have certain real-world meaning and a proper interpretation associated with them

The easier it is to explain the semantics of the relation, the better the relation schema design will be

Guideline 1

Design a relation schema so that it is easy to explain its meaning

Do not combine attributes from multiple entity types and relationship types into a single relation.

Examples of Violating Guideline 1

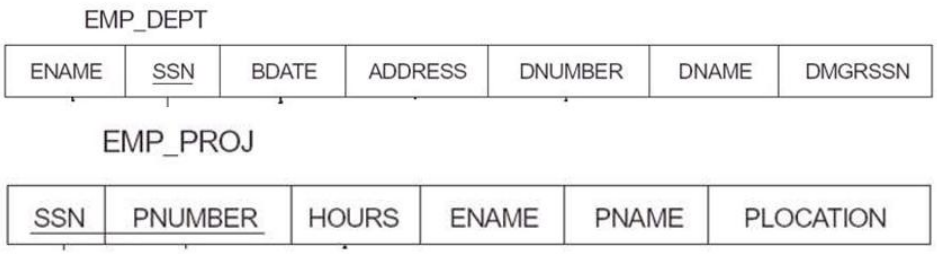


Fig: schema diagram for company database

- **Reducing the redundant information in tuples**

- One goal of schema design is to minimize the storage space used by the base relations
- Grouping attributes into relation schemas has a significant effect on storage space
- For example, compare the space used by the two base relations EMPLOYEE and DEPARTMENT with that for an EMP_DEPT base relation
- In EMP_DEPT, the attribute values pertaining to a particular department (Dnumber, Dname, Dmgr_ssn) are repeated for every employee who works for that department

EMPLOYEE

Fname	Minit	Lname	Ssn	Bdate	Address	gender	Salary	Super_ssn	Dno
John	B	Smith	123456789	1965-01-09	731 Fondren, Houston, TX	M	30000	333445555	5
Franklin	T	Wong	333445555	1955-12-08	638 Voss, Houston, TX	M	40000	888665555	5
Alicia	J	Zelaya	999887777	1968-01-19	3321 Castle, Spring, TX	F	25000	987654321	4
Jennifer	S	Wallace	987654321	1941-06-20	291 Berry, Bellaire, TX	F	43000	888665555	4
Ramesh	K	Narayan	666884444	1962-09-15	975 Fire Oak, Humble, TX	M	38000	333445555	5
Joyce	A	English	453453453	1972-07-31	5631 Rice, Houston, TX	F	25000	333445555	5
Ahmad	V	Jabbar	987987987	1969-03-29	980 Dallas, Houston, TX	M	25000	987654321	4
James	E	Borg	888665555	1937-11-10	450 Stone, Houston, TX	M	55000	NULL	1

Figure 1: One possible database state for the COMPANY relational database schema

DEPARTMENT

Dname	Dnumber	Mgr_ssn	Mgr_start_date
Research	5	333445555	1988-05-22
Administration	4	987654321	1995-01-01
Headquarters	1	888665555	1981-06-19

DEPT_LOCATIONS

Dnumber	Dlocation
1	Houston
4	Stafford
5	Bellaire
5	Sugarland
5	Houston

PROJECT

Pname	Pnumber	Plocation	Dnum
ProductX	1	Bellaire	5
ProductY	2	Sugarland	5
ProductZ	3	Houston	5
Computerization	10	Stafford	4
Reorganization	20	Houston	1
Newbenefits	30	Stafford	4

WORKS_ON

Essn	Pno	Hours
123456789	1	32.5
123456789	2	7.5
666884444	3	40.0
453453453	1	20.0
453453453	2	20.0
333445555	2	10.0
333445555	3	10.0
333445555	10	10.0
333445555	20	10.0
999887777	30	30.0
999887777	10	10.0
987987987	10	35.0
987987987	30	5.0
987654321	30	20.0
987654321	20	15.0
888665555	20	NULL

DEPENDENT

Essn	Dependent_name	gender	Bdate	Relationship
333445555	Alice	F	1986-04-05	Daughter
333445555	Theodore	M	1983-10-25	Son
333445555	Joy	F	1958-05-03	Spouse
987654321	Abner	M	1942-02-28	Spouse
123456789	Michael	M	1988-01-04	Son
123456789	Alice	F	1988-12-30	Daughter
123456789	Elizabeth	F	1967-05-05	Spouse

Redundancy

EMP_DEPT

Ename	Ssn	Bdate	Address	Dnumber	Dname	Dmgr_ssn
Smith, John B.	123456789	1965-01-09	731 Fondren, Houston, TX	5	Research	333445555
Wong, Franklin T.	333445555	1955-12-08	638 Voss, Houston, TX	5	Research	333445555
Zelaya, Alicia J.	999887777	1968-07-19	3321 Castle, Spring, TX	4	Administration	987654321
Wallace, Jennifer S.	987654321	1941-06-20	291 Berry, Bellaire, TX	4	Administration	987654321
Narayan, Ramesh K.	666884444	1962-09-15	975 FireOak, Humble, TX	5	Research	333445555
English, Joyce A.	453453453	1972-07-31	5631 Rice, Houston, TX	5	Research	333445555
Jabbar, Ahmad V.	987987987	1969-03-29	980 Dallas, Houston, TX	4	Administration	987654321
Borg, James E.	888665555	1937-11-10	450 Stone, Houston, TX	1	Headquarters	888665555

Redundancy Redundancy

EMP_PROJ

Ssn	Pnumber	Hours	Ename	Pname	Plocation
123456789	1	32.5	Smith, John B.	ProductX	Bellaire
123456789	2	7.5	Smith, John B.	ProductY	Sugarland
666884444	3	40.0	Narayan, Ramesh K.	ProductZ	Houston
453453453	1	20.0	English, Joyce A.	ProductX	Bellaire
453453453	2	20.0	English, Joyce A.	ProductY	Sugarland
333445555	2	10.0	Wong, Franklin T.	ProductY	Sugarland
333445555	3	10.0	Wong, Franklin T.	ProductZ	Houston
333445555	10	10.0	Wong, Franklin T.	Computerization	Stafford
333445555	20	10.0	Wong, Franklin T.	Reorganization	Houston
999887777	30	30.0	Zelaya, Alicia J.	Newbenefits	Stafford
999887777	10	10.0	Zelaya, Alicia J.	Computerization	Stafford
987987987	10	35.0	Jabbar, Ahmad V.	Computerization	Stafford
987987987	30	5.0	Jabbar, Ahmad V.	Newbenefits	Stafford
987654321	30	20.0	Wallace, Jennifer S.	Newbenefits	Stafford
987654321	20	15.0	Wallace, Jennifer S.	Reorganization	Houston
888665555	20	Null	Borg, James E.	Reorganization	Houston

Storing natural joins of base relations leads to an additional problem referred to as update anomalies. These can be classified into:

insertion anomalies

deletion anomalies,

modification anomalies

Guideline 2

Design the base relation schemas so that no insertion, deletion, or modification anomalies are present in the relations

If any anomalies are present, note them clearly and make sure that the programs that update the database will operate correctly

The second guideline is consistent with and, in a way, a restatement of the first guideline These guidelines may sometimes have to be violated in order to improve the performance of certain queries.

- **Reducing the NULL values in tuples**

If many of the attributes do not apply to all tuples in the relation, we end up with many NULLs in those tuples

- this can waste space at the storage level
- may lead to problems with understanding the meaning of the attributes
- may also lead to problems with specifying JOIN operations
- how to account for them when aggregate operations such as COUNT or SUM are applied

NULLs can have multiple interpretations, such as the following:

The attribute does not apply to this tuple. For example, Visa_status may not apply to U.S. students.

The attribute value for this tuple is unknown. For example, the Date_of_birth may be unknown for an employee.

The value is known but absent; that is, it has not been recorded yet. For example, the Home_Phone_Number for an employee may exist, but may not be available and recorded yet.

Guideline 3

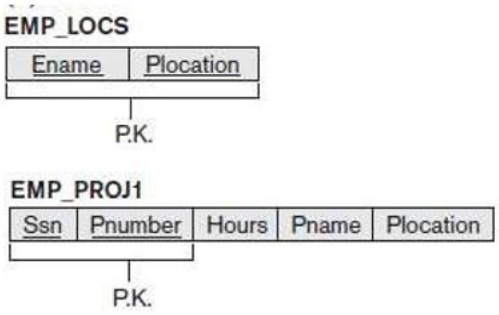
As far as possible, avoid placing attributes in a base relation whose values may frequently be NULL.

If NULLs are unavoidable, make sure that they apply in exceptional cases only and do not apply to a majority of tuples in the relation.

Using space efficiently and avoiding joins with NULL values are the two overriding criteria that determine whether to include the columns that may have NULLs in a relation or to have a separate relation for those columns with the appropriate key columns.

- **Disallowing the possibility of generating spurious tuples.**

Consider the two relation schemas EMP_LOCS and EMP_PROJ1 which can be used instead of the single EMP_PROJ.



A tuple in EMP_LOCS means that the employee whose name is Ename works on some project whose location is Plocation
A tuple in EMP_PROJ1 refers to the fact that the employee whose Social Security number is Ssn works Hours per week on the project whose name, number, and location are Pname, Pnumber, and Plocation.

Ssn	Pnumber	Hours	Pname	Plocation	Ename
123456789	1	32.5	ProductX	Bellaire	Smith, John B.
123456789	1	32.5	ProductX	Bellaire	English, Joyce A.
123456789	2	7.5	ProductY	Sugarland	Smith, John B.
123456789	2	7.5	ProductY	Sugarland	English, Joyce A.
123456789	2	7.5	ProductY	Sugarland	Wong, Franklin T.
666884444	3	40.0	ProductZ	Houston	Narayan, Ramesh K.
666884444	3	40.0	ProductZ	Houston	Wong, Franklin T.
453453453	1	20.0	ProductX	Bellaire	Smith, John B.
453453453	1	20.0	ProductX	Bellaire	English, Joyce A.
453453453	2	20.0	ProductY	Sugarland	Smith, John B.
453453453	2	20.0	ProductY	Sugarland	English, Joyce A.
453453453	2	20.0	ProductY	Sugarland	Wong, Franklin T.
333445555	2	10.0	ProductY	Sugarland	Smith, John B.
333445555	2	10.0	ProductY	Sugarland	English, Joyce A.
333445555	2	10.0	ProductY	Sugarland	Wong, Franklin T.
333445555	3	10.0	ProductZ	Houston	Narayan, Ramesh K.
333445555	3	10.0	ProductZ	Houston	Wong, Franklin T.
333445555	10	10.0	Computerization	Stafford	Wong, Franklin T.
333445555	20	10.0	Reorganization	Houston	Narayan, Ramesh K.
333445555	20	10.0	Reorganization	Houston	Wong, Franklin T.

Guideline 4

Design relation schemas so that they can be joined with equality conditions on attributes that are appropriately related (primary key, foreign key) pairs in a way that guarantees that no spurious tuples are generated
Avoid relations that contain matching attributes that are not (foreign key, primary key) combinations because joining on such attributes may produce spurious tuples

12. What is NOSQL? Explain CAP theorem.

NoSQL, or **"Not Only SQL,"** is a database management system (DBMS) designed to handle large volumes of unstructured and semi-structured data. Unlike traditional relational databases that use tables and pre-defined schemas, NoSQL databases provide **flexible data models** and support **horizontal scalability**, making them ideal for modern applications that require real-time data processing.

CAP theorem:

The CAP theorem is a fundamental concept in distributed systems theory that was first proposed by Eric Brewer in 2000 and subsequently shown by Seth Gilbert and Nancy Lynch in 2002. It asserts that all three of the following qualities cannot be concurrently guaranteed in any distributed data system:

1. Consistency

Consistency means that all the nodes (databases) inside a network will have the same copies of a replicated data item visible for various transactions. It guarantees that every node in a distributed cluster returns the same, most recent, and successful write. It refers to every client having the same view of the data. There are various types of consistency models. Consistency in CAP refers to sequential consistency, a very strong form of consistency.

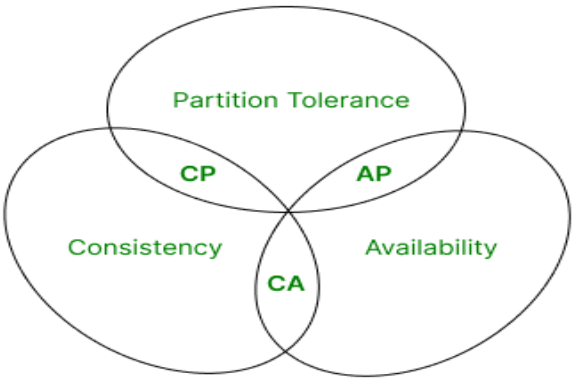
2. Availability

Availability means that each read or write request for a data item will either be processed successfully or will receive a message that the operation cannot be completed. Every non-failing node returns a response for all the read and write requests in a reasonable amount of time. The key word here is "every". In simple terms, every node (on either side of a network partition) must be able to respond in a reasonable amount of time.

3. Partition Tolerance

Partition tolerance means that the system can continue operating even if the network connecting the nodes has a fault that results in two or more partitions, where the nodes in each partition can only communicate among each other. That means, the system continues to function and upholds its consistency guarantees in spite of network partitions. Network partitions are a fact of life. Distributed systems guaranteeing partition tolerance can gracefully recover from partitions once the partition heals.

he CAP theorem states that distributed databases can have at most two of the three properties: consistency, availability, and partition tolerance. As a result, database systems prioritize only two properties at a time.



Venn diagram of CAP theorem