DBMS Question Bank

1. List the advantages of using database approach and Explain briefly.

Controlling Redundancy
Restricting Unauthorized Access
Providing Persistent Storage for Program Objects
Providing Storage Structures and Search Techniques for Efficient Query Processing
Providing Backup and Recovery
Providing Multiple User Interfaces
Representing Complex Relationships among Data
Enforcing Integrity Constraints
Permitting Inferencing and Actions Using Rules and Triggers
Potential for Enforcing Standards
Reduced Application Development Time
Flexibility.
Availability of Up-to-Date Information
Economies of Scale

2. Describe the following terms with suitable example 10 marks

i) Referential integrity

The Referential Integrity Constraint is specified between 2 relations and is used to maintain the consistency among tuples of the 2 relations. Informally, the referential integrity constraint states that 'a tuple in 1 relation that refers to another relation must refer to an existing tuple in that relation. We can diagrammatically display the referential integrity constraints by drawing a directed arc from each foreign key to the relation it references. The arrowhead may point to the primary key of the referenced relation.

- 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.
- 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.

ii) Super key

The set of fields that contains a key is called as a 'super key'. The set of 1 or more attributes that allows us to identify uniquely an entity in the entity set. A super key specifies a uniqueness constraint that no 2 distinct tuples can have the same value. Every relation has at least 1 default super key as the set of all attributes. Example:

Students (Relation) Name (Fields)
Login Age Gross

One of the super key = {Sid, Name, Login, Gross}
Any two distinct tuples t1 and t2 in a relation state r of R, we have the constraint that: $t1[SK] \neq t2[SK]$

iii) Meta-data

The DDL compiler processes schema definitions, specified in the DDL, and stores descriptions of the schemas (meta-data) in the DBMS catalog. The catalog includes information such as the names and sizes of files, names and data types of data items, storage details of each file, mapping information among schemas, and constraints.

or

The DBMS stores the descriptions of the schema constructs and constraints—also called the meta-data—in the DBMS catalog so that DBMS software can refer to the schema whenever it needs to. The schema is sometimes called the intension.

or

The complete definition or description of the database structure and constraints. This definition is stored in the DBMS catalog, which contains information such as the structure of each file, the type and storage format of each data item, and various constraints on the data. The information stored in the catalog is called meta-data

iv) Database Administrator

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. These are the people, who maintain and design the database daily. DBA is responsible for the following issues.

- Design of the conceptual and physical schemas: The DBA is responsible for interacting with the users of the system to understand what data is to be stored in the DBMS and how it is likely to be used. The DBA creates the original schema by writing a set of definitions and is Permanently stored in the 'Data Dictionary'.
- Security and Authorization: The DBA is responsible for ensuring the unauthorized data access is not permitted. The granting of different types of authorization allows the DBA to regulate which parts of the database various users can access.
- Storage structure and Access method definition: The DBA creates appropriate storage structures and access methods by writing a set of definitions, which are translated by the DDL compiler.

- Data Availability and Recovery from Failures: The DBA must take steps to ensure that if the system fails, users can continue to access as much of the uncorrupted data as possible. The DBA also work to restore the data to consistent state.

- Database Tuning: The DBA is responsible for modifying the database to ensure adequate Performance as requirements change.
- Integrity Constraint Specification: The integrity constraints are kept in a special system structure that is consulted by the DBA whenever an update takes place in the system.

v) Data independence

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), to change constraints, or to reduce the database

Physical data independence is the capacity to change the internal schema without having to change the conceptual schema. Hence, the external schemas need not be changed as well.

3. Construct an E-R diagram for a hospital database with a set of patients and a set of medical doctors. Associate with each patient a log of the various tests and examinations conducted. The patient has a history of Medicine purchased for their treatment based on prescription, and also specify price for those medicines. Identify the relation among the schemas and represent the cardinality ratio. 10 marks

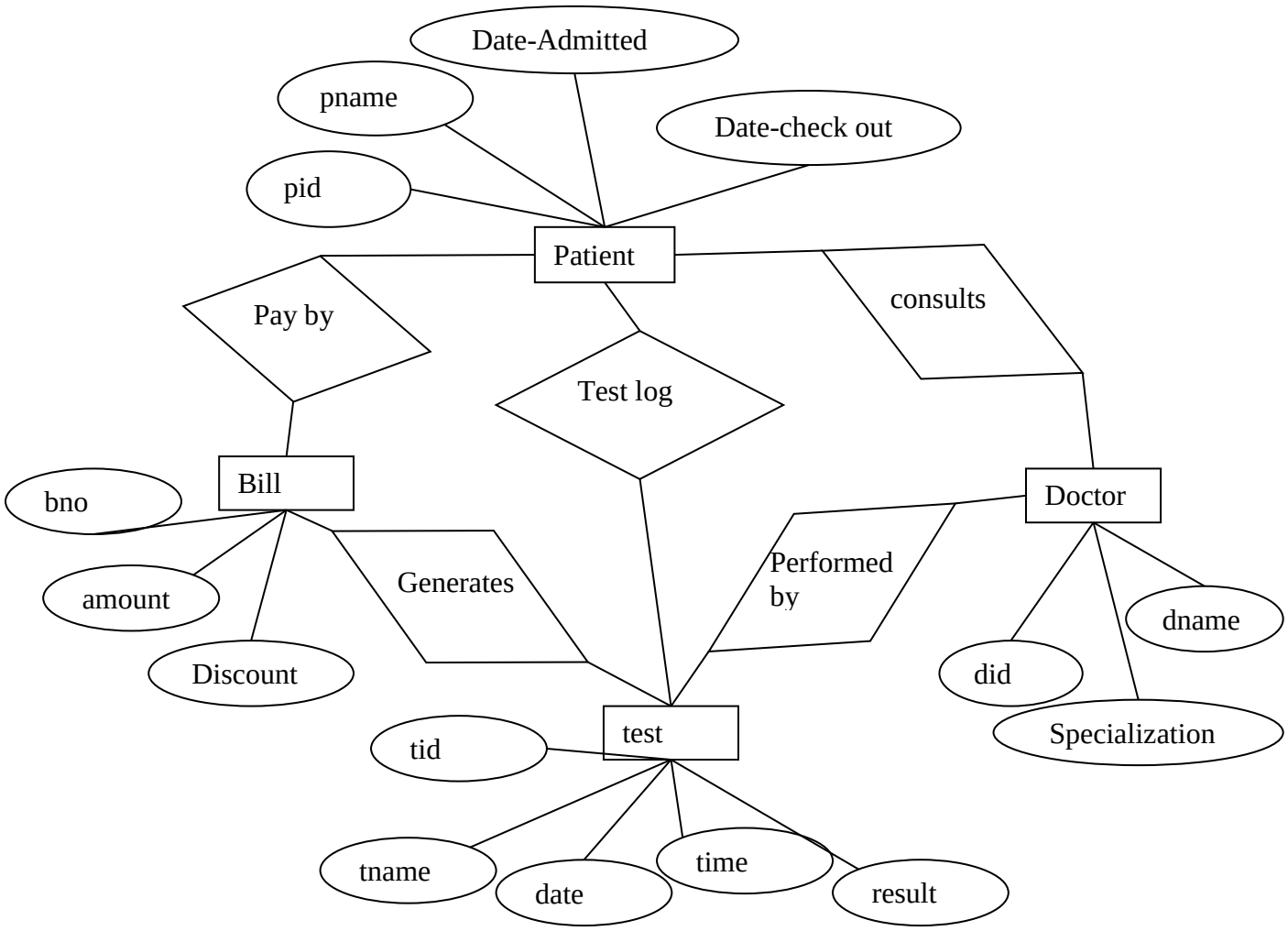
Schemas

Patient(Pid, Pname, date _admitted, Date_check_out)
Doctor(did, dname, specialization, Pid)
Test(tno,tname, tdate,ttime, result, pid,did)
Bill(bno, pid, tid,amount, discount)

Primary and foreign keys

Relationships

1. Patient is consulted by Doctor
2. Doctor performs the test
3. Patient has test records
4. Test generated the bill
5. Bill paid by patient



Cardinality ratio

4. Consider following relation schema and write the Relation Algebra
Sailors(sid: integer, sname: string, rating: integer, age: real)
Boats(bid: integer, bname: string, color: string)
Reserves(sid: integer, bid: integer, day: date) 10 marks

i) Retrieve names of sailors who have reserved a red and a green boat.
T1←Π(sanme) (σ(color=”red”)((sailors* reserves)* boats)
T2←Π(sanme) (σ(color=”green”)((sailors* reserves)* boats)
T3←T1∪T2

ii) Retrieve sid and names of sailors with age more than 20, who have not Reserved a Red boat.
T1←Π(sid,sanme) (σ(color≠”red” and age>20)((sailors* reserves)* boats)

iii) Retrieve the boat names which are not reserved by any Sailors.
T1←Π(banme)(boats)
T2←Π(banme) (reserves* boats)
T3←T1-T2

iv) Retrieve the sailor names who have reserved yellow boat on Sunday.
T1←Π(sanme) (σ(color=”yellow” and day=”Sunday”)((sailors* reserves)* boats)

v) Retrieve the boat names which has reserved by highest rating sailors.
T1←Π(banme)(σ(rating=(F max(rating) Sailors) ((sailors* reserves)* boats)

5. Consider following Schema and Write the SQL queries.
Flights(fino: integer, from_place: string, to_place: string, distance: integer, departs: time, arrives: time)
Aircraft(aid: integer, aname: string, cruisingrange: integer, fino: integer)
Certified(eid: integer, aid: integer)
Employees(eid: integer, ename: string, salary: integer)

Note that the Employees relation describes pilots and other kinds of employees as well; every pilot is certified for some aircraft (otherwise, he or she would not qualify as a pilot), and only pilots are certified to fly. 10 marks

i. Find the eids and employee names of pilots certified for some Boeing aircraft.

Select eid,ename from aircraft a,certified c, employees e where a.aid=c.aid and c.eid=e.eid and aname=”Boeing”

ii. Find the aids and aircraft names that can be used on non-stop flights from Bangalore to Madras.

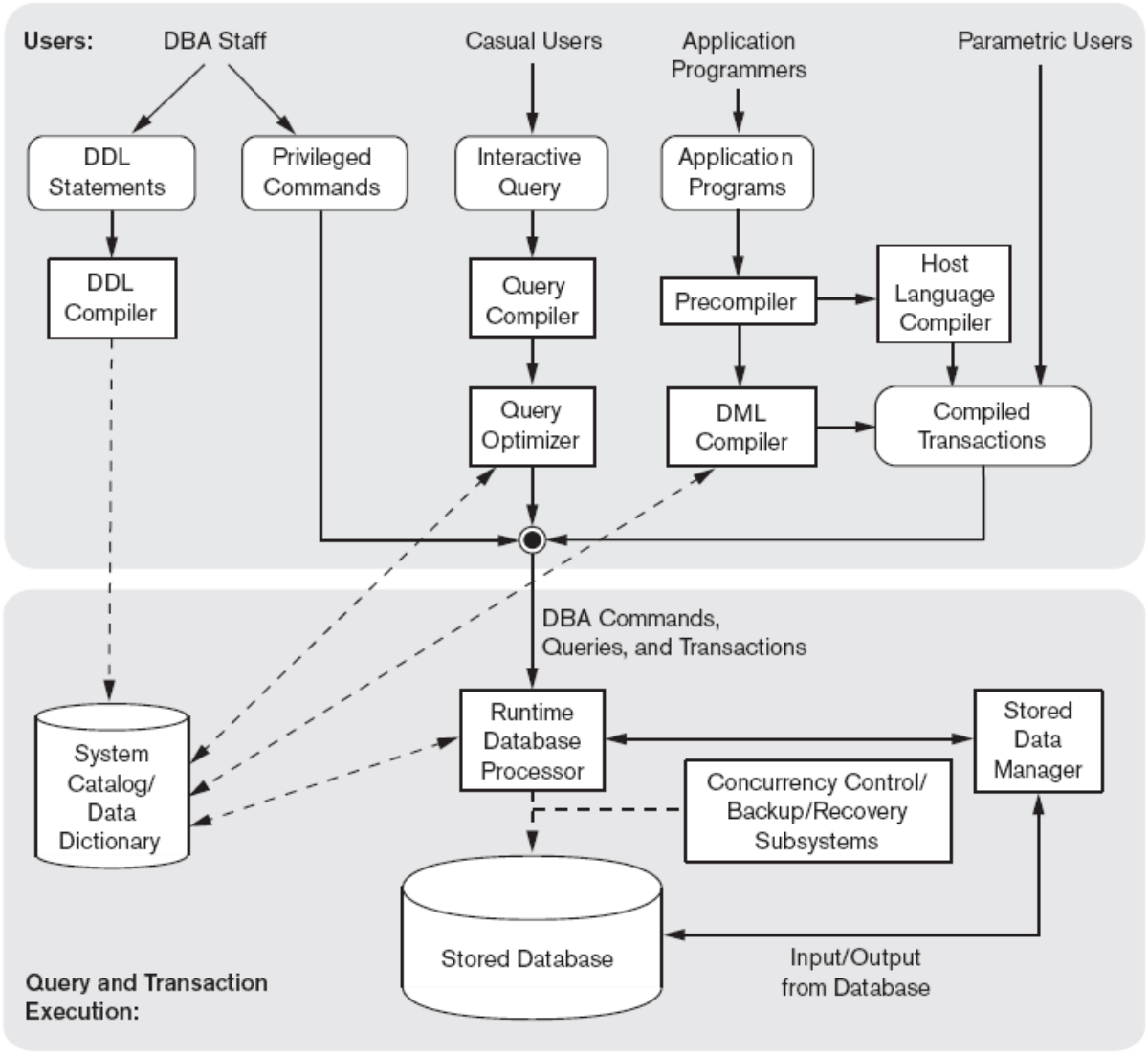
Select aid,aname form aircraft a, flights f where a.fino=f.fino and from_place=’bangalore’ and to_place=’madras’

iii. Identify the flights that can be piloted by every pilot whose salary is more than ₹100,000.
Select aid, aname from aircraft where aid in (select aid from certified where eid in(Select eid from employee where salary>100000))

iv. Find the names of pilots who can operate planes with a range greater than 3,000 miles but are not certified on any Boeing aircraft.
Select eid,ename from employee e, certified c where e.eid= c.eid and c.acid in (Select aid from aircraft where crusingrange>3000 and aname!=”Boeing”)

v. Find the eids and names of employees who are certified for the largest number of aircraft.
Create view maxcer as (select eid,count(*) as acount from certified group by(eid))
Slect eid,name from employee where eid in(select eid from certified where from certified group by (eid) having coun(*)=select from max(account) from maxcer)

6. Draw the DBMS component module diagram; explain their interactions with every module. 10 marks



Component modules of a DBMS and their interactions.

Users
DDL compiler
Query compiler
Stored data manger
Runtime processor
System catalog

7. Explain the select, project and rename operations in relational algebra with suitable example. 10 marks

Selection Operator(σ)

Selection and Projection are unary operators.
The selection operator is sigma: σ

The selection operation acts like a filter on a relation by returning only a certain number of tuples.
σC(R) Returns only those tuples in R that satisfy condition C
A condition C can be made up of any combination of comparison or logical operators that operate on the attributes of R. Comparison operators: >,<,<=,>=, ≠,==
Logical operators
ך - not, v - or
Example
Select only those Employees in the CS department:
σ Dept= 'CS'(EMP)

Projection(π)

Projection is also a Unary operator.
The Projection operator is pi: π
Projection limits the attributes that will be returned from the original relation.
The general syntax is: π attributes R
Where attributes is the list of attributes to be displayed and R is the relation.
The resulting relation will have the same number of tuples as the original relation (unless there are duplicate tuples produced).
The degree of the resulting relation may be equal to or less than that of the original relation
Project only the names and departments of the employees:
πname, dept(EMP)

Rename Operation (ρ)
The results of relational algebra are also relations but without any name. The rename operation allows us to rename the output relation. 'rename' operation is denoted with small Greek letter rho ρ.

Notation − ρ_x (E)

Where the result of expression **E** is saved with name of **x**.

8. Specify the list constraints in SQL with default and check, write proper syntax and examples. 10 marks

SQL Constraints

SQL constraints are used to specify rules for the data in a table.

Constraints are used to limit the type of data that can go into a table. This ensures the accuracy and reliability of the data in the table. If there is any violation between the constraint and the data action, the action is aborted.

Constraints can be column level or table level. Column level constraints apply to a column, and table level constraints apply to the whole table.

The following constraints are commonly used in SQL:

- **NOT NULL** - Ensures that a column cannot have a NULL value
- **UNIQUE** - Ensures that all values in a column are different
- **PRIMARY KEY** - A combination of a NOT NULL and UNIQUE. Uniquely identifies each row in a table
- **FOREIGN KEY** - Uniquely identifies a row/record in another table
- **CHECK** - Ensures that all values in a column satisfies a specific condition
- **DEFAULT** - Sets a default value for a column when no value is specified
- **INDEX** - Used to create and retrieve data from the database very quickly

```
CREATE TABLE EMPLOYEE ( … , Dno INT NOT NULL
DEFAULT 1,
                    CONSTRAINT EMPPK PRIMARY KEY (Ssn),
                    CONSTRAINT EMPSUPERFK FOREIGN KEY (Super_ssn)
                REFERENCES EMPLOYEE(Ssn) ON DELETE SET NULL ON
                UPDATE CASCADE,
                    CONSTRAINT EMPDEPTFK FOREIGN KEY(Dno)
                REFERENCES DEPARTMENT(Dnumber) ON DELETE SET DEFAULT
                ON UPDATE CASCADE);

CREATE TABLE DEPARTMENT ( … , Mgr_ssn CHAR(9) NOT NULL DEFAULT
'888665555', … ,
                    CONSTRAINT DEPTPK PRIMARY KEY(Dnumber),
                    CONSTRAINT DEPTSK UNIQUE (Dname),
                    CONSTRAINT DEPTMGRFK FOREIGN KEY (Mgr_ssn) REFERENCES
            EMPLOYEE(Ssn) ON DELETE SET DEFAULT ON UPDATE CASCADE);


CREATE TABLE DEPT_LOCATIONS ( … , PRIMARY KEY (Dnumber, Dlocation),
FOREIGN KEY (Dnumber) REFERENCES DEPARTMENT(Dnumber) ON
DELETE CASCADE ON UPDATE CASCADE);
```

9. Why JDBC is required? Describe the different steps involved in JDBC process.

ODBC and JDBC, short for Open Database Connectivity and Java Database Connectivity, also enable the integration of SQL with a general-purpose programming language. Both ODBC and JDBC expose database capabilities in a standardized way to the application programmer through an application programming interface (API). In contrast to Embedded SQL, ODBC and JDBC allow a single executable to access different DBMSs Without recompilation.

The architecture of JDBC has four main components: the application, the driver manager, several data source specific drivers, and the corresponding data SOUTces.

JDBC Driver Management

The following Java example code explicitly loads a JDBC driver:

Class.forName("oracle/jdbc.driver.OracleDriver");

There are two other ways of registering a driver. We can include the driver with -jdbc.drivers=oracle/jdbc.driver at the command line when we start the Java application.

Connections

A session with a data source is started through creation of a Connection object; A connection identifies a logical session with a data source; multiple connections within the same Java program can refer to different data sources or the same data source. Connections are specified through a JDBC URL, a URL that uses the jdbc protocol. Such a URL has the form

jdbc:<subprotocol>:<otherParameters>

String uri = ..jdbc:oracle:www.bookstore.com:3083..

Connection connection;

try {

Connection connection = DriverManager.getConnection(urI,userId,password);

} catch(SQLException excpt)

{ System.out.println(excpt.getMessageO);

return;

}

Executing SQL Statements

The PreparedStatement class dynamically generates precompiled SQL statements that can be used several times; these SQL statements can have parameters, but their structure is fixed when the PreparedStatement object is created.

```
String sql = "INSERT INTO Books VALUES('?, 7, '?, ?, 0, 7)";
PreparedStatement pstmt = con.prepareStatement(sql);
pstmt.clearParameters();
pstmt.setString(l, isbn);
 pstmt.setString(2, title);
pstmt.setString(3, author);
pstmt.setFloat(5, price); pstmt.setInt(6, year);
int numRows = pstmt.executeUpdate();
```

ResultSets

The statement executeQuery returns a, ResultSet object, which is similar to a cursor.

ResultSet rs=stmt.executeQuery(sqlQuery); / / rs is now a cursor / / first call to rs.nextO moves to the first record / /

```
rs.nextO moves to the next row String sqlQuery;
 ResultSet rs = stmt.executeQuery(sqlQuery)
while (rs.next())
{ / / process the data }
```

Exceptions and Warnings

Similar to the SQLSTATE variable, most of the methods in java. sql can throw an exception of the type SQLException if an error occurs. The information includes SQLState, a string that describes the error

```
try { stmt = con.createStatement();
warning = con.getWarnings();
while( warning != null)
{  / / handle SQLWarnings / / code to process warning
```

```
 warning = warning.getNextWarning(); / /get next warning
} con.clearWarnings();
stmt.executeUpdate( queryString );
warning = stmt.getWarnings();
while( warning != null)
{ / / handle SQLWarnings / / code to process warning
warning = warning.getNextWarning(); / /get next warning
}
} / / end try
catch ( SQLException SQLe)
{ / / code to handle exception
} / / end catch
```

Using a ResultSet Object While next () allows us to retrieve the logically next row in the query answer, we can move about in the query answer in other ways too:

• previous moves back one row.

• absolute (int num) moves to the row with the specified number.

• relative(int num) moves forward or backward (if num is negative) relative to the current position.

relative(-1) has the same effect as previous.

• first 0 moves to the first row, and last0 moves to the last row.

10. Define SQLJ. Explain the Process of declaring the SQLJ objects and write the SQLJ code for displaying the data from the table named STUDENT.

SQLJ (short for 'SQL-Java') was developed by the SQLJ Group, a group of database vendors and Sun. SQLJ was developed to complement the dynamic way of creating queries in JDBC with a static model. It is therefore very close to Embedded SQL.

Unlike JDBC, having semi-static SQL queries allows the compiler to perform SQL syntax checks, strong type checks of the compatibility of the host variables with the respective SQL attributes, and consistency of the query with the database schema-tables, attributes, views, and stored procedures--all at compilation time.

#sql books = { SELECT usn, name INTO :rollno, :sname FROM student WHERE phone= :mobile };

In JDBC, we can dynamically decide which host language variables will hold the query result. In the following example, we read the title of the book into variable ftitle if the book was written by Feynman, and into variable otitle otherwise:

```
/ / assume we have a ResultSet cursor rs author = rs.getString(3);
if (name==”kumar")
{
sname = rs.getString(2):
 }
else
{
lname = rs.getString(2);
}
```

SQLJ Code

```
String sname,lname, phone;
#sql iterator STUDENTS (String sname, String phone);
STUDENTS students;
/ / the application sets the author
/ / execute the query and open the cursor
 #sql students = { SELECT sname, phone INTO :sname, :phone FROM Students WHERE sname= :sname };
/ / retrieve results
while (students.next())
{
System.out.println(students.sname ()+ ", " + students.phone()); }
students.close();
```

The corresponding JDBC code fragment looks as follows (assuming we also declared phone, aname, and lname:

```
PrcparedStatcment stmt = connection.prepareStatement( "SELECT sname,phone FROM Students WHERE lname = ?");
/ / set the parameter in the query  execute it
stmt.setString(1, lname);
ResultSet  rs = stmt.executeQuery();
/ / retrieve the results
while (rs.next())
{
System.out.println(rs.getString(l) + ", " + rs.getString(2));
}
```

Comparing the JDBC and SQLJ code, we see that the SQLJ code is much easier to read than the JDBC code. Thus, SQLJ reduces software development and maintenance costs.

Let us consider the individual components of the SQLJ code in more detail. All SQLJ statements have the special prefix #sql. In SQLJ, we retrieve the results of SQL queries with iterator objects, which are basically cursors. An iterator is an instance of an iterator class. Usage of an iterator in SQLJ goes through five steps:

- Declare the Iterator Class: In the preceding code, this happened through the statement #sql iterator Books (String title, Float price); This statement creates a new Java class that we can use to instantiate objects.
- Instantiate an Iterator Object from the New Iterator Class: We instantiated our iterator in the statement Books books;.
- Initialize the Iterator Using a SQL Statement: In our example, this happens through the statement #sql books ;;;;;; ....
- Iteratively, Read the Rows From the Iterator Object: This step is very similar to reading rows through a ResultSet object in JDBC.
- Close the Iterator Object.

11. Describe the CASE STUDY implementing a online Internet book Shop web application with database and list of functionalities.

The application logic will be implemented in Java, and so they consider JDBC and SQLJ as possible candidates for interfacing the database system with application code.

Recall that Database settled on the following schema:

Books(isbn: CHAR(10), title: CHAR(8), author: CHAR(80), qty_in_stock: INTEGER, price: REAL, year_published: INTEGER)

Customers(cid: INTEGER, cname: CHAR(80), address: CHAR(200))

Orders(ordernum: INTEGER, isbn: CHAR(lO), cid: INTEGER, cardnum: CHAR(l6), qty: INTEGER, order_date: DATE, ship_date: DATE)

Now, Database considers the types of queries and updates that will arise. They first create a list of tasks that will be performed in the application. Tasks performed by customers include the following.

- Customers search books by author name, title, or ISBN.
- Customers register with the website. Registered customers might want to change their contact information. Database realize that they have to augment the Customers table with additional information to capture login and password information for each customer; we do not discuss this aspect any further.
- Customers check out a final shopping basket to complete a sale.
- Customers add and delete books from a 'shopping basket' at the website.
- Customers check the status of existing orders and look at old orders.
- Administrative ta.'3ks performed by employees of B&N are listed next.
- Employees look up customer contact information.
- Employees add new books to the inventory.
- Employees fulfill orders, and need to update the shipping date of individual books.
- Employees analyze the data to find profitable customers and customers likely to respond to special marketing campaigns.

Next database consider the types of queries that will arise out of these tasks. To support searching for books by name, author, title, or ISBN, database decide to write a stored procedure as follows:

Database Application Development

CREATE PROCEDURE SearchByISBN (IN book.isbn CHAR (10)) SELECT B.title, B.author, B.qty_instock,B.price, B.yearpublished FROM Books B WHERE B.isbn = book.isbn

Placing an order involves inserting one or more records into the Orders table. Since Database has not yet chosen the Java-based technology to program the application logic, they assume for now that the individual books in the order are stored at the application layer in a Java array. To finalize the order, they write the following JDBC code shown in Figure 6.11, which inserts the elements from the array into the Orders table. Note that this code fragment assumes several Java variables have been set beforehand.

```
String sql = "INSERT INTO Orders VALUES(?, ?, ?, ?, ?, ?)";
PreparedStatement pstmt = con.prepareStatement(sql);
con.setAutoCommit(false);
try {
 / / orderList is a vector of Order objects
 / / ordernum is the current order number
 / / dd is the ID of the customer, cardnum is the credit card number
 for (int i=0; iiorderList.length(); i++)

/ / now instantiate the parameters with values Order
{
currentOrder = orderList[i];
pstmt.clearParameters() ;
pstmt.setInt(l, ordernum);
pstmt.setString(2, Order.getlsbn());
pstmt.setInt(3, dd);
 pstmt.setString(4, creditCardNum);
pstmt.setlnt(5, Order.getQty());
```

```
pstmt.setDate(6, null);
pstmt.executeUpdate());
}
con.commit();
}catch (SQLException e)
{ con.rollback();
 System.out.println(e.getMessage()); }
```

Inserting a Completed Order into the Database

```
}
```

Database writes other JDBC code and stored procedures for all of the remaining tasks. They use code similar to some of the fragments that we have seen in this chapter.

- Establishing a connection to a database
- Adding new books to the inventory
- Processing results from SQL queries
- For each customer, showing how many orders he or she has placed. We showed a sample stored procedure for this query
- Increasing the available number of copies of a book by adding inventory,
- Ranking customers according to their purchases,

- Database application takes care to make the application robust by processing exceptions and warnings, as shown Database application also decide to write a trigger. Whenever a new order is entered into the Orders table, it is inserted with ship-date set to NULL. The trigger processes each row in the order and calls the stored procedure 'UpdateShipDate'. This stored procedure (whose code is not shown here) updates the (anticipated) ship_date of the new order to 'tomorrow', in case qty in stock of the corresponding book in the Books table is greater than zero. Otherwise, the stored procedure sets the ship_date to two weeks.

```
CREATE TRIGGER update_ShipDate AFTER INSERT ON Orders FOR EACH ROW
BEGIN CALL UpdatcShipDate(new);
END
```

12. **Draw the neat diagram of single, two, three tier architecture diagrams. Explain the functional components in those architectures.**

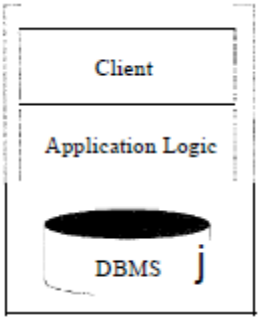
Data-intensive Internet applications can be understood in terms of three different functional components: data management, application logic, and presentation

Single-Tier and Client-Server Architectures

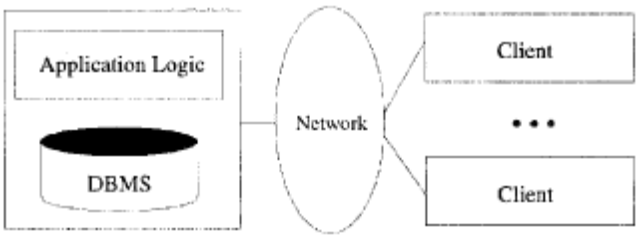
Single-tier architectures have a,n important drawback: Users expect graphical interfaces that require much more computational power than simple dumb terminals. Centralized computation of the graphical displays of such interfaces requires much more computational power than a single server have available, and thus single-tier architectures do not scale to thousands of users. The commoditization of the PC and the availability of cheap client computers led to the development of the two-tier architecture

Two-tier architectures, often also referred to a client-server architectures, consist of a client computer and a server computer, which interact through a well-defined protocol. What part of the functionality the client implements, and what part is left to the server, can vary. In the traditional client-server architecture, the client implements just the graphical user interface, and the server.

The thick-client model has several disadvantages when compared to the thin client model. First, there is no central place to update and maintain the business logic, since the application code runs at many client sites. Second, a large amount of trust is required between the server and the clients.



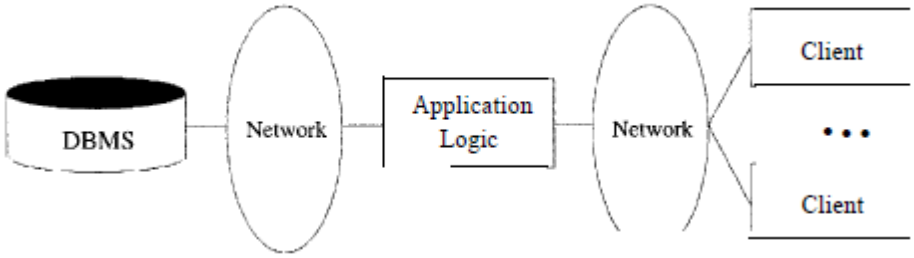
Single Tier



Two Tier

Three Tier Architectures

- Presentation Tier: Users require a natural interface to make requests, provide input, and to see results. The widespread use of the Internet has made Web-based interfaces increasingly popular.
- Middle Tier: The application logic executes here. An enterprise-class application reflects complex business processes, and is coded in a general purpose language such as C++ or Java.
- Data Management Tier: Data-intensive Web applications involve DBMSs, which are the subject of this book.



At the presentation layer, we need to provide forms through which the user can issue requests, and display responses that the middle tier generates.

The middle layer runs code that implements the business logic of the application: It controls what data needs to be input before an action can be executed, determines the control flow between multi-action steps, controls access to the database layer, and often assembles dynamically generated HTML pages from database query results.

Advantages of the Three-Tier Architecture

Heterogeneous Systems: Applications can utilize the strengths of different platforms and different software components at the different tiers. It is easy to modify or replace the code at any tier without affecting the other tiers.

- Thin Clients: Clients only need enough computation power for the presentation layer. Typically, clients are Web browsers.
- Integrated Data Access: In many applications, the data must be accessed from several sources. This can be handled transparently at the middle tier, where we can centrally manage connections to all database systems involved.
- Scalability to Many Clients: Each client is lightweight and all access to the system is through the middle tier. The middle tier can share database connections across clients, and if the middle tier becomes the bottle-neck, we can deploy several servers executing the middle tier code;

13. List of Client tier technologies used in presentation Layer. Describe with suitable Example.

HTML Forms

HTML forms are a common way of communicating data from the client tier to the middle tier. The general format of a form is the following:

```
<FORM ACTION="page.jsp" METHOD="GET" NAME="LoginForm">
</FORM>
```

A single HTML document can contain more than one form. Inside an HTML form, we can have any HTML tags except another FORM element.

The FORM tag has three important attributes:

- ACTION: Specifies the URI of the page to which the form contents are submitted; if the ACTION attribute is absent, then the URI of the current page is used. In the sample above, the form input would be submitted to the page named page. j sp, which should provide logic for processing the input from the form

- METHOD: The HTTP/1.0 method used to submit the user input from the filled-out form to the web server. There are two choices, GET and POST; we postpone their discussion to the next section.
- NAME: This attribute gives the form a name. Although not necessary, naming forms is good style. The client-side programs in JavaScript that refer to forms by name and perform checks on form fields.

Inside HTML forms, the INPUT, SELECT, and TEXTAREA tags are used to specify user input elements; a form can have many elements of each type. The simplest user input element is an INPUT field, a standalone tag with no terminating tag. An example of an INPUT tag is the following:

<INPUT TYPE=”text" NAME="title">

Passing Arguments to Server-Side Scripts

Using the GET method gives users the opportunity to bookmark the page with the constructed URI and thus directly jump to it in subsequent sessions; this is not possible with the POST method. The choice of GET versus POST should be determined' by the application and its requirements

JavaScript

JavaScript is often used for the following types of computation at the client:

- Browser Detection: JavaScript can be used to detect the browser type and load a browser-specific page.
- Form Validation: JavaScript is used to perform simple consistency checks on form fields. For example, a JavaScript program might check whether form input that asks for an email address contains the character '@,' or if all required fields have been input by the user.
- Browser Control: This includes opening pages in customized windows; examples include the annoying pop-up advertisements that you see at many websites, which are programmed using JavaScript.

JavaScript is usually embedded into an HTML document with a special tag, the SCRIPT tag.

```
<SCRIPT LANGUAGE=" JavaScript">
 alert(" Welcome to our bookstore");
 </SCRIPT
```

Style Sheets

```
BODY {BACKGROUND-COLOR: yellow}
H1 {FONT-SIZE: 36pt}
H3 {COLOR: blue}
P {MARGIN-LEFT: 50px; COLOR: red}
```

The use of style sheets has many advantages. First, we can reuse the same document many times and display it differently depending on the context. Second, we can tailor the display to the reader's preference such as font size, color style, and even level of detail. Third, we can deal with different output formats, such as different output devices.

14. Explain the informal guidelines used as a measures to determine the quality of relation schema design.

Informal Design Guidelines for Relation Schemas

Informal guidelines that may be used as measures to determine the quality of relation schema design:

- Making sure that the semantics of the attributes is clear in the schema
- Reducing the redundant information in tuples
- Reducing the NULL values in tuples
- Disallowing the possibility of generating spurious tuples

Imparting Clear Semantics to Attributes in Relations

The semantics of a relation refers to its meaning resulting from the interpretation of attribute values in a tuple

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. Intuitively, if a relation schema corresponds to one entity type or one relationship type, it is straightforward to explain its meaning. Otherwise, if the relation

corresponds to a mixture of multiple entities and relationships, semantic ambiguities will result and the relation cannot be easily explained.

Redundant Information in Tuples and Update Anomalies

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, and modification anomalies.

- Mixing attributes of multiple entities may cause problems
- Information is stored redundantly wasting storage
- Problems with update anomalies
  - 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.

NULL Values in Tuples

i. Attributes that are NULL frequently could be placed in separate relations (with the primary key)

ii. Reasons for nulls:

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

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

Generation of 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

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.

15. Define Normal Form. Explain 1NF, 2NF, 3NF and Boyce Codd Normal Forms with suitable Example.

- Normalization: The process of decomposing unsatisfactory "bad" relations by breaking up their attributes into smaller relations
- Normal form: Condition using keys and FDs of a relation to certify whether a relation schema is in a particular normal form

  – A superkey of a relation schema $R = \{A_1, A_2, ..., A_n\}$ is a set of attributes S *subset-of* R with the property that no two tuples t_1 and t_2 in any legal relation state r of R will have $t_1[S] = t_2[S]$

  – A **key** K is a superkey with the *additional property* that removal of any attribute from K will cause K not to be a superkey any more.

First Normal Form

Disallows composite attributes, multivalued attributes, and **nested relations**; attributes whose values *for an individual tuple* are non-atomic

EmpNum	EmpPhone	EmpDegrees
123	233-9876	
333	233-1231	BA, BSc, PhD
679	233-1231	BSc, MSc

Employee

EmpNum	EmpPhone
123	233-9876
333	233-1231
679	233-1231

EmployeeDegree

EmpNum	EmpDegree
333	BA
333	BSc
333	PhD
679	BSc
679	MSc

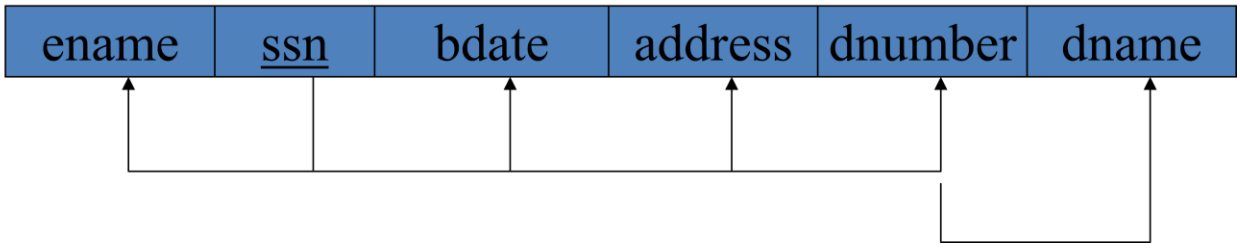
Second Normal Form

- Uses the concepts of **FD**s, **primary key**

Definitions:

- **Prime attribute** - attribute that is member of the primary key K
- **Full functional dependency** - a FD Y -> Z where removal of any attribute from Y means the FD does not hold any more
- A relation schema R is in **second normal form** (**2NF**) if every non-prime attribute A in R is fully functionally dependent on the primary key
- R can be decomposed into 2NF relations via the process of 2NF normalization

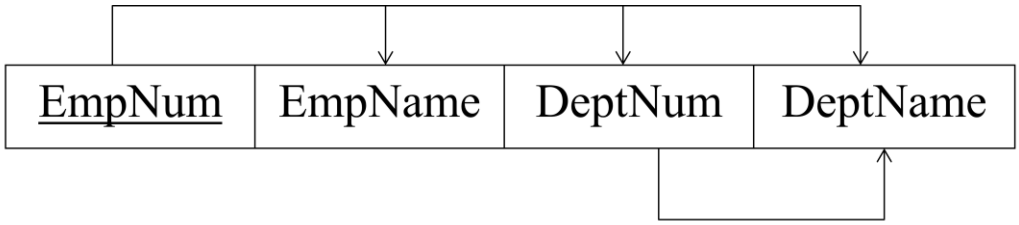
EmployeeDept



Third Normal Form

- **Transitive functional dependency** - a FD X -> Z that can be derived from two FDs X -> Y and Y -> Z
- 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

– R can be decomposed into 3NF relations via the process of 3NF normalization.



We correct the situation by decomposing the original relation into two 3NF relations. Note the decomposition is *lossless*.

BCNF

– A relation schema R is in **Boyce-Codd Normal Form** (**BCNF**) if whenever an FD X -> A holds in R, then X is a superkey of R

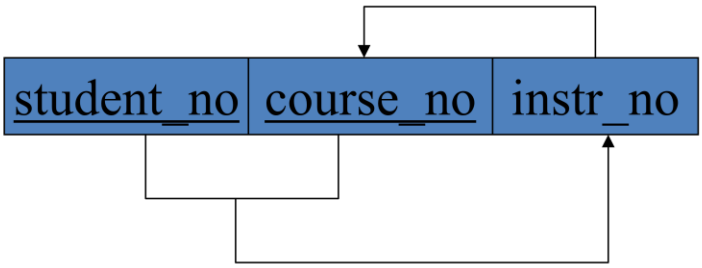
– Each normal form is strictly stronger than the previous one

  - Every 2NF relation is in 1NF
  - Every 3NF relation is in 2NF
  - Every BCNF relation is in 3NF

– There exist relations that are in 3NF but not in BCNF

– The goal is to have each relation in BCNF (or 3NF)

In 3NF, but not in BCNF:



Instructor teaches one course only.

Student takes a course and has one instructor.

{student_no, course_no} → instr_no
instr_no → course_no

since we have instr_no → course-no, but instr_no is not a Candidate key.

16. Write the Short Note on

i)Triggers

- Objective: to monitor a database and take action when a condition occurs
- Triggers are expressed in a syntax similar to assertions and include the following:
  - event (e.g., an update operation)
  - condition
  - action (to be taken when the condition is satisfied)
- A trigger to compare an employee's salary to his/her supervisor during insert or update operations:

```
CREATE TRIGGER INFORM_SUPERVISOR
BEFORE INSERT OR UPDATE OF
        SALARY, SUPERVISOR_SSN ON EMPLOYEE
```

```
FOR EACH ROW
        WHEN
        (NEW.SALARY> (SELECT SALARY FROM EMPLOYEE
         WHERE SSN=NEW.SUPERVISOR_SSN))
        INFORM_SUPERVISOR (NEW.SUPERVISOR_SSN,NEW.SSN;
```

ii)Stored Procedures

Stored procedures are also beneficial for software engineering reasons. Once a stored procedure is registered with the database server, different users can re-use the stored procedure, eliminating duplication of efforts in writing SQL queries or application logic, and making code maintenance easily

```
CREATE PROCEDURE ShowNumberOfOrders
SELECT C.cid, C.cname, COUNT(*)
FROM Customers C, Orders a
WHERE C.cid = O.cid
GROUP BY C.cid, C.cname
```

Stored procedures can also have parameters. These parameters have to be valid SQL types, and have one of three different modes: IN, OUT, or INOUT. IN parameters are arguments to' the stored procedure. OUT parameters are returned from the stored procedure; it assigns values to all OUT parameters that the user can process. INOUT parameters combine the properties of IN and OUT parameters: They contain values to be passed to the stored procedures, and the stored procedure can set their values as return values.

```
CREATE PROCEDURE Addlnventory (
IN book_isbn CHAR(lO),
IN addedQty INTEGER)
UPDATE Books
SET
WHERE
qty_in_stock = qtyjn_stock + addedQty
bookjsbn = isbn
```

iii)Nested Queries

- The nested query selects the number of the 'Research' department
- The outer query select an EMPLOYEE tuple if its DNO value is in the result of either nested query
- The comparison operator **IN** compares a value v with a set (or multi-set) of values V, and evaluates to **TRUE** if v is one of the elements in V
- In general, we can have several levels of nested queries
- A reference to an *unqualified attribute* refers to the relation declared in the *innermost nested query*
- In this example, the nested query is *not correlated* with the outer query
- If a condition in the WHERE-clause of a *nested query* references an attribute of a relation declared in the *outer query* , the two queries are said to be *correlated*
- The result of a correlated nested query is *different for each tuple (or combination of tuples) of the relation(s) the outer query*

```
SELECT      E.FNAME, E.LNAME
    FROM        EMPLOYEE AS E
    WHERE       E.SSN IN (SELECT   ESSN
                FROM DEPENDENT
                WHERE       ESSN=E.SSN AND
                        E.FNAME=DEPENDENT_NAME
```

iv) SQL JIONs

- Can specify a "joined relation" in the FROM-clause
- Looks like any other relation but is the result of a join

Allows the user to specify different types of joins (regular "theta" JOIN, NATURAL JOIN, LEFT OUTER JOIN, RIGHT OUTER JOIN, CROSS JOIN, etc)

```
SELECT      E.FNAME, E.LNAME, S.FNAME, S.LNAME
FROM        EMPLOYEE E S
WHERE       E.SUPERSSN=S.SSN
```

can be written as:

```
SELECT    E.FNAME, E.LNAME, S.FNAME, S.LNAME
    FROM      (EMPLOYEE E LEFT OUTER JOIN EMPLOYEES
        ON  E.SUPERSSN=S.SSN)

SELECT    FNAME, LNAME, ADDRESS
    FROM      EMPLOYEE, DEPARTMENT
    WHERE     DNAME='Research' AND DNUMBER=DNO
```

v) Views

- A view is a "virtual" table that is derived from other tables
- Allows for limited update operations (since the table may not physically be stored)
- Allows full query operations
- A convenience for expressing certain operations
- SQL command: CREATE VIEW
    - a table (view) name
    - a possible list of attribute names (for example, when arithmetic operations are specified or when we want the names to be different from the attributes in the base relations)
    - a query to specify the table contents

```
CREATE View WORKS_ON_NEW AS
    SELECT FNAME, LNAME, PNAME, HOURS
        FROM EMPLOYEE, PROJECT, WORKS_ON
    WHERE SSN=ESSN AND PNO=PNUMBER
        GROUP BY PNAME;
```

17 a) What are the ACID Properties? Explain with example

ACID properties

- **Atomicity**
  A transaction is an atomic unit of processing; it is either performed in its entirety or not performed at all.

- **Consistency preservation**
  A transaction is consistency preserving if its complete execution takes the database from one consistent state to another
- **Isolation**
  The execution of a transaction should not be interfered with by any other transactions executing concurrently
- **Durability**
  The changes applied to the database by a committed transaction must persist in the database. These changes must not be lost because of any failure

b) What is Concurrency? Explain the types of problem during occurs due concurrency.

DBMS has a *Concurrency Control* sub-system to assure database remains in consistent state despite concurrent execution of transactions

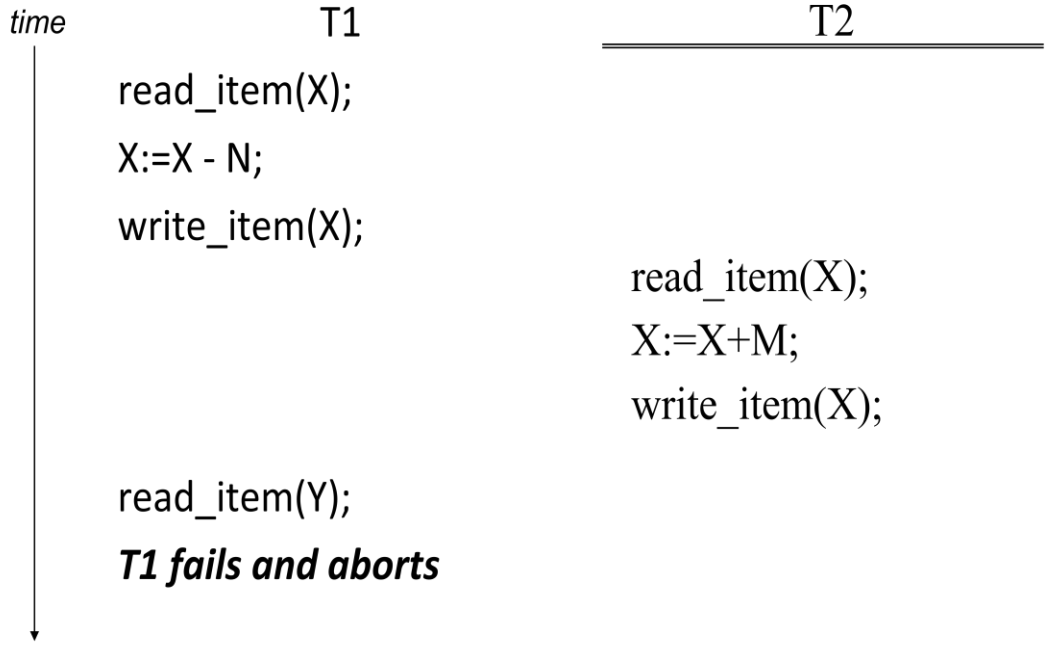
Three problems are occurs during concurrency

1. The lost update problem
2. The temporary update (dirty read) problem
3. Incorrect summary problem

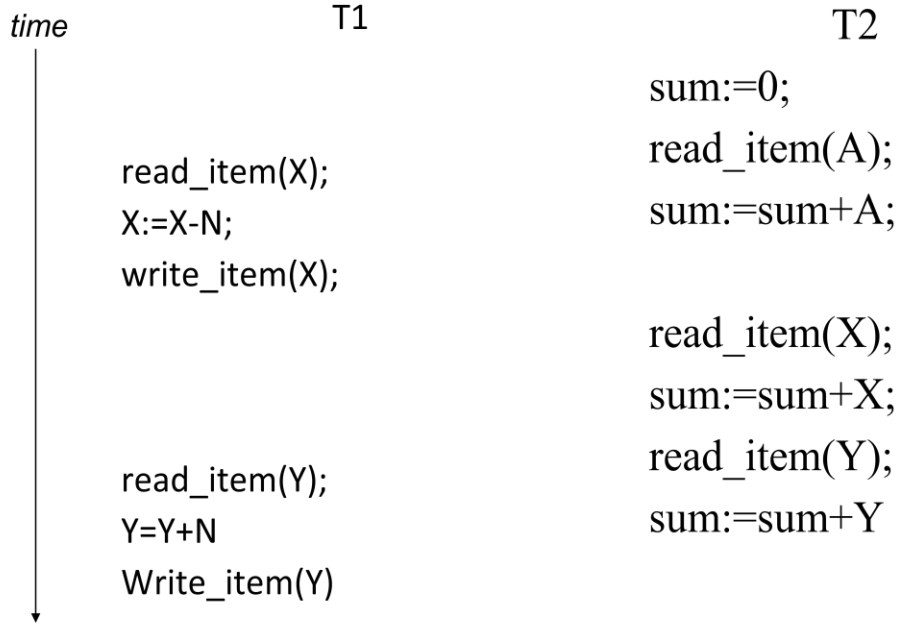
Lost Update Problem

time	T1	T2
	read_item(X);	
	X:=X - N;	
		read_item(X);
		X:=X+M;
	write_item(X);	
	read_item(Y);	
		write_item(X);
	Y:=Y + N;	
	write_item(Y);	

Temporary Update (Dirty Read)



Incorrect Summary Problem



18. Define lock. Write and explain the algorithm to control the concurrency using Two-Phase Locking Protocol.

Lock

- A variable associated with a data item
- Describes status of the data item with respect to operations that can be performed on it

Types of locks

- Binary locks
  - Locked/unlocked
  - Enforces mutual exclusion

Multiple-mode locks:

- Each data item can be in one of three lock states
  1. Read lock or shared lock
  2. Write lock or exclusive lock
  3. Unlock

Two-Phase Locking (2PL) Protocol

Transaction is said to follow the *two-phase-locking protocol* if all locking operations precede the *first* unlock operation

- Expanding (growing) phase
- Shrinking phase

During the shrinking phase no new locks can be acquired

- Downgrading ok
- Upgrading is not

2PL Example

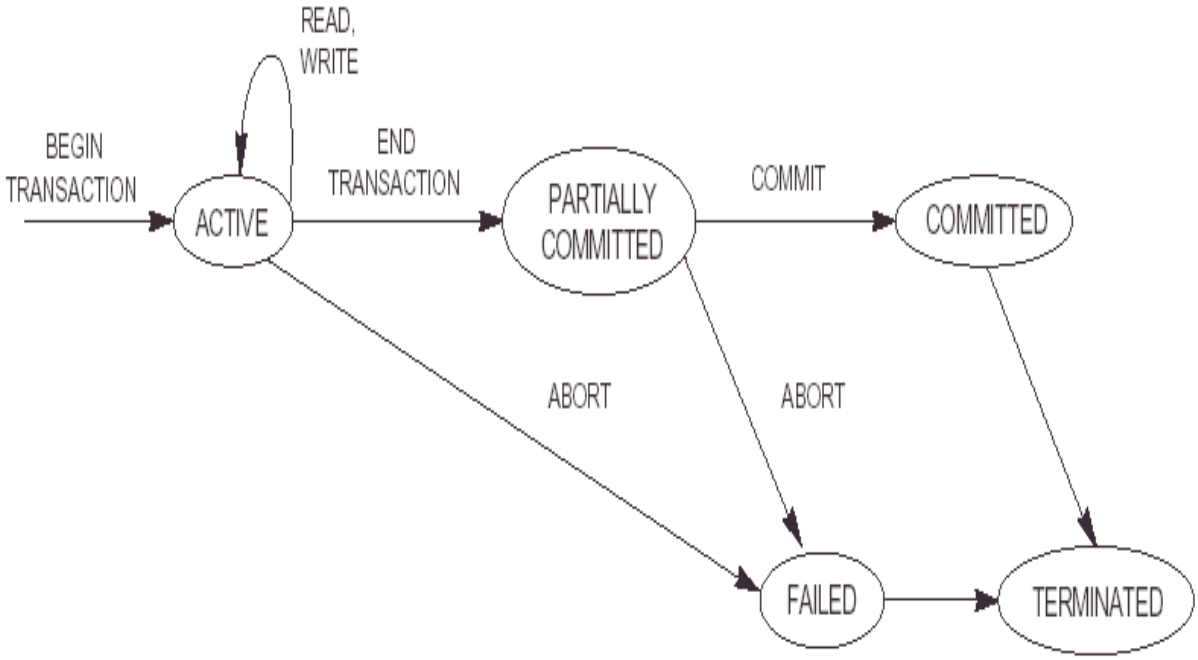
T1’	T2’
read_lock(Y);	d_lock(X);
read_item(Y);	d_item(X);
write_lock(X);	te_lock(Y);
unlock(Y);	ock(X);
read_item(X);	d_item(Y);
X:=X+Y;	=X+Y;
write_item(X);	te_item(Y);
unlock(X);	ock(Y);

- **Both T1’ and T2’ follow the 2PL protocol**
- **Any schedule including T1’ and T2’ is guaranteed to be serializable**
- **Limits the amount of concurrency**
- Two-phase locking protocol (2PL)
  All lock operations precede the first unlock operation
  - Expanding phase and shrinking phase
  - Upgrading of locks must be done in expanding phase and downgrading of locks must be done in shrinking phase
- If every transaction in a schedule follows 2PL protocol then the schedules is guaranteed to be serializable.
- Variants of 2PL
  Basic, conservative, strict, and rigorous

19 a) List the transaction states. Draw and explain state transition diagram of the transaction.

Transaction states

- BEGIN_TRANSACTION: marks start of transaction
- READ or WRITE: two possible operations on the data
- END_TRANSACTION: marks the end of the read or write operations; start checking whether everything went according to plan
- COMIT_TRANSACTION: signals successful end of transaction; changes can be "committed" to DB
- Partially committed
- ROLLBACK (or ABORT): signals unsuccessful end of transaction, changes applied to DB must be undone



b) Describe the list of steps include during the transaction read and write operations

- A database is represented as a collection of named data items
- Read-item (X)
    1. Find the address of the disk block that contains item *X*
    2. Copy the disk block into a buffer in main memory
    3. Copy the item X from the buffer to the program variable named X
- Write-item (X)
    1. Find the address of the disk block that contains item *X*.
    2. Copy that disk block into a buffer in main memory
    3. Copy item *X* from the program variable named *X* into its correct location in the buffer.
    4. Store the updated block from the buffer back to disk (either immediately or at some later point in time).

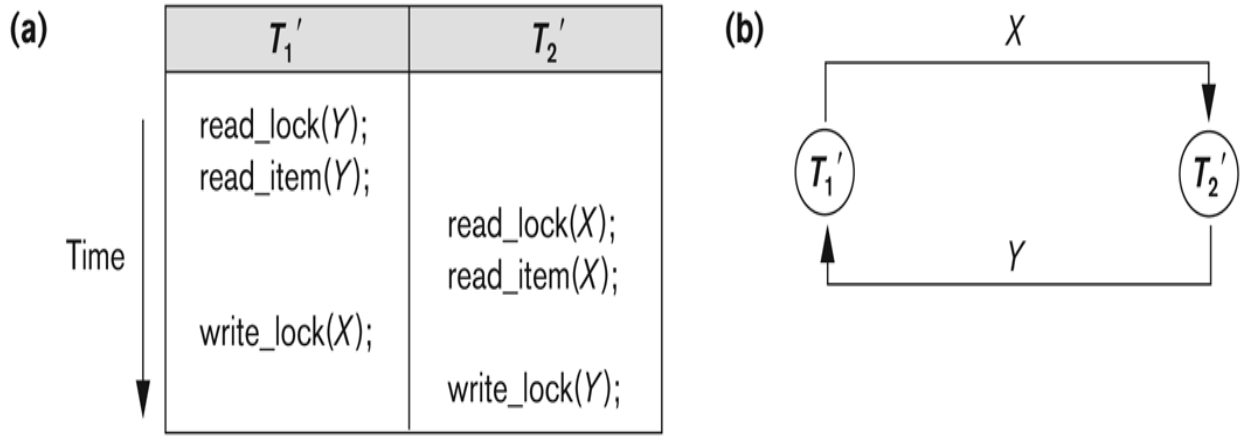
.

20. When the deadlock and Starvation occurs? Explain these problems can resolved.

Deadlocks and Starvation in 2PL

- 2PL can produce deadlocks
- Deadlock and starvation in 2PL

Deadlock occurs when each transaction T in a set of two or more transactions is waiting for some item that is locked by some other transaction T' in the set.



Illustrating the deadlock problem. (a) A partial schedule of T_1' and T_2' that is in a state of deadlock. (b) A wait-for graph for the partial schedule in (a).

Deadlock prevention protocols

- Conservative 2PL, lock all needed items in advance
- Ordering all items in the database

Possible actions if a transaction is involved in a possible deadlock situation

- Block and wait
- Abort and restart
- Preempt and abort another transaction

Two schemes that prevent deadlock (Timestamp based)

Wait-die

An older transaction is allowed to wait on a younger transaction whereas a younger transaction requesting an item from held by an older transaction is aborted and restarted with the same timestamp

Wound-wait

A younger transaction is allowed to wait on an older transaction whereas an older transaction requesting an item from held by a younger transaction preempts the younger transaction by aborting it.

Starvation
A transaction cannot proceed for an infinite period of time while other transactions in the system continue normally

- Unfair waiting scheme
- Victim selection

21. How to recover the transaction from failure? Explain the list of recovery concepts.

- Keeps information about operations made by transactions:
  - *Before-image* (*undo entry*) of updated items
  - *After-image* (*redo entry*) of updated items
- Enables restoring a consistent state after non-catastrophic failure (forward/backward).
- Alternatives:
  - *undo/no-redo*
  - *no-undo/redo*
  - *undo/redo*
  - *no-undo/no-redo.*

Write-Ahead Logging (WAL)

(1) No overwrite of disk data before *undo*-type
log records are forced to disk.
(2) *Both undo- and redo-type* log records
(= before- and after-images) must
be forced to disk before end of commit.

Backup

- Copy of database on archival storage
  (off-line, often on tape).
- Enables partial recovery from *catastrophic* failures:
  - For *committed* transactions:
    Load backup and apply redo operations from the
    log (if the log survived).
  - *Non-committed* transactions must be restarted
    (= re-executed).

Cache

- In-memory buffer for database pages.
- A *directory* (page table) keeps track of pages
  in cache.
- Page-replacement strategy needed, e.g.
  FIFO (First-In-First-Out), or
  LRU (Least Recently Used)
- *Dirty bit* tells for each page, if it has changed
- *Flushing* means (force-)writing buffer pages to disk.
- *Pin/unpin bit* tells if the page can be written

Rollback

- At failure, apply undo-type log records (before-images) to updated items.
- A recoverable schedule may allow *cascading rollback.*
- Most practical protocols avoid cascading rollbacks. Then no *read-entries* are required in the log (no dirty reads).

22. Explain the timestamp ordering techniques for concurrency control.

Timestamp --- a unique identifier created by the DBMS to identify a transaction

- read-TS(X)
  - The largest timestamp among all the timestamps of transactions that have successfully read item X
- write-TS(X)
  - The largest timestamp among all the timestamps of transactions that have successfully written item X

Timestamp Ordering (TO) algorithms

- Basic timestamp ordering
- Strict timestamp ordering
- Thomas's write rule

Basic timestamp ordering algorithm

Order transactions based on their timestamps

- T issues a write(X)
  - If read-TS(X) > TS(T) or if write-TS(X) > TS(T) the abort and rollback T and reject the operation.
  - If condition above does not occur then execute the operation and set write-TS(X) = TS(T)
- T issues a read(X)
  - If write-TS(X) > TS(T) then abort and rollback T and reject the operation.
  - If write-TS(X) ≤ TS(T) then execute the operation and set read-TS(X) = max (read-TS(X), TS(T))
- The schedules produced by basic TO are guaranteed to be conflict serializable
- No deadlocks but cyclic restart are possible (hence starvation)

Strict Timestamp Ordering (strict TO)

- A transaction T that issues a read-item(*X*) or write-item(*X*) such that TS(T) > write-TS(*X*) has its read or write operation *delayed* until the transaction T' that *wrote* the value of *X* (hence TS(T') = write-TS(*X*)) has committed or aborted.
- No deadlocks, since T waits for T' only if TS(T) > TS(T')
- **Strict TO** ensures that the schedules are both **strict** (for easy recoverability) and (conflict) serializable

Thomas's write rule

- It rejects fewer write operations, by modifying the basic TO checks for the write-item(*X*) operation as follows:
  1. If read-TS(*X*) > TS(T), then abort and roll back T and reject the operation.

2. If write-TS(X) > TS(T), then do not execute the write operation but continue processing.
   [This is because some transaction with timestamp greater than TS(T)—and hence after T in the timestamp ordering—has already written the value of X. Hence, we must ignore the write_item(X) operation of T because it is already outdated and obsolete. Notice that any conflict arising from this situation would be detected by case (1).]
3. If neither the condition in part (1) nor the condition in part (2) occurs, then execute the write-item(X) operation of T and set write-TS(X) to TS(T).

Thomas' write rule does not enforce conflict serializability

23.Define 4NF and 5NF. Explain with suitable example.

Fourth Normal Form (4NF)

Fourth Normal Form comes into picture when **Multi-valued Dependency** occur in any relation. In this tutorial we will learn about Multi-valued Dependency, how to remove it and how to make any table satisfy the fourth normal form.

A relation schema R is in **4NF** with respect to a set of dependencies F (that includes functional dependencies and multivalued dependencies) if, for every *nontrivial* multivalued dependency X —>> Y in F^+, X is a superkey for R.

c)The relation SUPPLY with no MVDs is in 4NF but not in 5NF if it has the JD(R1, R2, R3). (d) Decomposing the relation SUPPLY into the 5NF relations R1, R2, and R3
For a table to satisfy the Fourth Normal Form, it should satisfy the following two conditions:

(c) **SUPPLY**

SNAME	PARTNAME	PROJNAME
Smith	Bolt	ProjX
Smith	Nut	ProjY
Adamsky	Bolt	ProjY
Walton	Nut	ProjZ
Adamsky	Nail	ProjX
Adamsky	Bolt	ProjX
Smith	Bolt	ProjY

(d) **R1**

SNAME	PARTNAME
Smith	Bolt
Smith	Nut
Adamsky	Bolt
Walton	Nut
Adamsky	Nail

R2

SNAME	PROJNAME
Smith	ProjX
Smith	ProjY
Adamsky	ProjY
Walton	ProjZ
Adamsky	ProjX

R3

PARTNAME	PROJNAME
Bolt	ProjX
Nut	ProjY
Bolt	ProjY
Nut	ProjZ
Nail	ProjX

- 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)):

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

· $t_3[Y] = t_1[Y]$ 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 $Y = R$.

s_id	**course**	**hobby**
1	Science	Cricket
1	Maths	Hockey
2	C#	Cricket
2	Php	Hockey

1. As you can see in the table above, student with s_id **1** has opted for two courses, **Science** and **Maths**, and has two hobbies, **Cricket** and **Hockey**.

Fifth normal form (5NF)

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), every R_i is a superkey of R.

SUBJECT	**LECTURER**	**SEMESTER**
Computer	Anshika	Semester 1
Computer	John	Semester 1
Math	John	Semester 1
Math	Akash	Semester 2
Chemistry	Praveen	Semester 1

- In the above table, John takes both Computer and Math class for Semester 1 but he doesn't take Math class for Semester 2. In this case, combination of all these fields required to identify a valid data.

- Suppose we add a new Semester as Semester 3 but do not know about the subject and who will be taking that subject so we leave Lecturer and Subject as NULL. But all three columns together acts as a primary key, so we can't leave other two columns blank.
- So to make the above table into 5NF, we can decompose it into three relations P1, P2 & P3:

Lossless Decomposition

- If the information is not lost from the relation that is decomposed, then the decomposition will be lossless.
- The lossless decomposition guarantees that the join of relations will result in the same relation as it was decomposed.
- The relation is said to be lossless decomposition if natural joins of all the decomposition give the original relation.

24. Write the short note on

i)Domain Key Normal form

- **Defintion:**A relation schema is said to be in **DKNF** if all constraints and dependencies that should hold on the valid relation states can be enforced simply by enforcing the domain constraints and key constraints on the relation.
- The **idea** is to specify (theoretically, at least) the "*ultimate normal form*" that takes into account all possible types of dependencies and constraints. .
- For a relation in DKNF, it becomes very straightforward to enforce all database constraints by simply checking that each attribute value in a tuple is of the appropriate domain and that every key constraint is enforced.
- The practical utility of DKNF is limited

ii)Template Dependency

- Template dependencies provide a technique for representing constraints in relations that typically have no easy and formal definitions.
- The idea is to specify a template—or example—that defines each constraint or dependency.
- There are two types of templates: tuple-generating templates and constraint-generating templates.

A template consists of a number of **hypothesis tuples** that are meant to show an example of the tuples that may appear in one or more relations. The other part of the template is the **template conclusion.**

Templates for some common types of dependencies.
(a) Template for functional dependency *X* –> *Y*.
(b) Template for the multivalued dependency *X* —>> *Y* .
(c) Template for the inclusion dependency *R.X* < *S.Y*.

(a)

$R = \{ A , B , C , D \}$

hypothesis

$a_1 \quad b_1 \quad c_1$ $\qquad X = \{A , B\}$

$a_1 \quad b_1 \quad c_2$ $\qquad Y = \{C , D\}$

conclusion

$c_1 = c_2$ and $d_1 = d_2$

(b)

$R = \{ A , B , C , D \}$

hypothesis

$a_1 \quad b_1 \quad 1 \quad d_1$ $\qquad X = \{A , B\}$

$a_1 \quad b_1 \quad 2 \quad d_2$ $\qquad Y = \{C\}$

conclusion

$a_1 \quad b_1 \quad 2 \quad d_1$

$a_1 \quad b_1 \quad 1 \quad d_2$

(c)

$R = \{ A , B , C , D \}$ $\qquad S = \{ E , F , G \}$ $\qquad X = \{C , D\}$

hypothesis

$a_1 \quad b_1 \quad c_1 \quad d_1$ $\qquad Y = \{E , F\}$

conclusion

$c_1 \quad d_1 \quad g$

iii) Inclusion Dependency

An **inclusion dependency** $R.X < S.Y$ between two sets of attributes—X of relation schema R, and Y of relation schema S—specifies the constraint that, at any specific time when r is a relation state of R and s a relation state of S, we must have

$$p_X(r(R)) \supseteq p_Y(s(S))$$

To formalize two types of interrelational constraints which cannot be expressed using F.D.s or MVDs:

- Referential integrity constraints
- Class/subclass relationships

- **Inclusion dependency inference rules**

  IDIR1 (**reflexivity**): $R.X < R.X$.

  IDIR2 (**attribute correspondence**): If $R.X < S.Y$ where $X = \{A_1, A_2, ..., A_n\}$ and $Y = \{B_1, B_2, ..., B_n\}$ and A_i *Corresponds-to* B_i, then $R.A_i < S.B_i$ for $1 \leq i \leq n$.

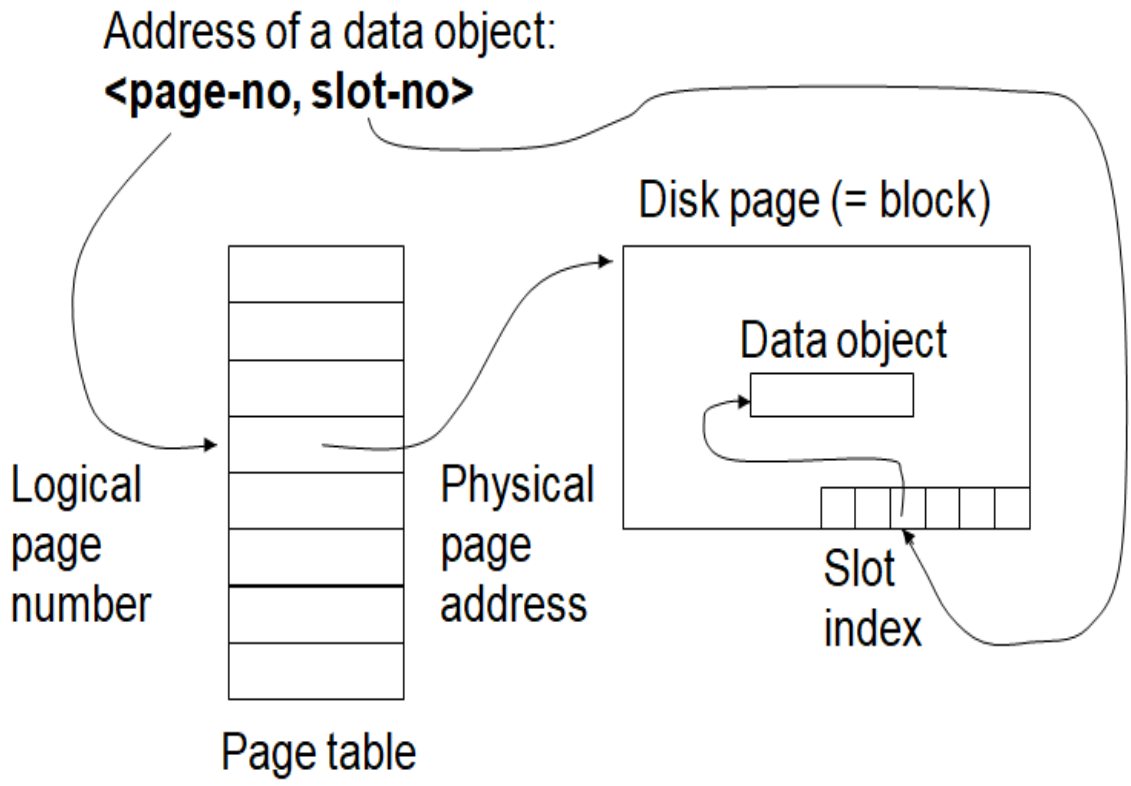
IDIR3 (**transitivity**): If *R.X* < *S.Y* and *S.Y* < *T.Z*, then *R.X* < *T.Z*.

iv) Shadow Paging

Shadow paging

Assumes an *indirect addressing* scheme:
References to database objects consist of
<*page-no*, *slot-no*> pairs, where *page-no* is
the index to a *page table (*'directory'), which contains the true physical addresses.

- Updated pages are written to *different* locations
- *Two* page tables are kept:
  - *Shadow* (pointing to original pages on disk)
  - *Current* (pointing to updated pages)



Advantages:

- Simple recovery, no log in single-user systems (in multi-user systems, logs and checkpoints needed for concurrency control).

Disadvantages:

- Fragmentation of storage (clustering lost).
- Writing the shadow table to disk takes time.
- Garbage collection of pages needed.

v) Immediate Update recovery

Immediate update

- Updated values from the buffer first to the log, then to the database (even before commit);
  WAL-rule
- The most common in practice.
- Rollback (undo) may occur
- Two alternatives:
  - All writes forced to disk before commit
    (UNDO/NO-REDO)
  - Some writes from buffer to disk after commit
    (UNDO/REDO)